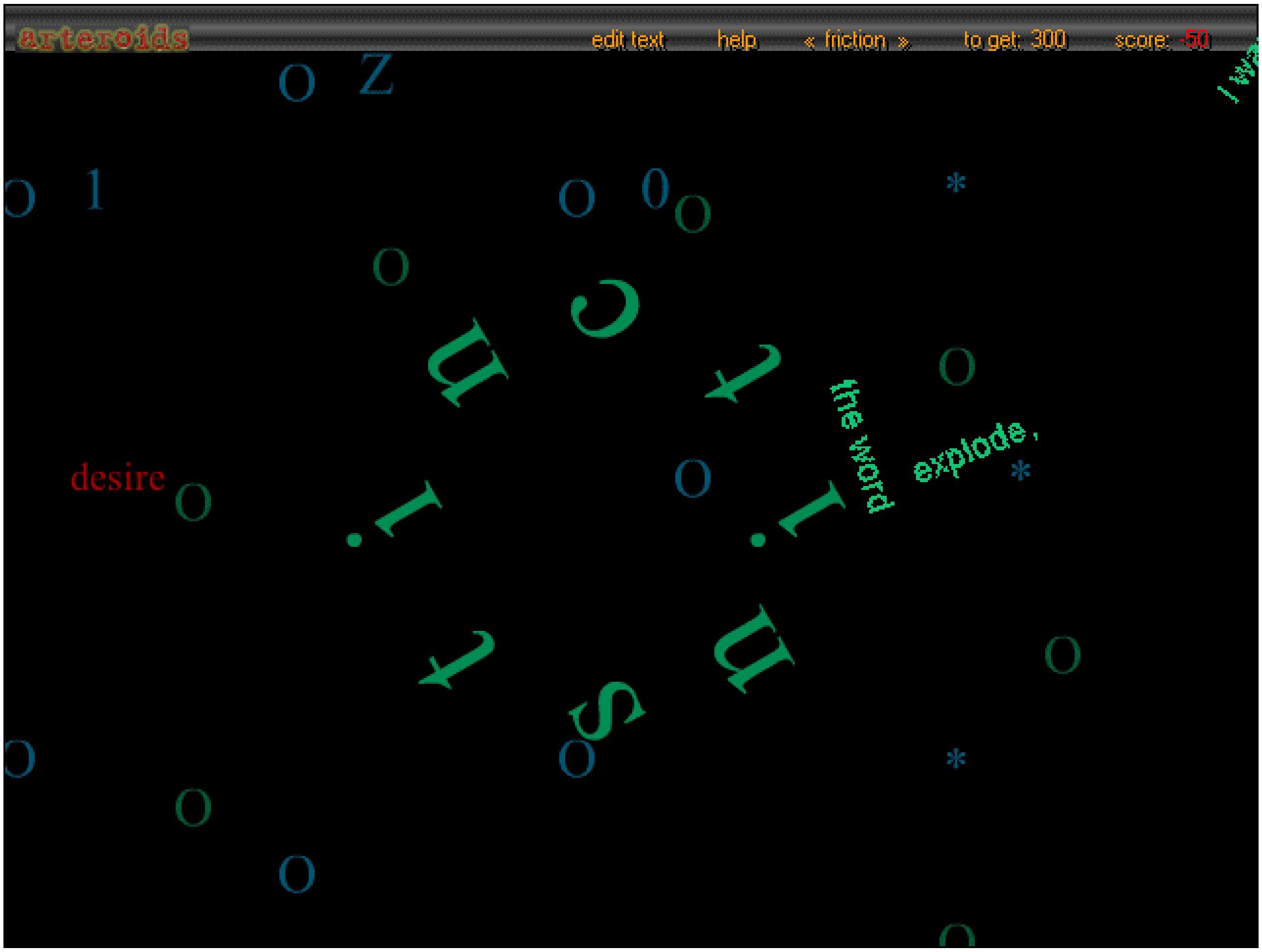
Arteroids

Version 1.0

Code Documentation



by Jim Andrews
www.vispo.com
jim@vispo.com

December 2001

Table of Contents

Arteroids: Overview and Flowchart

Reason for this doc

I put together this doc on the Arteroids code for a couple of reasons. I will probably try to add a few more levels to it at some point, and so would like to stand a chance of understanding the code when I return to it. But, also, I am releasing the code with the publication of Arteroids on www.theremediproject.com so that others can add levels or use it for parts themselves, and this doc will help you understand the code. It's a fairly extensive bit of code, so I thought some documentation beyond what's in the code itself would be helpful to both of us.

Also, if one is going to release code publicly, then it should be usable.

Structure of This Doc

If you look at the Table of Contents, you see that the progression of this doc is by channel. For each channel, I've provided the code for each corresponding sprite, and have broken it up into meaningful sections in the same way that the score is broken up into meaningful sections.

Conventions of the Program

Arteroids is such that there is a channel (horizontal row or 'layer' in Flash terminology) for each sprite in the movie just about always. In other words, there are hardly ever two sprites in the same channel.

Why? Well, for instance, if you make sprite A invisible, and then the movie continues, sprite A ends, and the same channel then contains sprite B, sprite B will also be invisible, unless you explicitly set it to be visible. And this general carryover of properties between sprites in the same channel can cause problems unless you make sure you set all the properties in an 'on beginsprite' handler. Unless you are quite sure you've done this, it is best to have only one sprite per channel for the duration of the movie.

So you have to scroll the score down to see the whole thing.

I have made Arteroids so that only movie scripts are not attached to some sprite onstage.

Note the three blue sprites named

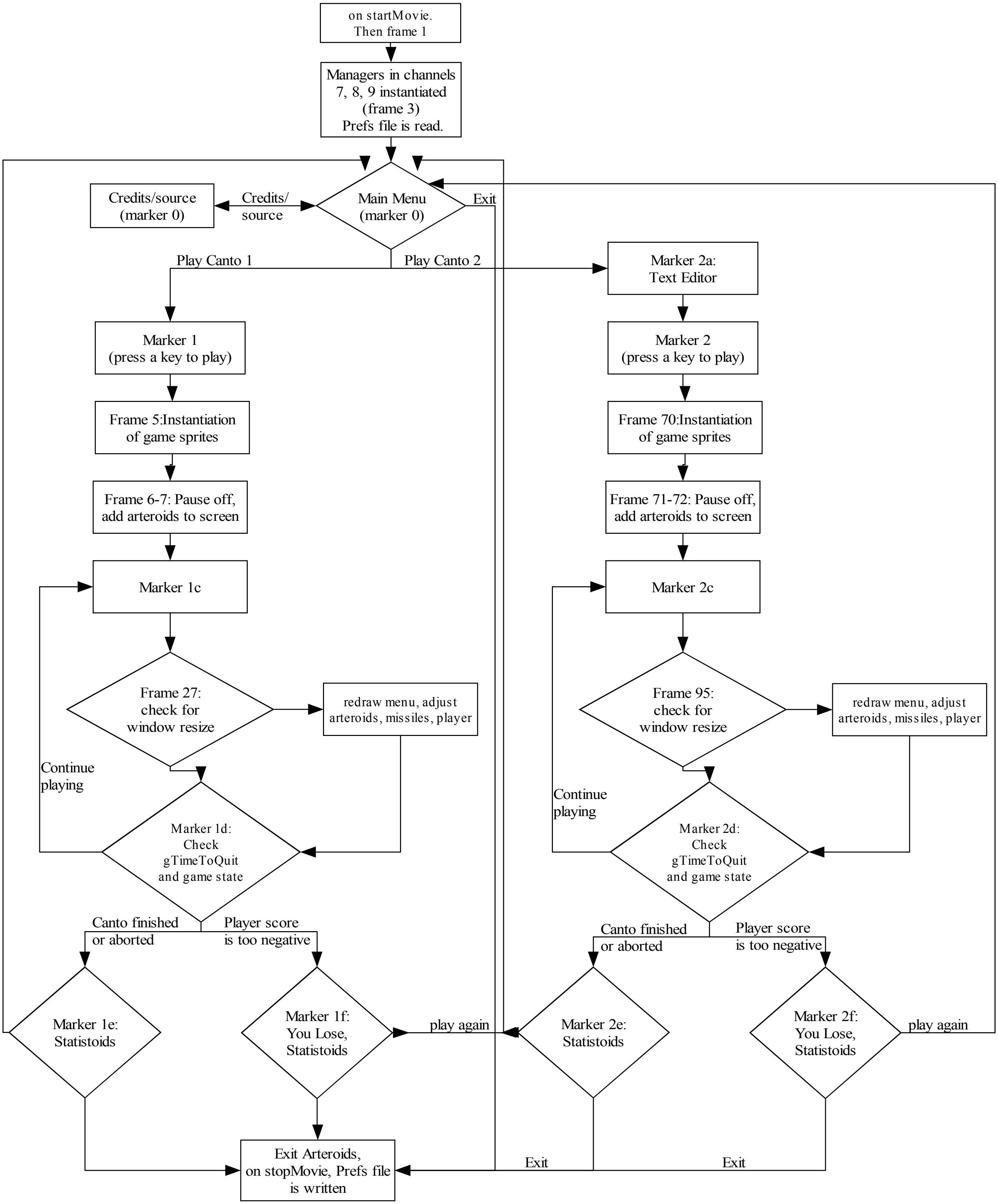
Score and Level Manager member (look above the stage to find me)
User Data Manager Member
Message Manager Member.

These three sprites, unlike the rest, extend basically from the beginning of the score to the end of the score, although the game has two levels. That is because these sprites have scripts attached to them that manage data throughout the entire game and need to be instantiated throughout the movie.

Some groups of sprites have managers attached to the first sprite in the group. For instance, the green exploding letters have a manager attached to the first green exploding letter, and so do the blue letters. The words (as opposed to exploding letters) have a 'Green and Blue Word Manager' attached to the first green word. So make a special point of checking both the first and second sprites in groups of sprites.

Score Flowchart

To read the code, you need to be familiar with how Shockwave 8.5 movies proceed. The flowchart on the next page describes the flow along the score/timeline. Have a look at the score in Director in conjunction with looking at the flowchart.


on startMovie.
Then frame 1
Managers in channels 7, 8, 9 instantiated (frame 3)
Prefs file is read.
Credits/source (marker 0)
Credits/ source
Main Menu (marker 0)
Exit
Play Canto 1
Play Canto 2
Marker 2a: Text Editor
Marker 1 (press a key to play)
Marker 2 (press a key to play)
Frame 5:Instantiation of game sprites
Frame 70:Instantiation of game sprites
Frame 6-7: Pause off, add arteroids to screen
Frame 71-72: Pause off, add arteroids to screen
Marker 1c
Marker 2c
Frame 27: check for window resize
redraw menu, adjust arteroids, missiles, player
Frame 95: check for window resize
redraw menu, adjust arteroids, missiles, player
Continue playing
Marker 1d: Check gTimeToQuit and game state
Continue playing
Marker 2d: Check gTimeToQuit and game state
Canto finished or aborted
Player score is too negative
Canto finished or aborted
Player score is too negative
Marker 1e: Statistoids
Marker 1f: You Lose, Statistoids
play again
Marker 2e: Statistoids
Marker 2f: You Lose, Statistoids
play again
Exit Arteroids, on stopMovie, Prefs file is written
Exit
Exit

Collision Detection

Collision detection in the program is implemented as follows. Check each onscreen arteroid each frame to see if it has collided with a missile. Stop collision detection for that arteroid as soon as you find it has been hit with a missile. There can be max 2 missiles onscreen at once, so you only have to check max 2 missiles for each onscreen arteroid.

This is done in the 'on prepareFrame' handler of the 'Green and Blue Word Manager' script, which is attached to the first green word.

Then, still concerning the same arteroid, check to see if it has collided with the player.

Note that the above collision detections must be performed. It would be a waste of CPU cycles to check to see if arteroids that aren't onstage have collided with missiles or the player, and it would be a waste of CPU cycles to check if missiles that aren't onstage collided with arteroids.

Green and Blue Explosions

When the player hits an arteroid with a missile, the arteroid explodes. What actually happens is that when the arteroid is hit, if it is a green arteroid, it contacts the Exploding Green Letter Manager (this script is attached to the first green exploding letter) and requests as many characters as are in the word (or phrase). There are 25 green exploding letters available, so it might not get all or any of the green letters it needs to explode with. Each exploding green letter is a Flash 4 SWF (the Green Exploding Letter member). Have a look at the documentation in the Green Exploding Letter Manager script and the Exploding Green Letter script, which is attached to each exploding green letter.

So the word is assigned as many available exploding green letters as it needs or are available, given that there can be more than one green explosion at a time, the phrases can be arbitrarily long, and there are only 25 green exploding letters available. More than that doesn't seem to work well for performance reasons. Since there are 25 green and 25 blue exploding letters, you can at times be changing 50 SWF on screen each frame. More than that is not a good idea until computers get much quicker. But it isn't really required, as you can see from the way the explosions appear onscreen now.

This request for green exploding letters originates in the Single Circle Green Asteroid 1 script in the 'on explode' handler, which is attached to each green arteroid. This handler also defines variables such as the total number of iterations the explosion happens over and the radius of explosion, the initial and terminal font size of the explosion, the initial and terminal blend values for the exploding letters and so on.

And of course flags are set to indicate that the word is now in the process of explosion.

For as many frames as are required to fulfill the number of iterations of the explosion, the Green and Blue Word Manager thereafter calls the 'on executeMyGeometry' handler in the Single Circle Green Asteroid 1 script to execute the animation of the explosion.

Once the explosion is finished, the exploding green letters involved in the explosion are returned to the Exploding Green Letter Manager to be used in subsequent explosions.

Much the same goes for the explosions of the blue words, only with other corresponding scripts.

I used Flash 4 letters for the explosions rather than Director text because the Flash letters animate quicker. It would be nice to have the words/phrases themselves be Flash animations rather than Director text for the same reason, ie, Director text animates slowly. But I don't know if this is possible. You would have to insert text into the SWF on the fly and you would have to size the SWF bounding rectangle so that the rectangle contained only the text and not blank space at the end of the word, otherwise collision detection would be such as to indicate collisions when there really shouldn't be any.

Pre-Game Code and Score Documentation

Startmovie Handler (Movie Script)

```
--********************************************************************************************

--ARTEROIDS by Jim Andrews, copyright 2001, based on code by Ian Clay.
--Go ahead and use this code, but please credit me, if you use any of this code, as I have credited Ian Clay
--and Michael Szabo: with a link to my site from within the piece itself in the
--Credits section. Please link to http://vispo.com.


--********************************************************************************************
--STARTMOVIE ETC

--This handler contains the 'on startMovie' handler that runs before other code. That handler
--just initializes some globals.

--It also contains some code that centers things on the screen, no big deal.

--The getURL handler is for the explosion of the id entity. The animations that display when
--the id entity explodes are Flash animations. These Flash 4 animations (members poetry3
--and poetry4) notify Director when they finish their last frame. This lets the program know
--that the explosion is finished and the id entity text can be placed back on stage.
--********************************************************************************************

global gMissile, gCurrentlyOnStageBullets, gCurrentlyOnStageBulletsCount, gCurrentLevel
global gBangMemberNumber, gBulletMemberNumber, gScoreAndLevelManager
global gAControlIsMousedDown, gWithinAControl, gCurrentlyEditedTextBox
global gFriction
global gUserDataManager
global gPlayerShipSpriteNum
global gStageWidth, gStageHeight

on StartMovie me
  gCurrentlyOnStageBullets=[]
  --Contains a list of the sprite numbers of bullets currently onstage
  gCurrentlyOnStageBulletsCount=0
  gMissile=[:]
  --This is the primary missile data storage structure
  gCurrentLevel=1
  --We always start out at level 1
  the floatPrecision = 2
  gAControlIsMousedDown=0
  gWithinAControl=FALSE
  --The above two are vars used for cursor control (see the Cursor Control script)
  gCurrentlyEditedTextBox=0
  --Indicates the currently edited text box in level 2
  gFriction=0.9
  --This determines the amount of friction on the player. I should change this to be read/written in the prefs file
  gStageWidth=the stageRight - the stageLeft
  gStageHeight=the stageBottom - the stageTop
end startMovie

on centerIt spriteToCenter
  --This centers a sprite on the stage. as in sprite(spritenum).loc=centerIt(spritenum)
  newH=(gStageWidth - (sprite(spriteToCenter).right-sprite(spriteToCenter).left))/2
  newV=(gStageHeight-(sprite(spriteToCenter).bottom-sprite(spriteToCenter).top))/2
  return point(newH, newV)
end centerIt

on centerH spriteToCenter
  --This centers a sprite horizontally, as in sprite(spritenum).locH=centerH(spritenum)
  return (gStageWidth - sprite(spriteToCenter).width)/2
end centerH


on stopMovie me
  --This causes the prefs file to be written to the user's hard disk on exit of arteroids.
  sprite(gUserDataManager).writePrefsFile()
end stopMovie


on getURL stringFromFlash
   --the poetry3 movie has reached the end
   sprite(gPlayerShipSpriteNum).pDeathLoopCount=sprite(gPlayerShipSpriteNum).pDeathLoopCount+1
   if sprite(gPlayerShipSpriteNum).pDeathLoopCount=1 then
     sprite(gPlayerShipSpriteNum).pDeathLoopCount=0
     sprite(gPlayerShipSpriteNum).iAmFinishedExploding()
   end if
end
```

Encryption (Movie Script)

This is Michael Szabo's code, not mine. There is a link in the Credits section of Arteroids to an article concerning this code. The link is http://www.director-online.com/buildArticle.cfm?id=1003. It is used to encrypt/decrypt the data stored on the player's hard disk. Thanks, Michael. This code is called in the User Data Manager behavior to decrypt/encrpt.

```
on EncryptString( argMsg, argKey )
  ----
  -- FUNCTION: EncryptString()
  ----
  -- Description:
  --   This function will modify a 'message' string based on
  --   the contents of a second 'key' string for the purposes
  --   of hiding any sensitive data from prying eyes when
  --   it is saved to the user's harddrive.
  ----
  -- Parameters:
  --   argMsg:    The message string to be encrypted.
  --   argKey:    The key string used to garble the message
  --              string. The key string can be any length,
  --              but any more than 16 chars is probably
  --              overkill. The key string used to encrypt a
  --              message is also the exact same string needed
  --              to decrypt the message, therefore, make sure
  --              that the value of the key string will not
  --              change over time or between user sessions.
  ----
  -- Returns:
  --   sOut:      The garbled version of the incoming message
  --              string.
  ----
  sOut = EMPTY
  iKeyLength = argKey.length
  iMsgLength = argMsg.length
  iKeyIndex = 1
  -- Encrypt the string one character at a time.
  repeat with i = 1 to iMsgLength
    iMsgChar = CharToNum(argMsg.char[i])
    iKeyChar = CharToNum(argKey.char[iKeyIndex])
    sOutChar = NumToChar( BitXOR(iMsgChar,iKeyChar) )
    put sOutChar after sOut
    -- Use the next character in the key string;
    -- wrap back to first char when the key length
    -- is exceeded.
    iKeyIndex = iKeyIndex + 1
    if iKeyIndex > iKeyLength then iKeyIndex = 1
  end repeat
  return sOut
end EncryptString


on EncodeString( argString )
  ----
  -- FUNCTION: EncodeString()
  ----
  -- Description:
  --   This function prepares a text string so that
  --   it can be output with the SetPref() function.
  --   SetPref() will not work properly with strings
  --   containing certain non-alphanumeric characters.
  --   Passing a string through this function will
  --   reduce the data to 4-bits-per-character instead
  --   of 8 and will return the string using only
  --   characters that SetPref() can handle.
  ----
  -- Parameters:
  --   argString:   The incoming string to be encoded.
  ----
  -- Returns:
  --   sOut:        The encoded string.
  ----
  iStrLength = argString.length
  sOut = EMPTY
  iHiMask = 15 -- Integer equivalent to the binary "00001111"
  iLoMask = 240 -- Integer equivalent to the binary "11110000"
  iValOffset = 97
  repeat with i = 1 to iStrLength
    iCharVal = CharToNum( argString.char[i] )
    iHiVal = BitNOT( BitOR( BitNOT( iCharVal ), iHiMask ) ) / 16 + iValOffset
    iLoVal = BitNOT( BitOR( BitNOT( iCharVal ), iLoMask ) ) + iValOffset
    put NumToChar( iHiVal ) & NumToChar( iLoVal ) after sOut
  end repeat
  return sOut
end EncodeString


on DecodeString( argString )
  ----
  -- FUNCTION: DecodeString()
  ----
  -- Description:
  --   This function removes the encoding from
  --   a string created by the companion
  --   EncodeString() function.
  ----
  -- Parameters:
  --   argString:   The incoming string to be decoded.
  ----
  -- Returns:
  --   sOut:       The decoded string.
  ----
  sOut = EMPTY
  iStrLength = argString.length
  iValOffset = 97
  i = 1
  repeat while i < iStrLength
    iHiVal = ( CharToNum( argString.char[i] ) - 97 ) * 16
    iLoVal = CharToNum( argString.char[i+1] ) - 97
    put NumToChar( iHiVal + iLoVal ) after sOut
    -- skip to the next character pair
    i = i + 2
  end repeat
  return sOut
end DecodeString
```

Channels 1-6: Texts for Canto 1

MOVIE FRAMERATE

The movie framerate is 32 fps. I set it relatively low so that low-end computers can perform at or close to it, yet it is fast enough that the animation appears quite continuous.

1-5: TEXT FOR CANTO 1

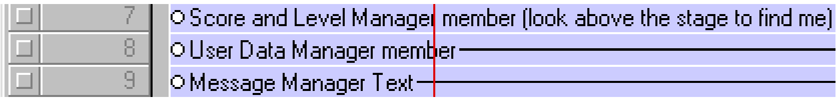
Channels 1-5 contain text sprites that are located above the stage (scroll the stage up to see them). These are the Outer Green, Inner Green, Outer Blue, Inner Blue, and Id-Entity texts for Canto 1. Canto 1 does not permit the player to modify the texts; the texts in channels 1-5 are the texts that will be displayed when the player playes canto 1. If you want to change these texts, edit these sprites.

6: INITIALIZING TEXTS

This sprite appears onstage as a text sprite and says "initializing texts…"

```
--*******************************************************************
--INITIALIZING TEXTS SCRIPT

--This script is at the beginning of the movie. It checks the
--texts for Canto 1 to make sure that the texts do not contain
--invisible ASCII characters and replaces invisible ASCII characters
--(except paragraph breaks) with the letter "a".
--*******************************************************************

property spritenum

on beginsprite me
    sprite(spritenum).loc=centerIt(spritenum)
end beginsprite

on exitFrame me
  repeat with i = 1 to member("Original Outer Green Level 1").char.count
    tempNum=charToNum(member("Original Outer Green Level 1").char[i..i])
    if (tempNum<32 or tempNum>127) and (tempNum<>13) then
      member("Original Outer Green Level 1").char[i..i]="a"
    end if
  end repeat
  repeat with i = 1 to member("Original Inner Green Level 1").char.count
    tempNum=charToNum(member("Original Inner Green Level 1").char[i..i])
    if (tempNum<32 or tempNum>127) and (tempNum<>13) then
      member("Original Inner Green Level 1").char[i..i]="a"
    end if
  end repeat
  repeat with i = 1 to member("Original Outer Blue Level 1").char.count
    tempNum=charToNum(member("Original Outer Blue Level 1").char[i..i])
    if (tempNum<32 or tempNum>127) and (tempNum<>13) then
      member("Original Outer Blue Level 1").char[i..i]="a"
    end if
  end repeat
  repeat with i = 1 to member("Original Inner Blue Level 1").char.count
    tempNum=charToNum(member("Original Inner Blue Level 1").char[i..i])
    if (tempNum<32 or tempNum>127) and (tempNum<>13) then
      member("Original Inner Blue Level 1").char[i..i]="a"
    end if
  end repeat
  repeat with i = 1 to member("Original Id entity text box, level 1").char.count
    tempNum=charToNum(member("Original Id entity text box, level 1").char[i..i])
    if (tempNum<32 or tempNum>127) and (tempNum<>13) then
      member("Original Id entity text box, level 1").char[i..i]="a"
    end if
  end repeat
end exitFrame
```

Channels 7-9: Score and Level, User Data, and Message Managers


7
Score and Level Manager member (look above the stage to find me)
8
User Data Manager member
9
Message Manager Text

These three sprites extend over the entire course of the game. They do not appeaar onstage but, instead, above the stage. They could just as well have been implemented as parent scripts, rather than as behaviors attached to sprites. I made them behaviors attached to sprites so that they would be visible in the score, so that the movie would be a bit more easily readable.

7: SCORE AND LEVEL MANAGER MEMBER (LOOK ABOVE THE STAGE TO FIND ME)

This sprite is located above the stage (scroll the stage up to see it). It does not appear onstage. Note that it exists continuously over the full length of the movie. This is because it maintains data throughout the movie concerned with the player's score and the current level information. It is referred to as gScoreAndLevelManager (which is a global) whenever other sprites need to call handlers in the Score and Level Manager.

```
--*************************************************************************************
--SCORE AND LEVEL MANAGER SCRIPT

--This has to exist over the full life of the game.

--It is attached to a sprite that has member "Score and Level Manager member (look above the stage
--to find me)". If you make this sprite visible in the score, you'll find it above the stage.
```

```
--This script manages the score and what level/canto we're in.

--This is a fairly well-formed object in that there is a public interface into
--the data managed by the Score and Level Manager. For instance, you do not access
--pScore directly but via calls such as
--sprite(gScoreAndLevelManager).addPoints(PointsToAdd).

--*******************SCORE DATA***********************

--This script maintains player score data during the game. Score data is also
--maintained in gUserDataManager. The score data maintained in gScoreAndLevelManager
--is thus temporary storage, as it were, concerning score data. Finally
--gUserDataManager writes score data to the prefs file (though that isn't implemented
--at all yet.

--*pGameCounter
--This is currently not doing much. It is always 1, currently. It is used
--in setScoreOverall in a kind of meaningless way. Currently no score data
--is written to the prefs file. So, currently, gGameCounter is an index into
--pScoreOverall, a list of final game scores, but the list only ever has
--one entry in it. When you want to save scores to the prefs file, you must
--populate the pScoreOverall list in one of a couple of ways. Is it going
--to hold data read from previous prefs files or just record scores for the
--games played during the session? If the latter, then you must provide the
--user with a 'New Game' button, which currently does not exist. So there are
--some important decisions to be made about this before mucking with it.

--*pScoreOverall[pGameCounter]:
--This is to be a list of game scores, but currently it only ever has one
--score in it ever. Read the above note about pGameCounter. pScoreOverall
--is only used, like pGameCounter, in setScoreOverall, currently.

--It needs to be noted that the
--current logic for calculating a game score is that you take the most recently
--played scores for each level and sum them. So, for instance, if in one session
--you play level 1 four times, only the last time you played level 1 counts toward
--your score.

--*pFinalLevelScore[gCurrentLevel][pScoreCount[i]]:
--Whereas the program uses pScore to put in entries while the player is playing,
--the pFinalLevelScore list of lists only contains values from levels that the
--player has finished. It is these values that are best for recording
--final info to be written to the user's hard disk. There is also a pFinalLevelScore
--list (a different one) in gUserDataManager, so the relation between this one and
--the one in gUserDataManager needs to be reviewed.

--*pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]:
--This list of lists holds the scores for individual levels that the player has
--attained for the current session, not sessions from the prefs. But perhaps
--the prefs should hold scores from levels, not just games.

--*pScoreCount[level]:
--Describes the number of scores at each level.

--*pScoreToGet
--Not used currently.

--*********************LEVEL DATA************************

--*pLevelAction[[level1],[level2]...]
--pLevelAction is an important little list. It contains a list describing certain things about each level of play.
```

```
--If you add a new level, you must update this list with a new entry.
--Each list contains the following info for each level of play:
--  #currentStage: you start a level in stage 2. Stage 1 occurs when the player's score reaches the
--    first entry (ie, when they are toast).
--  #totalStages: this is the total number of stages in the level. There are always at least 2,
--    since you start out in stage 2.
--  #cutOffPoints: This describes the number of points the player must have to pass into successive
--    stages of the level. For instance,  in level 1, you start out in stage 2. Then when you get 50 points,
--    you pass into stage 3. Then when you get 150 points, you pass into stage 4. Then when you get 300 points,
--    the level is over. Also, if you ever pass from stage 2 to stage 1, the level terminates.
--  #numberOfEnemySprites: This describes the number of Arteroids always present in a particular stage of the
--    level.
--  #maxTime: this is the number of milliseconds that is considered a normal play-time for the level. If the player
--    goes over this limit,  their score suffers.

--*pNumberOfLevels:
--This counts the number of [level] lists in pLevelAction. There is no gNumberOfLevels. This pNumberOfLevels
--is the var used throughout the game.

--*pExitedPrematurely:
--This variable is set to true when the user clicks 'Exit Canto' in the drop down menu while playing, thuse
--exiting prematurely.


--***********************************************************************************

global gScoreAndLevelManager, gUserDataManager
property pGameCounter, pScore, pScoreCount, pScoreOverall, pScoreToGet, spritenum, pFinalLevelScore
property pLevelAction, pNumberOfLevels
property pExitedPrematurely
global gCurrentLevel, gYourScore, gTimeToQuit

on beginsprite me
  gScoreAndLevelManager=spritenum
  --Other sprites reference this script using this global gScoreAndLevelmanager
  pGameCounter=1
  --Keeps track of the number of games that have been completed.
  pScoreOverall=[0]
  --This list has an entry in it, a score, for each game finished or in progress. See the description of pGameCounter above.
  --pScoreOverall currently only ever has one entry.
  pLevelAction=[[#currentStage:2, #totalStages:4, #cutOffPoints:[-300,50,150,300], #numberOfEnemySprites:[2,2,4,6],
#maxTime:120000],[#currentStage:2, #totalStages:4, #cutOffPoints:[-200,50,150,300], #numberOfEnemySprites:[2,4,6,8],
#maxTime:120000]]
  pNumberOfLevels = pLevelAction.count
  --So you see the importance of pLevelAction is quite wide ranging. There is no gNumberOfLevels.
  pScore=[]
  pScoreCount=[]
  pFinalLevelScore=[]
  repeat with i=1 to pNumberOfLevels
    pScore.append([])
    --when you play a session, you might play a level several times. As long as you finish, the score is recorded.
    --So you end up with various scores in this list of lists, a list of scores for each level.
    pScoreCount.append(0)
    --This counts the number of scores in each list in pScore.
    pFinalLevelScore.append([])
    --Whereas the program uses pScore to put in entries while the player is playing, the pFinalLevelScore list of lists
    --only contains values from levels that the player has finished. It is these values that are best for recording
    --final info to be written to the user's hard disk.
  end repeat
  pGameCounter=1
  --counts the number of games played.
  sprite(gScoreAndLevelManager).visible=TRUE
  --Like I said up top, this script is attached to a sprite that is above the stage.
  pExitedPrematurely=FALSE
  --This variable is set to true when the user clicks 'Exit Canto' in the drop down menu while playing, thuse exiting prematurely.
end beginsprite


--SCORE GET SET ROUTINES*****************************************************************


on addANewLevelScore me
  --This is called when you start a level (whether you've played the level previously or not).
  pScore[gCurrentLevel].append(0)
  pScoreCount[gCurrentLevel]=pScoreCount[gCurrentLevel]+1
end addANewLevelScore


on addPoints me, PointsToAdd
  --This adds points to the current score. It is called when the player hits an arteroid
  pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]=PointsToAdd + pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]
  --Add PointsToAdd points to the score for the current level, to the most recently played round of that level.
  sprite(gYourScore).member.text=string(pScore[gCurrentLevel][pScoreCount[gCurrentLevel]])
  --Update the member that displays the score to the player.
end addPoints


on subtractPoints me, PointsToSubtract
  --This subtracts points from the current score. It's called when a webarteroid intersects with the identity
  tempScore=pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]-PointsToSubtract
  pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]=tempScore
  sprite(gYourScore).member.text=string(tempScore)
end subtractPoints


on getBaseScore me
  --returns the current level score
  return pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]
end getBaseScore


on getTypicalAndEndScore me
  --returns the score at which the level ends, which is also the typical score
  return pLevelAction[gCurrentLevel][#cutOffPoints].getLast()
end getTypicalScore


on setLevelScore me
  if pExitedPrematurely then
    --if the player clicked 'Exit Canto' from the drop down menu before finishing the canto
    pFinalLevelScore[gCurrentLevel].append(-400.0)
    pExitedPrematurely=FALSE
  else
    --This computes and sets the Level score after the level is completed. It is called in the User Data Manager
    if pScore[gCurrentLevel][pScoreCount[gCurrentLevel]] > 0 then
      tempTime=sprite(gUserDataManager).getLevelFinalTime(gCurrentLevel)
      if tempTime>=pLevelAction[gCurrentLevel][#maxTime] then
        --if the player has exceeded #maxTime for the level, then the score is simply the product of the base score and accuracy,
        --so it won't be higher than the base score, which in levels 1 and 2 is 300.
        tempBaseScore=pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]
        tempAccuracy=sprite(gUserDataManager).getLevelFinalAccuracy(gCurrentLevel)
        pFinalLevelScore[gCurrentLevel].append(tempBaseScore*tempAccuracy)
      else
        --if the player finishes in a time better than #maxTime, then their score can be much higher.
        tempBaseScore=pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]
        tempAccuracy=sprite(gUserDataManager).getLevelFinalAccuracy(gCurrentLevel)
        tempTimeRanking=pLevelAction[gCurrentLevel][#maxTime]-tempTime
        --this is the amount of time they bettered #maxTime by.
        pFinalLevelScore[gCurrentLevel].append(tempBaseScore*tempAccuracy*(1+ float(tempTimeRanking)/10000))
        --the above formula multiplies their base score, accuracy, and a number that increases as tempTimeRanking increases.
      end if
    else
      pFinalLevelScore[gCurrentLevel].append(pScore[gCurrentLevel][pScoreCount[gCurrentLevel]])
      --looks like this would be 0
    end if
  end if
  return pFinalLevelScore[gCurrentLevel].getLast()
end setLevelScore


on getTempLevelScore me
  --returns the last level score, finished or not.
  return pScore[gCurrentLevel].getLast()
end


on setScoreOverall me
  --sets and returns the current overall score
  tempScore=0
  repeat with i=1 to pNumberOfLevels
    if pScoreCount[i]>0 then
      --if level i has been played at least once
      tempScore=tempScore+pFinalLevelScore[i][pScoreCount[i]]
      --note that pScoreCount[i] is the most recent score at level i
    end if
  end repeat
  pScoreOverall[pGameCounter]=tempScore
  return tempScore
end setScoreOverall


--LEVEL DATA ADJUSTING ROUTINES**********************************************


on checkToMoveToAnotherLevel me
  --This checks the score to see whether it's time to move to another level.
  --Looks like this is not called anywhere in the program.
  --The checking to see whether it's time to move to another level is carried out in the 'Block End of Level' script
  --that's attached to a frame at the end of the level. That way you don't keep on checking for an event that
  --doesn't happen very often, and you can be less than precise on the timing of it.
  if pScoreCount[gCurrentLevel]>0 then
    if pScore[gCurrentLevel][pScoreCount[gCurrentLevel]]<sprite(spritenum).getTypicalAndEndScore() then
      return FALSE
    else
      return TRUE
    end if
  else
    return FALSE
  end if
end checkToMoveToAnotherLevel


on resetCurrentStage me
  --sets the stage back to the starting stage, which is always stage 2
  pLevelAction[gCurrentLevel][#currentStage]=2
end resetCurrentStage
```

```
on adjustCurrentStageUp me
  --This is called in the 'Block end of level' script.
  --Every couple of seconds, the program checks to see whether
  --it's time to move to a new stage of the level. This routine
  --checks to see whether it's time to move a stage up.
  --If it is, it adjusts #current stage one stage up
  currentStage=pLevelAction[gCurrentLevel][#currentStage]
  relevantCutOffPoint=pLevelAction[gCurrentLevel][#cutOffPoints][currentStage]
  if pScore[gCurrentLevel][pScoreCount[gCurrentLevel]] >= relevantCutOffPoint and currentStage <>
pLevelAction[gCurrentLevel][#totalStages] then
    pLevelAction[gCurrentLevel][#currentStage]=pLevelAction[gCurrentLevel][#currentStage]+1
  end if
end


on adjustCurrentStageDown me
  --This does the same as adjustCurrentStageUp, only this routine
  --adjusts the current stage down one stage if that's appropriate.
  currentStage=pLevelAction[gCurrentLevel][#currentStage]
  oneStageDown=currentStage-1
  if oneStageDown >0 then
    relevantCutOffPoint=pLevelAction[gCurrentLevel][#cutOffPoints][oneStageDown]
    if pScore[gCurrentLevel][pScoreCount[gCurrentLevel]] < relevantCutOffPoint then
      pLevelAction[gCurrentLevel][#currentStage]=oneStageDown
    end if
  end if
end


on endLevel me, levelToEnd
  --This is called in the 'Block end of level' script.
  --Each level has a marker like 1e or 2e or 3e etc, which is where the player
  --is shown their final score for the level and their stats etc.
  --This routine takes them there.
  go to string(levelToEnd) & "e"
end


on playerHasLost me
  --This is called in the 'Block end of level' script.
  --Each level has a a marker 1f or 2f or 3f etc, which is where the player
  --is taken when they get a terrible score and lose the level.
  go to string(gCurrentLevel) & "f"
end
```

8: USER DATA MANAGER MEMBER

One of the main things this script does is read the prefs file written to the player's hard disk that contains the texts they create in Canto 2, and write the prefs file at the end of the game. Currently, no score data (as in the player's score) is written to the hard disk, but you can see I've made the User Data Manager to accomodate this eventually.

```
--*****************************************************************************************************
--USER DATA MANAGER

--This script controls the data that is written to the user's hard drive and some other data.
--It's attached to a sprite above the stage. The sprite is instantiated throughout
--the game.
```

```
--Store this info on the hard disc (not done yet):
--top ten level x scores (including accuracy and time)
--top ten game scores (including separate level scores for each such game, and the corresponding accuracy and time for each
level)
--Should be able to see a sorted list of top scores for level x, and correspondingly the accuracy and time data shows in the same
--row as the level score. Should also be able to sort by accuracy or by time.
--Similarly, if you view top ten overall scores, each row should have the overall score, and the score for each level, and
--the accuracy and time info for each level. Delphi would be handy here!

global gUserDataManager, gScoreAndLevelManager, gConfirmWindowLevel2Manager

global gCurrentLevel

--These are used in the readPrefsFile and the WritePrefsFile and the handlers that get and set saved texts
property pPrefsExistsAlready, pAllSavedUserData, pAllSavedTextData, pAllSavedScoreData
property pGreenTexts, pBlueTexts, pIdEntityTexts, pTextNames, pCurrentTextNums
property pTextBuffer, pOuterGreenTextBuffer, pInnerGreenSameAsOuterBuffer, pInnerGreenTextBuffer, pOuterBlueTextBuffer,
pInnerBlueSameAsOuterBuffer, pInnerBlueTextBuffer, pIdEntityTextBuffer, pTextNameBuffer, pCurrentTextNumBuffer
property pOuter, pInnerSameAsOuter, pInner
property pGreen, pBlue, pIdEntity, pNames, pCurrent

--These are used concerning score data.
property spritenum, pInitialTimes, pTotalPausedTime, pFinalTimes, pCompletedLevelTimes, pInitialPauseTime, pIsPaused
property pMissileHits, pMissileMisses, pCompletedLevelAccuracy
property pFinalLevelScores, pTotalScores

--This is used for encryption of user data
property pEncryptionKeyString

--*************************************
--beginsprite
--*************************************

on beginsprite me
  pEncryptionKeyString=" Change this string to something else just for you so that no one knows your encryption string."
  --This is used in EncryptString to encrypt and decrypt user data. The
  --encryption routines are stored in the 'Encryption' script.
  gUserDataManager=spritenum
  --other sprites reference this script via the name gUserDataManager
  --The below is score initialization (as in 'player's score')
  pCompletedLevelTimes=[]
  pInitialTimes=[]
  pFinalTimes=[]
  pTotalPausedTime=0
  pInitialPauseTime=the milliseconds
  pCompletedLevelAccuracy=[]
  pFinalLevelScores=[]
  repeat with i=1 to sprite(gScoreAndLevelManager).pNumberOfLevels
    pCompletedLevelTimes.append([])
    pInitialTimes.append(0)
    pFinalTimes.append(0)
    pCompletedLevelAccuracy.append([])
    pFinalLevelScores.append(0)
  end repeat
  pMissileMisses=1
  pMissileHits=0
  pIsPaused=FALSE
  pPrefsExistsAlready=FALSE
  sprite(gUserDataManager).readPrefsFile()
end beginsprite
```

```
--*************************************
-- File I/O  set prefs/ get prefs
--*************************************

on writePrefsFile me
  pAllSavedScoreData=[]
  tempString=string(pAllSavedUserData)
  --the above turns a list of lists into a string for writing to the hard disk
  e=EncryptString(tempString, pEncryptionKeyString)
  f=EncodeString(e)
  --the above encrypts the string. See the 'StartMovie' script for these routines.
  setPref "yourprefsfile", f
  --the above writes to disk. Change the name of the file to something specific to your project so
  --that you do not overwrite other peoples' projects if they use this name too. Change the filename here
  --and the one other occurrence below in readPrefsFile.
end


on readPrefsFile me
  pOuter=1
  pInnerSameAsOuter=2
  pInner=3
  pGreen=1
  pBlue=2
  pIdEntity=3
  pNames=4
  pCurrent=5
  pGreenTexts=[]
  pBlueTexts=[]
  pIdEntityTexts=[]
  pTextNames=[]
  pCurrentTextNums=[]
  pOuterGreenTextBuffer=[]
  pInnerGreenSameAsOuterBuffer=[]
  pInnerGreenTextBuffer=[]
  pOuterBlueTextBuffer=[]
  pInnerBlueSameAsOuterBuffer=[]
  pInnerBlueTextBuffer=[]
  pIdEntityTextBuffer=[]
  pTextNameBuffer=[]
  pCurrentTextNumBuffer=[]
  repeat with i=1 to sprite(gScoreAndLevelManager).pNumberOfLevels
    pGreenTexts.append([[],[],[]])
    --format: [Outer, InnerSameAsOuter, Inner], one such list for each level
    pBlueTexts.append([[],[],[]])
    --format: [Outer, InnerSameAsOuter, Inner], one such list for each level
    pIdEntityTexts.append([])
    pTextNames.append([])
    pCurrentTextNums.append(1)
    pOuterGreenTextBuffer.append("")
    pInnerGreenSameAsOuterBuffer.append(FALSE)
    pInnerGreenTextBuffer.append("")
    pOuterBlueTextBuffer.append("")
    pInnerBlueSameAsOuterBuffer.append(FALSE)
    pInnerBlueTextBuffer.append("")
    pIdEntityTextBuffer.append("")
    pTextNameBuffer.append("")
    pCurrentTextNumBuffer.append("")
  end repeat
  pTextBuffer=[pOuterGreenTextBuffer, pInnerGreenSameAsOuterBuffer, pInnerGreenTextBuffer, pOuterBlueTextBuffer,
pInnerBlueSameAsOuterBuffer, pInnerBlueTextBuffer, pIdEntityTextBuffer, pTextNameBuffer, pCurrentTextNumBuffer]
```

```
  pAllSavedTextData=[pGreenTexts, pBlueTexts, pIdentityTexts, pTextNames, pCurrentTextNums]
  pAllSavedScoreData=[]
  pAllSavedUserData=[pAllSavedTextData, pAllSavedScoreData]
  --the above is the format of the list stored to disk. Currently pAllSavedScoreData is empty, but that's for later.
  prefsIsCorrupt=FALSE
  a= getPref("yourprefsfile ")
  isVoid=voidP(a)
  if isVoid then
    tempPrefFile=a
  else
    b=DecodeString(a)
    c=EncryptString(b, pEncryptionKeyString)
    d=value(c)
    if ilk(d) <> #list then
      prefsIsCorrupt=TRUE
      alert("There appears to be a problem with the your saved texts. Sorry, they will have to be overwritten at the end of the game.")
    end if
  end if
  if isVoid or prefsIsCorrupt then
    --if there is no prefs file on the disk or it is corrupt, then initialize the data not from disk, but below
    pPrefsExistsAlready=FALSE
    --the following 9 lines put the original level 1 texts into place.
    pAllSavedTextData[pGreen][1][pOuter].append(member("Original Outer Green Level 1").text)
    --you can find these text members in the first frame, above the stage
    pAllSavedTextData[pGreen][1][pInnerSameAsOuter].append(TRUE)
    pAllSavedTextData[pGreen][1][pInner].append(member("Original Inner Green Level 1").text)
    pAllSavedTextData[pBlue][1][pOuter].append(member("Original Outer Blue Level 1").text)
    pAllSavedTextData[pBlue][1][pInnerSameAsOuter].append(FALSE)
    pAllSavedTextData[pBlue][1][pInner].append(member("Original Inner Blue Level 1").text)
    pAllSavedTextData[pIdEntity][1].append(member("Original Id entity text box, level 1").text)
    pAllSavedTextData[pNames][1].append("Streaming(Texts)")
    --this is where you customize the name of level 1
    pAllSavedTextData[pCurrent][1]=1
    --the following 9 lines put the original level 2 text into place.
    pAllSavedTextData[pGreen][2][pOuter].append(member("Original Outer Green Level 2").text)
    pAllSavedTextData[pGreen][2][pInnerSameAsOuter].append(FALSE)
    pAllSavedTextData[pGreen][2][pInner].append(member("Original Inner Green Level 2").text)
    pAllSavedTextData[pBlue][2][pOuter].append(member("Original Outer Blue Level 2").text)
    pAllSavedTextData[pBlue][2][pInnerSameAsOuter].append(FALSE)
    pAllSavedTextData[pBlue][2][pInner].append(member("Original Inner Blue Level 2").text)
    pAllSavedTextData[pIdEntity][2].append(member("Original Id entity text box, level 2").text)
    pAllSavedTextData[pNames][2].append("Writing(Arteroids)")
    pAllSavedTextData[pCurrent][2]=1
  else
    pPrefsExistsAlready=TRUE
    --put tempPrefFile
    pAllSavedUserData=d
    --The form is [AllSavedTextData, AllSavedScoreData]
    pAllSavedTextData=pAllSavedUserData[1]
    --The form is [pGreenTexts, pBlueTexts, pIdentityTexts, pTextNames]
    pAllSavedScoreData= pAllSavedUserData[2]
    --form is yet to be determined
  end if
  pGreenTexts=pAllSavedTextData[pGreen]
  --the form is [[OuterGreenTexts, OuterSameAsInner(boolean), InnerGreenTexts],[OuterGreenTexts,
OuterSameAsInner(boolean), InnerGreenTexts],...]
  --one such list as [OuterGreenTexts, OuterSameAsInner(boolean), InnerGreenTexts] for each level
  pBlueTexts=pAllSavedTextData[pBlue]
  --the form is [[OuterBlueTexts, OuterSameAsInner(boolean), InnerBlueTexts],[OuterBlueTexts, OuterSameAsInner(boolean),
InnerBlueTexts],...]
  --one such list as [OuterBlueTexts, OuterSameAsInner(boolean), InnerBlueTexts] for each level
  pIdEntityTexts=pAllSavedTextData[pIdEntity]
  --the form is [[level 1 identities], [level 2 identities]...]
  pTextNames=pAllSavedTextData[pNames]
  --the form is [[level 1 Textnames], [level 2 textNames]...]
  pCurrentTextNums=pAllSavedTextData[pCurrent]
  --put "pAllSavedData after readprefs: " & pAllSavedUserData
end readPrefsFile


--************************************
--these routines set and get the text elements and the check box data
--************************************

--PUBLIC INTERFACE:
--See also the docs for each handler mentioned in this public interface
--sprite(gUserDataManager).setBufferText(typeOfText, theText)
--sprite(gUserDataManager).getBufferText(typeOfText)
--sprite(gUserDataManager).getOuterGreenText()
--sprite(gUserDataManager).setOuterGreenText(position, theText)
--sprite(gUserDataManager).getOuterBlueText()
--sprite(gUserDataManager).setOuterBlueText(position, theText)
--sprite(gUserDataManager).getInnerSameAsOuter(theColor)
--sprite(gUserDataManager).setInnerSameAsOuter(theColor, thePosition, theValue)
--sprite(gUserDataManager).getInnerGreen()
--sprite(gUserDataManager).setInnerGreen(position, theText)
--sprite(gUserDataManager).getInnerBlue()
--sprite(gUserDataManager).setInnerBlue(position, theText)
--sprite(gUserDataManager).getIdEntityText()
--sprite(gUserDataManager).setIdEntityText(position, theText)
--sprite(gUserDataManager).getTextName()
--sprite(gUserDataManager).getTextName2(position)
--sprite(gUserDataManager).getTextName3(level)
--sprite(gUserDataManager).setTextName(position, theText)
--sprite(gUserDataManager).getCurrentTextNum()
--sprite(gUserDataManager).getNumberOfTexts()
--sprite(gUserDataManager).setCurrentTextNum(theValue)
--sprite(gUserDataManager).setAllTextData(position, greenO, greenI, blueO, blueI, identityT, textNameT, blueInnerSameAsOuter, greenInnerSameAsOuter)
--Note that greenO etc are texts above
--sprite(gUserDataManager).getAllTextDataAndChangeCurrentText(position, greenO, greenCheckBoxNum, greenI, blueO, blueCheckBoxNum, blueI, identityT, textNameT)
--Note that greenO etc are sprite numbers above.

--IMPLEMENTATION:

--pTextBuffer=[pOuterGreenTextBuffer, pInnerGreenSameAsOuterBuffer, pInnerGreenTextBuffer, pOuterBlueTextBuffer, pInnerBlueSameAsOuterBuffer, pInnerBlueTextBuffer, pIdEntityTextBuffer, pTextNameBuffer, pCurrentTextNumBuffer]

on setBufferText(me, typeOfText, theText)
  if typeOfText >0 and typeOfText< 10 then
    pTextBuffer[typeOfText][gCurrentLevel]=theText
  else
    alert("SetBufferText has been called with a bad value of typeOfText")
  end if
end setBufferText


on getBufferText(me, typeOfText)
  if typeOfText >0 and typeOfText< 10 then
    return pTextBuffer[typeOfText][gCurrentLevel]
  else
    alert("getBufferText has been called with a bad value of typeOfText")
  end if
end setBufferText


on getOuterGreenText me
  --This returns the Outer GreenText in gUserDataManager concerning the current level of play.
  --Note that in 'on beginsprite' above all the texts are populated. Moreover, note that
  --if the player does not have a prefs file on their computer for Arteroids, then the
  --text data in gUserDataManager is populated with one text for level 1 (based on member("Original Outer Green Level 1")
  --and a similar operation concerning one text for level 2. Consequently, when you use this getOuterGreenText, there will
  --certainly be at least one thing in it to begin with, regardless of whether you are at level 1 or level 2.
  theTextNum=pCurrentTextNums[gCurrentLevel]
  return pGreenTexts[gCurrentLevel][pOuter][theTextNum]
end getOuterGreenText


on setOuterGreenText me, position, theText
  --This sets the Outer Green Text in gUserDataManager to theText.
  --position refers to the position or number of this particular Outer Green Text at this level of play,
  --which can be retrieved with tempcurrent=sprite(gUserDataManager).getCurrentTextNum().
  --If you want to create a new saved text, you make position be greater than pTextNames[gCurrentLevel].count,
  --in other words you make it greater than the current number of texts at this level. It doesn't even have
  --to be the right position, in that case, as long as it is greater than the number of texts at this level.
  if position > pTextNames[gCurrentLevel].count then
    pGreenTexts[gCurrentLevel][pOuter].append(theText)
  else
    pGreenTexts[gCurrentLevel][pOuter][position]=theText
  end if
end setOuterGreenText


on getOuterBlueText me
  --This returns the Outer Blue Text at this level of play. Have a look at the documentation of
  --getOuterGreenText because the same applies here.
  theTextNum=pCurrentTextNums[gCurrentLevel]
  return pBlueTexts[gCurrentLevel][pOuter][theTextNum]
end getOuterGreenText


on setOuterBlueText me, position, theText
  --This sets the outer blue text. Have a look at the documentation of
  --setOuterGreenText, because the same applies here.
  if position > pTextNames[gCurrentLevel].count then
    pBlueTexts[gCurrentLevel][pOuter].append(theText)
  else
    pBlueTexts[gCurrentLevel][pOuter][position]=theText
  end if
end setOuterGreenText


on getInnerSameAsOuter me, theColor
  --Note that on the stage there are two check boxes in the canto 2 editor. This function
  --basically returns whether these are checked (TRUE) or not (FALSE).
  --You have to specify the color of the text you want to check. This will return
  --the appropriate boolean value for the current level and the current text.
  theTextNum=pCurrentTextNums[gCurrentLevel]
  if theColor="blue" then
    return pBlueTexts[gCurrentLevel][pInnerSameAsOuter][theTextNum]
  else
    return  pGreenTexts[gCurrentLevel][pInnerSameAsOuter][theTextNum]
  end if
end getInnerSameAsOuter
```

```
on setInnerSameAsOuter me, theColor, thePosition, theValue
  --Note that on the stage there are two check boxes in the canto 2 editor. This handler sets
  --the boolean value not for the checkbox but for the gUserDataManager datum for the checkbox.
  --You must stiupulate the color of the checkbox ("blue" or "green"), the position (see the docs
  --for setOuterGreenText to understand the meaning of 'thePosition'), and the boolean value
  --you want set. This should be called when the user clicks the box and also before exiting
  --the editor (just for safety's sake). And when saving a text. But that is included in
  --setAllTextData for your convenience. setAllTextData gathers all the 'set' routines into
  --one routine, since you should do all the 'sets' when saving a text.
  textCount=pTextNames[gCurrentLevel].count
  theTextNum=pCurrentTextNums[gCurrentLevel]
  if theColor="blue" then
    if thePosition > textCount then
      pBlueTexts[gCurrentLevel][pInnerSameAsOuter].append(theValue)
    else
      pBlueTexts[gCurrentLevel][pInnerSameAsOuter][thePosition]=theValue
    end if
  else
    if thePosition > textCount then
      pGreenTexts[gCurrentLevel][pInnerSameAsOuter].append(theValue)
    else
      pGreenTexts[gCurrentLevel][pInnerSameAsOuter][thePosition]=theValue
    end if
  end if
end setInnerSameAsOuter


on getInnerGreen me
  --This returns the inner green text in gUserDataManager at the current Level for the current text num.
  theTextNum=pCurrentTextNums[gCurrentLevel]
  return pGreenTexts[gCurrentLevel][pInner][theTextNum]
end getInnerGreen


on setInnerGreen me, position, theText
  --This sets the inner green text to theText at the current level for the current position.
  --  theTextNum=pCurrentTextNums[gCurrentLevel]
  --  pGreenTexts[gCurrentLevel][pInner][theTextNum]=theText
  if position > pTextNames[gCurrentLevel].count then
    pGreenTexts[gCurrentLevel][pInner].append(theText)
  else
    pGreenTexts[gCurrentLevel][pInner][position]=theText
  end if
end setInnerBlue


on getInnerBlue me
  --This returns the inner blue text in gUserDataManager at the current Level for the current text num.
  theTextNum=pCurrentTextNums[gCurrentLevel]
  return pBlueTexts[gCurrentLevel][pInner][theTextNum]
end getInnerGreen


on setInnerBlue me, position, theText
  --This sets the inner blue text to theText at the current level for the current position.
  --  theTextNum=pCurrentTextNums[gCurrentLevel]
  --  pBlueTexts[gCurrentLevel][pInner][theTextNum]=theText
  if position > pTextNames[gCurrentLevel].count then
    pBlueTexts[gCurrentLevel][pInner].append(theText)
  else
    pBlueTexts[gCurrentLevel][pInner][position]=theText
  end if
end setInnerBlue


on getIdEntityText me
  --This returns the id-entity text for the current level and the current text num.
  theTextNum=pCurrentTextNums[gCurrentLevel]
  return pIdEntityTexts[gCurrentLevel][theTextNum]
end getIdEntityText


on setIdEntityText me, position, theText
  --This sets the id-entity text to theText for the current level and the current text num.
  textCount=pTextNames[gCurrentLevel].count
  theTextNum=pCurrentTextNums[gCurrentLevel]
  if position > textCount then
    pIdEntityTexts[gCurrentLevel].append(theText)
  else
    pIdEntityTexts[gCurrentLevel][position]=theText
  end if
end setIdEntityText


on getTextName me
  --This returns the name of the current text at the current level. This is what
  --was in the Save As box.
  theTextNum=pCurrentTextNums[gCurrentLevel]
  return pTextNames[gCurrentLevel][theTextNum]
end getTextName


--sprite(gUserDataManager).getTextName2(position)
on getTextName2 me, position
  return pTextNames[gCurrentLevel][position]
end getTextName2


--sprite(gUserDataManager).getTextName3(level)
on getTextName3 me, level
  return pTextNames[level][pCurrentTextNums[level]]
end getTextName3


on setTextName me, position, theText
  --This sets the name of the current text at the current level  to theText, and at the given position.
  textCount=pTextNames[gCurrentLevel].count
  theTextNum=pCurrentTextNums[gCurrentLevel]
  if position > textCount then
    pTextNames[gCurrentLevel].append(theText)
  else
    pTextNames[gCurrentLevel][position]=theText
  end if
end setTextName


on getCurrentTextNum me
  --This gets the current text num for this level.
  return pCurrentTextNums[gCurrentLevel]
end getCurrentTextNums


on setCurrentTextNum me, theValue
```

```
  --This sets the current text num for this level.
  pCurrentTextNums[gCurrentLevel]=theValue
end setCurrentTextNum


on getNumberOfTexts me
  return sprite(spritenum).pTextNames[gCurrentLevel].count
end getNumberOfTexts


on setAllTextData me,  position, greenO, greenI, blueO, blueI, identityT, textNameT, blueInnerSameAsOuter,
greenInnerSameAsOuter
  --This combines all the above 'set' routines so you can save a text without having to do all the below calls.
  --This sets values in gUserDataManager, not in the onstage text members. So this should be used when saving a family of texts.
  sprite(spritenum).setOuterGreenText(position, greenO) --greenO is a text
  sprite(spritenum).setInnerGreen(position, greenI) --greenI is a text
  sprite(spritenum).setInnerSameAsOuter("green", position, greenInnerSameAsOuter)
  sprite(spritenum).setInnerSameAsOuter("blue", position, blueInnerSameAsOuter)
  sprite(spritenum).setOuterBlueText(position, blueO)
  sprite(spritenum).setInnerBlue(position, blueI)
  sprite(spritenum).setIdEntityText(position, identityT)
  sprite(spritenum).setTextName(position, textNameT)
  sprite(spritenum).setCurrentTextNum(position)
end setAllTextData


on getAllTextDataAndChangeCurrentText me,  position, greenO, greenCheckBoxNum, greenI, blueO, blueCheckBoxNum, blueI,
identityT, textNameT
  --This combines all the above 'set' routines so you can save a text without having to do all the below calls.
  --This sets values in gUserDataManager, not in the onstage text members. So this should be used when saving a family of texts.
  --Note that greenO is a sprite number now (as opposed to in setAllTextData)
  sprite(spritenum).setCurrentTextNum(position)
  --this sets the current text num, which the below handlers use.
  sprite(greenO).member.text=sprite(spritenum).getOuterGreenText()
  --getOuterGreenText
  innerGreenSameAsOuterB=sprite(spritenum).getInnerSameAsOuter("green")
  if innerGreenSameAsOuterB then
    sprite(greenCheckBoxNum).member=member("CheckBoxChecked copy")
    sprite(greenI).member=sprite(greenO).member
  else
    sprite(greenCheckBoxNum).member=member("CheckBoxUnchecked copy")
    sprite(greenI).member=member(sprite(greenI).pOriginalMemberNum)
    sprite(greenI).member.text=sprite(spritenum).getInnerGreen()
  end if
  --get the inner green check box and the inner green text
  sprite(blueO).member.text=sprite(spritenum).getOuterBlueText()
  --getOuterBlueText
  innerBlueSameAsOuterB=sprite(spritenum).getInnerSameAsOuter("blue")
  if innerBlueSameAsOuterB then
    sprite(blueCheckBoxNum).member=member("CheckBoxChecked copy")
    sprite(blueI).member=sprite(blueO).member
  else
    sprite(blueCheckBoxNum).member=member("CheckBoxUnchecked copy")
    sprite(blueI).member=member(sprite(blueI).pOriginalMemberNum)
    sprite(blueI).member.text=sprite(spritenum).getInnerBlue()
  end if
  --get the inner blue check box and the inner green text
  sprite(identityT).member.text=sprite(spritenum).getIdEntityText()
  --get the identity text
  sprite(textNameT).member.text=sprite(spritenum).getTextName()
  --get the text name
end setAllTextData
```

```
--*************************************
--Delete Texts
--*************************************

on deleteText me, textNumToDelete
  if sprite(spritenum).getNumberOfTexts()>1 then
    pGreenTexts[gCurrentLevel][pOuter].deleteAt(textNumToDelete)
    pGreenTexts[gCurrentLevel][pInnerSameAsOuter].deleteAt(textNumToDelete)
    pGreenTexts[gCurrentLevel][pInner].deleteAt(textNumToDelete)
    pBlueTexts[gCurrentLevel][pOuter].deleteAt(textNumToDelete)
    pBlueTexts[gCurrentLevel][pInnerSameAsOuter].deleteAt(textNumToDelete)
    pBlueTexts[gCurrentLevel][pInner].deleteAt(textNumToDelete)
    pIdEntityTexts[gCurrentLevel].deleteAt(textNumToDelete)
    pTextNames[gCurrentLevel].deleteAt(textNumToDelete)
    if pCurrentTextNums[gCurrentLevel]>=textNumToDelete then
      --then pCurrentTextNums needs adjustment. And what about pTextNumToLoad and pTextNumToLoadTo?
      if pCurrentTextNums[gCurrentLevel]>textNumToDelete then
        pCurrentTextNums[gCurrentLevel]=pCurrentTextNums[gCurrentLevel]-1
      else
        if textNumToDelete=1 then
          pCurrentTextNums[gCurrentLevel]=1
        else
          pCurrentTextNums[gCurrentLevel]=pCurrentTextNums[gCurrentLevel]-1
        end if
      end if
    else
      --what other variables in this unit need adjustment? Not finished.
    end if
    return 1
  else
    return 0
  end if
end deleteText


--*************************************
--Time info
--*************************************


on setLevelInitialTime me, LevelToSet
  --This sets up the scratch data for the timer to time a level. Paused times are not counted.
  pInitialTimes[LevelToSet]=the milliseconds
  --the above var could just as well not be a list but a single value since the info is not kept and you can
  --only play one level at a time.
  pTotalPausedTime=0
end setInitialTime


on setLevelFinalTime me, LevelToSet
  pFinalTimes[LevelToSet]= the milliseconds
  tempTime=pFinalTimes[LevelToSet]-pInitialTimes[LevelToSet]-pTotalPausedTime
  pCompletedLevelTimes[LevelToSet].append(tempTime)
  return tempTime
end setLevelFinalTime


on getLevelFinalTime me, LevelToSet
  return pCompletedLevelTimes[LevelToSet].getLast()
end getLevelFinalTime
```

```
on PauseButtonOn me, LevelToSet
  pInitialPauseTime=the milliseconds
  pIsPaused=TRUE
end PauseButtonOn


on isPaused me
  return pIsPaused
end isPaused


on PauseButtonOff me, LevelToSet
  tempTime= the milliseconds - pInitialPauseTime
  pTotalPausedTime=pTotalPausedTime+tempTime
  pIsPaused=FALSE
end PauseButtonOff


--*************************************
--Accuracy info
--sprite(gUserDataManager).getLevelFinalAccuracy(gCurrentLevel)
--*************************************


on addOneMissileHit me
  pMissileHits=pMissileHits+1
end addOneHit


on addOneMissileMiss me
  pMissileMisses=pMissileMisses+1
end addOneHit


on setLevelFinalAccuracy me, LevelToSet
  tempAccuracy=float(pMissileHits)/(pMissileMisses+pMissileHits)
  pCompletedLevelAccuracy[LevelToSet].append(tempAccuracy)
  return tempAccuracy
end setLevelFinalAccuracy


on getLevelFinalAccuracy me, LevelToSet
  return  pCompletedLevelAccuracy[LevelToSet].getLast()
end getLevelFinalAccuracy


on resetMissileHitsAndMissesCount me
  pMissileMisses=1
  pMissileHits=0
end resetMissileHitsAndMissesCount


on getMissileHits me
  return pMissileHits
end getMissileHits


on getMissileMisses
  return pMissileMisses
end
```

```
--*************************************
--Score info ('score' as in the player's score)
--*************************************


on setFinalLevelScore me, LevelToSet
  pFinalLevelScores[LevelToSet]=sprite(gScoreAndLevelManager).setLevelScore()
  return pFinalLevelScores[LevelToSet]
end setFinalLevelScore
```

9: MESSAGE MANAGER TEXT

```
--******************************************************************************
--MESSAGE MANAGER
--******************************************************************************

--This handles messages to the player.

global gCurrentLevel
global gMessageManager, gUserDataManager, gScoreAndLevelManager
global gTextEditorMessagePane
property pLevelTitles, pCommentTexts, pCommentTextsCount, pDeathNotices, pDeathNoticesCount
property spritenum

on beginsprite me
  gMessageManager=spritenum
  sprite(gMessageManager).visible=TRUE
  pLevelTitles=[]
  repeat with i= 1 to sprite(gScoreAndLevelManager).pNumberOfLevels
    pLevelTitles.append(sprite(gUserDataManager).getTextName3(i))
  end repeat
  pCommentTexts=[["Poetry has momentarily triumphed over the forces of dullness.", "Poetry survived. Next, save poetry from yourself."],["Thus endeth the canto. Did you save or destroy poetry? Is there a difference?", "You have successfully saved poetry from yourself. Thank you."]]
  pDeathNotices=[["Oops. Poetry is dead. And it's your fault.", "So much for poetry. When you get below -400, it's surely dead.", "Poetry has given up the ghost.", "I'm sorry, poetry is dead. Whatever else is true, we live many lives in this one.", "Poetry passed away with little resistance to its own demise " & the long date & ", " & the long time & ". RIPoetry."],["Your text-droid drama is over.", "This is how it ends.", "You apparently are toast.", "There's winning and losing, but which is which is confusing. Uh, this is losing, by the way. You lost. Or was that a battle between texts?"]]
  pCommentTextsCount=[]
  pDeathNoticesCount=[]
  repeat with i = 1 to  pCommentTexts.count
    pCommentTextsCount.append(pCommentTexts[i].count)
  end repeat
  repeat with i = 1 to  pDeathNotices.count
    pDeathNoticesCount.append(pDeathNotices[i].count)
  end repeat
end beginsprite


on levelTitle me, level
  repeat with i= 1 to sprite(gScoreAndLevelManager).pNumberOfLevels
    pLevelTitles[i]=sprite(gUserDataManager).getTextName3(i)
  end repeat
  return pLevelTitles[level]
end title

on levelComment me, gCurrentLevel
  return pCommentTexts[gCurrentLevel][random(pCommentTextsCount[gCurrentLevel])]
```

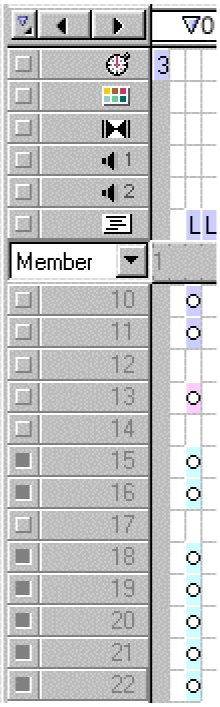
```
end comment


on deathNotice me, gCurrentLevel
  return pDeathNotices[gCurrentLevel][random(pDeathNoticesCount[gCurrentLevel])]
end

--sprite(gMessageManager).textEditorMessage(textMessage)
on textEditorMessage me, textMessage
  sprite(gTextEditorMessagePane).member.text=textMessage
end textEditorMessage
```

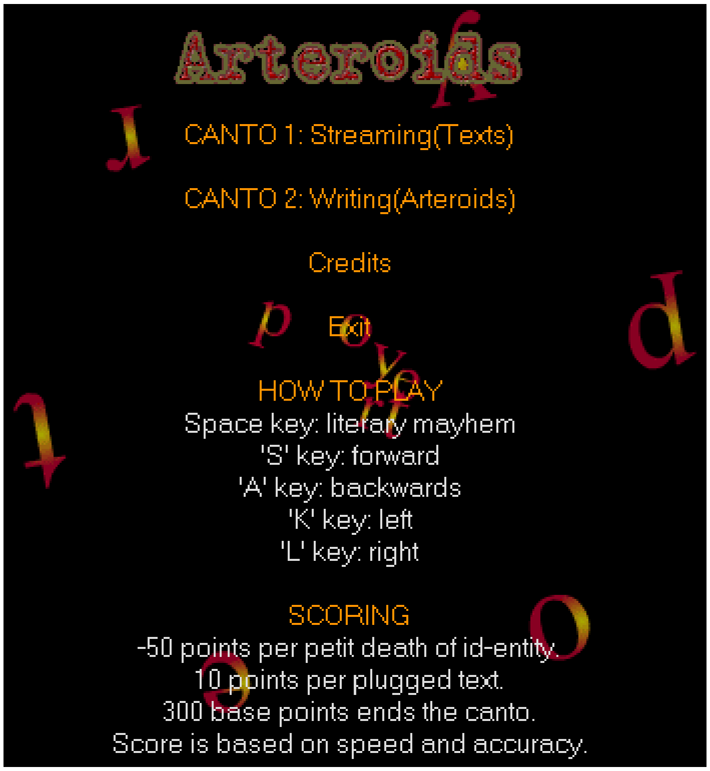
Channels 10-22: Main Menu and Credits

Sprites 10-22 display the main menu for the game 10-11 and the credits for the game (13-22).

Note that there is a marker 0 associated with this frame. Whenever the player is returned to the main menu, they are returned here via marker 0.

The frame script in this frame simply loops the playback head on this frame.





CANTO 1: Streaming(Texts)

CANTO 2: Writing(Arteroids)

Credits

Exit

HOW TO PLAY
Space key: literary mayhem
'S' key: forward
'A' key: backwards
'K' key: left
'L' key: right

SCORING
-50 points per petit death of id-entity.
10 points per plugged text.
300 base points ends the canto.
Score is based on speed and accuracy.

10: MAIN MENU ANIMATION

The animation of the word 'poetry' exploding is a Flash 4 animation. The name of the cast member is poetry3.

```
--************************************************
--MAIN MENU POETRY ANIMATION

--This is attached to the animation behind the main
--menu. It simply positions the animation and sets
--it playing.
--************************************************

property spritenum
global gStageWidth, gStageHeight

on beginsprite me
  sprite(spritenum).visible=FALSE
  sprite(spritenum).stop()
  sprite(spritenum).frame=1
  sprite(spritenum).locH = (gStageWidth)/2
  sprite(spritenum).locV= (gStageHeight)/2
  sprite(spritenum).ink=0
  sprite(spritenum).play()
  sprite(spritenum).locZ=spritenum
  sprite(spritenum).visible=TRUE
end beginsprite
```

11: MAIN MENU

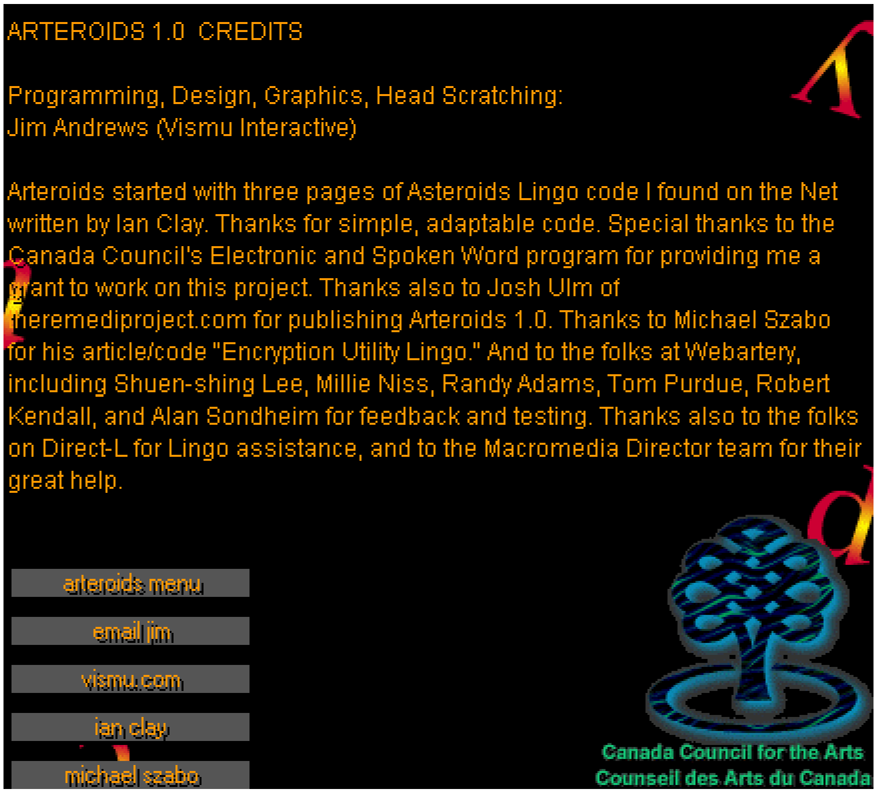
```
--*************************************************************************************
--MAIN MENU SCRIPT

--This script is attached to the main menu in the game. The Main Menu comes up at
--the beginning of the game and is reachable at the end of levels if the user
--selects to view it. There is a marker for this frame: 0. So when the user
--comes to the Main menu, they come to marker 0.
--*************************************************************************************

property spritenum
global  gMessageManager
--This unit of code is called to populate titles and so forth
property pCanto1, pCanto2, pCredits, pQuit
--These properties describe the paragraph numbers in the main menu that are hot.
property pCurrentParagraphNum, pOldParagraphNum
--This is used to control the mouseover coloring of text
global gMainMenu, gArteroidsOpeningGraphic
--These are the spritenums of visible elements in the Main Menu
global gCredits, gCanadaCouncil, gCreditsVismuButton, gCreditsEmailButton, gCreditsIanClayButton, gCreditsSzaboButton,
gCreditsArteroidsMenuButton
--These are the spritenums of visible elements in the Credits page, which is in the same frame as the Main Menu

on beginsprite me
  gMainMenu=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locZ=spritenum
  pCanto1=6
  pCanto2=8
  pCredits=10
  pQuit=12
  --If the Main Menu is revised so that the paragraph numbers of the hot (hyper) paragraphs change
  --then these numbers need to be changed to reflect the new situation.
  sprite(spritenum).member.paragraph[pCanto2]= "CANTO 2: " & sprite(gMessageManager).levelTitle(2)
  sprite(spritenum).loc=centerIt(spritenum)
  --This centers the menu on the screen (see the 'StartMovie etc' script).
  sprite(spritenum).visible=TRUE
  saveIt= the itemdelimiter
  the itemdelimiter=RETURN
  pCurrentParagraphNum=-1
  pOldParagraphNum=1
end beginsprite

on mouseup me
  pointClicked = the mouseLoc
  pCurrentparagraphNum = sprite(spriteNum).pointToParagraph(pointClicked)
  case pCurrentParagraphNum of
    pCanto1:
      --Go to Canto 1
      cursor 0
      sprite(spritenum).member.paragraph[pCanto1].color=rgb(255, 153, 0)
      go to "1"
      --Note that whenever you enter Canto 1, you send the playback head to marker '1'. Same with level 2, etc
      --(goes to marker  '2' etc)
    pCanto2:
      --Go to Canto 2
      cursor 0
      sprite(spritenum).member.paragraph[pCanto2].color=rgb(255, 153, 0)
      go to "2"
    pCredits:
      cursor 0
      sprite(spritenum).member.paragraph[pCredits].color=rgb(255, 153, 0)
      sprite(gArteroidsOpeningGraphic).visible=FALSE
      sprite(spritenum).visible=FALSE
      sprite(gCredits).visible=TRUE
      sprite(gCanadaCouncil).visible=TRUE
      sprite(gCreditsVismuButton).visible=TRUE
      sprite(gCreditsEmailButton).visible=TRUE
      sprite(gCreditsIanClayButton).visible=TRUE
      sprite(gCreditsSzaboButton).visible=TRUE
      sprite(gCreditsArteroidsMenuButton).visible=TRUE
    pQuit:
      --Quit arteroids
      --Note that parent.close() is used. This is because the HTML for arteroids is in a frameset.
      --This is to do fullfullfull screen in IE for the PC.
      cursor 0
      sprite(spritenum).member.paragraph[pQuit].color=rgb(255, 153, 0)
      gotonetpage "javascript:parent.close()"
    otherwise:
      cursor 0
  end case
end mouseup

on mousewithin me
  --This routine is used to color text when the user mouses over hot links.
  --The case statement needs to contain an entry for each hot line.
  pointClicked = the mouseLoc
  pCurrentparagraphNum = sprite(spriteNum).pointToParagraph(pointClicked)
  case pCurrentParagraphNum of
    pCanto1: sprite(spritenum).MakeMeRed(pCanto1)
    pCanto2: sprite(spritenum).MakeMeRed(pCanto2)
    pCredits: sprite(spritenum).MakeMeRed(pCredits)
    pQuit: sprite(spritenum).MakeMeRed(pQuit)
    otherwise:
      cursor 0
      sprite(spritenum).member.paragraph[pOldParagraphNum].color=rgb(255,153,0)
      pOldParagraphNum=1
  end case
end mousewithin

on MakeMeRed me, myParaNum
  cursor 280
  if myParaNum<>pOldParagraphNum then
    sprite(spritenum).member.paragraph[myParaNum].color=rgb(255,0,0)
    sprite(spritenum).member.paragraph[pOldParagraphNum].color=rgb(255,153,0)
  end if
  pOldParagraphNum=pCurrentParagraphNum
end

on mouseleave me
  sprite(spritenum).member.paragraph[pOldparagraphNum].color=rgb(255,153,0)
  pOldParagraphNum=1
  cursor 0
end mouseleave
```

13: MAIN MENU GRAPHIC

```
--******************************************************************************
--MAIN MENU GRAPHIC

--This is attached to the Main Menu graphic word 'Arteroids'. This script
--establishes the global name for this sprite, the loc, and the vis=TRUE
--******************************************************************************

property spritenum
global gArteroidsOpeningGraphic, gMainMenu

on beginsprite me
  gArteroidsOpeningGraphic=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locH=centerH(spritenum)
  sprite(spritenum).locV=sprite(gMainMenu).member.linePosToLocV(sprite(gMainMenu).pCanto1) + sprite(gMainMenu).locV - sprite(spritenum).height - 2*sprite(gMainMenu).member.fixedLineSpace
  --The above line puts the graphic locv one line above the first visible line of the Main menu, assuming that is line pCanto1.
  sprite(spritenum).locZ=spritenum
  sprite(spritenum).visible=TRUE
end beginsprite
```

15: CREDITS SCRIPT

```
--*************************************************
--CREDITS SCRIPT

--This is the script that controls the behavior of the Credits main text field.
--This didn't seem like a sufficiently complex window to need a Manager, so each
--link on this page contains its own behaviors controlling the color and action.
--*************************************************


property spritenum
global gCredits

on beginsprite me
  gCredits=spritenum
  sprite(spritenum).loc=centerIt(spritenum)
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locZ=spritenum
end beginsprite
```



ARTEROIDS 1.0 CREDITS

Programming, Design, Graphics, Head Scratching:
Jim Andrews (Vismu Interactive)

Arteroids started with three pages of Asteroids Lingo code I found on the Net written by Ian Clay. Thanks for simple, adaptable code. Special thanks to the Canada Council's Electronic and Spoken Word program for providing me a grant to work on this project. Thanks also to Josh Ulm of theremediproject.com for publishing Arteroids 1.0. Thanks to Michael Szabo for his article/code "Encryption Utility Lingo." And to the folks at Webartery, including Shuen-shing Lee, Millie Niss, Randy Adams, Tom Purdue, Robert Kendall, and Alan Sondheim for feedback and testing. Thanks also to the folks on Direct-L for Lingo assistance, and to the Macromedia Director team for their great help.

16: CANADA COUNCIL LOGO

This graphical sprite also has the cursor control behavior attached to it, which controls the changing of the cursor on mouseover to indicate the graphic is clickable.

```
--************************************************************************************
--CREDITS CANADA COUNCIL LOGO

--The Canada Council logo appears at bottom right of the Credits main field member.
--This script handles that action. And the logo is initially invisible, like all
--the other Credits elements.
--************************************************************************************

property spritenum
global gCredits, gCanadaCouncil

on beginsprite me
  gCanadaCouncil=spritenum
  sprite(spritenum).visible=false
  sprite(spritenum).locH=sprite(gCredits).right
  sprite(spritenum).locV=sprite(gCredits).bottom
  sprite(spritenum).locZ=spritenum
end beginsprite

on mouseup me
--If you look in the HTML page, you see that there is
  --javascript and VBScript to handle the opening of a new
  --browser window containing the Canada Council page.
  externalEvent ("canadacouncil()")
end mouseup
```

18: MICHAEL SZABO LINK

This sprite also has attached to it the Menu Colors script which contains code to handle the cursor and color changes on mouseover etc.

```
--************************************************************
--CREDITS SZABO LINK

--This is attached to the sprite displaying the link to
--Michael Szabo's work (he wrote the article on encryption
--used in Arteroids
--************************************************************

property spritenum
global gCreditsSzaboButton, gCredits

on beginsprite me
  gCreditsSzaboButton=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locH=sprite(gCredits).locH + 7
  sprite(spritenum).locV=sprite(gCredits).bottom -sprite(spritenum).height
  --This line places the button at the bottom of the Credits text field.
  sprite(spritenum).locZ=spritenum
end beginsprite

on mouseup me
  --If you look in the HTML file, you'll find code to open the Szabo window.
  externalEvent ("szabo()")
end mouseup


--************************************************************************************
```

```
--MENU COLORS

--This is a general purpose routine that handles the colors for hypertext menu
--elements in the program. rgb(255, 153, 0) is orange, and rgb(255,0,0) is red.
--*******************************************************************************

global gAControlIsMousedDown, gWithinAControl
property spritenum

on beginsprite me
  sprite(spritenum).member.color=rgb(255,153,0)
  gWithinAControl=FALSE
  gAControlIsMousedDown=FALSE
end beginsprite

on mouseenter me
  if not gAControlIsMousedDown then
    sprite(spritenum).member.color=rgb(255,0,0)
  end if
  cursor 280
  gWithinAControl=TRUE
end mouseenter

on mousedown me
  cursor 280
  gAControlIsMousedDown=spritenum
end mousedown

on mouseleave me
  if not gAControlIsMousedDown then
    cursor 0
    sprite(spritenum).member.color=rgb(255,153,0)
  end if
  gWithinAControl=FALSE
end mouseleave

on mouseup me
  gAControlIsMousedDown=0
end mouseup

on mouseupOutside me
  if gAControlIsMousedDown=spritenum then
    gAControlIsMousedDown=0
  end if
  if not gWithinAControl then
    cursor 0
  end if
  sprite(spritenum).member.color=rgb(255,153,0)
end mouseupOutside
```

19: IAN CLAY LINK

This sprite also has attached to it the Menu Colors script which controls the mouseover coloring and cursor change. See channel 18 above.

```
--***********************************************************************************
--CREDITS IAN CLAY LINK

--This behavior positions the sprite that you click to open a browser window to Ian Clay's
```

```
--work, on whose code Arteroids is based. The mouseup behavior calls code in the HTML page
--that opens the new browser window.
--*************************************************************************************************


property spritenum
global gCreditsIanClayButton, gCredits

on beginsprite me
  gCreditsIanClayButton=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locH=sprite(gCredits).locH + 7
  sprite(spritenum).locV=sprite(spritenum-1).locV - sprite(spritenum).height - 10
  sprite(spritenum).locZ=spritenum
end beginsprite

on mouseup me
  --If you look in the HTML file, you'll find code to open the Ian Clay window.
  externalEvent ("clay()")
end mouseup
```

20: VISMU LINK

This sprite also has attached to it the Menu Colors script which controls the mouseover coloring and cursor change. See channel 18 above.

```
--*****************************************************************************************
--CREDITS VISMU BUTTON

--When the user clicks the button in Credits that says 'vismu.com', a new browser
--window named 'vismu' is opened. This script handles that action
--*****************************************************************************************

property spritenum
global gCreditsVismuButton, gCredits

on beginsprite me
  gCreditsVismuButton=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locH=sprite(gCredits).locH + 7
  sprite(spritenum).locV=sprite(spritenum-1).locV - sprite(spritenum).height - 10
  sprite(spritenum).locZ=spritenum
end beginsprite

on mouseup me
  --If you look in the HTML file, you'll find code to open the Vispo window.
  externalEvent ("vispo()")
end mouseup
```

21: EMAIL LINK

This sprite also has attached to it the Menu Colors script which controls the mouseover coloring and cursor change. See channel 18 above.

```
--*****************************************************************************************
```

```
--CREDITS EMAIL BUTTON

--When the user clicks the button in Credits that says 'email jim@vispo.com', an
--email client window is opened with the subject 'Arteroids' and info from
--the environment (see Lingo Dictionary for 'environment'). This script handles
--that action.
--********************************************************************************

property spritenum
global gCreditsEmailButton, gCredits

on beginsprite me
  gCreditsEmailButton=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locH=sprite(gCredits).locH + 7
  sprite(spritenum).locV=sprite(spritenum-1).locV - sprite(spritenum).height - 10
  sprite(spritenum).locZ=spritenum
end beginsprite

on mouseup me
  set subjectString = "Arteroids"
  set messageString = "Debugging info (delete if you like).   OS: " & the environment.platform & ". OS version: " & the environment.osVersion & ". Language: " & the environment.osLanguage & ". UILang: " & the environment.uiLanguage & ". Thanks. This is to help me debug if Arteroids went wonky on you."
  gotonetpage "mailto:"& "jim@vispo.com" &"?subject="&subjectString &"&body="& messageString
  --for a good macro on this, see www.mediamacros.com and search on email
end mouseup
```

22: BACK TO ARTEROIDS MENU

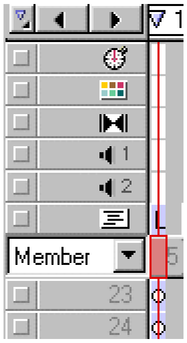
This sprite also has attached to it the Menu Colors script which controls the mouseover coloring and cursor change. See channel 18 above.

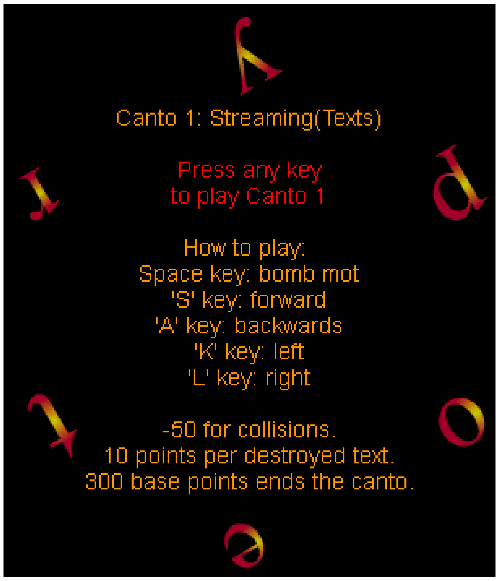
```
--********************************************************************************
--CREDITS BACK TO MENU BUTTON

--When the user clicks the 'arteroids menu' button in the Credits, they return to
--the arteroids menu. This handles that action
--********************************************************************************

property spritenum
global gCredits, gCreditsSzaboButton, gCreditsArteroidsMenuButton
global gMainMenu, gArteroidsOpeningGraphic, gCanadaCouncil, gCreditsVismuButton, gCreditsEmailButton, gCreditsIanClayButton

on beginsprite me
  gCreditsArteroidsMenuButton=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locH=sprite(gCredits).locH + 7
  sprite(spritenum).locV=sprite(spritenum-1).locV - sprite(spritenum).height - 10
  sprite(spritenum).locZ=spritenum
end beginsprite

on mouseup me
  sprite(gArteroidsOpeningGraphic).visible=TRUE
  sprite(gMainMenu).visible=TRUE
  sprite(gCredits).visible=FALSE
  sprite(gCanadaCouncil).visible=FALSE
  sprite(gCreditsVismuButton).visible=FALSE
  sprite(gCreditsEmailButton).visible=FALSE
  sprite(gCreditsIanClayButton).visible=FALSE
  sprite(gCreditsSzaboButton).visible=FALSE
  sprite(gCreditsArteroidsMenuButton).visible=FALSE
end mouseup
```

Canto 1 Score and Code Documentation

Channels 23-24 in Frame 4: Initialize Canto/Level 1

This screen serves two functions. It loops while the user gets ready to play Canto 1 and it initializes variables for Canto 1.





Canto 1: Streaming(Texts)

Press any key
to play Canto 1

How to play:
Space key: bomb mot
'S' key: forward
'A' key: backwards
'K' key: left
'L' key: right

-50 for collisions.
10 points per destroyed text.
300 base points ends the canto.

MARKER 1

When players choose to play Canto 1 from the main menu, they are taken to marker 1. In general, level x starts with marker x.

FRAME SCRIPT: LEVEL 1 INITIALIZATION

```
--*********************************************************************************
--LEVEL 1 INITIALIZATION

--This is attached to the frame that initializes level 1. Hardly any of the sprites
--exist at this point in the time line. The frame to which it is attached is the
--first frame in level 1, as evidenced by the '1' marker for this frame.
--*********************************************************************************

global  gUserDataManager, gScoreAndLevelManager
--gUserDataManager is called reset the Missile hits/misses count and set the new
--start time for this level. gScoreAndLevelManager is used to initialize score
--and level info.

global gCurrentlyOnStageBullets, gCurrentlyOnStageBulletsCount, gMissile,
global gCurrentLevel, gBangMemberNumber, gBulletMemberNumber, gLastScore


on beginsprite me
  cursor 0
  gCurrentlyOnStageBullets=[]
  --no bullets and no player on stage right now
  gCurrentlyOnStageBulletsCount=0
  --no bullets and no player on stage right now
  gMissile=[:]
  --this initializes the main data structure for the missiles
  gCurrentLevel=1
  --we're in level 1
  gBangMemberNumber=member("Bang").number
  gBulletMemberNumber=member("bullet").number
  --these two lines initialize what the bullets look like
  the floatPrecision = 2
  --this is for display purposes at the end of the level
  sprite(gUserDataManager).resetMissileHitsAndMissesCount()
  --this resets the data used to calculate accuracy, which is also used in calculating the score
  sprite(gUserDataManager).setLevelInitialTime(gCurrentLevel)
  --this sets the time to 0
  sprite(gScoreAndLevelManager).resetCurrentStage()
  --sets the current stage to stage 2, which is where levels begin.
  --Stage 1 is when the score is negative.
  sprite(gScoreAndLevelManager).addANewLevelScore()
  --we're going into a new level, so add an entry to the pScore list.
  sprite(gUserDataManager).pauseButtonOn(gCurrentLevel)
  --put the pause button on
  gLastScore=0
  --used in the 'Block End of Level' script to compare recent previous score with current
end startMovie


on exitframe me
  --this works better than a keydown handler in this situation
  if the keypressed <> "" then
    go to the frame + 1
  else
    go to the frame
  end if
end
```

23: MAIN MENU POETRY ANIMATION

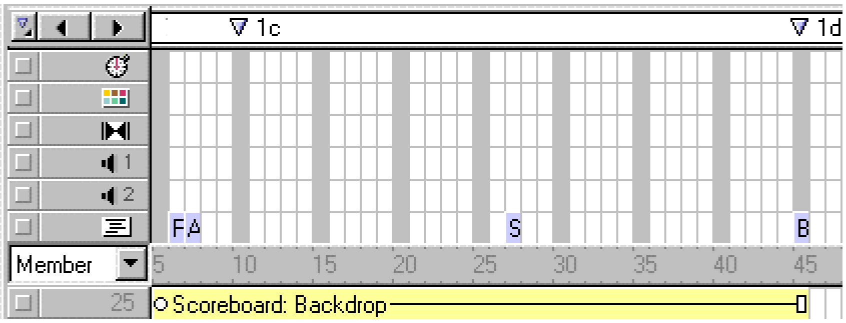
See 10: Main Menu Animation. This is the same member with the same script attached to it.

24: LEVEL X FINAL ENTRY POINT

This script is attached to a text member that looks as above. It appears just prior to entering a level. It gives the player an opportunity to get his/her hands on the keyboard before the level starts.

```
--*********************************************************************************
--LEVEL X FINAL ENTRYPOINT

--After the user selects from the Main Menu a Canto to enter, they are presented
--with a screen that tells them how to play and to press a key to enter the level.
--This script is attached to the text that shows this info.
--*********************************************************************************


property spritenum
global gCurrentLevel, gMessageManager

on beginsprite me
  sprite(spritenum).visible=FALSE
  cursor 0
  sprite(spritenum).member.paragraph[4]= "Canto" && string(gCurrentLevel) & ":" &&
sprite(gMessageManager).levelTitle(gCurrentLevel)
  sprite(spritenum).member.paragraph[7]= "to play Canto" && string(gCurrentLevel)
  sprite(spritenum).member.paragraph[7].color=rgb(255,0,0)
  sprite(spritenum).loc=centerIt(spritenum)
  sprite(spritenum).locZ=spritenum
  sprite(spritenum).visible=TRUE
end
```

Canto 1 Frame scripts and Markers During Play



The above shows frames 5-45. When you first enter to actually play Canto 1, two frame scripts are first executed (Pause Button Off and Add Enemy Texts) in frames 6 and 7 after the sprites for the Canto have been instantiated in frame 5.

MARKERS 1C AND 1D

When you are actually playing Canto 1, the playback head goes from frame 10, which has marker 1c attached to it, to frame 45, which has marker 1d attached to it. In a sense, then, I could have made Canto 1 be a one-frame section. But it is desirable to stretch it over 40 frames (an arbitrary number) because there are computing tasks that need periodically to be checked but it would be a waste of CPU cycles to peform these tasks each frame. The computing tasks I'm referring to are carried out in frames 27 (Stage Dimension Update Script) and 45 (Block End of Level).

FRAME SCRIPT (FRAME 6): PAUSE BUTTON OFF

```
--******************************************************************
--PAUSE BUTTON OFF

--The PauseButtonOn handler is run when the Canto is initialized.
--This is to restart the clock but pause it. However, now that we
--are entering the playing field, we want to turn the pause button
--off
--******************************************************************

global gUserDataManager, gCurrentLevel
global gAControlIsMousedDown, gWithinAControl

on prepareFrame me
  sprite(gUserDataManager).pauseButtonOff(gCurrentLevel)
  gAControlIsMousedDown=0
  gWithinAControl=0
end
```

FRAME SCRIPT (FRAME 7): ADD ENEMY TEXTS

```
--*******************************************************************************
```

```
--ADD ENEMY TEXTS

--This gets attached to an early frame to start out some texts onstage
--initially in Level 1 and 2.
--This initiates the process of throwing Arteroids onto stage. Thereafter,
--checkToAddMoreEnemyTexts() is called after an arteroid finishes exploding.
--********************************************************************************

global gEnemyWordManager

on prepareFrame me
 sprite(gEnemyWordManager).checkToAddMoreEnemyTexts()
end
```

FRAME SCRIPT (FRAME 27): STAGE DIMENSION UPDATE SCRIPT

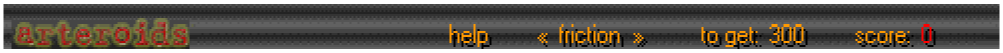
```
--************************************************************
--STAGE DIMENSION UPDATE SCRIPT

--Since the player can resize the stage, there needs to be some updating of the stage dimensions periodically
--because the stage dimensions are used by several of the handlers in the code. But rather than re-calculating the
--stage dimensions every time a handler needs to use them, which is somewhat wasteful of CPU cycles, I've chosen to
--create this script. It is a frame script that is executed once about every second and a half. In level 1, the
--script is in a frame between markers 1c and 1d. More generally, in level x, it should be between markers xc and xd.
--************************************************************

global gCurrentLevel, gEnemyWordManager, gMissile, gStageWidth, gStageHeight, gPlayerShipSpriteNum,
gScoreBoardBackDrop, gYourScore, gScoreboardScoreLabel, gScoreToGet
global gScoreboardScoreToGetLabel, gFrictionPlus, gFrictionDisplaySprite, gFrictionMinus, gHelpButton, gHelpSprite,
gArteroidsGraphic, gClickToEditButton

on prepareFrame me
 if ((the stageRight- the stageLeft)<>gStageWidth) or ((the stageBottom - the stageTop)<>gStageHeight) then
  --then the window has been resized and the refresh routines should be performed
  gStageWidth=the stageRight - the stageLeft
  gStageHeight=the stageBottom - the stageTop
  --reset the globals
  sprite(gPlayerShipSpriteNum).stageDimensionsForPlayerUse()
  sprite(gScoreBoardBackDrop).stageDimensionsForScoreboardBackdrop()
  sprite(gYourScore).stageDimensionsForScoreDisplay()
  sprite(gScoreboardScoreLabel).stageDimensionsForScoreBoardPositioning()
  sprite(gScoreToGet).stageDimensionsForScoreBoardPositioning()
  sprite(gScoreboardScoreToGetLabel).stageDimensionsForScoreBoardPositioning()
  sprite(gFrictionPlus).stageDimensionsForScoreBoardPositioning()
  sprite(gFrictionDisplaySprite).stageDimensionsForScoreBoardPositioning()
  sprite(gFrictionMinus).stageDimensionsForScoreBoardPositioning()
  sprite(gHelpButton).stageDimensionsForScoreBoardPositioning()
  if gCurrentLevel=2 then
   sprite(gClickToEditButton).stageDimensionsForScoreBoardPositioning()
  end if
  sprite(gArteroidsGraphic).positionArteroidsGraphic()
  sprite(gHelpSprite).displayHelpText()
  --the above redraws the scoreboard
  greenArteroids=sprite(gEnemyWordManager).pEnemyWordSpritesByType["#SingleCircleGreen"]
  repeat with i=1 to greenArteroids.count
   sprite(greenArteroids[i]).reCalcStageDimensions()
  end repeat
  --the above loop sets the green arteroids aright
  repeat with i=1 to gMissile.count
   sprite(gMissile.getPropAt(i)).calcMissileBorders()
```

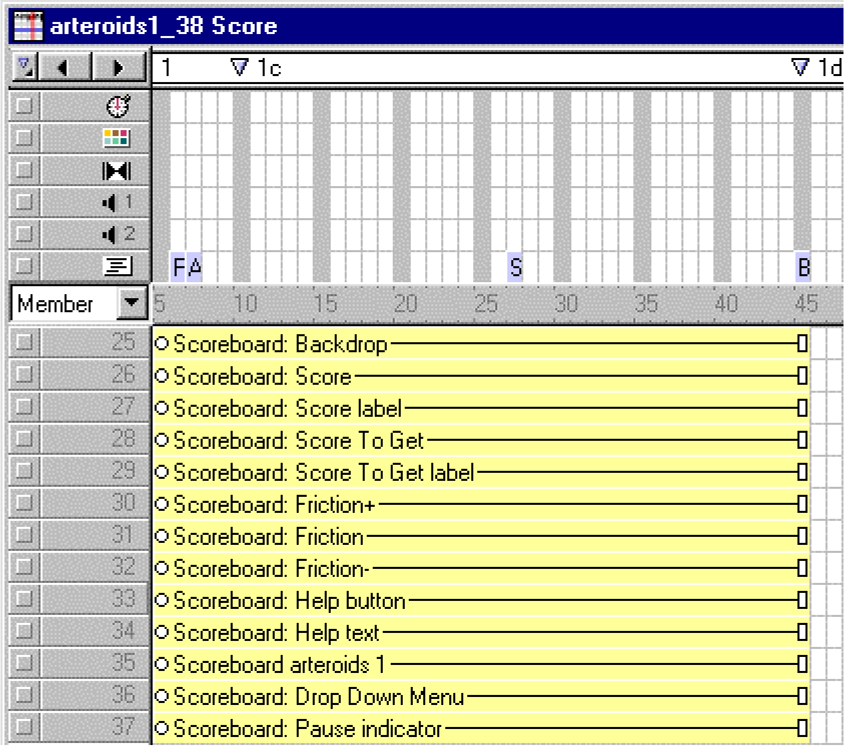
```
    end repeat
    --the above loop sets the missiles aright
  end if
end
```

FRAME SCRIPT (FRAME 45): BLOCK END OF LEVEL

```
--*******************************************************************************************
--BLOCK END OF LEVEL

--This is attached to the frame at the end of a level.
--Note that this is the only type of script with an 'on exitframe' handler.
--The rest of the frame events are 'on prepareframe'. This is important.
--There's a lot of processing involved in each frame. This script
--can move the playback head to a new level, so you want there to be
--no subsequent frame processing for this frame when it makes such a move.
--*******************************************************************************************

global gScoreAndLevelManager, gCurrentLevel, gTimeToQuit
global gLastScore


on exitFrame me
  tempScore=sprite(gScoreAndLevelManager).getTempLevelScore()
  tempFinalStage=sprite(gScoreAndLevelManager).pLevelAction[gCurrentLevel][#totalStages]
  tempEndOfLevelScore=sprite(gScoreAndLevelManager).pLevelAction[gCurrentLevel][#cutOffPoints][tempFinalStage]
  case 1 of
    gTimeToQuit:
      --end the current level
      gTimeToQuit=FALSE
      sprite(gScoreAndLevelManager).endLevel(gCurrentLevel)
    (tempScore > gLastScore):
      if (tempScore > tempEndOfLevelScore)  then
        --end the current level
        sprite(gScoreAndLevelManager).endLevel(gCurrentLevel)
      else
        --adjustStageUp
        sprite(gScoreAndLevelManager).adjustCurrentStageUp() --it might be time to move to another stage
        gLastScore=tempScore
        go to string(gCurrentLevel) & "c" --so the naming convention of the markers is important
      end if
    (tempScore < gLastScore):
      if (tempScore < sprite(gScoreAndLevelManager).pLevelAction[gCurrentLevel][#cutOffPoints][1]) then
        --poetry is dead
        sprite(gScoreAndLevelManager).playerHasLost()
      else
        --adjustStageDown
        sprite(gScoreAndLevelManager).adjustCurrentStageDown() --player has lost points, so it might be time to go down a stage
        gLastScore=tempScore
        go to string(gCurrentLevel) & "c"
      end if
    otherwise:
      go to string(gCurrentLevel) & "c"
  end case
end exitFrame
```

Channels 25-37: Canto 1 Scoreboard





25	Scoreboard: Backdrop
26	Scoreboard: Score
27	Scoreboard: Score label
28	Scoreboard: Score To Get
29	Scoreboard: Score To Get label
30	Scoreboard: Friction+
31	Scoreboard: Friction
32	Scoreboard: Friction-
33	Scoreboard: Help button
34	Scoreboard: Help text
35	Scoreboard arteroids 1
36	Scoreboard: Drop Down Menu
37	Scoreboard: Pause indicator

The yellow sprites in the score diagram define the canto 1 scoreboard, shown above the score diagram. It should be noted that they also include the drop down menu (channel 36) accessible by mouseover of the 'arteroids' graphic above.

25: Scoreboard BackDrop

```
--**********************************************************************************
--SCOREBOARD BACKDROP SCRIPT

--The yellow sprites in the score are scoreboard sprites. This script is attached
--to the first among them, which is the gray scale backdrop to the scoreboard at
--the top of the screen. Note that this script makes sure that the scoreboard
--displays across the whole screen, however big or small the screen is.
--**********************************************************************************

property spritenum
global gScoreboardBackDrop, gStageWidth

on beginsprite me
  gScoreBoardBackDrop=spritenum
  sprite(gScoreBoardBackDrop).stageDimensionsForScoreboardBackdrop()
end beginsprite

--sprite(gScoreBoardBackDrop).stageDimensionsForScoreboardBackdrop()
on stageDimensionsForScoreboardBackdrop me
  sprite(spritenum).height=25
  sprite(spritenum).locH = -1
  sprite(spritenum).top=0
  --sprite(spritenum).bottom=sprite(spritenum).height
  sprite(spritenum).locV=-1
  sprite(spritenum).width=gStageWidth + 1
end stageDimensionsForScoreboardBackdrop
```

26: SCOREBOARD SCORE

```
--***********************************************************************************
--SCOREBOARD: SCORE SCRIPT

--This displays the player's score, which is updated by the Score and Level
--Manager in addPoints and SubtractPoints, for instance.
--***********************************************************************************


property spritenum
global gYourScore, gScoreBoardBackDrop, gStageWidth

on beginsprite me
  gYourScore=spritenum
  sprite(spritenum).member.text="0"
  sprite(gYourScore).stageDimensionsForScoreDisplay()
  sprite(spritenum).locZ=50000
end beginsprite

on stageDimensionsForScoreDisplay me
  sprite(spritenum).locH= gStageWidth - sprite(spritenum).width -5
  sprite(spritenum).locV=sprite(gScoreBoardBackDrop).bottom - sprite(spritenum).height
end stageDimensionsForScoreDisplay
```

27: SCOREBOARD SCORE LABEL

```
--********************************************************
--SCOREBOARD SCORE LABEL SCRIPT

--This is attached to the text member that says 'score: '
--********************************************************


property spritenum
global gScoreboardScoreLabel

on beginsprite me
  gScoreboardScoreLabel=spritenum
end beginsprite



--***********************************************************************************
--SCOREBOARD: POSITIONING SCRIPT

--This is attached to several of the scoreboard sprites to position them. The
--gScoreBackdrop sprite gets positioned toward the right top of the backdrop sprite,
--and then the scoreboard sprites get positioned relative to the sprite they're
--next to (hence the spritenum-1 deal below).
--***********************************************************************************
```

```
property spritenum

on beginsprite me
  sprite(spritenum).stageDimensionsForScoreBoardPositioning()
end beginsprite

on stageDimensionsForScoreBoardPositioning me
  sprite(spritenum).locH= sprite(spritenum-1).left - 1 - sprite(spritenum).width
  sprite(spritenum).locV=sprite(spritenum-1).locV
end stageDimensionsForScoreBoardPositioning
```

28: SCOREBOARD SCORE TO GET

This also has the Scoreboard Positioning script attached to it (see Channel 27).

```
--*********************************************************************************
--SCOREBOARD: SCORE TO GET

--The scoreboard shows a 'Score To Get' number. This number is got as shown below.
--*********************************************************************************


property spritenum
global gScoreToGet, gScoreAndLevelManager

on beginsprite me
  gScoreToGet=spritenum
  sprite(spritenum).member.text=string(sprite(gScoreAndLevelManager).getTypicalAndEndScore())
end beginsprite
```

29: SCOREBOARD SCORE TO GET LABEL

This also has the Scoreboard Positioning script attached to it (see Channel 27).

```
--***********************************************
--SCOREBOARD SCORE TO GET LABEL SCRIPT

--This is attached to the text sprite that says
--'to get'
--***********************************************

property spritenum
global gScoreboardScoreToGetLabel

on beginsprite me
  gScoreboardScoreToGetLabel=spritenum
end beginsprite
```

30: SCOREBOARD FRICTION+

This also has the Scoreboard Positioning script attached to it (see Channel 27).

It also has the Menu Colors script for controlling rollover color and cursor shape (see Channel 18).

```
--******************************************************************************
--SCOREBOARD: FRICTION+ SCRIPT

--This script controls the player's ability to increase friction. gFriction is
--currently initialized in the 'Startmovie etc' script to .9. That is displayed to
--the user as (1-gFriction)*10, or 1.0 (the norm). When the user clicks the
--friction+ button, gFriction is decreased by 0.01, to a value of .89. Which is
--displayed to the user as (1-.89)*10=1.1. When you increase the friction, you
--should be displayed a higher number, right, not a lower one. Hence this scheme.
--To see how gFriction affects the player, look at gFriction in the 'Player' script.
--******************************************************************************

property spritenum
global gFriction
global gFrictionDisplaySprite, gFrictionPlus

on beginsprite me
  gFrictionPlus=spritenum
end beginsprite

on mouseenter me
  --If you play with the friction while the game is playing, you see
  --that on mouseenter a number is displayed. This does it.
  sprite(gFrictionDisplaySprite).member.text= string((1-gFriction)*10)
end mouseenter

on mouseleave me
  sprite(gFrictionDisplaySprite).member.text= "friction"
end mouseleave

on mousedown me
  --gFriction varies between .8 and 1. When gFriction=.8, there is
  --the most friction. When gFriction=1, there is no friction.
  if gFriction > .8 then
    gFriction=gFriction - 0.01
  end if
  sprite(gFrictionDisplaySprite).member.text= string((1-gFriction)*10)
end mousedown
```

31: SCOREBOARD FRICTION

This also has the Scoreboard Positioning script attached to it (see Channel 27).

It also has the Menu Colors script for controlling rollover color and cursor shape (see Channel 18).

```
--******************************************************************************
--SCOREBOARD: FRICTION

--This initializes the Scoreboard word friction.
--******************************************************************************


property spritenum
global gFrictionDisplaySprite

on beginsprite me
  gFrictionDisplaySprite=spritenum
  sprite(spritenum).member.text="friction"
```

```
end beginsprite
```

32: SCOREBOARD FRICTION -

This also has the Scoreboard Positioning script attached to it (see Channel 27).

It also has the Menu Colors script for controlling rollover color and cursor shape (see Channel 18).

```
--********************************************************************************
--SCOREBOARD: FRICTION-

--This controls the player's ability to decrease friction. See the documentation
--in the 'Scoreboard: Friction+' script to understand the below, because it is
--much the same, only gFriction is being increased below, not decreased.
--********************************************************************************

property spritenum
global gFriction
global gFrictionDisplaySprite, gFrictionMinus

on beginsprite me
  gFrictionMinus=spritenum
end beginsprite


on mouseenter me
  sprite(gFrictionDisplaySprite).member.text= string((1-gFriction)*10)
end mouseenter

on mouseleave me
  sprite(gFrictionDisplaySprite).member.text= "friction"
end mouseleave


on mousedown me
  if gFriction < 1 then
    gFriction=gFriction + 0.01
  end if
  sprite(gFrictionDisplaySprite).member.text= string((1-gFriction)*10)
end mousedown
```

33: SCOREBOARD HELP BUTTON

This also has the Scoreboard Positioning script attached to it (see Channel 27).

It also has the Menu Colors script for controlling rollover color and cursor shape (see Channel 18).

```
--********************************************************************************
--SCOREBOARD: HELP BUTTON

--When the player clicks 'help' on the scoreboard, Help (gHelpSprite) is displayed
--and the game is paused.
--********************************************************************************
```

```
property spritenum
global gHelpSprite, gDropDownArteroidsMenu, gHelpButton

on beginsprite me
  gHelpButton=spritenum
end beginsprite


on mouseup me
  sprite(gHelpSprite).visible=true
  cursor 0
  sprite(gDropDownArteroidsMenu).pausePlay()
end mouseup
```

34: SCOREBOARD HELP TEXT

```
--*************************************************************************************
--SCOREBOARD: HELP TEXT

--When the user clicks 'help' on the Scoreboard, then some Help text is displayed.
--This is the script that controls the field member that contains the Help text.
--*************************************************************************************


property spritenum
global gHelpSprite
global gStageWidth, gStageHeight

on beginsprite me
  gHelpSprite=spritenum
  sprite(spritenum).visible=FALSE
  sprite(gHelpSprite).displayHelpText()
end beginsprite

on displayHelpText me
  --sprite(gHelpSprite).displayHelpText()
  myWidth=sprite(spritenum).width
  myHeight=sprite(spritenum).height
  sprite(spritenum).locH = (gStageWidth-myWidth)/2
  sprite(spritenum).locV = (gStageHeight-myHeight)/2
end displayHelpText
```

35: SCOREBOARD ARTEROIDS GRAPHIC TOP LEFT

```
--*************************************************************************************
--SCOREBOARD: ARTEROIDS GRAPHIC TOP LEFT

--The scoreboard contains an arteroids graphic top left which, on mouseenter,
--displays some menu items. This is the script that controls the display of those
--menu items.
--*************************************************************************************


property spritenum
global gArteroidsGraphic, gDropDownArteroidsMenu, gScoreBoardBackDrop

on beginsprite me
  gArteroidsGraphic=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).member=member("Scoreboard arteroids 1")
  sprite(spritenum).positionArteroidsGraphic()
  sprite(spritenum).visible=TRUE
end beginsprite

on mouseenter me
  sprite(spritenum).member=member("Scoreboard arteroids 1b")
  sprite(gDropDownArteroidsMenu).visible=true
end mouseenter

on positionArteroidsGraphic me
  sprite(spritenum).locH=5
  sprite(spritenum).locV=sprite(gScoreBoardBackDrop).bottom - sprite(spritenum).height
end positionArteroidsGraphic


--****************************************************
--CURSOR CONTROL

--This behavior is attached to sprites that are
--clickable. This behavior controls cursor display
--****************************************************

global gAControlIsMousedDown, gWithinAControl
property spritenum

on mouseenter me
  put spritenum
  cursor 280
  gWithinAControl=TRUE
end mouseenter

on mousedown me
  cursor 280
  gAControlIsMousedDown=spritenum
end mousedown

on mouseleave me
  if not gAControlIsMousedDown then
    cursor 0
  end if
  gWithinAControl=FALSE
end mouseleave

on mouseup me
  gAControlIsMousedDown=0
end mouseup

on mouseupOutside me
  if gAControlIsMousedDown=spritenum then
    gAControlIsMousedDown=0
  end if
  if not gWithinAControl then
    cursor 0
  end if
end mouseupOutside
```

36: SCOREBOARD DROP DOWN MENU SCRIPT

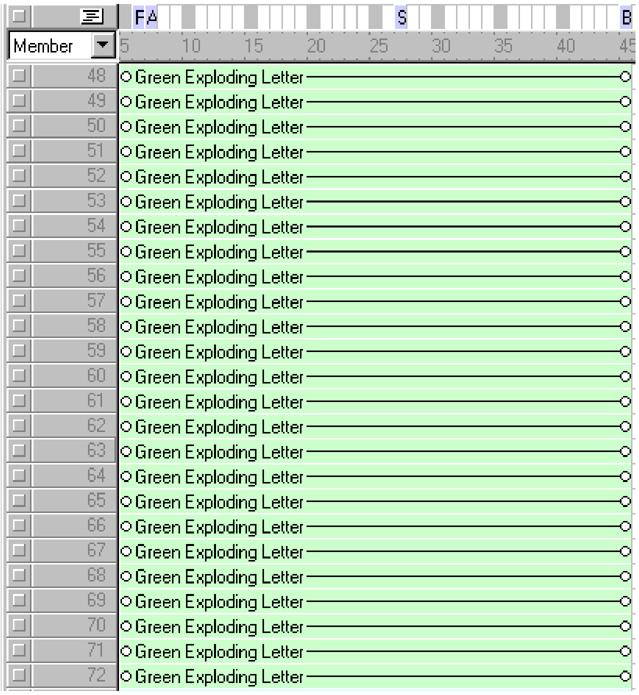
This sprite also has the Cursor Control behavior attached to it (see channel 35)

```
--*********************************************************************************
--SCOREBOARD: MENU SCRIPT

--This script is attached to a drop down menu reachable by mouseover of the
--arteroids graphic top left. It lets the user select and display other texts
--s/he has saved.
--*********************************************************************************

global gUserDataManager, gScoreAndLevelManager
property spritenum, pOldSelectedTextNum, pPausing
global gTimeToQuit, gDropDownArteroidsMenu, gArteroidsGraphic, gPauseIndicator,
global gCurrentLevel, gHelpSprite

on beginsprite me
  gDropDownArteroidsMenu=spritenum
  sprite(spritenum).visible=FALSE
  gTimeToQuit=FALSE
  --One of the menu items is 'Exit Canto'. When the user clicks to exit the canto,
  --gTimeToQuit is set to true, and acted on when the score hits the 'Block end of
  --level' script. Why only then? Because the 'Block end of level' script contains
  --an 'on frameexit' script. If you just exit the level any old time, there are
  --bound to be problems. Note that the 'Block end of level' script is one of the
  --only handlers with an 'on frameexit' script.
  sprite(spritenum).locH=5
  sprite(spritenum).locV=sprite(gArteroidsGraphic).bottom
  sprite(spritenum).refreshMe()
end beginsprite


on refreshme me
  set the foreColor of field sprite(spritenum).member = 27
  --27 is orange
  pOldSelectedTextNum=0
  --This is used (as elsewhere in the program) to control color display of text.
end refreshme


on mouseup me
  tempSelectedTextNum=sprite(spritenum).pointToLine(the MouseLoc)
  case tempSelectedTextNum of
    1:
      --Pause
      sprite(spritenum).pausePlay()
    2:
      --Exit Canto
      gTimeToQuit=TRUE
      sprite(gScoreAndLevelManager).pExitedPrematurely=TRUE
      sprite(gPauseIndicator).member.text="preparing to exit canto"
    3:
      --Exit Arteroids
      gotonetpage "javascript:parent.close()"
      --Note that this is parent.close(), not window.close(). The reason is
      --because to achieve the full full fullscreen for IE, the html files
      --are in a frameset.
    otherwise:
  end case
end mouseup

on pausePlay me
  --This routine pauses play. It is called when the user selects
  --the 'Pause' menu item. It is also called when the user clicks
  --'help' from the scoreboard.
  sprite(gPauseIndicator).member.text="press a key to go"
  cursor 0
  updatestage
  pPausing=1
  --this indicates we're in pause mode
  flag=0
  sprite(gUserDataManager).pauseButtonOn(gCurrentLevel)
  --the above line pauses the timer.
  repeat while pPausing
    --This is a tight repeat loop. Note that when you pause the game,
    --the playback head does not move. This is because it is in this
    --repeat loop.
    if the keypressed <>"" then
      if flag = 1 then
        pPausing=0
        flag = 0
        sprite(gPauseIndicator).member.text=""
        sprite(gHelpSprite).visible=FALSE
        sprite(gUserDataManager).pauseButtonOff(gCurrentLevel)
        --This resumes the timer.
      else
        flag = 1
      end if
    end if
  end repeat
end pausePlay


on mousewithin me
  --This controls the text color on mousewithin
  tempNewSelectedTextNum=sprite(spritenum).pointToLine(the MouseLoc)
  if tempNewSelectedTextNum>0 then
    if pOldSelectedTextNum<>tempNewSelectedTextNum then
      if pOldSelectedTextNum >0 then
        set the foreColor of line pOldSelectedTextNum of field sprite(spritenum).member = 27
        --orange
      end if
      set the foreColor of line tempNewSelectedTextNum of field sprite(spritenum).member = 6
      --red
      pOldSelectedTextNum=tempNewSelectedTextNum
    end if
  end if
end mousewithin

on mouseleave me
  set the foreColor of field sprite(spritenum).member = 27
  sprite(spritenum).visible=FALSE
  pOldSelectedTextNum=0
  sprite(gArteroidsGraphic).member=member("Scoreboard arteroids 1")
end mouseleave
```

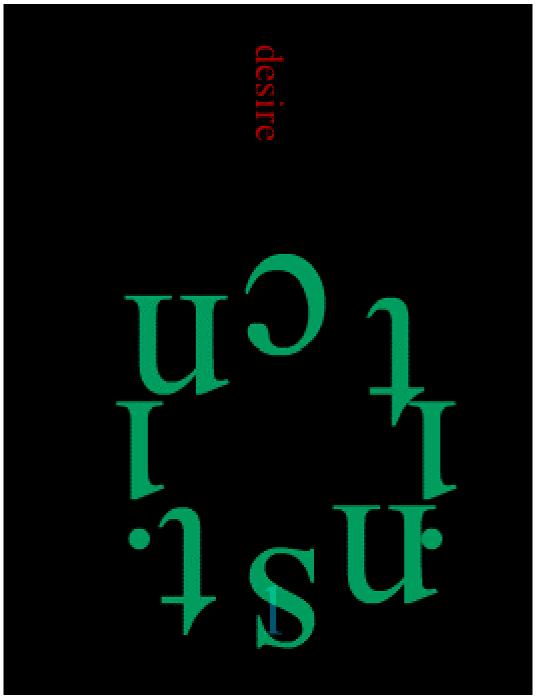
37: PAUSE INDICATOR SCRIPT

```
--*********************************************************************************
--SCOREBOARD: PAUSE INDICATOR SCRIPT

--When the user clicks to pause the game, the scoreboard says, in red, 'press a
--key to go'. This script is attached to that field member sprite.
--*********************************************************************************

property spritenum
global gPauseIndicator, gScoreBoardBackDrop, gArteroidsGraphic

on beginsprite me
  gPauseIndicator=spritenum
  sprite(gPauseIndicator).visible=FALSE
  sprite(gPauseIndicator).member.text=""
  sprite(gPauseIndicator).locH=sprite(gArteroidsGraphic).right + 10
  sprite(gPauseIndicator).locV=sprite(gScoreBoardBackDrop).bottom-sprite(gPauseIndicator).height
  sprite(gPauseIndicator).visible=TRUE
end beginsprite
```

Channels 48-72: Green Exploding Letters



48	Green Exploding Letter
49	Green Exploding Letter
50	Green Exploding Letter
51	Green Exploding Letter
52	Green Exploding Letter
53	Green Exploding Letter
54	Green Exploding Letter
55	Green Exploding Letter
56	Green Exploding Letter
57	Green Exploding Letter
58	Green Exploding Letter
59	Green Exploding Letter
60	Green Exploding Letter
61	Green Exploding Letter
62	Green Exploding Letter
63	Green Exploding Letter
64	Green Exploding Letter
65	Green Exploding Letter
66	Green Exploding Letter
67	Green Exploding Letter
68	Green Exploding Letter
69	Green Exploding Letter
70	Green Exploding Letter
71	Green Exploding Letter
72	Green Exploding Letter



These sprites are the green exploding letters you see when you hit a text onscreen in Canto 1. They all have the same member: the Flash 4 SWF called Green Exploding Letter. This SWF contains many frames, and in each frame there is a different ASCII character, and there is one frame for each visible ASCII character. Moreover, the ASCII characters in the SWF are in appropriately numbered frames in the SWF. By this I mean that letter "A", for instance, is in frame 65 of the SWF because charToNum("A")=65. As you will see in the code below, this makes for being able to throw it onto the screen quite fast.

Note that there are only 25 exploding green letters. You don't really want a lot more than that for performance reasons:

when you get 25 Flash animations onscreen at once (plus the 25 blue ones) that does take a bit of a toll on performance speed.

Note that the exploding green letter in channel 48 has the 'Exploding Green Letter Manager' script attached to it. This script manages the 25 exploding green letters. Channel 48 has this script attached to it, whereas the rest of the sprites only have the 'Exploding Green Letter script' attached to them.

48: EXPLODING GREEN LETTER MANAGER

```
--**************************************************************************************
--EXPLODING GREEN LETTER MANAGER

--This script is attached to the first green exploding letter. As with some other
--groups of same-colored sprites in the score, the green letters have a manager
--sprite. Manager sprites are always the first of the sprites, if the group has
--a manager at all. The reason that the manager is the first of the sprites is
--because the manager initializes variables that the other same-colored sprites
--need access to, and the higher up in the score a sprite is, the sooner it's
--routines execute.

--When the Green Letter Manager 'on beginsprite' handler executes, it creates the
--pAvailable list. Subsequent exploding green letters add themselves to this list.
--**************************************************************************************


property pAvailable, spritenum
global gGreenLetterManager, gLayerManagerLocZ

on beginsprite me
  gGreenLetterManager=spritenum
  pAvailable=[]
  --pAvailable is the list of green letters available for explosions.
  --Green letters add themselves to this list on beginsprite (see the behavior attached to them)
  gLayerManagerLocZ=-40000
  --This LocZ is incremented as letters explode. This incremented value is assigned to be the
  --LocZ of exploded letters once they are finished exploding, so they are underneath everything.
end beginsprite


on getChannels me, NumberOfChannelsRequested, ForSprite, StringToExplode
  --This is called by an arteroid right before it explodes. This routine
  --returns a list of green letter sprite numbers to the exploding text for it to explode with.
  --There are currently 25 green letters. Each of them is a flash animation.
  --I didn't want a lot more than 25 for performance reasons.
  NumberOfChannelsAllocated=0
  ListOfChannelsToAllocate=[]
  i=1
  repeat while (NumberOfChannelsAllocated < NumberOfChannelsRequested) and (pAvailable <> [])
    --repeat while you haven't allocated enough channels and there are channels to allocate
    tempChannel=pAvailable[1]
    ListOfChannelsToAllocate.append(tempChannel)
    NumberOfChannelsAllocated=NumberOfChannelsAllocated+1
    pAvailable.deleteAt(1)
  end repeat
  repeat with i = 1 to NumberOfChannelsAllocated
    sprite(ListOfChannelsToAllocate[i]).locZ=gLayerManagerLocZ
  end repeat
  gLayerManagerLocZ=gLayerManagerLocZ+1
  return ListOfChannelsToAllocate.duplicate()
end getChannels


on returnChannels me, channelList
  --Green exploding Arteroids call this after they've finished exploding
  --to return the green letters they used to explode with (rentals only, you can't take them with you).
  repeat with i=1 to channelList.count
    sprite(channelList[i]).ink=0
    sprite(channelList[i]).rotation=0
    pAvailable.append(channelList[i])
  end repeat
end returnChannels
```

48-72: EXPLODING GREEN LETTER SCRIPT

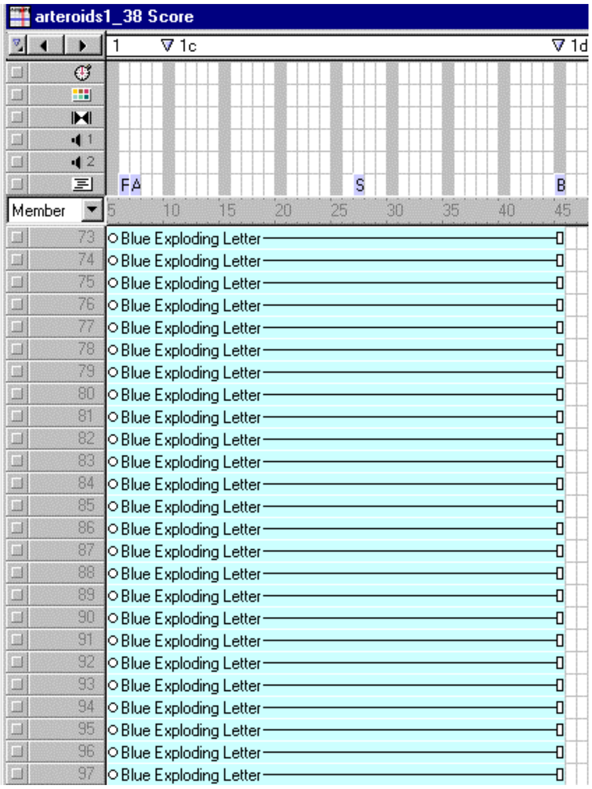
```
--***************************************************************************************
--EXPLODING GREEN LETTER SCRIPT

--This script gets attached to each of the green exploding letters in the score.
--It initializes the letter and also lets the Green Letter Manager
--know its ready for action.
--***************************************************************************************


property spritenum
global gGreenLetterManager

on beginSprite me
  sprite(gGreenLetterManager).pAvailable.append(spritenum)
  sprite(spritenum).blend=100
  sprite(spritenum).rotation=0
  sprite(spritenum).locH=-500
  sprite(spritenum).locV=-500
  sprite(spritenum).ink=0
  sprite(spritenum).visible=TRUE
end beginSprite
```

Channels 73-97: Blue Exploding Letters

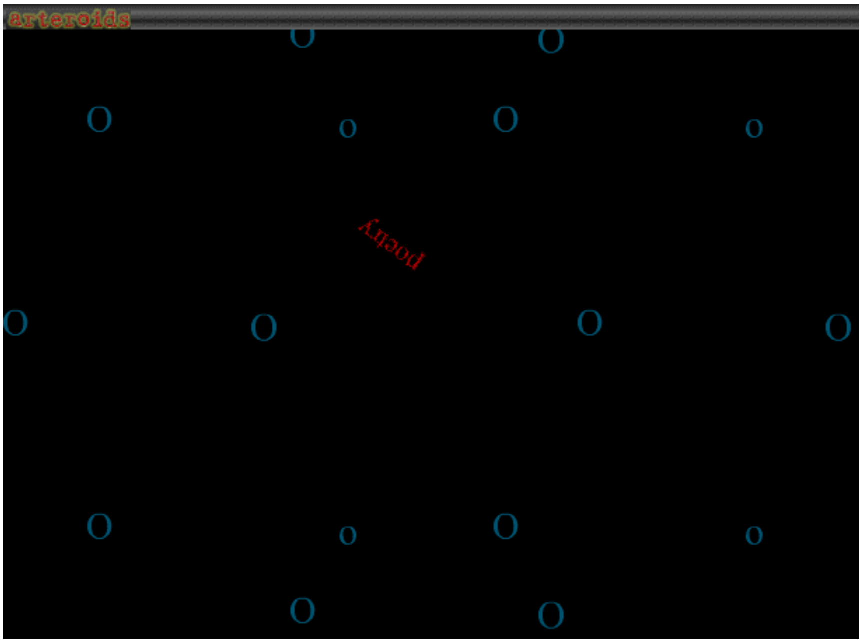
Just as the exploding green letters use a single imported Flash 4 SWF, so too do the exploding blue letters use a single, though different, imported Flash 4 SWF. The one for the blue letters is exactly like the SWF for the green letters except that the letters are blue. See the "Channels 48-72: Green Exploding Letters" section of this doc for a lowdown on the green letters.

In Canto 1, the texts for the exploding blue letters is drawn from the Inner Blue Text in frame 1 of the movie. If you change that text, you change the exploding blue texts that occur in Canto 1.

The code for the exploding blue letters is much the same as the code for the exploding green letters.



73 Blue Exploding Letter
74 Blue Exploding Letter
75 Blue Exploding Letter
76 Blue Exploding Letter
77 Blue Exploding Letter
78 Blue Exploding Letter
79 Blue Exploding Letter
80 Blue Exploding Letter
81 Blue Exploding Letter
82 Blue Exploding Letter
83 Blue Exploding Letter
84 Blue Exploding Letter
85 Blue Exploding Letter
86 Blue Exploding Letter
87 Blue Exploding Letter
88 Blue Exploding Letter
89 Blue Exploding Letter
90 Blue Exploding Letter
91 Blue Exploding Letter
92 Blue Exploding Letter
93 Blue Exploding Letter
94 Blue Exploding Letter
95 Blue Exploding Letter
96 Blue Exploding Letter
97 Blue Exploding Letter



73: EXPLODING BLUE LETTER MANAGER

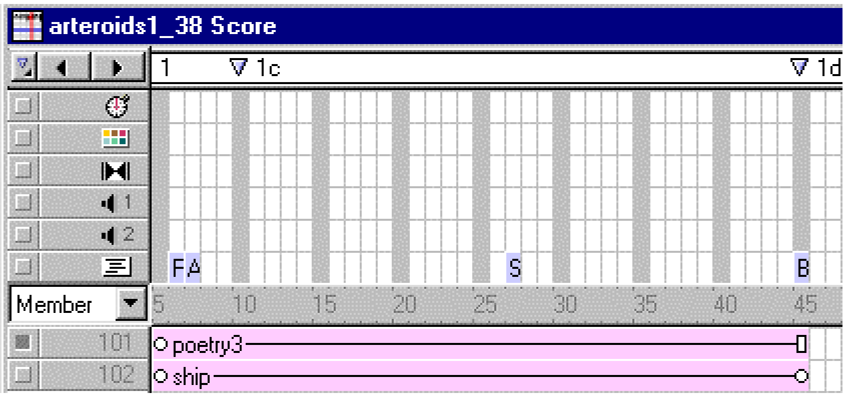
```
--*************************************************************************************
--EXPLODING BLUE LETTER MANAGER

--This script is attached to the first blue exploding letter. As with some other
--groups of same-colored sprites in the score, the blue letters have a manager
--sprite. Manager sprites are always the first of the sprites, if the group has
--a manager at all. The reason that the manager is the first of the sprites is
--because the manager initializes variables that the other same-colored sprites
--need access to, and the higher up in the score a sprite is, the sooner it's
--routines execute.

--When the Blue Letter Manager 'on beginsprite' handler executes, it creates the
--pAvailable list.
--*************************************************************************************
```

```
property pAvailable, spritenum
global gBlueLetterManager, gLayerManagerLocZ

on beginsprite me
  gBlueLetterManager=spritenum
  pAvailable=[]
  gLayerManagerLocZ=-30000
end beginsprite


on getChannels me, NumberOfChannelsRequested, ForSprite, StringToExplode
  --This is called by a blue webarteroid when the webarteroid is hit. This handler
  --allocates some blue letters to the webarteroid to explode with.
  NumberOfChannelsAllocated=0
  ListOfChannelsToAllocate=[]
  i=1
  repeat while (NumberOfChannelsAllocated < NumberOfChannelsRequested) and (pAvailable <> [])
    tempChannel=pAvailable[1]
    ListOfChannelsToAllocate.append(tempChannel)
    NumberOfChannelsAllocated=NumberOfChannelsAllocated+1
    pAvailable.deleteAt(1)
  end repeat
  repeat with i = 1 to NumberOfChannelsAllocated
    sprite(ListOfChannelsToAllocate[i]).locZ=gLayerManagerLocZ
  end repeat
  return ListOfChannelsToAllocate.duplicate()
end getChannels


on returnChannels me, channelList
  --This is called when an explosion is finished. It returns the
  --letter channels to the Blue Letter Manager so they can be used
  --again for other explosions.
  repeat with i=1 to channelList.count
    sprite(channelList[i]).ink=0
    sprite(channelList[i]).rotation=0
    sprite(channelList[i]).locZ=gLayerManagerLocZ
    pAvailable.append(channelList[i])
  end repeat
  gLayerManagerLocZ=gLayerManagerLocZ+1
  --channelList=[]
end returnChannels
```

73-97: Exploding Blue Letter Script

```
--************************************************************************************
--EXPLODING BLUE LETTER SCRIPT

--This script gets attached to each of the blue exploding letters in the score.
--It initializes the letter and also lets the Blue Letter Manager
--know its ready for action.
--************************************************************************************

property spritenum
global gBlueLetterManager

on beginSprite me
  sprite(gBlueLetterManager).pAvailable.append(spritenum)
  --This lets the Blue letter manager know that this sprite is availalbe for action.
  sprite(spritenum).blend=100
  sprite(spritenum).rotation=0
  sprite(spritenum).locH=-100
  sprite(spritenum).locV=-100
  sprite(spritenum).ink=0
  sprite(spritenum).visible=TRUE
end beginSprite
```

Channels 101-102: The Player (Id-Entity) and Explosion of Player



The first of these sprites is the animation triggered when the player explodes. The second sprite is the red text the player drives in Canto 1.

101: Poetry Explosion

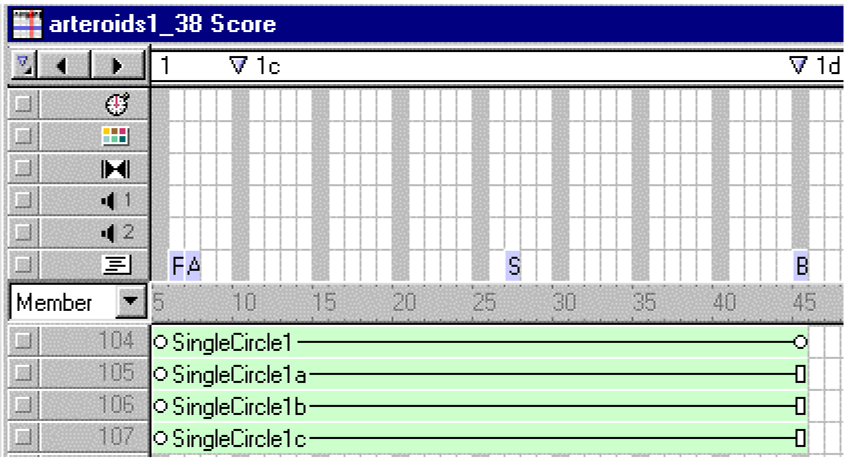
```
--*********************************************************
--POETRY EXPLOSION

--This is attached to the sprite with the Flash 4 import
--member that is displayed when the player's red text
--explodes.
--*********************************************************

property spritenum
global gPoetryExplosion

on beginsprite me
  gPoetryExplosion=spritenum
  sprite(spritenum).visible=FALSE
  sprite(spritenum).stop()
  sprite(spritenum).frame=1
  sprite(spritenum).loc=point(-1000, -1000)
  sprite(spritenum).ink=0
end beginsprite
```

102: The Player (otherwise known as the Id-Entity)

```
--**********************************************************
--PLAYER INIT LEVEL 1

--This gets attached to the player (identity) in Canto 1
--**********************************************************
```

```
property spritenum

on beginsprite me
  sprite(spritenum).member.text="poetry"
  sprite(spritenum).visible=TRUE
end beginsprite


--*************************************************************************************
--PLAYER WIDTH AND REGPOINT INIT

--This gets attached to the id entity.
--This behavior adjusts the width of the id entity so that it's never longer than
--300 pixels, but if it's shorter than that, it's just the right width so that
--it doesn't get hit in invisible spaces by being too long or too wide.
--*************************************************************************************

property pMyTextWidth, spritenum

on beginsprite me
  tempLength=sprite(spritenum).member.line[1].length
  --length of the first line of the identity
  firstApprox=max([16, 6.2*tempLength+8])
  --an ad hoc formula I came up with
  secondApprox= integer(min([300, firstApprox]))
  --Dont make it longer than 300 pixels
  pMyTextWidth=secondApprox
  --Is this used anywhere?
  sprite(spritenum).member.width=pMyTextWidth
  --Besides here I mean.
  theText=sprite(spritenum).member.text
  --This is the id entity's text
  theLineHeight=sprite(spritenum).member.charPosToLoc(1).locV
  --the height of the first character
  theLineCount=sprite(spritenum).member.line.count
  --the number of paragraphs
  repeat with i=1 to theLineCount
    --do this for each paragraph
    sprite(spritenum).member.text=theText.line[i]
    --now the member just has the one paragraph in it
    lastChar=theText.line[i].char.count
    --figure out what the number of the last char is so you can compare its height with the height of the first char in the paragraph
    repeat while sprite(spritenum).member.width <300 and theLineHeight < sprite(spritenum).member.charPosToLoc(lastChar).locV
      --repeat while the width is less than 300 and the lineHeight of the first char is less than the lineHeight of the last char
      sprite(spritenum).member.width=sprite(spritenum).member.width+2
    end repeat
  end repeat
  sprite(spritenum).member.text=theText
  --now that the width is adjusted, put the whole text back in there
  sprite(spritenum).member.regpoint=point(sprite(spritenum).member.width/2,sprite(spritenum).member.height/2)
  --also adjust the regpoint to be the center point so that it rotates around its center point (nicer)
end beginsprite


--*******************************************************************************
--PLAYER

--This gets attached to the id-entity, ie, the player's ship or token or
--whatever you want to call it.
--*******************************************************************************

global gmissile, gPlayerShipSpriteNum, gBulletMemberNumber, gMissileManager
global gCurrentlyOnStageBullets, gCurrentlyOnStageBulletsCount
global gFriction, gStageWidth, gStageHeight

property pMyLocH,pMyLocV,pMe, pJet,pAngle,pFlag,pMySpeedFactor
property sr, sl, st, sb, spritenum

on beginsprite me
  pFlag = 0
  pMe = me.spritenum
  gPlayerShipSpriteNum=pMe
  pAngle = 0
  pJet = float(0)
  pMyLocH = sprite(pMe).loch
  pMyLocV = sprite(pMe).locv
  sprite(pMe).loc=point(sprite(pMe).member.width/2, sprite(pMe).member.height/2)
  pMySpeedFactor=10
  sprite(gPlayerShipSpriteNum).stageDimensionsForPlayerUse()
  sprite(spritenum).visible=TRUE
end


on stageDimensionsForPlayerUse me
  sr=gStageWidth+pMySpeedFactor --sr is stage right plus a bit
  sl=-pMySpeedFactor
  st=-pMySpeedFactor
  sb=gStageHeight+pMySpeedFactor
end stageDimensionsForPlayerUse


on prepareFrame me
  if sprite(spritenum).pIAmNotExploding then
    if keypressed("a") then
      if pJet > -3 then
        pJet = pJet - 0.3
      end if
    end if

    if keypressed("s") then
      if pJet < 3 then
        pJet = pJet + 0.3
        --this speeds it up
      end if
    end if

    pJet = pJet * gFriction
    --this line slows it down when no key is pressed

    if keypressed ("k") then
      pAngle = pAngle -9
    end if
    if keypressed ("l") then
      pAngle = pAngle + 9
    end if

    mpAngle = pAngle * pi()/180
    pMyLocH = pMyLocH +pMySpeedFactor*cos(mpAngle)*pJet
    pMyLocV = pMyLocV +pMySpeedFactor*sin(mpAngle)*pJet
    onstage me
    --the onstage routine keeps it on the stage
    sprite(pMe).loch = pMyLocH
    sprite(pMe).locv = pMyLocV
    sprite(pMe).rotation = pAngle
    if keypressed (" ") then
      --this fires missiles
      if pFlag = 1 then
        --good use of the pFlag to make it so that if you keep the space key down, it only fires once.
        if sprite(gMissileManager).pMissilesInMotionListCount<2 then
          mymissile = getone(gmissile,0)
          if mymissile <> 0 then
            sprite(mymissile).member=member(gBulletMemberNumber)
            --set the member of sprite mymissile to "bullet"
            sendsprite mymissile #shoot, pMe,pMyLocH,pMyLocV,pAngle
            setprop(gmissile,mymissile,1)
          end if
        end if
      end if
      pFlag = 0
    else
      pFlag = 1
    end if
  end if
end prepareFrame


on onstage me
  case 1 of
    (sprite(pMe).left>sr): pMyLocH= sl --stage left
    (sprite(pMe).right<sl): pMyLocH= sr  --stage right
    (sprite(pMe).top>sb): pMyLocV= st  --stage top
    (sprite(pMe).bottom<st): pMyLocV= sb  --stage bottom
  end case
end onstage


--************************************************************
--PLAYER HIT

--This is attached to the id entity. It involves routines
--for collision detection and the events after the player
--is hit. But the collision detection is actually managed
--for the entire program in The Green and Blue Word Manager.
--************************************************************

property spritenum, pfirstbanger, pOriginalLength, pOriginalWidth, pDeathLoopCount, pIAmNotExploding,
pRepositionEnemyWords
global gScoreAndLevelManager, gPoetryExplosion

on beginsprite me
  pfirstbanger=0
  pDeathLoopCount=0
  pIAmNotExploding=TRUE
  pRepositionEnemyWords=TRUE
end beginsprite


on noCollision me, collidedWith
  --this is the collsion detection routine for the player
  if pfirstbanger=0 then
    --pfirstbanger is useless right now, because it is always 0. But if you want the player, like the Arteroids,
    --to only get hit once and then explode, then you do some stuff with pfirstbanger. As it is, it's useless
    --computing right now.
    if sprite collidedWith intersects spritenum then
      pfirstbanger=1
      sprite(gScoreAndLevelManager).subtractPoints(50)
      pIamNotExploding=FALSE
      sprite(gPoetryExplosion).loc=sprite(spritenum).loc
      sprite(gPoetryExplosion).ink=36
      sprite(gPoetryExplosion).play()
      sprite(gPoetryExplosion).visible=TRUE
      sprite(spritenum).visible=FALSE
      pRepositionEnemyWords=TRUE
    end if
  end if
end


on iAmFinishedExploding me
  pIAmNotExploding=TRUE
  pFirstBanger=0
  sprite(gPoetryExplosion).visible=FALSE
  sprite(gPoetryExplosion).stop()
  sprite(gPoetryExplosion).frame=1
  sprite(gPoetryExplosion).ink=0
  sprite(gPoetryExplosion).loc= point(-1000,-1000)
  sprite(spritenum).visible=TRUE
end iAmFinishedExploding
```

Channels 104-107: Green Texts



These sprites are for green texts that the player shoots in Canto 1. Since there are four such sprites, there can be at most four green sprites on the stage at once in Canto 1. The number of green sprites onstage at once is controlled by the #numberOfEnemySprites property in the pLevelAction property in the Score and Level Manager.

The green texts are Director text members. They travel in an arbitrary straight line on the stage. The text that they display is assigned to them just before they go out on stage. Once out on the stage, they travel around until the player hits them. When that happens, they explode, and are then taken offstage and stay there until the checkToAddMoreEnemyTexts handler in the Green and Blue Word Manager throws them out onstage again.

Channel 104 has a behavior attached to it that is quite special: the Green and Blue Word Manager. This behavior, like the name implies, manages all the blue and green texts (not the exploding texts). Note that there's an 'on prepareFrame' handler in the Green and Blue Word Manager. This is crucial to the program. It manages the collision detection in the program. Basically what it does is it checks each onstage green and blue word to see if they have collided with missiles, and it checks also to see if the player has collided with any blue or green texts. It also trigger makes sure that if there any blue or green texts exploding, then their animations are attended to each frame.

104: GREEN AND BLUE WORD MANAGER

```
--****************************************************************************************************
--GREEN AND BLUE WORD MANAGER

--Attach this to the first enemy word in the score, ie, the first textual arteroid. This manages
--all the enemy words, ie, all the textual Arteroids, both blue and green.
--****************************************************************************************************


global gEnemyWordManager
--This sprite
global gScoreAndLevelManager
--used to send messages to the scorekeeper
global gUserDataManager
--used to get buffer data to initialize texts
global gCurrentLevel
--the current level. Controlled in the LevelManager
global gMessageManager
--used to communicate with the Message Manager
global gCurrentlyOnStageBullets, gCurrentlyOnStageBulletsCount
global gPlayerShipSpriteNum
global gStageWidth, gStageHeight
property spritenum
--This sprite's num
property pEnemyWordSpriteTypes
--A linear list of the types of sprites. Each type has a different behavior associated with it.
property pEnemyWordSpriteTypesByRelativeFrequency
--When the EnemyWordManager timer checks (every second) to see whether it should send more enemy text onto the stage
--it checks the pLevelAction list to see how many enemy texts should be on-stage at the moment. If it needs to add enemy texts,
--it looks at the relative frequency with which each type should occur. This is a number between 1 and 100. The higher the number,
--the more frequently that type will occur.
property pEnemyWordSpriteTypesByOrdering
--Values can be "random" or "sequential" depending on whether the texts for this type of enemy text should be displayed in
sequence
--according to the pEnemyWordSpriteTypesByTexts order or whether the texts can be displayed in random order.
property pEnemyWordSpriteTypesByTexts
--This lists the texts each type of enemy sprite can display
property  pEnemyWordSpriteTypesByTotalNumberOfTexts
--For each type, contains the total number of texts for the particular type.
property pEnemyWordSpriteTypesByTextsCurrentlyDisplayed
--This lists the number of the currently displayed text for each type.
property pEnemyWordSpriteTypesByPointsAwarded
--This lists the number of points for each type of enemy word sprite.
property pEnemyWordSpritesByType
property pResoivoir, pResoivoirCount
--This holds, for each type of enemy word sprite, a list of the spritenums of available sprites that can hold enemy text.
property pEnemyWordSpriteTypesCurrentlyInPlay
--For each type of enemy word sprite, this holds a list of the spritenums of those sprites now on stage in battle.
property pTotalEnemyWordSpritesByTypeCurrentlyOnStage
--Strongly related to the one above it.
property pEnemyWordSpriteTypesAvailable
--For each type, a list of the spritenums that can now be used (aren't already busy).
property pTotalEnemyWordSpritesAvailable
--Total number of sprites available
property pTotalEnemyWordSpritesCurrentlyOnStage
--Numeric total of the enemy word sprites on stage regardless of type.
property pEnemyWordSpritesInMotion, pEnemyWordSpritesInMotionCount
property pCurrentlyInPlayAndNotExploding, pCurrentlyInPlayAndNotExplodingCount
--a subset of the ones in motion.
property pIAmExploding, pIAmExplodingCount

on beginsprite me
  gEnemyWordManager=spritenum
  pEnemyWordSpriteTypes=[]
  pEnemyWordSpriteTypesByRelativeFrequency=[:]
  pEnemyWordSpriteTypesByOrdering=[:]
  pEnemyWordSpriteTypesByTexts=[:]
  pEnemyWordSpriteTypesByTotalNumberOfTexts=[:]
  pEnemyWordSpriteTypesByTextsCurrentlyDisplayed=[:]
  pEnemyWordSpriteTypesByPointsAwarded=[:]
  pEnemyWordSpritesByType=[:]
  pEnemyWordSpriteTypesCurrentlyInPlay= [:]
  pTotalEnemyWordSpritesByTypeCurrentlyOnStage=[:]
  pEnemyWordSpriteTypesAvailable=[:]
  pTotalEnemyWordSpritesAvailable=0
  pTotalEnemyWordSpritesCurrentlyOnStage=0
  pEnemyWordSpritesInMotion=[]
  pEnemyWordSpritesInMotionCount=0
  pCurrentlyInPlayAndNotExploding=[]
  pCurrentlyInPlayAndNotExplodingCount=0
  pIAmExploding=[]
  pIAmExplodingCount=0
  pResoivoir=["OOOOOOOO", "69696969", "@@@@@@@@", "::::::::::", "**********", "EXESEXEXES", "JOYJOYJOY",
"LOVELOVE", "HEARTHEART", "(:)(:)(:)(:)", "WMWMWMWM", "ZNZNZNZN"]
  pResoivoirCount=pResoivoir.count
  sprite(spritenum).checkTextsAndMakeAdjustments()
end beginsprite

on prepareFrame me
  --THIS HANDLER DOES MUCH OF THE PROCESSING IN THE PROGRAM.
  --The collision detection is done here, among other things.
  --I wanted to minimize the number of collision detection checks done.
  --What collisions need to be checked for occurence? Well, if a missile
  --fired by the player collides with an arteroid, that is a collision.
  --And if the player (the id-entity) collides with an arteroid,
  --that is a collision. Concerning the former type of collision, what
  --we do below is check each arteroid in pCurrentlyInPlayAndNotExploding
  --to see if it has collided with a missile or the player. This handler
  --also repositions arteroids and the player in the event that the player
  --collides with an arteroid. And this handler initiates explosion
  --processing each frame also.
  tempDroidsToCheck=pCurrentlyInPlayAndNotExploding.duplicate()
  --This list is duplicated at the outset of the procedure because during
  --the course of the handler, pCurrentlyInPlayAndNotExploding can change,
  --which messes up the index into this list. So the duplicate is used
  --to carry out the checking. The duplicate does not change during the
  --course of this handler.
  tempDroidCount=pCurrentlyInPlayAndNotExplodingCount
  j=1
  repeat with i= 1 to tempDroidCount
    --check each arteroid in play and not exploding
    didNotCollide=TRUE
    repeat while j<= gCurrentlyOnStageBulletsCount and didNotCollide
      --Check each bullet until either you find a collision or you run out of bullets to check
      didNotCollide=sprite(gCurrentlyOnStageBullets[j]).noCollision(tempDroidsToCheck[i])
      j=j+1
    end repeat
    j=1
    if sprite(gPlayerShipSpriteNum).pIAmNotExploding then
```

```
      --if the player is not exploding, check whether the player has collided with this arteroid.
      sprite(gPlayerShipSpriteNum).noCollision(tempDroidsToCheck[i])
    end if
  end repeat
  if sprite(gPlayerShipSpriteNum).pIAmNotExploding then
    --if the player is not exploding, then each arteroid needs to have its new position calculate.
    repeat with i=1 to pEnemyWordSpritesInMotionCount
      sendSprite (pEnemyWordSpritesInMotion[i], #calcNewPosition)
    end repeat
  else
    --if the player is exploding, then the arteroids onscreen and the player need to be
    --repositioned. This is done only once during the explosion of the player.
    if sprite(gPlayerShipSpriteNum).pRepositionEnemyWords then
      --This is set to TRUE when the player collides with an arteroid.
      repeat with i=1 to tempDroidCount
        --reposition each arteroid onstage to the edge of the screen unless it is exploding,
        --in which case leave it where it is.
        exploder=pIAmExploding.getOne(tempDroidsToCheck[i])
        --check to see if this arteroid is currently exploding
        if exploder=0 then
          --if it isn't, then reposition it to a random position along the edge of the stage
          HIs0orStageWidth=random(2)-1
          if HIs0orStageWidth then
            His0=random(2)-1
            if His0 then
              mylocH=0
              mylocV=random(gStageHeight)
            else
              mylocH= gStageWidth
              mylocV=random(gStageHeight)
            end if
          else
            Vis0=random(2)-1
            if Vis0 then
              mylocH=random(gStageWidth)
              mylocV=0
            else
              myLocH=random(gStageWidth)
              myLocV=gStageHeight
            end if
          end if
          sprite(tempDroidsToCheck[i]).myLocH=myLocH
          sprite(tempDroidsToCheck[i]).locH=myLocH
          sprite(tempDroidsToCheck[i]).myLocV=myLocV
          sprite(tempDroidsToCheck[i]).locV=myLocV
        end if
      end repeat
      --now reposition the id-entity to the middle of the stage
      halfStageWidth=gStageWidth/2
      halfStageHeight=gStageHeight/2
      sprite(gPlayerShipSpriteNum).pMyLocH=halfStageWidth
      sprite(gPlayerShipSpriteNum).locH=halfStageWidth
      sprite(gPlayerShipSpriteNum).pMyLocV=halfStageHeight
      sprite(gPlayerShipSpriteNum).locV=halfStageHeight
      sprite(gPlayerShipSpriteNum).pRepositionEnemyWords=FALSE
    end if
  end if
  repeat with i=1 to pIAmExplodingCount
    --this loop does the processing of explosions of arteroids
    sendSprite(pIAmExploding[i], #executeMyGeometry)
  end repeat
end
```

```
on addMeToMovingList me, SpriteToAdd
  --This is called by the 'Follow id-entity' and 'Random Line Motion' scripts in their
  --'setMeInMotion' handlers. In other words, addMeToMovingList is called when an
  --arteroid is sent onto the stage. So pEnemyWordSpritesInMotion is a list that
  --contains the spritenums of arteroids currently on stage.
  pEnemyWordSpritesInMotion.append(SpriteToAdd)
  pEnemyWordSpritesInMotionCount=pEnemyWordSpritesInMotionCount+1
end addMeToMotionList


on deleteMeFromMovingList me, SpriteToDelete
  --This is called by the iveBeenHitByThePlayer handler in the
  --Green And Blue Word Manager script, which is in turn called
  --by scripts such as 'Single Circle Green Asteroid 1' after an
  --arteroid has finished exploding. Note, then, that an arteroid
  --may be exploding and still in this list. The reason is this:
  --note that when arteroids explode, the explosion moves in the
  --same direction as the arteroid had been moving. Keeping arteroids
  --in the pEnemyWordSpritesInMotion list while they are exploding
  --permits this. See also the below handlers.
  pEnemyWordSpritesInMotion.deleteOne(SpriteToDelete)
  pEnemyWordSpritesInMotionCount=pEnemyWordSpritesInMotionCount-1
end deleteMeFromMovingList


on AddMeToCurrentlyInPlayAndNotExploding me, SpriteToAdd
  --Arteroids in this list are currently in play and not exploding,
  --unlike arteroids in pEnemyWordSpritesInMotion which are in play
  --and either exploding or not exploding. AddMeToCurrentlyInPlayAndNotExploding
  --is called, as is addMeToMovingList, in the 'setMeInMotion' handler attached
  --to arteroids that is called just prior to sending an arteroid onstage.
  --It is the pCurrentlyInPlayAndNotExploding list that is used for collision
  --detection in the 'on prepareFrame' handler below. In other words, if an
  --arteroid is exploding, it should not be checked for collision detection.
  pCurrentlyInPlayAndNotExploding.append(SpriteToAdd)
  pCurrentlyInPlayAndNotExplodingCount=pCurrentlyInPlayAndNotExplodingCount+1
end AddMeToCurrentlyInPlayAndNotExploding


on deleteMeFromCurrentlyInPlayAndNotExploding me, SpriteToDelete
  --This is called by arteroids immediately upon their beginning to explode.
  --Arteroids that are exploding are not part of the collision detection scheme.
  pCurrentlyInPlayAndNotExploding.deleteOne(SpriteToDelete)
  pCurrentlyInPlayAndNotExplodingCount=pCurrentlyInPlayAndNotExplodingCount-1
end deleteMeFromCurrentlyInPlayAndNotExploding

on addMeToExplodingList me, SpriteToAdd
  --This is called in the same place as deleteMeFromCurrentlyInPlayAndNotExploding,
  --ie, as soon as an arteroid begins to explode. The pIAmExploding list is used
  --to process explosions (see the 'on prepareFrame' handler below).
  pIAmExploding.append(SpriteToAdd)
  pIAmExplodingCount=pIAmExplodingCount+1
end addmetoexplodinglist

on deleteMeFromExplodingList me, SpriteToDelete
  --This is called in iveBeenHitByThePlayer (below) when an arteroid has finished exploding.
  pIAmExploding.deleteOne(SpriteToDelete)
  pIAmExplodingCount=pIAmExplodingCount-1
end deleteMeFromExplodingList
```

```
on iveBeenHitByThePlayer me, spriteToTakeOutOfPlay, typeOfSprite
  --call this from the exploded sprite AFTER it has finished exploding
  sprite(spriteToTakeOutOfPlay).pCurrentlyInPlay=FALSE
  --an arteroid is still considered in play when it is exploding (even though it is invisible then).
  --Only after it is finished exploding do you give it a false value.
  pEnemyWordSpriteTypesCurrentlyInPlay[typeOfSprite].deleteOne(spriteToTakeOutOfPlay)
pTotalEnemyWordSpritesByTypeCurrentlyOnStage[typeOfSprite]=pTotalEnemyWordSpritesByTypeCurrentlyOnStage[typeOfSprite] - 1
  pEnemyWordSpriteTypesAvailable[typeOfSprite].append(spriteToTakeOutOfPlay)
  pTotalEnemyWordSpritesAvailable=pTotalEnemyWordSpritesAvailable+1
  pTotalEnemyWordSpritesCurrentlyOnStage=pTotalEnemyWordSpritesCurrentlyOnStage-1
  sprite(gScoreAndLevelManager).addPoints(pEnemyWordSpriteTypesByPointsAwarded[typeOfSprite])
  sprite(spritenum).deleteMeFromMovingList(spriteToTakeOutOfPlay)
  sprite(spritenum).deleteMeFromExplodingList(spriteToTakeOutOfPlay)
  checkToAddMoreEnemyTexts
  sprite(spriteToTakeOutOfPlay).locH=-110
  sprite(spriteToTakeOutOfPlay).locV=-110
end iveBeenHitByThePlayer


on checkToAddMoreEnemyTexts me
  --This is called in a framescript near the beginning of a level called 'Add Enemy Texts' to add some textual onscreen Arteroids
  --and it is called after a textual webarteroid explodes to accomplish the same goal.
  --The code here is fairly tricky.
  repeat while (pTotalEnemyWordSpritesCurrentlyOnStage < sprite(gScoreAndLevelManager).pLevelAction[gCurrentLevel][#numberOfEnemySprites][sprite(gScoreAndLevelManager).pLevelAction[gCurrentLevel][#currentStage]]) and (pTotalEnemyWordSpritesAvailable > 0) then
    --repeat while there are fewer textual Arteroids onscreen than there should be *and* while there are some available to add.
    tempAvailableSpriteTypes=[]
    tempNumberOfAvailableSpriteTypes=0
    tempCumulativeHistory=0
    tempCumulativeHistoryList=[]
    repeat with i = 1 to pEnemyWordSpriteTypes.count
      --Here's the idea: each type of textual webarteroid has a pRelativeFrequency assigned to it (0-100). This indicates how often it occurs
      --onscreen relative to other types of textual Arteroids. So, if there are sprites available within the type, consider choosing
      --one of this type, and add the relative frequency to a cumulative history that consists of the sum of all the relative frequencies
      --of the available types. Then, conceptually, construct a number line from one to the sum of the relative frequencies, and partition
      --the line so that each partition has the length of some available type's relative frequency. Then choose a random number
      --somewhere on the line. Wherever the random number is, choose the corresponding textual webarteroid type to deploy on the screen.
      if pEnemyWordSpriteTypesAvailable[i] <> [] then
        tempAvailableSpriteTypes.append(i)
        tempNumberOfAvailableSpriteTypes=tempNumberOfAvailableSpriteTypes+1
        tempCumulativeHistory=tempCumulativeHistory + pEnemyWordSpriteTypesByRelativeFrequency[i]
        tempCumulativeHistoryList.append(tempCumulativeHistory)
      end if
    end repeat
    --The above code constructs the number line mentioned above.
    tempRandomSpot=random(tempCumulativeHistory)
    --Now choose the random number.
    i=1
    repeat while (tempRandomSpot > tempCumulativeHistoryList[i]) and (i<= tempNumberOfAvailableSpriteTypes)
      i=i+1
    end repeat
    --The above loop figures out which partition of the number line the random number falls in
    sprite(spritenum).putThisSpriteTypeIntoPlay(pEnemyWordSpriteTypes[tempAvailableSpriteTypes[i]])
    --This last line is the crucial one that actually puts the arteroid into play.
  end repeat
end checktoAddMoreEnemyTexts
```

```
on putThisSpriteTypeIntoPlay me, spriteType
  --This is called by the above handler when it's time to actually put a particular type onto the stage.
  tempSprite=pEnemyWordSpriteTypesAvailable[spriteType][1]
  pEnemyWordSpriteTypesAvailable[spriteType].deleteAt(1)
  if pEnemyWordSpriteTypesByOrdering[spriteType]="sequential" then
    --if the type is 'sequential', then the texts are displayed one after another.
    tempNewTextNumber= (pEnemyWordSpriteTypesByTextsCurrentlyDisplayed[spriteType] mod
pEnemyWordSpriteTypesByTotalNumberOfTexts[spriteType]) + 1
    sprite(tempSprite).member.text=pEnemyWordSpriteTypesByTexts[spriteType][tempNewTextNumber]
  else
    --else the type is random, and random texts in the list are displayed
    tempNewTextNumber=random(pEnemyWordSpriteTypesByTotalNumberOfTexts[spriteType])
    sprite(tempSprite).member.text= pEnemyWordSpriteTypesByTexts[spriteType][tempNewTextNumber]
  end if
  sprite(tempSprite).pMyCurrentExplodingText=sprite(tempSprite).pMyExplodingTexts[tempNewTextNumber]
  sprite(tempSprite).member.width=sprite(tempSprite).pMyTextWidths[tempNewTextNumber]
  pEnemyWordSpriteTypesByTextsCurrentlyDisplayed[spriteType]=tempNewTextNumber
  pEnemyWordSpriteTypesCurrentlyInPlay[spriteType].append(tempSprite)

pTotalEnemyWordSpritesByTypeCurrentlyOnStage[spriteType]=pTotalEnemyWordSpritesByTypeCurrentlyOnStage[spriteType]+1
  pTotalEnemyWordSpritesAvailable=pTotalEnemyWordSpritesAvailable-1
  pTotalEnemyWordSpritesCurrentlyOnStage=pTotalEnemyWordSpritesCurrentlyOnStage+1
  sendsprite(tempSprite, #setMeInMotion)
end putThisSpriteTypeIntoPlay


--sprite(gEnemyWordManager).addMeAndMyType(spritenum, pMyTypeOfSprite, pMyRelativeFrequency, pMyOrdering, pMyTexts,
pMyTotalNumberOfTexts, pMyCurrentTextNum, pMyPointsAwarded)
on addMeAndMyType me, SpriteToAdd, TypeOfSprite, RelativeFrequency, Ordering, Texts, TotalNumberOfTexts,
CurrentTextNumber, PointsAwarded
  --This is called by an arteroid on beginsprite. It adds the different types of arteroids to the data of this script.
  tempType=pEnemyWordSpriteTypes.getOne(TypeOfSprite)
  if tempType=0 then
    pEnemyWordSpriteTypes.append(TypeOfSprite)
    pEnemyWordSpriteTypesByRelativeFrequency.addProp(TypeOfSprite, RelativeFrequency)
    pEnemyWordSpriteTypesByOrdering.addProp(TypeOfSprite, Ordering)
    pEnemyWordSpriteTypesByTexts.addProp(TypeOfSprite, Texts)
    pEnemyWordSpriteTypesByTotalNumberOfTexts.addProp(TypeOfSprite, TotalNumberOfTexts)
    pEnemyWordSpriteTypesByTextsCurrentlyDisplayed.addProp(TypeOfSprite, CurrentTextNumber)
    pEnemyWordSpriteTypesByPointsAwarded.addProp(TypeOfSprite, PointsAwarded)
    pEnemyWordSpritesByType.addProp(TypeOfSprite, [SpriteToAdd])
    pEnemyWordSpriteTypesCurrentlyInPlay.addProp(TypeOfSprite, [])
    pEnemyWordSpriteTypesAvailable.addProp(TypeOfSprite, [SpriteToAdd])
    pTotalEnemyWordSpritesByTypeCurrentlyOnStage.addProp(TypeOfSprite, 0)
  else
    pEnemyWordSpritesByType[TypeOfSprite].append(SpriteToAdd)
    pEnemyWordSpriteTypesAvailable[TypeOfSprite].append(SpriteToAdd)
  end if
  pTotalEnemyWordSpritesAvailable=pTotalEnemyWordSpritesAvailable+1
end addMeAndMyType


on checkTextsAndMakeAdjustments me
  --This checks the texts in level 1 and 2 to see if they're hunky dorry. If you add another level to the game and it contains text, then
  --you need to add some code here. You will hear about it if you don't.
  if gCurrentLevel=1 then
    texts=[member("Original Outer Green Level 1"), member("Original Inner Green Level 1"), member("Original Outer Blue Level 1"),
member("Original Inner Blue Level 1")]
  else
    if gCurrentLevel=2 then
```

```
      texts=[member("Outer Green Level 2"), member("Inner Green Level 2"), member("Outer Blue Level 2"), member("Inner Blue
Level 2")]
    else
      alert("Bug 5. Please tell jim@vispo.com that you encountered bug 5. Sorry. Oops. Better quit the program, this is unforeseen.")
    end if
  end if
  repeat with i=1 to 4
    --once for each of the four texts
    j=1
    repeat while j<= texts[i].line.count
      if texts[i].line[j].char.count >0 then
        repeat while charToNum(texts[i].line[j].char[1]) <33 and texts[i].line[j].char.count >0
          texts[i].line[j].char[1].delete()
        end repeat
      end if
      if texts[i].line[j]=EMPTY then
        if texts[i].line.count > 1 then
          texts[i].line[j].delete()
        else
          texts[i].line[j]="coretext "
        end if
      else
        j=j+1
      end if
    end repeat
  end repeat

  if not sprite(gUserDataManager).getBufferText(2) then
    if texts[1].line.count - texts[2].line.count > 0 then
      numOflinesToFill=texts[1].line.count - texts[2].line.count
      repeat with i=texts[2].line.count+1 to texts[1].line.count
        texts[2].line[i]=pResoivoir[random(pResoivoirCount)]
      end repeat
    else
      if texts[2].line.count - texts[1].line.count > 0 then
        repeat with i=texts[1].line.count+1 to texts[2].line.count
          texts[1].line[i]=pResoivoir[random(pResoivoirCount)]
        end repeat
      end if
    end if
  end if

  if not sprite(gUserDataManager).getBufferText(5) then
    if texts[3].line.count - texts[4].line.count > 0 then
      numOflinesToFill=texts[3].line.count - texts[4].line.count
      repeat with i=texts[4].line.count+1 to texts[3].line.count
        texts[4].line[i]=pResoivoir[random(pResoivoirCount)]
      end repeat
    else
      if texts[4].line.count - texts[3].line.count > 0 then
        repeat with i=texts[3].line.count+1 to texts[4].line.count
          texts[3].line[i]=pResoivoir[random(pResoivoirCount)]
        end repeat
      end if
    end if
  end if
end checkTextsAndMakeAdjustments


on dipIntoTextResoivoir me
  return pResoivoir[random(pResoivoirCount)]
end
```

104-107: TEXT HANDLING FOR GREEN ARTEROIDS

```
--**************************************************************************************************************
--SINGLE CIRCLE GREEN ASTEROID 1

--This behavior gets attached to sprites that are text sprites to be exploded, and explode in a single circle.
--If the sprite that's the Enemy Word Manager is of this type
--then this script must be attached AFTER the Enemy Word Manager script, because there are some variables used in this script that rely
--on values from the other script.
--**************************************************************************************************************

property spritenum, pStringToExplode, pCurrentText, pStringLength, pLetterChannels, pNumberOfLetterChannels, pIAmExploding, pCurrentlyInPlay
property pIterationOfExplosion,pSpeedFactor, pRadiusOfExplosion
property pBlendIncrement, pInitialExplosionBlend, pTotalIterations, pInitialExplosionFontSize, pFontSizeIncrement, pRotationFactor, pMyTypeOfSprite
property pCurrentExplosionBlend, pCurrentExplosionFontSize, pCurrentRotation, pAnglePart
property pMyRelativeFrequency, pMyOrdering, pMyTexts, pMyTotalNumberOfTexts, pMyTextWidths, pMyExplodingTexts, pMyCurrentExplodingText, pMyCurrentTextNum, pMyPointsAwarded
global gGreenLetterManager, gEnemyWordManager

on beginsprite me
  pMyTypeOfSprite="#SingleCircleGreen"
  --This variable defines the type of Asteroid we're dealing with here. It's used in the
  sprite(spritenum).member.centerRegPoint=TRUE
  --This makes the explosions be centered at the center of the text to explode.
  pMyRelativeFrequency=100
  --This  is used in the Enemy Letter Manager to determine the relative frequency with which this type of sprite is selected
  --to be onstage, given other types to choose among. Though the program will function correctly if there is only one type.
  pMyOrdering="sequential"
  --Possible values are "sequential" and "random". If the type is "sequential" then it will display the texts below sequentially
  --as they are destroyed and reappear.
  pMyTexts=[]
  repeat with i= 1 to member("Original Outer Green Level 1").paragraph.count
    t=member("Original Outer Green Level 1").paragraph[i]
    pMyTexts.add(t)
  end repeat
  --This defines the texts that will appear in the asteroid.
  pMyExplodingTexts=pMyTexts
  --The texts that are displayed don't have to be the same as the text that the text explodes into. If you define pMyExplodingTexts
  --differently from pMyTexts, make sure that there are the same number of texts in the list.
  pMyTotalNumberOfTexts=pMyTexts.count
  --The total Number of texts in pMyTexts and in pMyExplodingTexts
  pMyTextWidths=[]
  --This array holds the widths, in pixels, that the text object should be when it contains a given text from pMyTexts.
  --Unfortunately Director does not adjust this correctly automatically.
  repeat with i=1 to pMyTotalNumberOfTexts
    tempLength=pMyTexts[i].length
    firstApprox=max([16, 9.2*tempLength+8])
    --No text can be shorter than 16 characters, and the formula there is a rough approximation to the right length of any
    --string, in pixels. It isn't bang on, but it isn't too bad for a rude guess.
    secondApprox= integer(min([300, firstApprox]))
    --Don't let any text be longer than 300 pixels. Just because otherwise you get very long texts that take up a whole part
    --of the screen
    pMyTextWidths.append(secondApprox)
  end repeat
  if pMyOrdering="sequential" then
    pMyCurrentTextNum=pMyTotalNumberOfTexts
    --make the first text be the last text...because in Enemy Word Manager it is then modded back to the first text.
    pCurrentText=pMyTexts[pMyCurrentTextNum]
    pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
  else
    --else it's a random selection among the texts
    pMyCurrentTextNum=random(pMyTotalNumberOfTexts)
    pCurrentText=pMyTexts[pMyCurrentTextNum]
    pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
  end if
  sprite(spritenum).member.text=pCurrentText
  pMyPointsAwarded=10
  --This is the number of points awarded to the player when s/he blasts this type of text
  pStringToExplode=""
  --This is initialized later.
  pStringLength=0
  --This is initialized later
  pLetterChannels=[]
  --When a text explodes, what happens is that the displayed letters in the explosion are actually drawn from a flash
  --animation cast member called 'alphabet'. This is because Flash animations are much more quickly manipulated
  --than is Director text. Note that in the score there are many sprites whose member is the 'alphabet' member.
  --The pLetterChannels list holds the spritenumbers of the 'alphabet' sprites used for each letter of the exploding text.
  --In other words, if the text to explode is 'fart' then pLetterChannels will have four values, one for each of the letters
  --in 'fart'.
  pCurrentlyInPlay=FALSE
  --A sprite has pCurrentlyInPlay=TRUE if it is onstage or has been blasted but the explosion is still happening.
  pIAmExploding=FALSE
  --This is true during the interval when the sprite is exploding and false otherwise.
  pIterationOfExplosion=0
  --You must define here the number of frames it takes for the the explosion to occur.
  pSpeedFactor=3+ (random(10))
  --This is a handy little number that you can change to speed up/slow down sprites.
  --This is basically the number of pixels per frame that the thing moves by.
  pInitialExplosionFontSize=30 + random(75)
  --The initial size (not actually the font size) of the letters (actually its the height and width of the 'alphabet' sprites.
  pRadiusOfExplosion=120
  --This defines the radius length of the explosion.
  pInitialExplosionBlend=100
  --This determines what the Blend (or alpha) values is of the letters  when they first explode.
  pTotalIterations=20
  --You can see that geometrically this value makes sense. The radius of explosion is the total distance
  --that each letter traverses during the explosion, and the speed factor is the number of pixels the letter
  --moves on each iteration of the explosion, so the total number of iterations is the one divided by the other.
  pBlendIncrement= 3
  --This is the inrement by which the blend is decreases each iteration. Change this to change the final
  --blend of the letters after the explosion is finished, so you do get some text corpses onstage after the explosion.
  pFontSizeIncrement=integer(((pInitialExplosionFontSize-18.0)/pTotalIterations)-0.5)
  --The increment by which the 'alphabet' sprite is changed each iteration.
  pRotationFactor=30
  --The number of degrees by which the exploding letter is rotated each iteration.
  pCurrentExplosionBlend=pInitialExplosionBlend
  pCurrentExplosionFontSize=pInitialExplosionFontSize
  pCurrentRotation=0
  sprite(gEnemyWordManager).addMeAndMyType(spritenum, pMyTypeOfSprite, pMyRelativeFrequency, pMyOrdering, pMyTexts,
pMyTotalNumberOfTexts, pMyCurrentTextNum, pMyPointsAwarded)
  --so the Enemy Word Manager must be attached BEFORE this script to the sprite that is both the Enemy Word Manager and the
Word To Explode 1
end beginsprite


on explode me
  sprite(gEnemyWordManager).deleteMeFromCurrentlyInPlayAndNotExploding(spritenum)
  sprite(spritenum).visible=FALSE
  pStringToExplode=sprite(spritenum).pMyCurrentExplodingText
  pStringLength=pStringToExplode.length
  pLetterChannels=sprite(gGreenLetterManager).getChannels(pStringLength, spritenum, pStringToExplode)
  pNumberOfLetterChannels=pLetterChannels.count
  if pNumberOfLetterChannels >0 then
    pRadiusOfExplosion=20+pStringLength*25
    pAnglePart=6.2832/pNumberOfLetterChannels
    repeat with i = 1 to pNumberOfLetterChannels
      tempNum=charToNum(chars(pMyCurrentExplodingText, i,i))
      sprite(pLetterChannels[i]).frame = tempNum
      sprite(pLetterChannels[i]).ink=36
    end repeat
    sprite(gEnemyWordManager).addMeToExplodingList(spritenum)
  else
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
end explode me


on executeMyGeometry me
  --this can change depending on the geometry
  if pIterationOfExplosion < pTotalIterations then
    pCurrentExplosionBlend=pCurrentExplosionBlend-pBlendIncrement
    pCurrentExplosionFontSize=pCurrentExplosionFontSize-pFontSizeIncrement
    pCurrentRotation=(pCurrentRotation+pRotationFactor) mod 360
    theRadius=pRadiusOfExplosion*pIterationOfExplosion.float/pTotalIterations
    repeat with i= 1 to pNumberOfLetterChannels
      tempi=i-1
      sprite(pLetterChannels[i]).blend=pCurrentExplosionBlend
      sprite(pLetterChannels[i]).width=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).height=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).rotation=pCurrentRotation
      theAngle=tempi*pAnglePart
      Hextension=theRadius*cos(theAngle)
      Vextension=theRadius*sin(theAngle)
      sprite(pLetterChannels[i]).locH= sprite(spritenum).locH + Hextension
      sprite(pLetterChannels[i]).locV= sprite(spritenum).locV + Vextension
    end repeat
    pIterationOfExplosion=pIterationOfExplosion+1
  else
    pCurrentExplosionBlend=pInitialExplosionBlend
    pCurrentExplosionFontSize=pInitialExplosionFontSize
    pCurrentRotation=0
    sprite(gGreenLetterManager).returnChannels(pLetterChannels)
    pLetterChannels=[]
    pIterationOfExplosion=0
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
  --put pIterationOfExplosion
end executeMyGeometry


--************************************************************************
--RANDOM LINE MOTION

--This controls the motion of the text to which it is attached.
--This behavior makes the sprites to which it is attached move in an
--arbitrary straight line around the stage
--**********************************************************************

global gEnemyWordManager
global gStageWidth, gStageHeight
property myloch, mylocv, mys, angle, mangle, mySpeedFactor,
property sr, sl, sb, st

on beginsprite me
  mys = me.spritenum
  angle = random(170)+5
  myloch = sprite(mys).loch
  mylocv = sprite(mys).locv
  mySpeedFactor=6.0+random(10)
  sprite(mys).loch=-100
  sprite(mys).locv=-100
end


on setMeInMotion me
  angle=random(350)
  mangle = angle * pi()/180
  mySpeedFactor=float(random(6)+1)
  sprite(mys).visible=true

  HIs0orStageWidth=random(2)-1
  if HIs0orStageWidth then
    His0=random(2)-1
    if His0 then
      mylocH=0
      mylocV=random(gStageHeight)
    else
      mylocH= gStageWidth
      mylocV=random(gStageHeight)
    end if
  else
    Vis0=random(2)-1
    if Vis0 then
      mylocH=random(gStageWidth)
      mylocV=0
    else
      myLocH=random(gStageWidth)
      myLocV=gStageHeight
    end if
  end if

  sprite(mys).loch=myLocH
  sprite(mys).locv=myLocV
  sprite(mys).rotation = angle
  --sprite(mys).pCurrentlyInPlayAndNotExploding=TRUE
  sprite(gEnemyWordManager).addMeToCurrentlyInPlayAndNotExploding(mys)
  sprite(mys).pCurrentlyInPlay=TRUE
  sr=gStageWidth+mySpeedFactor
  sl=-mySpeedFactor
  sb=gStageHeight+mySpeedFactor
  st=-mySpeedFactor
  sprite(gEnemyWordManager).addMeToMovingList(mys)
end reset me

on calcNewPosition me
    myloch = myloch +mySpeedFactor*cos(mangle)
    mylocv = mylocv +mySpeedFactor*sin(mangle)
    onstage me
    sprite(mys).loch = myloch
    sprite(mys).locv = mylocv
end

on onstage me
```

```
  --This keeps the thing on the stage rather than drifting off.
  case 1 of
    (sprite(mys).left>sr): myloch= sl
    (sprite(mys).right<sl): myloch= sr
    (sprite(mys).top>sb): mylocv= st
    (sprite(mys).bottom<st): mylocv= sb
  end case
end onstage
```

Channels 109-114: Blue Texts

```
--*********************************************************************************************
--SINGLE CIRCLE BLUE ARTEROID LEVEL 1

--This behavior gets attached to blue text Arteroids that are text sprites to be exploded, and explode in a single circle.
--If the sprite that's the Enemy Word Manager is of this type
--then this script must be attached AFTER the Enemy Word Manager script, because there are some variables used in this script that rely
--on values from the other script.

--This behavior gets attached to sprites that are text sprites to be exploded, and explode in a single circle.
--If the sprite that's the Enemy Word Manager is of this type
--then this script must be attached AFTER the Enemy Word Manager script, because there are some variables used in this script that rely
--on values from the other script.
--*********************************************************************************************

property spritenum, pStringToExplode, pCurrentText, pStringLength, pLetterChannels, pNumberOfLetterChannels, pIAmExploding, pCurrentlyInPlay
property pIterationOfExplosion,pSpeedFactor, pRadiusOfExplosion
property pBlendIncrement, pInitialExplosionBlend, pTotalIterations, pInitialExplosionFontSize, pFontSizeIncrement, pRotationFactor, pMyTypeOfSprite
property pCurrentExplosionBlend, pCurrentExplosionFontSize, pCurrentRotation, pAnglePart
property pMyRelativeFrequency, pMyOrdering, pMyTexts, pMyTotalNumberOfTexts, pMyTextWidths, pMyExplodingTexts, pMyCurrentExplodingText, pMyCurrentTextNum, pMyPointsAwarded
global gBlueLetterManager, gEnemyWordManager

on beginsprite me
  pMyTypeOfSprite="#SingleCircleBlue"
  --This variable defines the type of Asteroid we're dealing with here. It's used in the
  sprite(spritenum).member.centerRegPoint=TRUE
  --This makes the explosions be centered at the center of the text to explode.
  pMyRelativeFrequency=100
  --This  is used in the Enemy Letter Manager to determine the relative frequency with which this type of sprite is selected
  --to be onstage, given other types to choose among. Though the program will function correctly if there is only one type.
  pMyOrdering="sequential"
  --Possible values are "sequential" and "random". If the type is "sequential" then it will display the texts below sequentially
  --as they are destroyed and reappear.
  pMyTexts=[]
  repeat with i= 1 to member("Original Outer Blue Level 1").paragraph.count
    pMyTexts.add(member("Original Outer Blue Level 1").paragraph[i])
  end repeat
  --This defines the texts that will appear in the asteroid.
  pMyExplodingTexts=[]
  repeat with i= 1 to member("Original Inner Blue Level 1").paragraph.count
    pMyExplodingTexts.add(member("Original Inner Blue Level 1").paragraph[i])
  end repeat
  --The texts that are displayed don't have to be the same as the text that the text explodes into. If you define pMyExplodingTexts
  --differently from pMyTexts, make sure that there are the same number of texts in the list.
```

```
pMyTotalNumberOfTexts=pMyTexts.count
--The total Number of texts in pMyTexts and in pMyExplodingTexts
pMyTextWidths=[]
--This array holds the widths, in pixels, that the text object should be when it contains a given text from pMyTexts.
--Unfortunately Director does not adjust this correctly automatically.
repeat with i=1 to pMyTotalNumberOfTexts
  tempLength=pMyTexts[i].length
  firstApprox=max([16, 9.2*tempLength+8])
  --No text can be shorter than 16 characters, and the formula there is a rough approximation to the right length of any
  --string, in pixels. It isn't bang on, but it isn't too bad for a rude guess.
  secondApprox= integer(min([300, firstApprox]))
  --Don't let any text be longer than 300 pixels. Just because otherwise you get very long texts that take up a whole part
  --of the screen
  pMyTextWidths.append(secondApprox)
end repeat
if pMyOrdering="sequential" then
  pMyCurrentTextNum=pMyTotalNumberOfTexts
  --make the first text be the last text...because in Enemy Word Manager it is then modded back to the first text.
  pCurrentText=pMyTexts[pMyCurrentTextNum]
  pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
else
  --else it's a random selection among the texts
  pMyCurrentTextNum=random(pMyTotalNumberOfTexts)
  pCurrentText=pMyTexts[pMyCurrentTextNum]
  pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
end if
sprite(spritenum).member.text=pCurrentText
pMyPointsAwarded=10
--This is the number of points awarded to the player when s/he blasts this type of text
pStringToExplode=""
--This is initialized later.
pStringLength=0
--This is initialized later
pLetterChannels=[]
--When a text explodes, what happens is that the displayed letters in the explosion are actually drawn from a flash
--animation cast member called 'alphabet'. This is because Flash animations are much more quickly manipulated
--than is Director text. Note that in the score there are many sprites whose member is the 'alphabet' member.
--The pLetterChannels list holds the spritenumbers of the 'alphabet' sprites used for each letter of the exploding text.
--In other words, if the text to explode is 'fart' then pLetterChannels will have four values, one for each of the letters
--in 'fart'.
pCurrentlyInPlay=FALSE
--A sprite has pCurrentlyInPlay=TRUE if it is onstage or has been blasted but the explosion is still happening.
pIAmExploding=FALSE
--This is true during the interval when the sprite is exploding and false otherwise.
pIterationOfExplosion=0
--You must define here the number of frames it takes for the the explosion to occur.
pSpeedFactor=3+ (random(10))
--This is a handy little number that you can change to speed up/slow down sprites.
--This is basically the number of pixels per frame that the thing moves by.
pInitialExplosionFontSize=30 + random(75)
--The initial size (not actually the font size) of the letters (actually its the height and width of the 'alphabet' sprites.
pRadiusOfExplosion=120
--This defines the radius length of the explosion.
pInitialExplosionBlend=100
--This determines what the Blend (or alpha) values is of the letters  when they first explode.
pTotalIterations=15
--You can see that geometrically this value makes sense. The radius of explosion is the total distance
--that each letter traverses during the explosion, and the speed factor is the number of pixels the letter
--moves on each iteration of the explosion, so the total number of iterations is the one divided by the other.
pBlendIncrement= 3
--This is the inrement by which the blend is decreases each iteration. Change this to change the final
--blend of the letters after the explosion is finished, so you do get some text corpses onstage after the explosion.
```

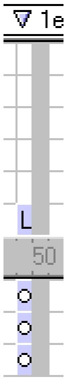
```
  pFontSizeIncrement=integer(((pInitialExplosionFontSize-18.0)/pTotalIterations)-0.5)
  --The increment by which the 'alphabet' sprite is changed each iteration.
  pRotationFactor=30
  --The number of degrees by which the exploding letter is rotated each iteration.
  pCurrentExplosionBlend=pInitialExplosionBlend
  pCurrentExplosionFontSize=pInitialExplosionFontSize
  pCurrentRotation=0
  sprite(gEnemyWordManager).addMeAndMyType(spritenum, pMyTypeOfSprite, pMyRelativeFrequency, pMyOrdering, pMyTexts, pMyTotalNumberOfTexts, pMyCurrentTextNum, pMyPointsAwarded)
  --so the Enemy Word Manager must be attached BEFORE this script to the sprite that is both the Enemy Word Manager and the Word To Explode 1
end beginsprite


on explode me
  sprite(gEnemyWordManager).deleteMeFromCurrentlyInPlayAndNotExploding(spritenum)
  sprite(spritenum).visible=FALSE
  pStringToExplode=sprite(spritenum).pMyCurrentExplodingText
  pStringLength=pStringToExplode.length
  pLetterChannels=sprite(gBlueLetterManager).getChannels(pStringLength, spritenum, pStringToExplode)
  pNumberOfLetterChannels=pLetterChannels.count
  if pNumberOfLetterChannels >0 then
    pRadiusOfExplosion=20+pStringLength*25
    pAnglePart=6.2832/pNumberOfLetterChannels
    repeat with i = 1 to pNumberOfLetterChannels
      tempNum=charToNum(chars(pMyCurrentExplodingText, i,i))
      sprite(pLetterChannels[i]).frame = tempNum
      sprite(pLetterChannels[i]).ink=36
    end repeat
    sprite(gEnemyWordManager).addMeToExplodingList(spritenum)
  else
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
end explode me


on executeMyGeometry me
  --this can change depending on the geometry
  if pIterationOfExplosion < pTotalIterations then
    pCurrentExplosionBlend=pCurrentExplosionBlend-pBlendIncrement
    pCurrentExplosionFontSize=pCurrentExplosionFontSize-pFontSizeIncrement
    pCurrentRotation=(pCurrentRotation+pRotationFactor) mod 360
    theRadius=pRadiusOfExplosion*pIterationOfExplosion.float/pTotalIterations
    repeat with i= 1 to pNumberOfLetterChannels
      tempi=i-1
      sprite(pLetterChannels[i]).blend=pCurrentExplosionBlend
      sprite(pLetterChannels[i]).width=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).height=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).rotation=pCurrentRotation
      theAngle=tempi*pAnglePart
      Hextension=theRadius*cos(theAngle)
      Vextension=theRadius*sin(theAngle)
      sprite(pLetterChannels[i]).locH= sprite(spritenum).locH + Hextension
      sprite(pLetterChannels[i]).locV= sprite(spritenum).locV + Vextension
    end repeat
    pIterationOfExplosion=pIterationOfExplosion+1
  else
    pCurrentExplosionBlend=pInitialExplosionBlend
    pCurrentExplosionFontSize=pInitialExplosionFontSize
    pCurrentRotation=0
    sprite(gBlueLetterManager).returnChannels(pLetterChannels)
    pLetterChannels=[]
    pIterationOfExplosion=0
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
end executeMyGeometry


--**********************************************************************
--FOLLOW ID ENTITY NEW

--This behavior is attached to sprites where you want them to follow
--the id entity. More particularly, this behavior, unlike the 'Follow
--Id Entity' behavior, calculates the distance between the id entity
--and this sprite, so the manner in which it follows the id entity is
--a bit different, and better.
--**********************************************************************

global gEnemyWordManager
global gStageWidth, gStageHeight, gPlayerShipSpriteNum
property myloch, mylocv, mySpeedFactor
property sr, sl, sb, st
property spritenum, ptargetSprite, pSpeedFactor


on beginsprite me
  pTargetSprite=gPlayerShipSpriteNum
  --This sets the number of the target sprite.
  MyLocH=-100
  MyLocV=-100
  sprite(spritenum).locH=-100
  sprite(spritenum).locV=-100
end beginsprite

on setMeInMotion me
  --This is called just before the sprite is sent onstage.
  pSpeedFactor=float(random(8)+4)
  --This determines the speed of the sprite
  sprite(spritenum).visible=true

  --This positions the sprite somewhere along the periphery
  --of the stage.
  HIs0orStageWidth=random(2)-1
  if HIs0orStageWidth then
    His0=random(2)-1
    if His0 then
      mylocH=0
      mylocV=random(gStageHeight)
    else
      mylocH= gStageWidth
      mylocV=random(gStageHeight)
    end if
  else
    Vis0=random(2)-1
    if Vis0 then
      mylocH=random(gStageWidth)
      mylocV=0
    else
      myLocH=random(gStageWidth)
      myLocV=gStageHeight
    end if
  end if

  sprite(gEnemyWordManager).addMeToCurrentlyInPlayAndNotExploding(spritenum)
  sprite(spritenum).pCurrentlyInPlay=TRUE
  sprite(gEnemyWordManager).addMeToMovingList(spritenum)
end reset me


on calcNewPosition me
  targetV=sprite(pTargetSprite).locV
  targetH=sprite(pTargetSprite).locH
  deltaV=targetV-myLocV
  deltaH=targetH-myLocH
  d=sqrt(power(deltaH,2)+power(deltaV,2))
  --the above is the distance between the sprite and id-entity.
  if d<>0 then
    MyLocV= MyLocV+ (deltaV*pSpeedFactor/d)
    MyLocH= MyLocH+ (deltaH*pSpeedFactor/d)
    sprite(spritenum).locV=MyLocV
    sprite(spritenum).locH=MyLocH
  end if
end calcNewPos
```

Channels 119-123: Missiles

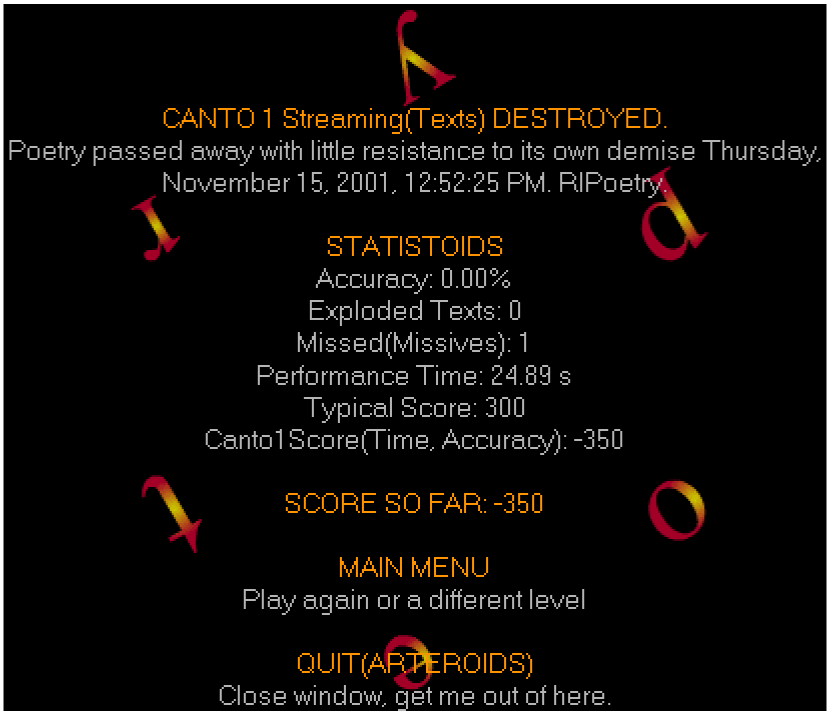
```
--***************************************************************************
--MISSILE MANAGER

--This manages the missiles. There are only ever max 2 missiles on the
--screen at one time. This can be seen by examining the 'Player' behavior
--attached to the player (id-entity). I limited it to 2 missiles onstage
--max at once to minimize collision detection. There are 5 missiles in
--the score, though there really only needs to be 2 or 3. I think.
--Notice that the Missile Manager doesn't use gMissile at all, which is
--the list that contains the missiles (see the 'missile behavior'
--behavior. This is because the Missile Manager is concerned with
--managing the pMissilesInMotionList, which is not a list of all the
--missiles but just those that are currently visible onstage.
--***************************************************************************

property spritenum, pMissilesInMotionList, pMissilesInMotionListCount
global gMissileManager

on beginsprite me
  gMissileManager=spritenum
  pMissilesInMotionList=[]
  pMissilesInMotionListCount=0
  gMissileMisses=0
  gMissileHits=0
end beginsprite


on addMeToMissilesInMotionList me, SpriteToAdd
  pMissilesInMotionList.append(SpriteToAdd)
  pMissilesInMotionListCount=pMissilesInMotionListCount+1
end addMeToMotionList

on deleteMeFromMissilesInMotionList me, SpriteToDelete
  pMissilesInMotionList.deleteOne(SpriteToDelete)
  pMissilesInMotionListCount=pMissilesInMotionListCount-1
end deleteMeFromMissilesInMotionList
```

```
on prepareFrame me
  repeat with i=1 to pMissilesInMotionListCount
    sendSprite (pMissilesInMotionList[i], #calcNewPosition)
  end repeat
end

--*******************************************************************
--MISSILE BEHAVIOR

--This is attached to the missiles.
--*******************************************************************


global gmissile, gBangMemberNumber, gBulletMemberNumber, gMissileManager, gUserDataManager
global gCurrentlyOnStageBullets, gCurrentlyOnStageBulletsCount, gStageWidth, gStageHeight
property mys, flag, myloch, mylocv, myangle, mySpeedFactor, pMyBangCounter
property mangle
property pTop, pBottom, pLeft, pRight

on beginsprite me
  mys = (me).spritenum
  sprite(mys).visible = 0
  addprop (gmissile, mys,0)
  flag = 0
  sprite(mys).calcMissileBorders()
end

on calcMissileBorders me
  pTop=-50
  pBottom=gStageHeight+50
  pLeft=-50
  pRight=gStageWidth+50
end calcMissileBorders


on shoot me,fromsprite,theloch,thelocv,angle
  sprite(gMissileManager).addMeToMissilesInMotionList(mys)
  gCurrentlyOnStageBullets.append(mys)
  gCurrentlyOnStageBulletsCount=gCurrentlyOnStageBulletsCount+1
  sprite(mys).visible = 1
  sprite(mys).loc = sprite(fromsprite).loc
  myloch = theloch
  mylocv = thelocv
  myangle = angle
  pMyBangCounter=0
  mangle = myangle * pi()/180
  mySpeedFactor=27
end


on noCollision me, this
  --This gets called from the gEnemyWordManager sprite.
  if sprite this intersects mys then
    didDelete=gCurrentlyOnStageBullets.deleteOne(mys)
    if didDelete then
      gCurrentlyOnStageBulletsCount=gCurrentlyOnStageBulletsCount-1
    end if
    sprite(mys).member=member(gBangMemberNumber)
    mySpeedFactor=3
    pMyBangCounter=1
    sprite(gUserDataManager).addOneMissileHit()
    sendsprite this, #explode
    return FALSE
  else
    return TRUE
  end if
end


on calcNewPosition me
  if (myloch > pRight) or (myloch<pLeft) or (mylocv > pBottom) or (mylocv<pTop) then
    setprop(gmissile,mys,0)
    sprite(mys).visible = 0
    sprite(mys).member=member(gBulletMemberNumber)
    sprite(mys).loc = point(-1000,-1000)
    pMyBangCounter=0
    sprite(gUserDataManager).addOneMissileMiss()
    sprite(gMissileManager).deleteMeFromMissilesInMotionList(mys)
    didDelete=gCurrentlyOnStageBullets.deleteOne(mys)
    if didDelete then
      gCurrentlyOnStageBulletsCount=gCurrentlyOnStageBulletsCount-1
    end if
  else
    myloch = myloch +mySpeedFactor*cos(mangle)
    mylocv = mylocv +mySpeedFactor*sin(mangle)
    sprite(mys).loc = point(myloch,mylocv)
    if pMyBangCounter > 0 then
      pMyBangCounter=pMyBangCounter+1
      if pMyBangCounter > 2 then
        setprop(gmissile,mys,0)
        sprite(mys).visible = 0
        sprite(mys).member=member(gBulletMemberNumber)
        sprite(mys).loc = point(-1000,-1000)
        pMyBangCounter=0
        sprite(gMissileManager).deleteMeFromMissilesInMotionList(mys)
      end if
    end if
  end if
end
```

Channels 130-132: Statistoids for Canto 1



After you finish playing canto 1, or exit canto 1 prematurely by clicking the 'Exit Canto' item in the dropdown menu, you are shown the statistoids for Canto 1 and can either return to the Main Menu or close Arteroids altogether. This happens in frame 49, as shown to the left.



CANTO 1 Streaming(Texts) DESTROYED.
Poetry passed away with little resistance to its own demise Thursday, November 15, 2001, 12:52:25 PM. RIPoetry.

STATISTOIDS
Accuracy: 0.00%
Exploded Texts: 0
Missed(Missives): 1
Performance Time: 24.89 s
Typical Score: 300
Canto1Score(Time, Accuracy): -350

SCORE SO FAR: -350

MAIN MENU
Play again or a different level

QUIT(ARTEROIDS)
Close window, get me out of here.

MARKER 1E (FRAME 49)

The playback head is sent to marker 1e in the endLevel handler of the Score and Level Manager when the player finishes Canto 1 or exits the Canto prematurely. Actually, the endLevel handler consists of the line

```
go to string(levelToEnd) & "e"
```

So any given level x should have a marker xe in it above the frame to which the movie is sent to display the statistoids.

FRAME 49 FRAME SCRIPT: LOOP ON FRAME

```
--***************************************
--LOOP ON FRAME

--This just loops the movie on this frame.
--***************************************

on exitFrame me
  go to the frame
end
```

130: MAIN MENU POETRY ANIMATION

See 10: Main Menu Animation. This is the same script and animation used in several places in the program.

131: STATISTOIDS

```
--****************************************************************************************
--CLOSURE CANTO SCRIPT

--This script is attached to the sprite with text member "CLOSURE(CANTO Y)". This
--text member displays the statistoids. This script is used to display the statistoids
--properly and to provide a link back to the main menu and an exit link from Arteroids.
--****************************************************************************************
```

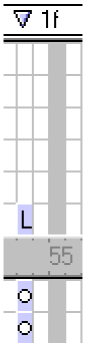
```
property spritenum, pCurrentParagraphNum, pOldParagraphNum
property pClosurePara, pCommentPara, pStatsHeaderPara, pAccuracyPara, pHitsPara, PMissesPara, pTimePara,
pBaseScorePara, pLevelScorePara
property pTotalScorePara, pMainMenu
property pQuitArteroidsPara, pLevelListingFirstPara, pLevelListingSecondPara
global gScoreDisplayer, gCurrentLevel, gUserDataManager, gScoreAndLevelManager, gMessageManager

on beginsprite me
  gScoreDisplayer=spritenum
  sprite(spritenum).visible=FALSE
  pClosurePara=5
  pCommentPara=6
  pAccuracyPara=9
  pHitsPara=10
  pMissesPara=11
  pTimePara=12
  pBaseScorePara=13
  pLevelScorePara=14
  pTotalScorePara=16
  pMainMenu=19
  pQuitArteroidsPara=22
  saveIt= the itemdelimiter
  the itemdelimiter=RETURN
  pCurrentParagraphNum=-1
  pOldParagraphNum=1
  sprite(spritenum).member.paragraph[pClosurePara]= "CLOSURE(CANTO " & string(gCurrentLevel) & ": " &
sprite(gMessageManager).levelTitle(gCurrentLevel) & ")"
  sprite(spritenum).member.paragraph[pClosurePara].color=rgb(255,153,0)
  sprite(spritenum).member.paragraph[pCommentPara]= sprite(gMessageManager).levelComment(gCurrentLevel)
  tempAccuracy=sprite(gUserDataManager).setLevelFinalAccuracy(gCurrentLevel)
  sprite(spritenum).member.paragraph[pAccuracyPara]= "Accuracy: " & string(100*tempAccuracy) & "%"
  sprite(spritenum).member.paragraph[pHitsPara]= "Exploded Texts: " & string(sprite(gUserDataManager).getMissileHits())
  sprite(spritenum).member.paragraph[pMissesPara]= "Missed(Missives): " & string(sprite(gUserDataManager).getMissileMisses())
  sprite(spritenum).member.paragraph[pTimePara] = "Performance Time: " &
string(sprite(gUserDataManager).setLevelFinalTime(gCurrentLevel)/float(1000)) & " s"
  sprite(spritenum).member.paragraph[pBaseScorePara] = "Typical Score: " &
string(sprite(gScoreAndLevelManager).getTypicalAndEndScore())
  sprite(spritenum).member.paragraph[pLevelScorePara] = "Canto" & string(gcurrentLevel) & "Score(Time, Accuracy): " &
string(sprite(gUserDataManager).setFinalLevelScore(gCurrentLevel))
  sprite(spritenum).member.paragraph[pTotalScorePara] = "SCORE SO FAR: " &
string(sprite(gScoreAndLevelManager).setScoreOverall())
  sprite(spritenum).member.paragraph[pTotalScorePara].color=rgb(255,153,0)
  sprite(spritenum).loc=centerIt(spritenum)
  sprite(spritenum).visible=TRUE
end beginsprite


on mousewithin me
  pointClicked = the mouseLoc
  pCurrentparagraphNum = sprite(spriteNum).pointToParagraph(pointClicked)
  case pCurrentParagraphNum of
    pMainMenu: sprite(spritenum).MakeMeRed(pMainMenu)
    pQuitArteroidsPara: sprite(spritenum).MakeMeRed(pQuitArteroidsPara)
    otherwise:
      cursor 0
      sprite(spritenum).member.paragraph[pOldParagraphNum].color=rgb(187,187,187)
  end case
end


on MakeMeRed me, myParaNum
```

```
  cursor 280
  sprite(spritenum).member.paragraph[pOldParagraphNum].color=rgb(187,187,187)
  sprite(spritenum).member.paragraph[myParaNum].color=rgb(255,0,0)
  pOldParagraphNum=pCurrentParagraphNum
end


on mouseleave me
  sprite(spritenum).member.paragraph[pOldparagraphNum].color=rgb(187,187,187)
  pOldParagraphNum=1
  cursor 0
end mouseleave


on mousedown me
  pointClicked = the mouseLoc
  pCurrentparagraphNum = sprite(spriteNum).pointToParagraph(pointClicked)
  case pCurrentParagraphNum of
    pMainMenu:
      cursor 0
      sprite(spritenum).member.paragraph[pMainMenu].color=rgb(187,187,187)
      go to "0"
    pQuitArteroidsPara:
      cursor 0
      sprite(spritenum).member.paragraph[pQuitArteroidsPara].color=rgb(187,187,187)
      gotonetpage "javascript:parent.close()"
    otherwise:
      cursor 0
  end case
end mousedown
```

132: ARTEROIDS GRAPHIC

```
--*************************************************************
--ARTEROIDS GRAPHIC FOR SCORE TIME

--This gets attached to the graphic of the word 'Arteroids'
--to position it.
--*************************************************************

property spritenum
global gScoreDisplayer

on beginsprite me
  sprite(spritenum).visible=FALSE
  sprite(spritenum).locH= centerH(spritenum)
  sprite(spritenum).locV=sprite(gScoreDisplayer).locV + 5
  sprite(spritenum).visible=TRUE
end beginsprite
```

Channels 135-136: Player Loses



135: MAIN MENU POETRY ANIMATION

This channel holds a sprite that has poetry3 as member (the poetry explosion animation). The behavior attached to it is the same as in 10: Main Menu Animation.

136: PLAYER LOSES

```
--*****************************************************************************************************
--PLAYER LOSES SCRIPT

--When the player gets a very negative score, they lose. This script is attached to the text member that
--displays the statistoids for this event.
--*****************************************************************************************************

property spritenum, pCurrentParagraphNum, pOldParagraphNum
property pClosurePara, pCommentPara, pStatsHeaderPara, pAccuracyPara, pHitsPara, PMissesPara, pTimePara,
pBaseScorePara, pLevelScorePara
property pTotalScorePara, pMainMenu
property pQuitArteroidsPara, pLevelListingFirstPara, pLevelListingSecondPara
global gScoreDisplayer, gCurrentLevel, gUserDataManager, gScoreAndLevelManager, gMessageManager

on beginsprite me
  gScoreDisplayer=spritenum
  sprite(spritenum).visible=FALSE
  pClosurePara=5
  pCommentPara=6
  pAccuracyPara=9
  pHitsPara=10
  pMissesPara=11
  pTimePara=12
  pBaseScorePara=13
  pLevelScorePara=14
  pTotalScorePara=16
  pMainMenu=19
  pQuitArteroidsPara=22
  saveIt= the itemdelimiter
  the itemdelimiter=RETURN
  pCurrentParagraphNum=-1
  pOldParagraphNum=1
  sprite(spritenum).member.paragraph[pClosurePara]= "CANTO " & string(gCurrentLevel) &&
sprite(gMessageManager).levelTitle(gCurrentLevel) & " DESTROYED."
  sprite(spritenum).member.paragraph[pClosurePara].color=rgb(255,153,0)
  sprite(spritenum).member.paragraph[pCommentPara]= sprite(gMessageManager).deathNotice(gCurrentLevel)
  tempAccuracy=sprite(gUserDataManager).setLevelFinalAccuracy(gCurrentLevel)
  sprite(spritenum).member.paragraph[pAccuracyPara]= "Accuracy: " & string(100*tempAccuracy) & "%"
  sprite(spritenum).member.paragraph[pHitsPara]= "Exploded Texts: " & string(sprite(gUserDataManager).getMissileHits())
  sprite(spritenum).member.paragraph[pMissesPara]= "Missed(Missives): " & string(sprite(gUserDataManager).getMissileMisses())
  sprite(spritenum).member.paragraph[pTimePara] = "Performance Time: " &
string(sprite(gUserDataManager).setLevelFinalTime(gCurrentLevel)/float(1000)) & " s"
  sprite(spritenum).member.paragraph[pBaseScorePara] = "Typical Score: " &
string(sprite(gScoreAndLevelManager).getTypicalAndEndScore())
  sprite(spritenum).member.paragraph[pLevelScorePara] = "Canto" & string(gcurrentLevel) & "Score(Time, Accuracy): " &
```

```
string(sprite(gUserDataManager).setFinalLevelScore(gCurrentLevel))
  sprite(spritenum).member.paragraph[pTotalScorePara] = "SCORE SO FAR: " &
string(sprite(gScoreAndLevelManager).setScoreOverall())
  sprite(spritenum).member.paragraph[pTotalScorePara].color=rgb(255,153,0)
  sprite(spritenum).loc=centerIt(spritenum)
  sprite(spritenum).visible=TRUE
end beginsprite


on mousewithin me
  pointClicked = the mouseLoc
  pCurrentparagraphNum = sprite(spriteNum).pointToParagraph(pointClicked)
  case pCurrentParagraphNum of
    pMainMenu: sprite(spritenum).MakeMeRed(pMainMenu)
    pQuitArteroidsPara: sprite(spritenum).MakeMeRed(pQuitArteroidsPara)
    otherwise:
      cursor 0
      sprite(spritenum).member.paragraph[pOldParagraphNum].color=rgb(187,187,187)
  end case
end


on MakeMeRed me, myParaNum
  cursor 280
  sprite(spritenum).member.paragraph[pOldParagraphNum].color=rgb(187,187,187)
  sprite(spritenum).member.paragraph[myParaNum].color=rgb(255,0,0)
  pOldParagraphNum=pCurrentParagraphNum
end


on mouseleave me
  sprite(spritenum).member.paragraph[pOldparagraphNum].color=rgb(187,187,187)
  pOldParagraphNum=1
  cursor 0
end mouseleave


on mousedown me
  pointClicked = the mouseLoc
  pCurrentparagraphNum = sprite(spriteNum).pointToParagraph(pointClicked)
  case pCurrentParagraphNum of
    pMainMenu:
      cursor 0
      sprite(spritenum).member.paragraph[pMainMenu].color=rgb(187,187,187)
      go to "0"
    pQuitArteroidsPara:
      cursor 0
      sprite(spritenum).member.paragraph[pQuitArteroidsPara].color=rgb(187,187,187)
      gotonetpage "javascript:parent.close()"
    otherwise:
      cursor 0
  end case
end mousedown
```

Canto 2 Initialization

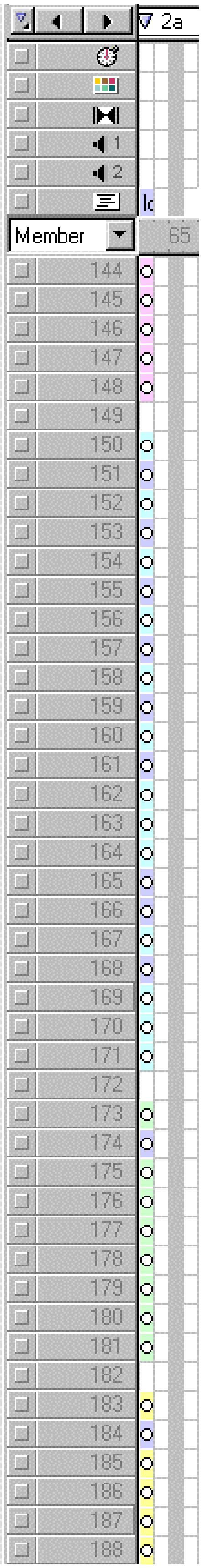
Frame 60

Frame 60 contains the below frame script (and nothing else) that initializes Canto 2. It is specific to canto 2 only in that it sets gCurrentLevel=2.

There is a marker associated with frame 60: marker 2. Each level x needs to have such a marker x at its start.

```
--*****************************************************************************************************
--INITIALIZE CANTO 2

--Unlike the Initialize 1 script, this one doesn't have a loop in frame handler.
--That's because we want to initialize the vars and then move into the text editor.
--Otherwise, it's much the same as the Initialize 1 routine.
--*****************************************************************************************************

global gMissile, gCurrentlyOnStageBullets, gCurrentlyOnStageBulletsCount, gCurrentLevel, gStageWidth, gStageHeight
global gBangMemberNumber, gBulletMemberNumber, gUserDataManager, gScoreAndLevelManager, gLastScore

on beginsprite me
  gStageWidth=the stageRight - the StageLeft
  gStageHeight= the stageBottom - the StageTop
  gCurrentlyOnStageBullets=[]
  gCurrentlyOnStageBulletsCount=0
  gMissile=[:]
  gCurrentLevel=2
  gBangMemberNumber=member("Bang").number
  gBulletMemberNumber=member("bullet").number
  the floatPrecision = 2
  sprite(gUserDataManager).resetMissileHitsAndMissesCount()
  sprite(gScoreAndLevelManager).resetCurrentStage()
  sprite(gScoreAndLevelManager).addANewLevelScore()
  sprite(gUserDataManager).setLevelInitialTime(gCurrentLevel)
  sprite(gUserDataManager).pauseButtonOn(gCurrentLevel)
  gLastScore=0
end beginsprite
```

Canto 2 Text Editor

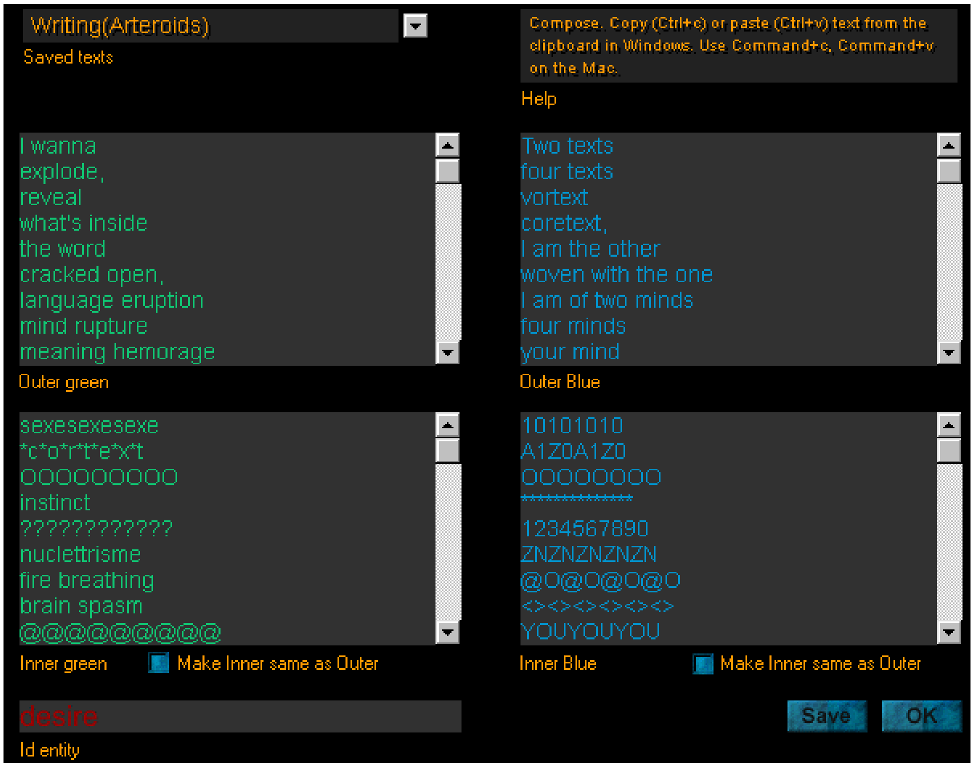


These are text sprites located above the stage. The texts are the original canto 2 texts if the player has not saved any

These are the text editor elements. The blue ones are not simply labels, whereas the purple ones are merely labels.

These are the sprites in the Save window.

These are the sprites in the Confirm Save window.


Writing(Arteroids)
Saved texts

Compose. Copy (Ctrl+c) or paste (Ctrl+v) text from the clipboard in Windows. Use Command+c, Command+v on the Mac.
Help

I wanna
explode,
reveal
what's inside
the word
cracked open,
language eruption
mind rupture
meaning hemorage
Outer green

Two texts
four texts
vortext
coretext,
I am the other
woven with the one
I am of two minds
four minds
your mind
Outer Blue

sexesexesexe
*c*o*r*t*e*x*t
OOOOOOOOOO
instinct
????????????
nuclettrisme
fire breathing
brain spasm
@@@@@@@@@
Inner green ☑ Make Inner same as Outer

10101010
A1Z0A1Z0
OOOOOOOOO
****

1234567890
ZNZNZNZNZN
@O@O@O@O
<><><><><><>
YOUYOUYOU
Inner Blue ☑ Make Inner same as Outer

desire
Id entity

Frame 63 Frame Script and Marker 2a

Notice that there is no marker 1a in Canto 1. Marker 2a is for the frame that contains the text editor, which does not exist in Canto 1.

```
--*******************************************************
--LOOP FRAME

--This is the frame script attached to the frame of the
--text editor
--*******************************************************

global gAControlIsMousedDown, gWithinAControl

on exitframe me
  go to the frame
end exitframe

on beginsprite me
  set the keyboardFocusSprite = 0
  cursor 0
  gAControlIsMousedDown=0
  gWithinAControl=0
end beginsprite
```

Channels 150-171: Text Editor Window

150: MESSAGE PANE

```
--******************************************************************
--TEXT EDITOR MESSAGE PANE INIT

--This is attached to the text member in the top right of the
--text editor, which is the Message Pane, ie, as the player
--mouses over stuff in the text editor, different pHelpMessage
--texts associated with the different sprites show in the
--Message Pane.
--******************************************************************

property spritenum, pHelpMessage
global gTextEditorMessagePane

on beginsprite me
  gTextEditorMessagePane=spritenum
  pHelpMessage="Compose. Copy (Ctrl+c) or paste (Ctrl+v) text from the clipboard in Windows. Use Command+c, Command+v on the Mac."
end beginsprite


--*********************************************************************************************
--TEXT EDITOR WINDOW LEVEL 2 MANAGER

--Shockwave does not support modal dialogue boxes or the idea of different windows in the same
--Shockwave app. So I have had to basically improvise on this matter, because there are some
--dialogue boxes that come up in the Text Editor when you go to Save a text and then Confirm
--the Save. So each of these windows has a Window Manager. What the manager does is make the
--window visible and invisible at the right points. Each particular window has
--its own pWindowElementList. The spritenums of all the sprites in that particular window
--are in the pWindowElementList managed by the Manager.
--*********************************************************************************************

property spritenum, pWindowElementList, pRefreshMe, pCurrentTextNum, pHasBeenEdited, pTextNumToLoad, pTextNumToSaveTo, pGreenInnerSameAsOuter, pBlueInnerSameAsOuter, pAllWindowSpritesAreLoaded
global gTextEditorWindowLevel2Manager, gUserDataManager
global gDropDownBLevel2, gOuterGreenLevel2, gGreenLevel2CheckBoxNum, gInnerGreenLevel2, gOuterBlueLevel2, gBlueLevel2CheckBoxNum, gInnerBlueLevel2, gIdEntityTextBoxLevel2, gDropDownALevel2

on beginsprite me
  gTextEditorWindowLevel2Manager=spritenum
  pWindowElementList=[]
  pHasBeenEdited=FALSE
  pTextNumToLoad=sprite(gUserDataManager).getCurrentTextNum()
  --sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad
  pTextNumToSaveTo=sprite(gUserDataManager).getCurrentTextNum()
  --sprite(gTextEditorWindowLevel2Manager).pTextNumToSaveTo
  pGreenInnerSameAsOuter=sprite(gUserDataManager).getInnerSameAsOuter("green")
  pBlueInnerSameAsOuter=sprite(gUserDataManager).getInnerSameAsOuter("blue")
  pCurrentTextNum=sprite(gUserDataManager).getCurrentTextNum()
  --sprite(gTextEditorWindowLevel2Manager).pCurrentTextNum
  pAllWindowSpritesAreLoaded=FALSE
  --sprite(gTextEditorWindowLevel2Manager).pAllWindowSpritesAreLoaded
  pRefreshMe=0
end beginsprite
```

```
--sprite(gTextEditorWindowLevel2Manager).makeWindowVisible()
on makeWindowVisible me
  if pAllWindowSpritesAreLoaded then
    --Read text data from gUserDataManager only when loading a text that is different from the current text
    if pRefreshMe then
      sprite(gTextEditorWindowLevel2Manager).refreshMe()
      --pCurrentTextNum=sprite(gUserDataManager).getCurrentTextNum() -- right side used to be pTextNumToLoad
    end if
  end if
  repeat with i=1 to sprite(gTextEditorWindowLevel2Manager).pWindowElementList.count
    sprite(sprite(gTextEditorWindowLevel2Manager).pWindowElementList[i]).visible=TRUE
  end repeat
  pHasBeenEdited=FALSE
  sprite(gDropDownBLevel2).visible=FALSE
  pRefreshme=0
end makeWindowVisible


--sprite(gTextEditorWindowLevel2Manager).refreshMe()
on refreshMe me
  position=pTextNumToLoad
  greenO=gOuterGreenLevel2
  greenCheckBoxNum=gGreenLevel2CheckBoxNum
  greenI=gInnerGreenLevel2
  blueO=gOuterBlueLevel2
  blueCheckBoxNum=gBlueLevel2CheckBoxNum
  blueI=gInnerBlueLevel2
  identityT=gIdEntityTextBoxLevel2
  textNameT=gDropDownALevel2
  sprite(gUserDataManager).getAllTextDataAndChangeCurrentText(position, greenO, greenCheckBoxNum, greenI, blueO,
blueCheckBoxNum, blueI, identityT, textNameT)
  pCurrentTextNum=sprite(gUserDataManager).getCurrentTextNum()
  pHasBeenEdited=FALSE
  pTextNumToLoad=sprite(gUserDataManager).getCurrentTextNum()
  --sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad
  pTextNumToSaveTo=pTextNumToLoad
  --sprite(gTextEditorWindowLevel2Manager).pTextNumToSaveTo
  pGreenInnerSameAsOuter=sprite(gUserDataManager).getInnerSameAsOuter("green")
  pBlueInnerSameAsOuter=sprite(gUserDataManager).getInnerSameAsOuter("blue")
  sprite(greenO).member.color=sprite(greenO).myColor() --next four lines are  to refresh color
  sprite(greenI).member.color=sprite(greenI).myColor()
  sprite(blueO).member.color=sprite(blueO).myColor()
  sprite(blueI).member.color=sprite(blueI).myColor()
end refreshMe


--sprite(gTextEditorWindowLevel2Manager).makeWindowInvisible()
on makeWindowInvisible me
  repeat with i=1 to sprite(gTextEditorWindowLevel2Manager).pWindowElementList.count
    sprite(sprite(gTextEditorWindowLevel2Manager).pWindowElementList[i]).visible=FALSE
  end repeat
end makeWindowInvisible


on writeToTextBuffer me,
  sprite(gUserDataManager).setBufferText(1, member("Outer Green Level 2").text)
  sprite(gUserDataManager).setBufferText(2, sprite(gTextEditorWindowLevel2Manager).pGreenInnerSameAsOuter)
  sprite(gUserDataManager).setBufferText(3, member("Inner Green Level 2").text)
  sprite(gUserDataManager).setBufferText(4, member("Outer Blue Level 2").text)
  sprite(gUserDataManager).setBufferText(5, sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter)
  sprite(gUserDataManager).setBufferText(6, member("Inner Blue Level 2").text)
  sprite(gUserDataManager).setBufferText(7, member("Id entity text box, level 2").text)
  sprite(gUserDataManager).setBufferText(8, member("Drop Down A Level 2").text)
  sprite(gUserDataManager).setBufferText(9, sprite(gUserDataManager).getCurrentTextNum())
end writeToTextBuffer


--************************************************************************************
--TEXT EDITOR WINDOW LEVEL 2 ELEMENTS

--This gets attached to each sprite in the Text Editor window.
--************************************************************************************

property spritenum
global gTextEditorWindowLevel2Manager
global gMessageManager, gTextEditorMessagePane

on beginsprite me
  sprite(spritenum).visible=TRUE
  sprite(gTextEditorWindowLevel2Manager).pWindowElementList.append(spritenum)
end beginsprite

on mouseenter me
  sprite(gMessageManager).textEditorMessage(sprite(spritenum).pHelpMessage)
end mouseenter

on mouseleave me
  sprite(gMessageManager).textEditorMessage(sprite(gTextEditorMessagePane).pHelpMessage)
end mouseenter
```

151: TEXT EDITOR HELP LABEL

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--*********************************************************************
--TEXT EDITOR HELP LABEL SCRIPT

--This is attached to the word 'Help' below the Message Pane
--*********************************************************************

property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="That which is named Help is not necessarily helpful."
end beginsprite
```

152: OUTER GREEN TEXT

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--*****************************************************************************
--GREEN OUTER EDITOR TEXT INIT LEVEL 2

--This is attached to the outer green text member in the text editor. It
--initializes some vars and identifies the color of the text.
--*****************************************************************************

global gOuterGreenLevel2, gUserDataManager
property twin, spritenum, pHelpMessage
```

```
on beginsprite me
  gOuterGreenLevel2=spritenum
  twin=spritenum+4
  sprite(spritenum).member.color= myColor()
  set the keyboardFocusSprite = 0
  sprite(spritenum).member.text=sprite(gUserDataManager).getOuterGreenText()
  pHelpMessage="When you click OK to play, this text is the green text you see."
end

on myColor me
  return rgb(20,200,120)
end myColor


--******************************************************************************
--GREEN LEVEL 2 SCRIPT

--This is attached to the outer and inner green texts in the text editor.
--******************************************************************************

property spritenum, pClicked
global gTextEditorWindowLevel2Manager

on beginsprite me
  pClicked=false
end beginsprite

on mousedown me
  pClicked=true
  pass
end mousedown

on mouseup me
  if pClicked then
   sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited=TRUE
    moi=sprite(spritenum).member.number
    member(moi).color=sprite(spritenum).myColor()
    lineNumber=sprite(spritenum).pointToLine(the mouseloc)
    if lineNumber > 0 then
      member(moi).line[linenumber].color=rgb(187,187,187)
      sprite(sprite(spritenum).twin).colorTextRemotely(lineNumber)
      sprite(sprite(spritenum).twin).member.scrollTop=member(moi).scrollTop
    end if
  end if
  pClicked=FALSE
end mousedown

on mouseupoutside me
  pClicked=FALSE
end mouseupoutside

on colorTextRemotely me, lineNumber
  if sprite(spritenum).member.line.count >= lineNumber then
    sprite(spritenum).member.color=sprite(spritenum).myColor()
    sprite(spritenum).member.line[lineNumber].color=rgb(187,187,187)
  end if
end colorTextRemotely
```

153: OUTER GREEN LABEL

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--******************************************************************************
--TEXT EDITOR OUTER GREEN LABEL SCRIPT

--This is attached to the label for the outer green text.
--******************************************************************************

property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="The Outer green is the green outer core of the poem, is streamed with the blue into a new double text."
end beginsprite
```

154: OUTER BLUE TEXT

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--******************************************************************************
--BLUE OUTER TEXT INIT LEVEL 2

--This is attached to the outer blue text and initializes some vars and
--identifies the color of its text.
--******************************************************************************

global gOuterBlueLevel2, gUserDataManager
property twin, spritenum, pHelpMessage

on beginsprite me
  gOuterBlueLevel2=spritenum
  twin=spritenum+4
  --The twin is the inner blue text sprite. I usually try to avoid this sort of +4 reference, because it
  --necessitates that if the outer blue is sprite x, then the inner blue needs to be +4 sprites away
  --in the score. But when this 'on beginsprite' handler runs, the inner blue sprite does not exist,
  --so you can't refer to it by gInnerBlueLevel2.
  sprite(spritenum).member.color= myColor()
  sprite(spritenum).member.text=sprite(gUserDataManager).getOuterBlueText()
  pHelpMessage="Click to edit, as with the other texts. This is the blue text you see when you click OK to play."
end

on myColor me
  return rgb(0,153,204)
end myColor

--******************************************************************************
--BLUE LEVEL 2 SCRIPT

--This is attached to the Outer and Inner Blue text.
--******************************************************************************

property spritenum, pClicked
global gTextEditorWindowLevel2Manager

on beginsprite me
  pClicked=false
end beginsprite

on mousedown me
```

```
  pClicked=true
  pass
end mousedown

on mouseup me
  if pClicked then
    sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited=TRUE
    moi=sprite(spritenum).member.number
    member(moi).color=sprite(spritenum).myColor()
    lineNumber=sprite(spritenum).pointToLine(the mouseloc)
    if lineNumber > 0 then
      member(moi).line[linenumber].color=rgb(187,187,187)
      sprite(sprite(spritenum).twin).colorTextRemotely(lineNumber)
      sprite(sprite(spritenum).twin).member.scrollTop=member(moi).scrollTop
    end if
  end if
  pClicked=FALSE
end mousedown

on mouseupoutside me
  pClicked=FALSE
end mouseupoutside

on colorTextRemotely me, lineNumber
  if sprite(spritenum).member.line.count >= lineNumber then
    sprite(spritenum).member.color=sprite(spritenum).myColor()
    sprite(spritenum).member.line[lineNumber].color=rgb(187,187,187)
  end if
end colorTextRemotely
```

155: Outer Blue Label

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--****************************************************************************
--TEXT EDITOR OUTER BLUE LABEL SCRIPT

--This is attached to the label for the outer blue text in the text editor.
--****************************************************************************


property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="The Outer blue is the blue outer core of the poem, is streamed with the green into a new double text."
end beginsprite
```

156: Inner Green Text

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--****************************************************************************
--GREEN INNER EDITOR TEXT INIT LEVEL 2

--This is attached to the inner green text in the text editor. It initializes
--some vars for that sprite and identifies the color of the text.
--****************************************************************************
```

```
global gInnerGreenLevel2, gOuterGreenLevel2, gUserDataManager
property twin, spritenum, pOriginalMemberNum, pHelpMessage

on beginsprite me
  gInnerGreenLevel2=spritenum
  twin=gOuterGreenLevel2
  pOriginalMemberNum=member("Inner Green Level 2").number
  if sprite(gUserDataManager).getInnerSameAsOuter("green") then
    sprite(spritenum).member=member("Outer Green Level 2")
  else
    sprite(spritenum).member=member("Inner Green Level 2")
    sprite(spritenum).member.text=sprite(gUserDataManager).getInnerGreen()
  end if
  sprite(spritenum).member.color= myColor()
  pHelpMessage="This is the text into which the Outer green explodes. When clicked, gray lines correspond."
end

on myColor me
  return rgb(20,200,120)
end myColor


--**************************************************************************
--GREEN LEVEL 2 SCRIPT

--This is attached to the inner and outer green texts in the text editor.
--***************************************************************************

property spritenum, pClicked
global gTextEditorWindowLevel2Manager

on beginsprite me
  pClicked=false
end beginsprite

on mousedown me
  pClicked=true
  pass
end mousedown

on mouseup me
  if pClicked then
   sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited=TRUE
    moi=sprite(spritenum).member.number
    member(moi).color=sprite(spritenum).myColor()
    lineNumber=sprite(spritenum).pointToLine(the mouseloc)
    if lineNumber > 0 then
      member(moi).line[linenumber].color=rgb(187,187,187)
      sprite(sprite(spritenum).twin).colorTextRemotely(lineNumber)
      sprite(sprite(spritenum).twin).member.scrollTop=member(moi).scrollTop
    end if
  end if
  pClicked=FALSE
end mousedown

on mouseupoutside me
  pClicked=FALSE
end mouseupoutside

on colorTextRemotely me, lineNumber
  if sprite(spritenum).member.line.count >= lineNumber then
```

```
    sprite(spritenum).member.color=sprite(spritenum).myColor()
    sprite(spritenum).member.line[lineNumber].color=rgb(187,187,187)
  end if
end colorTextRemotely
```

157: INNER GREEN LABEL

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--******************************************************************************
--TEXT EDITOR INNER GREEN LABEL SCRIPT

--This is attached to the label for the inner green text in the text editor.
--******************************************************************************

property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="The Inner green is the text into which the Outer green explodes. The inner of the outer word."
end beginsprite
```

158: INNER BLUE TEXT

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--******************************************************************************
--BLUE INNER TEXT EDITOR INIT LEVEL 2

--This is attached to the inner blue text in the text editor for initialization.
--******************************************************************************

global gInnerBlueLevel2, gOuterBlueLevel2,  gUserDataManager
property twin, spritenum, pOriginalMemberNum, pHelpMessage

on beginsprite me
  gInnerBlueLevel2=spritenum
  twin=gOuterBlueLevel2
  pOriginalMemberNum=member("Inner Blue Level 2").number
  if sprite(gUserDataManager).getInnerSameAsOuter("blue") then
    sprite(spritenum).member=member("Outer Blue Level 2")
  else
    sprite(spritenum).member=member("Inner Blue Level 2")
    sprite(spritenum).member.text=sprite(gUserDataManager).getInnerBlue()
  end if
  sprite(spritenum).member.color= myColor()
  pHelpMessage="This is the text into which the Outer Blue explodes. The inner and concentrically expanding blue word of the blue word."
end

on myColor me
  return rgb(0,153,204)
end myColor

--******************************************************************************
--BLUE LEVEL 2 SCRIPT

--This is attached to the Outer and Inner Blue text.
--******************************************************************************
```

```
property spritenum, pClicked
global gTextEditorWindowLevel2Manager

on beginsprite me
  pClicked=false
end beginsprite

on mousedown me
  pClicked=true
  pass
end mousedown

on mouseup me
  if pClicked then
    sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited=TRUE
    moi=sprite(spritenum).member.number
    member(moi).color=sprite(spritenum).myColor()
    lineNumber=sprite(spritenum).pointToLine(the mouseloc)
    if lineNumber > 0 then
      member(moi).line[linenumber].color=rgb(187,187,187)
      sprite(sprite(spritenum).twin).colorTextRemotely(lineNumber)
      sprite(sprite(spritenum).twin).member.scrollTop=member(moi).scrollTop
    end if
  end if
  pClicked=FALSE
end mousedown

on mouseupoutside me
  pClicked=FALSE
end mouseupoutside

on colorTextRemotely me, lineNumber
  if sprite(spritenum).member.line.count >= lineNumber then
    sprite(spritenum).member.color=sprite(spritenum).myColor()
    sprite(spritenum).member.line[lineNumber].color=rgb(187,187,187)
  end if
end colorTextRemotely
```

159: INNER BLUE LABEL

```
--******************************************************************************
--TEXT EDITOR INNER BLUE LABEL SCRIPT

--This is attached to the label for the inner blue text in the text editor.
--******************************************************************************

property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="The Inner blue is the text into which the Outer blue explodes. The inner blue of the outer blue word."
end beginsprite
```

160: ID-ENTITY TEXT

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it.

```
--******************************************************************************
--IDENTITY TEXTBOX LEVEL 2 SCRIPT
```

```
--This is attached to the id-entity text in the text editor.
--********************************************************************************


property spritenum, pHelpMessage
global gIdEntityTextBoxLevel2, gTextEditorWindowLevel2Manager, gUserDataManager

on beginsprite me
  set the keyboardFocusSprite = 0
  gIdEntityTextBoxLevel2=spritenum
  sprite(spritenum).member.text=sprite(gUserDataManager).getIdEntityText()
  pHelpMessage="This is the text you drive. You are in some sense on the side of this text or it is on yours or the drama you compose is otherwise."
end beginsprite

on mouseup me
  sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited=TRUE
end mousedown
```

161: ID-ENTITY LABEL

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it.

```
--*******************************************************************************************
--TEXT EDITOR ID ENTITY LABEL SCRIPT

--This is attached to the id entity label in the text editor
--*******************************************************************************************


property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="This text is what you drive, is either that which opposes the green and blue texts, or otherwise destroys/is destroyed by them. Or something else?"
end beginsprite
```

162: CHECK BOX FOR GREEN TEXTS

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it and the Cursor Control behavior.

```
--*******************************************************************************************
--CHECK BOX GREEN, LEVEL 2

--This is attached to the check box for the green texts in the text editor
--*******************************************************************************************


property spritenum, pClicked, pHelpMessage
global gUserDataManager, gInnerGreenLevel2, gOuterGreenLevel2, gGreenLevel2CheckBoxNum,
gTextEditorWindowLevel2Manager


on beginsprite me
  gGreenLevel2CheckBoxNum=spritenum
  if sprite(gUserDataManager).getInnerSameAsOuter("green") then
```

```
    sprite(spritenum).member=member("CheckBoxChecked copy")
    sprite(gTextEditorWindowLevel2Manager).pGreenInnerSameAsOuter=TRUE
  else
    sprite(spritenum).member=member("CheckBoxUnchecked copy")
    sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter=FALSE
  end if
  pClicked=FALSE --just a cursor thing
  pHelpMessage="When checked, Inner green is made to be the same as Outer green. You may lose text by clicking this if it is unsaved."
end beginsprite

on mousedown me
  pClicked=TRUE
end mousedown

on mouseup me
  nothing
  if pClicked then
    sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited=TRUE
    if sprite(gTextEditorWindowLevel2Manager).pGreenInnerSameAsOuter then
      sprite(gTextEditorWindowLevel2Manager).pGreenInnerSameAsOuter=FALSE
      sprite(gInnerGreenLevel2).member=member("Inner Green Level 2")
      sprite(gInnerGreenLevel2).member.text=sprite(gOuterGreenLevel2).member.text
      sprite(spritenum).member=member("CheckBoxUnchecked copy")
    else
      sprite(gTextEditorWindowLevel2Manager).pGreenInnerSameAsOuter=TRUE
      sprite(gInnerGreenLevel2).member=member("Outer Green Level 2")
      sprite(spritenum).member=member("CheckBoxChecked copy")
    end if
  end if
  pClicked=FALSE
end mouseup

on mouseupoutside me
  pClicked=FALSE
end mouseupoutside
```

163: CHECK BOX FOR BLUE TEXTS

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it and the Cursor Control behavior.

```
--*************************************************************************************
--CHECK BOX BLUE, LEVEL 2

--This is attached to the check box for the blue texts in the text editor
--*************************************************************************************


property spritenum, pClicked, pHelpMessage
global gUserDataManager, gTextEditorWindowLevel2Manager
global gInnerBlueLevel2, gOuterBlueLevel2, gBlueLevel2CheckBoxNum

on beginsprite me
  gBlueLevel2CheckBoxNum=spritenum
  if sprite(gUserDataManager).getInnerSameAsOuter("blue") then
    sprite(spritenum).member=member("CheckBoxChecked copy")
    sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter=TRUE
  else
    sprite(spritenum).member=member("CheckBoxUnchecked copy")
    sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter=FALSE
  end if
  pClicked=FALSE --just a cursor thing
  pHelpMessage="When checked, Inner blue is made to be the same as Outer blue. You may lose inner text by clicking if it is unsaved."
end beginsprite

on mousedown me
  pClicked=TRUE
end mousedown


on mouseupoutside me
  pClicked=FALSE
end mouseupoutside

on mouseup me
  if pClicked then
    sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited=TRUE
    if sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter then
      sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter=FALSE
      sprite(gInnerBlueLevel2).member=member("Inner Blue Level 2")
      sprite(gInnerBlueLevel2).member.text=sprite(gOuterBlueLevel2).member.text
      sprite(spritenum).member=member("CheckBoxUnchecked copy")
    else
      sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter=TRUE
      sprite(gInnerBlueLevel2).member=member("Outer Blue Level 2")
      sprite(spritenum).member=member("CheckBoxChecked copy")
    end if
  end if
  pClicked=FALSE
end mouseup
```

164: OK BUTTON FOR TEXT EDITOR

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it and the Cursor Control behavior.

```
--*****************************************************************************************
--OK BUTTON FOR LEVEL 2

--This is attached to the OK button in the text editor
--*****************************************************************************************


property spritenum
property pClicked, pHelpMessage
global gOKButtonLevel2
global gTextEditorWindowLevel2Manager, gConfirmWindowLevel2Manager, gUserDataManager

on beginsprite me
  gOKButtonLevel2=spritenum
  sprite(spritenum).member=member("OK Up")
  pClicked=FALSE
  pHelpMessage="Click to play (with/against (?)) the current texts."
end beginsprite

on mousedown me
  sprite(spritenum).member=member("OK Down")
  pClicked=TRUE
```

```
end mousedown


on mouseupoutside me
  pClicked=FALSE
  sprite(spritenum).member=member("OK Up")
end mouseupoutside


on mouseup me
  if pClicked then
    sprite(spritenum).member=member("OK Up")
    if member("Outer Green Level 2").text="" or member("Outer Blue Level 2").text="" then
      alert("Please write in at least the top two text boxes.")
    else
      if sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited then
        sprite(gConfirmWindowLevel2Manager).makeWindowVisible("Click OK to save text and play. Click Cancel to edit the text.")
        sprite(gConfirmWindowLevel2Manager).actionPath("OK: Save and Then Play", "Cancel: Do Not Play, Edit")
      else
        sprite(gOKButtonLevel2).proceedToPlay()
      end if
    end if
  end if
  pClicked=FALSE
end mousedown


on proceedToPlay me
  --this is called when you proceed to play from the confirm window or elsewhere
  sprite(gTextEditorWindowLevel2Manager).writeToTextBuffer()
  sprite(spritenum).changeIdEntity()
  go to the frame + 1
end proceedToPlay


on changeIdEntity me
  member("ship").text= member("Id entity text box, level 2").text
end changeIdEntity
```

165: Green Check Box Label

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--************************************************************************************
--TEXT EDITOR GREEN INNER SAME AS OUTER LABEL SCRIPT

--This is attached to the label of the checkbox for the green texts.
--************************************************************************************



property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="When checked, Inner green is made to be the same as Outer green. You may lose text by clicking this if it is unsaved."
end beginsprite
```

166: BLUE CHECK BOX LABEL

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--**************************************************************************************
--TEXT EDITOR BLUE MAKE INNER OUTER LABEL SCRIPT

--This is attached to the label for the check box for the blue texts in the text editor.
--**************************************************************************************


property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="When checked, Inner blue is made to be the same as Outer blue. You may lose inner text by clicking if it is unsaved."
end beginsprite
```

167: SAVE BUTTON

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it and the Cursor Control behavior

```
--************************************************************************************
--SAVE BUTTON LEVEL 2

--This is attached to the Save button in the text editor
--************************************************************************************

property spritenum
property pClicked, pHelpMessage
global gSaveWindowLevel2Manager, gTextEditorWindowLevel2Manager, gUserDataManager


on beginsprite me
  sprite(spritenum).member=member("Save Up")
  pClicked=FALSE
  pHelpMessage="Click to save the current text to your machine. You will be able to recall it on subsequent visits."
end beginsprite


on mousedown me
  sprite(spritenum).member=member("Save Down")
  pClicked=TRUE
end mousedown

on mouseupoutside me
  pClicked=FALSE
  sprite(spritenum).member=member("Save Up")
end mouseupoutside


on mouseup me
  if pClicked then
    cursor 0
    sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad=sprite(gUserDataManager).getCurrentTextNum()
    sprite(gTextEditorWindowLevel2Manager).pTextNumToSaveTo=sprite(gUserDataManager).getCurrentTextNum()
    sprite(gSaveWindowLevel2Manager).makeWindowVisible("Click" && QUOTE & "Save" & QUOTE && "above to save current
text or rename it to save as a new text.")
  end if
  sprite(spritenum).member=member("Save Up")
  pClicked=FALSE
end mousedown
```

168: SAVED TEXTS LABEL

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it

```
--**************************************************************************************
--SAVED TEXTS LABEL LEVEL 2

--This is attached to the label of the drop down in the text editor
--**************************************************************************************


property spritenum, pHelpMessage

on beginsprite me
  pHelpMessage="Saved texts are not necessarily redeemable."
end beginsprite
```

169: DROP DOWN B SCRIPT

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it and the Cursor Control behavior.

```
--**************************************************************************************
--DROP DOWN B SCRIPT

--This script is attached to a drop down menu in level 2. It lets the user
--select and display other texts s/he has saved.
--**************************************************************************************


property spritenum, pOldSelectedTextNum, pHelpMessage
global gUserDataManager, gCurrentLevel, gDropDownALevel2, gDropDownBLevel2, gTextEditorWindowLevel2Manager

global gOuterGreenLevel2, gGreenLevel2CheckBoxNum, gInnerGreenLevel2, gOuterBlueLevel2, gBlueLevel2CheckBoxNum,
gInnerBlueLevel2
global gConfirmWindowLevel2Manager, gIdEntityTextBoxLevel2

on beginsprite me
  gDropDownBLevel2=spritenum
  pOldSelectedTextNum=0
  sprite(spritenum).visible=false
  sprite(spritenum).refreshme()
  pHelpMessage="Click a saved text to edit it. Each saved, named text consists of Outer green, Inner green, Outer blue, Inner blue,
and Id-entity texts."
end beginsprite
```

```
on refreshme me
  sprite(spritenum).member.text=""
  repeat with i=1 to sprite(gUserDataManager).getNumberOfTexts()
    sprite(spritenum).member.line[i]=sprite(gUserDataManager).getTextName2(i)
  end repeat
  set the foreColor of field sprite(spritenum).member = 27
  pOldSelectedTextNum=0
end refreshme

on mouseup me
  tempSelectedTextNum=sprite(spritenum).pointToLine(the MouseLoc)
  if tempSelectedTextNum>0 then

    sprite(spritenum).member.line[tempSelectedTextNum].hilite()
    if sprite(gTextEditorWindowLevel2Manager).pHasBeenEdited then
      --then you should prompt to save edited current before moving to new selected text
      --open window
      --pass parameters needed to save or simply open new window
      if tempSelectedTextNum=sprite(gUserDataManager).getCurrentTextNum() then
        alert("This text is currently open.")
      else
        sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad=tempSelectedTextNum
        sprite(gTextEditorWindowLevel2Manager).pTextNumToSaveTo=sprite(gTextEditorWindowLevel2Manager).pCurrentTextNum
        sprite(gConfirmWindowLevel2Manager).makeWindowVisible("Would you like to save the current text before opening the one 
you clicked? Seems like you have edited it.")
        sprite(gConfirmWindowLevel2Manager).actionPath("OK: Display Save Window", "Cancel: Open new text and close modals")
      end if
    else
      if tempSelectedTextNum=sprite(gUserDataManager).getCurrentTextNum() then
        alert("This text is currently open.")
      else
        --then current text has not been edited, so make the switch
        sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad=tempSelectedTextNum
        sprite(gTextEditorWindowLevel2Manager).refreshMe()
      end if
    end if
    sprite(spritenum).visible=FALSE
  end if
end mouseup


on mousewithin me
  tempNewSelectedTextNum=sprite(spritenum).pointToLine(the MouseLoc)
  if tempNewSelectedTextNum>0 then
    if pOldSelectedTextNum<>tempNewSelectedTextNum then
      if pOldSelectedTextNum >0 then
        set the foreColor of line pOldSelectedTextNum of field sprite(spritenum).member = 27
      end if
      set the foreColor of line tempNewSelectedTextNum of field sprite(spritenum).member = 6
      pOldSelectedTextNum=tempNewSelectedTextNum
    end if
  end if

end mousewithin

on mouseleave me
  set the foreColor of field sprite(spritenum).member = 27
  sprite(spritenum).visible=FALSE
  pOldSelectedTextNum=0
end mouseleave
```

170: DROP DOWN A MEMBER

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it and the Cursor Control behavior.

```
--************************************************************************************
--DROP DOWN A LEVEL 2 SCRIPT

--The drop down menu at top left of the text editor is composed of two elements:
--there's a text element which, when clicked, displays a drop down. This script
--is attached to the former text element.
--************************************************************************************

global gDropDownBLevel2, gDropDownALevel2, gUserDataManager
property spritenum, pHelpMessage

on beginsprite me
  gDropDownALevel2=spritenum
  sprite(spritenum).member.text=sprite(gUserDataManager).getTextName()
  pHelpMessage="Show/hide the list of saved texts."
end beginsprite

on mouseup me
  sprite(gDropDownBLevel2).locZ=spritenum
  if sprite(gDropDownBLevel2).visible then
    sprite(gDropDownBLevel2).visible=FALSE
  else
    sprite(gDropDownBLevel2).refreshMe()
    sprite(gDropDownBLevel2).visible=TRUE
  end if
end mouseup
```

171: DROP DOWN BUTTON

This sprite also has the 'Text Editor Window Level 2 Elements' script attached to it and the Cursor Control behavior.

```
--************************************************************************
--LAST SPRITE IN THE TEXT EDITOR WINDOW

--I'm not sure whether this script is necessary or not.
--************************************************************************

global gTextEditorWindowLevel2Manager
on beginsprite me
  sprite(gTextEditorWindowLevel2Manager).pAllWindowSpritesAreLoaded=TRUE
end beginsprite


--************************************************************************
--DROP DOWN BUTTON

--There's a button you can click to drop down the drop down menu at top
--left in the text editor. This script is attached to that button.
--************************************************************************

global gDropDownBLevel2
property pHelpMessage
```

```
on beginsprite me
  pHelpMessage="Show/hide the list of saved texts."
end beginsprite

on mouseup me
  if sprite(gDropDownBLevel2).visible then
    sprite(gDropDownBLevel2).visible=FALSE
  else
    sprite(gDropDownBLevel2).refreshMe()
    sprite(gDropDownBLevel2).visible=TRUE
  end if
end mouseup
```

Channels 173-181: Save Window

When you click the Save button in the Text Editor, the Save Window appears. Shockwave does not support modal dialog boxes, which is what the Save Window encapsulates not too badly. So, to have a modal dialogue box, what I have done is create a Window Manager behavior attached to sprite 174. Basically the idea is that the sprite with the smallest number in a particular window is a 1 pixel bitmap that stretches to fill the screen. This sprite blocks the possibility of being able to mouseover stuff in the Text Editor while the Save Window is open. And the little 1 pixel bitmap has the Window Manager behavior attached to it.

This idea is similar to structures you see in Delphi, for instance, or Visual Basic.

173: MODAL BACKGROUND BITMAP

Channel 174 has a 1 pixel bitmap in it. This is stretched to cover the full screen. It is to block the possibility of being able to mouseover the text editor while the Save Window is open. The modal background bitmap also has the Save Window Level 2 Manager attached to it, which basically makes the whole save window (made up of sprites 174-181) visible or invisible at the right times. The Save Window Manager also contains a lot of the routines that you would normally put in behaviors attached to the sprites that trigger the behaviors.

```
--*****************************************************************************
--SAVE WINDOW LEVEL 2 MANAGER

--You can save texts in the text editor if you click the Save button.
--When you click the Save button, basically a modal dialogue box appears.
--But Shockwave does not support modal dialogue boxes, which are basically
--windows. So I have written this window manager for the Save modal dialog
--box window. Each element of the modal dialog box adds itself on beginsprite
--to the pWindowElementList so that the window can be made to be visible and
--invisible at the right points.
--*****************************************************************************

property spritenum, pWindowElementList, pTextNumToDelete
global gUserDataManager, gSaveWindowLevel2Manager, gTextEditorWindowLevel2Manager, gConfirmWindowLevel2Manager
global gLevel2TextListBox, gSaveAsTextBoxLevel2, gSaveWindowMessagePane

on beginsprite me
  gSaveWindowLevel2Manager=spritenum
  pWindowElementList=[]
  pTextNumToDelete=0
end beginsprite

--sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
on makeWindowInvisible me
  repeat with i=1 to sprite(gSaveWindowLevel2Manager).pWindowElementList.count
```

```
    sprite(sprite(gSaveWindowLevel2Manager).pWindowElementList[i]).visible=FALSE

sprite(sprite(gSaveWindowLevel2Manager).pWindowElementList[i]).locZ=sprite(gSaveWindowLevel2Manager).pWindowElementLi
st[i]
  end repeat
end makeWindowInvisible


--sprite(gSaveWindowLevel2Manager).makeWindowVisible(textMessage)
on makeWindowVisible me, textMessage
  sprite(gSaveWindowMessagePane).member.text=textMessage
  sprite(spritenum).refreshMe()
  repeat with i=1 to sprite(gSaveWindowLevel2Manager).pWindowElementList.count
    mySpriteNum=sprite(gSaveWindowLevel2Manager).pWindowElementList[i]
    sprite(mySpriteNum).locZ=mySpriteNum
    sprite(mySpriteNum).visible=TRUE
  end repeat

end makeWindowVisible

on refreshMe me
  sprite(gLevel2TextListBox).refreshMe()
  sprite(gSaveAsTextBoxLevel2).refreshMe()
  pTextNumToDelete=0
end refreshMe



on onSaveButtonUp me
  case sprite(gSaveAsTextBoxLevel2).member.text of
    "":
     --The player is trying to save an unnamed text.
     sprite(gSaveWindowMessagePane).visible=TRUE
     sprite(gSaveWindowMessagePane).member.text="Please name the current text. Not named => not saved."
    otherwise:
     i=1
     tempTargetLine=sprite(gUserDataManager).getNumberOfTexts() + 1
     --Let's see if the text title the user typed in gSaveAsTextBoxLevel2 matches any of the titles
     --already in gLevel2TextListBox.
     repeat while i <= sprite(gUserDataManager).getNumberOfTexts() and
tempTargetLine=sprite(gUserDataManager).getNumberOfTexts() + 1
       if sprite(gSaveAsTextBoxLevel2).member.text = sprite(gUserDataManager).getTextName2(i) then
         tempTargetLine=i
       end if
       i=i+1
     end repeat
     sprite(gTextEditorWindowLevel2Manager).pTextNumToSaveTo=tempTargetLine

     if tempTargetLine=sprite(gUserDataManager).getNumberOfTexts() + 1 then
       --no other canto has the same name as what's in the save as box, so do the save
       sprite(spritenum).setupAndSave()
       --sprite(gSaveWindowMessagePane).member.text="Current texts successfully saved."
       sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
       sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad=tempTargetLine --this was commented out. Why?
       sprite(gTextEditorWindowLevel2Manager).pRefreshMe=1 --(full refresh)
       sprite(gTextEditorWindowLevel2Manager).makeWindowVisible()
     else
       --pTargetLine is a number that corresponds with what's in the save as box, so open the confirm window, hold the save
       sprite(gConfirmWindowLevel2Manager).makeWindowVisible("The current text will overwrite the named text. Click OK to
overwrite or Cancel to cancel the save.")
       sprite(gConfirmWindowLevel2Manager).actionPath("OK: Save and Close Modals", "Cancel: Cancel Save, Close Modals")
     end if
  end case
end saveAsBoxIsSavable


on setupAndSave me
  --Text data is written go gUserDataManager only on save here, nowhere else except in getprefs.
  greenO=member("Outer Green Level 2").text
  if sprite(gTextEditorWindowLevel2Manager).pGreenInnerSameAsOuter then
    greenI=""
  else
    greenI=member("Inner Green Level 2").text
  end if
  blueO=member("Outer Blue Level 2").text
  if sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter then
    blueI=""
  else
    blueI=member("Inner Blue Level 2").text
  end if
  identityT=member("Id entity text box, level 2").text
  textNameT=member("Save As Box Level 2").text
  blueInnerSameAsOuter=sprite(gTextEditorWindowLevel2Manager).pBlueInnerSameAsOuter
  greenInnerSameAsOuter=sprite(gTextEditorWindowLevel2Manager).pGreenInnerSameAsOuter
  targetLine=sprite(gTextEditorWindowLevel2Manager).pTextNumToSaveTo
  sprite(gUserDataManager).setAllTextData(targetLine, greenO, greenI, blueO, blueI, identityT, textNameT,
blueInnerSameAsOuter, greenInnerSameAsOuter)
end setupAndSave


--sprite(gSaveWindowLevel2Manager).cancelButtonUp()
on cancelButtonUp me
  sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
  sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
  sprite(gTextEditorWindowLevel2Manager).makeWindowVisible()
end cancelButtonUp

on deleteIsRequested me
  --oldCurrentTextNum=sprite(gUserDataManager).getCurrentTextNum() --this is wrong. Text to delete may not be current text.
Could be text clicked in file list.
  if pTextNumToDelete>0  then
    --There is probably a valid textnum selected in the list box to delete.
    if pTextNumToDelete <= sprite(gUserDataManager).getNumberOfTexts() then
      --then there does exist a text to delete.
      if sprite(gUserDataManager).getNumberOfTexts()= 1 then
        --then there is only one text so it should not be deleted
        alert("You must keep at least one text, or the game becomes quite ineffable or conceptual, is wholely within the unsaid and
unwritten, more than now.")
      else
        sprite(gConfirmWindowLevel2Manager).makeWindowVisible("Click OK to delete the selected text from your hard drive, or
click Cancel to not delete.")
        sprite(gConfirmWindowLevel2Manager).actionPath("OK: Delete Selected Text, Close Confirm Window, Update Properties",
"Cancel: Do Not Delete, Close Confirm Window")
      end if
    else
      alert("To delete a text, first select it in the list box.")
    end if
  else
    alert("To delete a text, first select it in the list box.")
  end if
end deleteIsRequested


on deleteDone me
```

```
  deleteOperationResult=sprite(gUserDataManager).deleteText(pTextNumToDelete)
  if deleteOperationResult = 0 then
    alert("We have bug 4 here, sorry. Please email jim@vispo.com, telling me you encountered bug 4. Sorry. Best quit the
program.")
  else
    
    --adjust sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad with logic from guserdatamanger
    tempTextNumToLoad=sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad
    if tempTextNumToLoad>=pTextNumToDelete then
      --then pCurrentTextNums needs adjustment. And what about pTextNumToLoad and pTextNumToLoadTo?
      if tempTextNumToLoad>pTextNumToDelete then
        sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad=tempTextNumToLoad-1
      else
        if pTextNumToDelete=1 then
          sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad=1
        else
          sprite(gTextEditorWindowLevel2Manager).pTextNumToLoad=tempTextNumToLoad-1
        end if
      end if
    else
      --what other variables in this unit need adjustment? Not finished.
    end if
    pTextNumToDelete=0
  end if
end deleteDone



--*****************************************************************************
--SAVE WINDOW LEVEL 2 ELEMENTS

--Each sprite in the Save window has this sprite attached to it.
--*****************************************************************************


property spritenum
global gSaveWindowLevel2Manager

on beginsprite me
  sprite(gSaveWindowLevel2Manager).pWindowElementList.append(spritenum)
  sprite(spritenum).visible=FALSE
end beginsprite


--*****************************************************************************
--SAVE WINDOW LEVEL 2 SCREEN MODAL SCRIPT

--This behavior traps mouse events so that when the Save window is displayed,
--you can't click the text editor below it.
--*****************************************************************************


property spritenum

on beginsprite me
  sprite(spritenum).left=0
  sprite(spritenum).right=the stageright
  sprite(spritenum).top=0
  sprite(spritenum).bottom=the stagebottom
  sprite(spritenum).blend=50
end beginsprite me
```

```
on mousedown me
  --this is just to trap events
end mousedown

on mouseup me
  --this is just to trap events
end mouseup

on mouseupoutside me
  --this is just to trap events
end mouseupoutside

on mouseenter me
  --this is just to trap events
end mouseenter

on mouseleave me
  --this is just to trap events
end mouseleave
```

174: MODAL BACKGROUND 6

When you click the Save Button in the Text Editor, you see a graphic background in blues and greens. This sprite is that graphical member.

```
--****************************************************************************
--SAVE WINDOW LEVEL 2 ELEMENTS

--Each sprite in the Save window has this sprite attached to it.
--****************************************************************************


property spritenum
global gSaveWindowLevel2Manager

on beginsprite me
  sprite(gSaveWindowLevel2Manager).pWindowElementList.append(spritenum)
  sprite(spritenum).visible=FALSE
end beginsprite
```

175: SAVE AS LABEL

This is the label for the Save As window. Like all the sprites in the Save Window, this sprite has the Save Window Level 2 Elements script attached to it.

176: SAVE BUTTON

The Save button also has the Save Window Level 2 Elements script attached to it and the Cursor Control behavior.

```
--****************************************************************************
--SAVE WINDOW LEVEL 2 SAVE BUTTON

--This Saves texts on mouseup.
--****************************************************************************
```

```
property spritenum
property pClicked, pTargetLine
global gTextEditorWindowLevel2Manager, gSaveWindowLevel2Manager, gUserDataManager, gConfirmWindowLevel2Manager,
gCurrentLevel
global gOuterGreenLevel2, gInnerGreenLevel2
global gOuterBlueLevel2, gInnerBlueLevel2
global gIdEntityTextBoxLevel2
global gSaveButton, gSaveAsTextBoxLevel2, gSaveWindowMessagePane, gLevel2TextListBox

on beginsprite me
  gSaveButton=spritenum
  sprite(spritenum).member=member("Save Up")
  pClicked=FALSE
  pTargetLine=0
end beginsprite


on mousedown me
  sprite(spritenum).member=member("Save Down")
  pClicked=TRUE
end mousedown

on mouseupoutside me
  pClicked=FALSE
  sprite(spritenum).member=member("Save Up")
end mouseupoutside


on mouseup me
  sprite(spritenum).member=member("Save Up")
  if pClicked then
    --saveAsIsOK=saveAsBoxIsSavable()
    sprite(gSaveWindowLevel2Manager).onSaveButtonUp()
  end if
  --can maybe put stuff here
  pClicked=FALSE
end mousedown


on showUserMessage me, saveAsStatus

end showUserMessage
```

177: DELETE BUTTON

This sprite also has the Save Window Level 2 Elements script attached to it and the Cursor Control behavior.

```
--*******************************************************************************
--SAVE WINDOW LEVEL 2 DELETE BUTTON

--You can delete saved texts. This initiates that process on mouseup.
--*******************************************************************************

property spritenum
property pClicked
global gSaveWindowDeleteButton, gSaveWindowLevel2Manager

on beginsprite me
```

```
  gSaveWindowDeleteButton=spritenum
  sprite(spritenum).member=member("Delete Up")
  pClicked=FALSE
end beginsprite

on mousedown me
  sprite(spritenum).member=member("Delete Down")
  pClicked=TRUE
end mousedown

on mouseupoutside me
  pClicked=FALSE
  sprite(spritenum).member=member("Delete Up")
end mouseupoutside

on mouseup me
  sprite(spritenum).member=member("Delete Up")
  if pClicked then
    sprite(gSaveWindowLevel2Manager).deleteIsRequested()
  end if
  sprite(spritenum).member=member("Delete Up")
  pClicked=FALSE
end mousedown
```

178: CANCEL BUTTON

This sprite also has the Save Window Level 2 Elements script attached to it and the Cursor Control behavior.

```
--****************************************************************************
--SAVE WINDOW LEVEL 2 CANCEL BUTTON

--The cancel button cancels a save file operation on mouseup
--****************************************************************************


property spritenum
property pClicked
global gSaveWindowLevel2Manager

on beginsprite me
  sprite(spritenum).member=member("Cancel Up")
  pClicked=FALSE
end beginsprite


on mousedown me
  sprite(spritenum).member=member("Cancel Down")
  pClicked=TRUE
end mousedown

on mouseupoutside me
  pClicked=FALSE
  sprite(spritenum).member=member("Cancel Up")
end mouseupoutside


on mouseup me
  sprite(spritenum).member=member("Cancel Up")
  if pClicked then
    sprite(gSaveWindowLevel2Manager).cancelButtonUp()
  end if
  sprite(spritenum).member=member("Cancel Up")
  pClicked=FALSE
end mousedown
```

179: LIST BOX

This sprite also has the Save Window Level 2 Elements script attached to it

```
--*****************************************************************************
--TEXT LIST BOX LEVEL 2 SCRIPT

--There's a list box that lists saved files in the Save Window. This behavior
--defines the actions that happen on mousedown to it.
--*****************************************************************************


property spritenum
global gLevel2TextListBox, gSaveWindowLevel2Manager, gUserDataManager, gSaveWindowDeleteButton, gCurrentLevel,
gSaveAsTextBoxLevel2, gTextEditorWindowLevel2Manager

on beginsprite me
  gLevel2TextListBox=spritenum
  sprite(spritenum).member.text=""
  sprite(spritenum).refreshMe()
end beginsprite

on mousedown me
  tempSelectedTextNum=sprite(spritenum).pointToLine(the mouseloc)
  if tempSelectedTextNum>0 then
    sprite(gTextEditorWindowLevel2Manager).pTextNumToSaveTo=tempSelectedTextNum
    sprite(spritenum).member.line[tempSelectedTextNum].hilite()
    sprite(gSaveAsTextBoxLevel2).member.text=sprite(spritenum).member.line[tempSelectedTextNum]
    sprite(gSaveWindowLevel2Manager).pTextNumToDelete=tempSelectedTextNum
  end if
end mouseup

--sprite(gLevel2TextListBox).refreshMe()
on refreshMe me
  sprite(spritenum).member.text=""
  repeat with i= 1 to sprite(gUserDataManager).getNumberOfTexts()
    sprite(spritenum).member.line[i]=sprite(gUserDataManager).getTextName2(i)
  end repeat
end refreshMe
```

180: LIST BOX LABEL

This text sprite says 'texts saved'. This sprite has the Save Window Level 2 Elements script attached to it.

181: SAVE AS BOX

This sprite also has the Save Window Level 2 Elements script attached to it.

```
--***************************************************************************
--SAVE WINDOW, SAVE AS BOX LEVEL 2

--The Save As box shows the name of the text to be saved.
--***************************************************************************


property spritenum
global gSaveAsTextBoxLevel2, gTextEditorWindowLevel2Manager, gUserDataManager

on beginsprite me
  gSaveAsTextBoxLevel2=spritenum
  sprite(spritenum).refreshMe()
end beginsprite

--sprite(gSaveAsTextBoxLevel2).refreshMe()
on refreshMe me
  position= sprite(gTextEditorWindowLevel2Manager).pCurrentTextNum
  if position>0 then
    sprite(spritenum).member.text=sprite(gUserDataManager).getTextName2(position)
  else
    sprite(spritenum).member.text=sprite(gUserDataManager).getTextName()
  end  if
  set the keyboardFocusSprite = spritenum
end refreshMe
```

Channels 183-188: Confirm Window

When you save a text, you may be requested to confirm the Save at certain points. Much like we experience other modal dialog boxes when we save files in programs like Photoshop or whatever. Sprites 183-188 form one such window, which is managed by the Confirm Window Level 2 Manager script.

183: MODAL BACKGROUND BITMAP

As in the Save Window, the Confirm Window's first sprite is a 1 pixel bitmap that is stretched across the screen to capture mouse events so that when the Confirm Window is open, you can't access the Save Window or the Text Editor. This bitmap also has the below behaviors attached to it.

```
--***************************************************************************
--CONFIRM WINDOW LEVEL 2 MANAGER

--When you Save a text, you are asked to confirm the save in many cases, such
--as when you will overwrite a text with the same name. This confirmation
--essentially raises another modal dialog box. This behavior manages the confirm
--window.
--***************************************************************************


property spritenum, pWindowElementList, pOnOK, pOnCancel
global gConfirmWindowLevel2Manager, gSaveWindowLevel2Manager, gTextEditorWindowLevel2Manager, gUserDataManager
global gSaveWindowMessagePane, gOKButtonLevel2

on beginsprite me
  gConfirmWindowLevel2Manager=spritenum
  pWindowElementList=[]
  pOnOK=""
  --possible values: "", "OK: Display Save Window", "OK: Save and Close Modals"
  --accessed by sprite(gConfirmWindowLevel2Manager).pOnOK
  pOnCancel=""
  --possible values: "", "Cancel: Open new text and close modals"
  --accessed by sprite(gConfirmWindowLevel2Manager).pOnCancel
end beginsprite

--sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
on makeWindowInvisible me
  repeat with i=1 to sprite(gConfirmWindowLevel2Manager).pWindowElementList.count
    sprite(sprite(gConfirmWindowLevel2Manager).pWindowElementList[i]).visible=FALSE
  end repeat
  pOnOK=""
  pOnCancel=""
end makeWindowInvisible

--sprite(gConfirmWindowLevel2Manager).makeWindowVisible("Put message here")
on makeWindowVisible me, textMessage
  sprite(gSaveWindowMessagePane).member.text=textMessage
  repeat with i=1 to sprite(gConfirmWindowLevel2Manager).pWindowElementList.count
    myspriteNum=sprite(gConfirmWindowLevel2Manager).pWindowElementList[i]
    sprite(mySpriteNum).locZ=mySpriteNum
    sprite(mySpriteNum).visible=TRUE
  end repeat
  set the keyboardFocusSprite = 0
end makeWindowInvisible

--sprite(gConfirmWindowLevel2Manager).actionPath(selectedText, "OK: Display Save Window", "Cancel: Open new text and close modals")
--possible values: "", "OK: Display Save Window", "OK: Save and Close Modals"
--accessed by sprite(gConfirmWindowLevel2Manager).pOnOK
on actionPath me, onOK, onCancel
  pOnOK=onOK
  pOnCancel=onCancel
end actionPath

--sprite(gConfirmWindowLevel2Manager).messageText(myMessage)
on messageText me, myMessage
  sprite(gSaveWindowMessagePane).member.text=myMessage
end messageText


--sprite(gConfirmWindowLevel2Manager).OKButtonUp()
on OKButtonUp me
  case pOnOK of
    "":
      --this should not be possible
      alert("You have encountered a bug. This is bug number 1. Please report this to jim@vispo.com.")
    "OK: Display Save Window":
      --This occurs if the user started by selecting a text from the main window drop down
      sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
      sprite(gSaveWindowLevel2Manager).makeWindowVisible("Click" && QUOTE & "Save" & QUOTE && "to save current text or rename it to save as a new text.")
    "OK: Save and Close Modals":
      sprite(gSaveWindowLevel2Manager).setUpAndSave()
      sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
      sprite(gSaveWindowLevel2Manager).makeWindowInvisible() --new
      sprite(gTextEditorWindowLevel2Manager).pRefreshMe=1 --full refresh
      sprite(gTextEditorWindowLevel2Manager).makeWindowVisible()
    "OK: Save and Then Play":
      sprite(gSaveWindowLevel2Manager).setUpAndSave()
      sprite(gOKButtonLevel2).proceedToPlay()
    "OK: Close Confirm Window":
```

```
    sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
    sprite(gTextEditorWindowLevel2Manager).makeWindowVisible()
   "OK: Delete Selected Text, Close Confirm Window, Update Properties": --to do
    sprite(gSaveWindowLevel2Manager).deleteDone()
    sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
    sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
    sprite(gTextEditorWindowLevel2Manager).pRefreshMe=1
    sprite(gTextEditorWindowLevel2Manager).makeWindowVisible()

   otherwise:
    alert("You have encountered a bug. This is bug number 2. Please report this to jim@vispo.com.")
  end case
end OKButtonUp


global gOuterGreenLevel2, gGreenLevel2CheckBoxNum, gInnerGreenLevel2, gOuterBlueLevel2, gBlueLevel2CheckBoxNum,
gInnerBlueLevel2, gIdEntityTextBoxLevel2, gDropDownALevel2

on cancelButtonUp me
  case pOnCancel of
   "Cancel: Open new text and close modals":
    --check that this is the right value for here
    --this is for when cancel is clicked after the user selects a text from the drop down selector in the main window
    sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
    sprite(gSaveWindowLevel2Manager).makeWindowInvisible() --new
    sprite(gTextEditorWindowLevel2Manager).pRefreshMe=1
    sprite(gTextEditorWindowLevel2Manager).makeWindowVisible()
   "Cancel: Cancel Save, Close Modals":
    sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
    sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
   "Cancel: Do Not Play, Edit":
    sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
    sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
   "Cancel: Close Confirm Window":
    sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
    sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
   "Cancel: Do Not Delete, Close Confirm Window":
    sprite(gConfirmWindowLevel2Manager).makeWindowInvisible()
    sprite(gSaveWindowLevel2Manager).makeWindowInvisible()
   otherwise:
    alert("You have encountered a bug. This is bug number 3. Please report this to jim@vispo.com.")
  end case
end cancelButtonUp


--****************************************************************************
--CONFIRM WINDOW LEVEL 2 ELEMENTS

--This gets attached to all the sprites in the Confirm Window so that the
--Confirm Window Manager can make the window visible/invisible.
--****************************************************************************

property spritenum
global gConfirmWindowLevel2Manager

on beginsprite me
  sprite(gConfirmWindowLevel2Manager).pWindowElementList.append(spritenum)
  sprite(spritenum).visible=FALSE
end beginsprite


--****************************************************************************
```

```
--CONFIRM WINDOW LEVEL 2 SCREEN MODAL SCRIPT

--This behavior traps mouse events so that when the Confirm Window is open,
--you can't do anything with the Save Window.
--******************************************************************************

property spritenum

on beginsprite me
  sprite(spritenum).left=0
  sprite(spritenum).right=the stageright
  sprite(spritenum).top=0
  sprite(spritenum).bottom=the stagebottom
  sprite(spritenum).blend=40
end beginsprite me


on mousedown me
  --this is just to trap events
end mousedown

on mouseup me
  --this is just to trap events
end mouseup

on mouseupoutside me
  --this is just to trap events
end mouseupoutside

on mouseenter me
  --this is just to trap events
end mouseenter

on mouseleave me
  --this is just to trap events
end mouseleave
```

184: MODAL BACKGROUND 7

This is the graphical background to the Confirm Window. No special reason why the member is called Modal Background 7 apart from it was the 7th attempt at such a graphic. This sprite has the Confirm Window Level 2 Elements script attached to it.

185: MESSENGE PANE

The Messenge pane is the same member for both the Save and Confirm windows. It has three behaviors attached to it: Save Window Level 2 Elements, Confirm Window Level 2 Elements, and the below.

```
--******************************************************************************
--SAVE WINDOW MESSENGE PANE LEVEL 2

--The Save Window Messenge Pane displays messenges to the user.
--******************************************************************************


property spritenum
global gSaveWindowMessagePane
```

```
on beginsprite me
  gSaveWindowMessagePane=spritenum
end beginsprite
```

186: OK BUTTON

The OK button, which confirms the Save or delete, also has the Confirm Window Level 2 Elements behavior and the Cursor Control behavior assigned to it.

```
--******************************************************************************
--CONFIRM WINDOW OK BUTTON LEVEL 2

--On mouseup of the Confirm Window OK button, the save or delete is performed
--******************************************************************************

property spritenum
property pClicked
global gConfirmWindowLevel2Manager

on beginsprite me
  sprite(spritenum).member=member("OK Up")
  pClicked=FALSE
end beginsprite


on mousedown me
  sprite(spritenum).member=member("OK Down")
  pClicked=TRUE
end mousedown

on mouseupoutside me
  pClicked=FALSE
  sprite(spritenum).member=member("OK Up")
end mouseupoutside


on mouseup me
  if pClicked then
    sprite(spritenum).member=member("OK Up")
    sprite(gConfirmWindowLevel2Manager).OKButtonUp()
  end if
  pClicked=FALSE
end mousedown
```

187: CANCEL BUTTON

It also has the Confirm Window Level 2 Elements behavior and the Cursor Control behavior assigned to it.

```
--******************************************************************************
--CONFIRM WINDOW CANCEL BUTTON

--This button cancels the save or delete operation.
--******************************************************************************
```

```
property spritenum
property pClicked
global gConfirmWindowLevel2Manager
global gConfirmWindowCancelButton

on beginsprite me
  sprite(spritenum).member=member("Cancel Up")
  pClicked=FALSE
  gConfirmWindowCancelButton=spritenum
end beginsprite


on mousedown me
  sprite(spritenum).member=member("Cancel Down")
  pClicked=TRUE
end mousedown

on mouseupoutside me
  pClicked=FALSE
  sprite(spritenum).member=member("Cancel Up")
end mouseupoutside


on mouseup me
  sprite(spritenum).member=member("Cancel Up")
  if pClicked then
    sprite(gConfirmWindowLevel2Manager).cancelButtonUp()
  end if
  pClicked=FALSE
end mousedown
```

188: MESSAGES FROM HELL LEVEL 2

This is the label for the Messenge Pane. It is the same label as for the Save Window. This text sprite is used in the Save window and Confirm window. It has two behaviors attached to it: Save Window Level 2 Elements and Confirm Window Level 2 Elements.

Pre Canto 2 Processing

Frame 67: Press a Key to Play

Frame 67 is the 'press a key to play' frame for Canto 2. There is no significant processing in this frame. The Canto initialization routine has already been performed prior to entering the Text Editor, which we have passed, at this point.

The frame has the following behavior attached to it, which just waits for a key to be pressed.

```
--****************************************************************************
--PRESS A KEY TO ENTER LEVEL 2

--This behavior is attached to the frame in which the player must press a key
--to enter level 2.
--****************************************************************************


on beginsprite me
  cursor 0
end beginsprite

on exitframe me
  if the keypressed <> "" then
    go to the frame + 1
  else
    go to the frame
  end if
end
```

Frame 71-72: Pause Button Off, Add Enemy Texts

The Pause Button Off and Add Enemy Texts behaviors in frames 71 and 72 occur in Canto 2 as they do in Canto 1. See the docs on Canto 1 for more info

Canto 2 Play

Canto 2 is much like Canto 1--probably too much like Canto 1. The differences are these: you use text from the Text Editor for the Outer Green, Inner Green, Outer Blue, Inner Blue, and Id Entity texts rather than, as in Canto 1, text that the player cannot change; also, when you are playing, there is the option to 'Edit Text', that is, return to the text editor and edit the text; finally, there are more arteroids sent into play. This is controlled via the settings in the Score and Level Manager behavior, in particular by the pLevelAction variable.

So I'm only going to document code that is different from Canto 1 here. This doc is too long already. Will anybody ever use it? Probably for parts, which is OK. But I thought I'd better document the thing so that it can be used in whatever way you like, so that it can be understood rather than offering source code that is unintellibable.

Scoreboard

202: CLICK TO EDIT

```
--***************************************************************************
--CLICK TO EDIT TEXT LEVEL 2

--This is specific to level 2. It's attached to the sprite that says
--'click to edit text'. When clicked, this sprite takes the player to the
--text editor.
--***************************************************************************


global gCurrentLevel, gClickToEditButton
property spritenum

on mousedown me
  go to string(gCurrentLevel) & "a"
end

on beginsprite me
  gClickToEditButton=spritenum
  sprite(spritenum).visible=TRUE
end beginsprite
```

Green Texts

```
--****************************************************************************************************************
--GREEN TEXT INIT FOR LEVEL 2

--This is attached to the first green webarteroid in level 2. It initializes
--the texts that the green Arteroids will use.
--All the green Arteroids use the below texts in pMyTextsMain, and
--the green Arteroids read pMyTextsMain on beginsprite, if you look at their behavior.
--THIS USES VALUES FROM GUSERDATAMANAGER, CANNOT USE VALUES FROM SOME OF THE TEXT EDITOR
ELEMENTS THEMSELVES (THEY NO LONGER EXIST)
--****************************************************************************************************************


property pMyTextsMain, pMyExplodingTextsMain, pMyTotalNumberOfTextsMain, pMyTextWidthsMain, spritenum
```

```
global gGreenTextAdjuster, gUserDataManager

on beginsprite me
  gGreenTextAdjuster=spritenum
  pMyTextsMain=[]
  repeat with i=1 to member("Outer Green Level 2").paragraph.count
    pMyTextsMain[i]= member("Outer Green Level 2").paragraph[i]
  end repeat
  pMyExplodingTextsMain=[]
  if sprite(gUserDataManager).getBufferText(2) then
    repeat with i=1 to sprite(gUserDataManager).getBufferText(1).paragraph.count
      pMyExplodingTextsMain.append(sprite(gUserDataManager).getBufferText(1).paragraph[i])
    end repeat
  else
    repeat with i=1 to sprite(gUserDataManager).getBufferText(3).paragraph.count
      pMyExplodingTextsMain.append(sprite(gUserDataManager).getBufferText(3).paragraph[i])
    end repeat
  end if
  --The texts that are displayed don't have to be the same as the text that the text explodes into. If you define
pMyExplodingTextsMain
  --differently from pMyTextsMain, make sure that there are the same number of texts in the list.
  pMyTotalNumberOfTextsMain=pMyTextsMain.count
  --The total Number of texts in pMyTextsMain and in pMyExplodingTextsMain
  pMyTextWidthsMain=[]
  --This array holds the widths, in pixels, that the text object should be when it contains a given text from pMyTextsMain.
  --Unfortunately Director does not adjust this correctly automatically.
  repeat with i=1 to pMyTotalNumberOfTextsMain
    tempLength=pMyTextsMain[i].length
    firstApprox=max([16, 6.2*tempLength+8])
    --No text can be shorter than 16 characters, and the formula there is a rough approximation to the right length of any
    --string, in pixels. It isn't bang on, but it isn't too bad for a rude guess.
    secondApprox= integer(min([300, firstApprox]))
    --Don't let any text be longer than 300 pixels. Just because otherwise you get very long texts that take up a whole part
    --of the screen
    pMyTextWidthsMain.append(secondApprox)
  end repeat
  theLineheight=member("SingleCircle1").charPosToLoc(1).locV
  repeat with i= 1 to pMyTotalNumberOfTextsMain
    member("SingleCircle1").text=pMyTextsMain[i]
    member("SingleCircle1").width=pMyTextWidthsMain[i]
    lastChar=pMyTextsMain[i].length
    repeat while member("SingleCircle1").width <300 and theLineHeight < member("SingleCircle1").charPosToLoc(lastChar).locV
      member("SingleCircle1").width=member("SingleCircle1").width+2
    end repeat
    pMyTextWidthsMain[i]=member("SingleCircle1").width
  end repeat
end beginsprite


--**********************************************************************************************************
--SINGLE CIRCLE GREEN ARTEROID LEVEL 2

--This behavior gets attached to green sprites in level 2 that are Director text sprites to be exploded.
--If the sprite that's the Enemy Word Manager is of this type then this script must be attached AFTER the
--Enemy Word Manager script, because there are some variables used in this script that rely on values from the other script.
--**********************************************************************************************************


property spritenum, pStringToExplode, pCurrentText, pStringLength, pLetterChannels, pNumberOfLetterChannels, pIAmExploding,
pCurrentlyInPlay
property pIterationOfExplosion,pSpeedFactor, pRadiusOfExplosion
property pBlendIncrement, pInitialExplosionBlend, pTotalIterations, pInitialExplosionFontSize, pFontSizeIncrement,
pRotationFactor, pMyTypeOfSprite
property pCurrentExplosionBlend, pCurrentExplosionFontSize, pCurrentRotation, pAnglePart
property pMyRelativeFrequency, pMyOrdering, pMyTexts, pMyTotalNumberOfTexts, pMyTextWidths, pMyExplodingTexts,
pMyCurrentExplodingText, pMyCurrentTextNum, pMyPointsAwarded
global gGreenLetterManager, gEnemyWordManager, gGreenTextAdjuster

on beginsprite me
 pMyTypeOfSprite="#SingleCircleGreen"
 --This variable defines the type of Asteroid we're dealing with here. It's used in the
 sprite(spritenum).member.centerRegPoint=TRUE
 --This makes the explosions be centered at the center of the text to explode.
 pMyRelativeFrequency=100
 --This  is used in the Enemy Letter Manager to determine the relative frequency with which this type of sprite is selected
 --to be onstage, given other types to choose among. Though the program will function correctly if there is only one type.
 pMyOrdering="sequential"
 --Possible values are "sequential" and "random". If the type is "sequential" then it will display the texts below sequentially
 --as they are destroyed and reappear.
 pMyTexts=sprite(gGreenTextAdjuster).pMyTextsMain
 --This defines the texts that will appear in the asteroid.
 pMyExplodingTexts=sprite(gGreenTextAdjuster).pMyExplodingTextsMain
 --The texts that are displayed don't have to be the same as the text that the text explodes into. If you define pMyExplodingTexts
 --differently from pMyTexts, make sure that there are the same number of texts in the list.
 pMyTotalNumberOfTexts=sprite(gGreenTextAdjuster).pMyTotalNumberOfTextsMain
 --The total Number of texts in pMyTexts and in pMyExplodingTexts
 pMyTextWidths=sprite(gGreenTextAdjuster).pMyTextWidthsMain
 --This array holds the widths, in pixels, that the text object should be when it contains a given text from pMyTexts.
 --Unfortunately Director does not adjust this correctly automatically.

 if pMyOrdering="sequential" then
  pMyCurrentTextNum=pMyTotalNumberOfTexts
  --make the first text be the last text...because in Enemy Word Manager it is then modded back to the first text.
  pCurrentText=pMyTexts[pMyCurrentTextNum]
  pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
 else
  --else it's a random selection among the texts
  pMyCurrentTextNum=random(pMyTotalNumberOfTexts)
  pCurrentText=pMyTexts[pMyCurrentTextNum]
  pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
 end if
 sprite(spritenum).member.text=pCurrentText

 pMyPointsAwarded=10
 --This is the number of points awarded to the player when s/he blasts this type of text
 pStringToExplode=""
 --This is initialized later.
 pStringLength=0
 --This is initialized later
 pLetterChannels=[]
 --When a text explodes, what happens is that the displayed letters in the explosion are actually drawn from a flash
 --animation cast member called 'alphabet'. This is because Flash animations are much more quickly manipulated
 --than is Director text. Note that in the score there are many sprites whose member is the 'alphabet' member.
 --The pLetterChannels list holds the spritenumbers of the 'alphabet' sprites used for each letter of the exploding text.
 --In other words, if the text to explode is 'fart' then pLetterChannels will have four values, one for each of the letters
 --in 'fart'.
 pCurrentlyInPlay=FALSE
 --A sprite has pCurrentlyInPlay=TRUE if it is onstage or has been blasted but the explosion is still happening.
 pIAmExploding=FALSE
 --This is true during the interval when the sprite is exploding and false otherwise.
 pIterationOfExplosion=0
 --You must define here the number of frames it takes for the the explosion to occur.
 pSpeedFactor=3+ (random(10))
 --This is a handy little number that you can change to speed up/slow down sprites.
```

```
  --This is basically the number of pixels per frame that the thing moves by.

  pInitialExplosionFontSize=50 + random(70)
  --The initial size (not actually the font size) of the letters (actually its the height and width of the 'alphabet' sprites.
  pRadiusOfExplosion=120
  --This defines the radius length of the explosion.
  pInitialExplosionBlend=80
  --This determines what the Blend (or alpha) values is of the letters  when they first explode.
  pTotalIterations=15
  --You can see that geometrically this value makes sense. The radius of explosion is the total distance
  --that each letter traverses during the explosion, and the speed factor is the number of pixels the letter
  --moves on each iteration of the explosion, so the total number of iterations is the one divided by the other.
  pBlendIncrement= 3
  --This is the inrement by which the blend is decreases each iteration. Change this to change the final
  --blend of the letters after the explosion is finished, so you do get some text corpses onstage after the explosion.
  pFontSizeIncrement=integer(((pInitialExplosionFontSize-18.0)/pTotalIterations)-0.5)
  --The increment by which the 'alphabet' sprite is changed each iteration.
  pRotationFactor=30
  --The number of degrees by which the exploding letter is rotated each iteration.
  pCurrentExplosionBlend=pInitialExplosionBlend
  pCurrentExplosionFontSize=pInitialExplosionFontSize
  pCurrentRotation=0
  sprite(gEnemyWordManager).addMeAndMyType(spritenum, pMyTypeOfSprite, pMyRelativeFrequency, pMyOrdering, pMyTexts,
pMyTotalNumberOfTexts, pMyCurrentTextNum, pMyPointsAwarded)
  --so the Enemy Word Manager must be attached BEFORE this script to the sprite that is both the Enemy Word Manager and the
Word To Explode 1
end beginsprite


on explode me
  sprite(gEnemyWordManager).deleteMeFromCurrentlyInPlayAndNotExploding(spritenum)
  sprite(spritenum).visible=FALSE
  pStringToExplode=sprite(spritenum).pMyCurrentExplodingText
  pStringLength=pStringToExplode.length
  pLetterChannels=sprite(gGreenLetterManager).getChannels(pStringLength, spritenum, pStringToExplode)
  pNumberOfLetterChannels=pLetterChannels.count
  if pNumberOfLetterChannels >0 then
    pRadiusOfExplosion=20+pStringLength*25
    pAnglePart=6.2832/pNumberOfLetterChannels
    repeat with i = 1 to pNumberOfLetterChannels
      tempNum=charToNum(chars(pMyCurrentExplodingText, i,i))
      sprite(pLetterChannels[i]).frame = tempNum
      sprite(pLetterChannels[i]).ink=36
    end repeat
    sprite(gEnemyWordManager).addMeToExplodingList(spritenum)
  else
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
end explode me


on executeMyGeometry me
  --this can change depending on the geometry
  if pIterationOfExplosion < pTotalIterations then
    pCurrentExplosionBlend=pCurrentExplosionBlend-pBlendIncrement
    pCurrentExplosionFontSize=pCurrentExplosionFontSize-pFontSizeIncrement
    pCurrentRotation=(pCurrentRotation+pRotationFactor) mod 360
    theRadius=pRadiusOfExplosion*pIterationOfExplosion.float/pTotalIterations
    repeat with i= 1 to pNumberOfLetterChannels
      tempi=i-1
      sprite(pLetterChannels[i]).blend=pCurrentExplosionBlend
      sprite(pLetterChannels[i]).width=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).height=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).rotation=pCurrentRotation
      theAngle=tempi*pAnglePart
      Hextension=theRadius*cos(theAngle)
      Vextension=theRadius*sin(theAngle)
      sprite(pLetterChannels[i]).locH= sprite(spritenum).locH + Hextension
      sprite(pLetterChannels[i]).locV= sprite(spritenum).locV + Vextension
    end repeat
    pIterationOfExplosion=pIterationOfExplosion+1
  else
    pCurrentExplosionBlend=pInitialExplosionBlend
    pCurrentExplosionFontSize=pInitialExplosionFontSize
    pCurrentRotation=0
    sprite(gGreenLetterManager).returnChannels(pLetterChannels)
    pLetterChannels=[]
    pIterationOfExplosion=0
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
end executeMyGeometry
```

Blue Texts

```
--************************************************************************************************************
--SINGLE CIRCLE BLUE ARTEROID LEVEL 2

--This behavior gets attached to sprites that are text sprites to be exploded, and explode in a single circle.
--If the sprite that's the Enemy Word Manager is of this type
--then this script must be attached AFTER the Enemy Word Manager script, because there are some variables used in this script that rely
--on values from the other script.
--************************************************************************************************************


property spritenum, pStringToExplode, pCurrentText, pStringLength, pLetterChannels, pNumberOfLetterChannels, pIAmExploding, pCurrentlyInPlay
property pIterationOfExplosion,pSpeedFactor, pRadiusOfExplosion
property pBlendIncrement, pInitialExplosionBlend, pTotalIterations, pInitialExplosionFontSize, pFontSizeIncrement, pRotationFactor, pMyTypeOfSprite
property pCurrentExplosionBlend, pCurrentExplosionFontSize, pCurrentRotation, pAnglePart
property pMyRelativeFrequency, pMyOrdering, pMyTexts, pMyTotalNumberOfTexts, pMyTextWidths, pMyExplodingTexts, pMyCurrentExplodingText, pMyCurrentTextNum, pMyPointsAwarded
global gBlueLetterManager, gEnemyWordManager, gBlueTextAdjuster

on beginsprite me
  pMyTypeOfSprite="#SingleCircleBlue"
  --This variable defines the type of Asteroid we're dealing with here. It's used in the
  sprite(spritenum).member.centerRegPoint=TRUE
  --This makes the explosions be centered at the center of the text to explode.
  pMyRelativeFrequency=100
  --This  is used in the Enemy Letter Manager to determine the relative frequency with which this type of sprite is selected
  --to be onstage, given other types to choose among. Though the program will function correctly if there is only one type.
  pMyOrdering="sequential"
  --Possible values are "sequential" and "random". If the type is "sequential" then it will display the texts below sequentially
  --as they are destroyed and reappear.
  pMyTexts=sprite(gBlueTextAdjuster).pMyTextsMain
  --This defines the texts that will appear in the asteroid.
  pMyExplodingTexts=sprite(gBlueTextAdjuster).pMyExplodingTextsMain
  --The texts that are displayed don't have to be the same as the text that the text explodes into. If you define pMyExplodingTexts
  --differently from pMyTexts, make sure that there are the same number of texts in the list.
```

```
pMyTotalNumberOfTexts=sprite(gBlueTextAdjuster).pMyTotalNumberOfTextsMain
--The total Number of texts in pMyTexts and in pMyExplodingTexts
pMyTextWidths=sprite(gBlueTextAdjuster).pMyTextWidthsMain
--This array holds the widths, in pixels, that the text object should be when it contains a given text from pMyTexts.
--Unfortunately Director does not adjust this correctly automatically.

if pMyOrdering="sequential" then
  pMyCurrentTextNum=pMyTotalNumberOfTexts
  --make the first text be the last text...because in Enemy Word Manager it is then modded back to the first text.
  pCurrentText=pMyTexts[pMyCurrentTextNum]
  pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
else
  --else it's a random selection among the texts
  pMyCurrentTextNum=random(pMyTotalNumberOfTexts)
  pCurrentText=pMyTexts[pMyCurrentTextNum]
  pMyCurrentExplodingText=pMyExplodingTexts[pMyCurrentTextNum]
end if
sprite(spritenum).member.text=pCurrentText
pMyPointsAwarded=10
--This is the number of points awarded to the player when s/he blasts this type of text
pStringToExplode=""
--This is initialized later.
pStringLength=0
--This is initialized later
pLetterChannels=[]
--When a text explodes, what happens is that the displayed letters in the explosion are actually drawn from a flash
--animation cast member called 'alphabet'. This is because Flash animations are much more quickly manipulated
--than is Director text. Note that in the score there are many sprites whose member is the 'alphabet' member.
--The pLetterChannels list holds the spritenumbers of the 'alphabet' sprites used for each letter of the exploding text.
--In other words, if the text to explode is 'fart' then pLetterChannels will have four values, one for each of the letters
--in 'fart'.

pCurrentlyInPlay=FALSE
--A sprite has pCurrentlyInPlay=TRUE if it is onstage or has been blasted but the explosion is still happening.
pIAmExploding=FALSE
--This is true during the interval when the sprite is exploding and false otherwise.
pIterationOfExplosion=0
--You must define here the number of frames it takes for the the explosion to occur.
pSpeedFactor=3+ (random(10))
--This is a handy little number that you can change to speed up/slow down sprites.
--This is basically the number of pixels per frame that the thing moves by.
pInitialExplosionFontSize=30 + random(40)
--The initial size (not actually the font size) of the letters (actually its the height and width of the 'alphabet' sprites.
pRadiusOfExplosion=120
--This defines the radius length of the explosion.
pInitialExplosionBlend=80
--This determines what the Blend (or alpha) values is of the letters  when they first explode.
pTotalIterations=12
--You can see that geometrically this value makes sense. The radius of explosion is the total distance
--that each letter traverses during the explosion, and the speed factor is the number of pixels the letter
--moves on each iteration of the explosion, so the total number of iterations is the one divided by the other.
pBlendIncrement= 2
--This is the inrement by which the blend is decreases each iteration. Change this to change the final
--blend of the letters after the explosion is finished, so you do get some text corpses onstage after the explosion.
pFontSizeIncrement=integer(((pInitialExplosionFontSize-18.0)/pTotalIterations)-0.5)
--The increment by which the 'alphabet' sprite is changed each iteration.
pRotationFactor=30
--The number of degrees by which the exploding letter is rotated each iteration.
pCurrentExplosionBlend=pInitialExplosionBlend
pCurrentExplosionFontSize=pInitialExplosionFontSize
pCurrentRotation=0
sprite(gEnemyWordManager).addMeAndMyType(spritenum, pMyTypeOfSprite, pMyRelativeFrequency, pMyOrdering, pMyTexts,
pMyTotalNumberOfTexts, pMyCurrentTextNum, pMyPointsAwarded)
  --so the Enemy Word Manager must be attached BEFORE this script to the sprite that is both the Enemy Word Manager and the
Word To Explode 1
end beginsprite


on explode me
  sprite(gEnemyWordManager).deleteMeFromCurrentlyInPlayAndNotExploding(spritenum)
  sprite(spritenum).visible=FALSE
  pStringToExplode=sprite(spritenum).pMyCurrentExplodingText
  pStringLength=pStringToExplode.length
  pLetterChannels=sprite(gBlueLetterManager).getChannels(pStringLength, spritenum, pStringToExplode)
  pNumberOfLetterChannels=pLetterChannels.count
  if pNumberOfLetterChannels >0 then
    pRadiusOfExplosion=20+pStringLength*25
    pAnglePart=6.2832/pNumberOfLetterChannels
    repeat with i = 1 to pNumberOfLetterChannels
      tempNum=charToNum(chars(pMyCurrentExplodingText, i,i))
      sprite(pLetterChannels[i]).frame = tempNum
      sprite(pLetterChannels[i]).ink=36
    end repeat
    sprite(gEnemyWordManager).addMeToExplodingList(spritenum)
  else
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
end explode me


on executeMyGeometry me
  --this can change depending on the geometry
  if pIterationOfExplosion < pTotalIterations then
    pCurrentExplosionBlend=pCurrentExplosionBlend-pBlendIncrement
    pCurrentExplosionFontSize=pCurrentExplosionFontSize-pFontSizeIncrement
    pCurrentRotation=(pCurrentRotation+pRotationFactor) mod 360
    theRadius=pRadiusOfExplosion*pIterationOfExplosion.float/pTotalIterations
    repeat with i= 1 to pNumberOfLetterChannels
      tempi=i-1
      sprite(pLetterChannels[i]).blend=pCurrentExplosionBlend
      sprite(pLetterChannels[i]).width=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).height=pCurrentExplosionFontSize
      sprite(pLetterChannels[i]).rotation=pCurrentRotation
      theAngle=tempi*pAnglePart
      Hextension=theRadius*cos(theAngle)
      Vextension=theRadius*sin(theAngle)
      sprite(pLetterChannels[i]).locH= sprite(spritenum).locH + Hextension
      sprite(pLetterChannels[i]).locV= sprite(spritenum).locV + Vextension
    end repeat
    pIterationOfExplosion=pIterationOfExplosion+1
  else
    pCurrentExplosionBlend=pInitialExplosionBlend
    pCurrentExplosionFontSize=pInitialExplosionFontSize
    pCurrentRotation=0
    sprite(gBlueLetterManager).returnChannels(pLetterChannels)
    pLetterChannels=[]
    pIterationOfExplosion=0
    sprite(gEnemyWordManager).iveBeenHitByThePlayer(spritenum, sprite(spritenum).pMyTypeOfSprite)
  end if
  --put pIterationOfExplosion
end executeMyGeometr
```