Ok, time to wrap things up. So far we’ve covered sprites, collisions and controls. This month we’ll tidy things up a bit by adding a start screen, scoring, lives, a boss and an ending.

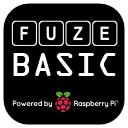
However before we get started I’d like to make a suggestion. At FUZE we’re busy working on a new version of the editor. You can download this straight to your Raspbian desktop right away. Visit <http://www.fuze.co.uk/resources-2> then GET FUZE BASIC for details on downloading.

The work in progress version has a more standard editor and as such is easier to save, load and edit programs, however if you prefer to stick to the original then you can run FUZE BASIC from the Run Menu.

First off then you will need to load a few new sprites so please download the following from the Resource page above then within the Tutorials Tab.

You need to download and save them to the MagPi folder where your program is saved;

“themagpi.bmp”  
“fblogo.bmp”  
“blackberry.bmp”

I’m afraid you have a lot of work on your hands this month as this is the final and finished program. The problem is that so much has changed it really needs typing in again from the beginning. Also it would take too much space to add everything individually so we are going to go through the whole thing section by section. The downside is that this means the game won’t work until we’ve finished entering the complete listing. Welcome to the eighties folks!

Once again please bring up the FUZE BASIC environment using the icon on the Desktop. If you’re in the older version go to the Editor with **F2** and start with;

|  |
| --- |
| **The Editor – The Main LOOP** |
| **// MagPi Game**  **PROC SetupMain**  **CYCLE**  **PROC SetupGame**  **PROC attract**  **WHILE lives > 0 CYCLE**  **PROC CheckControls**  **PROC DrawShip**  **PROC DrawEnemy**  **PROC DrawBullet**  **PROC BOSS**  **IF WIN then PROC wellDone**  **PROC updateScreen**  **UPDATE**  **REPEAT**  **PROC gameOver**  **REPEAT**  **END** |

Remember, you’ll only get errors if you try and RUN this so just be patient for now and the rewards will be great later!

Firstly we call the main setup routine to load all the sprites and variables. These items are required each time the game is executed for the first time but not when the game restarts within the program. This way we can keep a High Score, have lives and levels.  
  
Next is the main loop followed by SetupGame and attract procedures. The first resets the main variables to start each new game while attract displays the start-up screen (known as attract mode in the old days).

Next we check to see if we have any lives left and then check for keys pressed, draw everything and then check to see if it is ‘Boss time’ yet. All going well, and as long as there are lives left then it will REPEAT back to WHILE lives>0. If we’re out of lives then the gameOver procedure is called and we can start again.

Add the following after the END statement;

|  |
| --- |
| **The Editor – Screen Updates** |
| **DEF PROC updateScreen**  **CLS2**  **INK = Yellow**  **fontScale(3,3)**  **hvTab (0, 0)**  **PRINT "Score "; score**  **INK=RED**  **hvTab (tWidth/2-7,0)**  **PRINT "Hi Score "; hiScore**  **INK=YELLOW**  **hvTab (tWidth-7,0)**  **PRINT "Level "; Level**  **FOR num = 0 TO lives - 1 CYCLE**  **plotImage (shipPic, 0 + getImageW (shipPic) \* num, -10)**  **REPEAT**  **COLOUR = Silver**  **RECT (0, 62, gWidth, 4, 1)**  **ENDPROC** |

This procedure prepares and displays various pieces of information like the player’s score, the hi-score and the number of lives remaining.  
  
CLS2, as previously covered, clears the buffer screen. fontScale(3,3) sets the current font size to 3 times normal size, both horizontally and vertically.  
  
hvTab positions the text cursor at (x,y).

The FOR loop draws the number of lives as player ships along the bottom and the RECT command draws a thin silver rectangle along the bottom of the screen.

|  |
| --- |
| **The Editor - The BOSS!** |
| **DEF PROC BOSS**  **IF Enemy(EnemyMax,1)<=0 OR Enemy(EnemyMax,3)=0 THEN Warning=1**  **IF Warning Then**  **WarningCount=WarningCount+1**  **IF WarningCount<=150 THEN**  **INK=RND(15)+1**  **fontScale(3,5)**  **hvTab (tWidth/2 - LEN (Warning$) / 2, tHeight/4)\*3**  **PRINT Warning$**  **UPDATE**  **ELSE**  **Warning=0**  **bossActive=1**  **ENDIF**  **ENDIF**  **IF bossActive THEN**  **bossX=bossX-2\*Level**  **IF bossX< -getSpriteW(boss) then PROC gameOver**  **bossAng=bossAng+3**  **bossX=bossX+bossXX\*COS(bossAng)**  **bossY=bossY+bossYY\*SIN(-bossAng)**  **plotSprite(boss,bossX,bossY,0)**  **IF bossAng>=360 then bossAng=0**  **ENDIF**  **ENDPROC** |

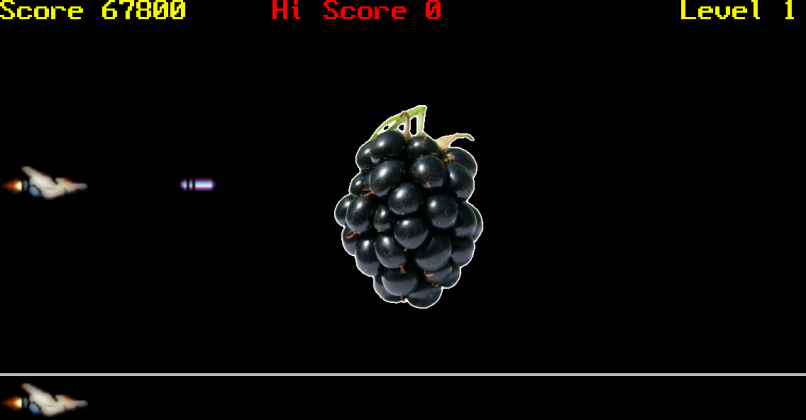
From here-on-in just add each new section to the end of the existing program.

The boss springs into action only if the Warning has been activated and this is only activated if the very last enemy has either left the screen to the left or has been destroyed.

The “Warning” is then displayed, and when finished, the boss itself appears.

The boss follows a simple circular pattern but its speed across the screen increases due to the statement; bossX = bossX - 2 \* Level.

Once again don’t bother trying to **RUN** this yet as it just doesn’t have enough to go on but here’s a preview to whet your appetite.



|  |
| --- |
| **The Editor – Bullet Time…** |
| **DEF PROC DrawBullet**  **IF Shot(1) > gWidth THEN**  **hideSprite (Shot(0))**  **Shot(3) = 0**  **Fire = 0**  **ENDIF**  **IF Shot(3) THEN**  **Shot(1) = Shot(1) + 8**  **plotSprite (Shot(0), Shot(1), Shot(2), 0)**  **Hit = spriteCollidePP (Shot(0), 2)**  **IF Hit > 0 AND Hit <= 64 THEN**  **score = score + Enemy(Hit - 1, 5) \* Level**  **EnemyCount=EnemyCount+1**  **Enemy(Hit - 1, 3) = 0**  **hideSprite (Hit)**  **hideSprite (Shot(0))**  **Shot(3) = 0**  **Fire = 0**  **ENDIF**  **IF Hit=68 THEN**  **score=score+500**  **bossHit=bossHit-1**  **hideSprite (Hit)**  **hideSprite (Shot(0))**  **Shot(3) = 0**  **Fire = 0**  **IF bossHit<=0 then**  **WIN=1**  **bossActive=0**  **Level=Level+1**  **PROC SetupGame**  **ENDIF**  **ENDIF**  **ENDIF**  **ENDPROC** |

You are only allowed a single bullet on screen at a time. First we check if it is still on screen and if not hide it then reset the Fire variable so another can be shot.

The bullet moves along at eight pixels at a time and is checked to see if it has collided with anything.



If it hits a rock then the rock is removed, the score is increased and the bullet removed so another can be fired.

If it comes into contact with the boss then the boss hit count decreases. If the bossHit counter gets to zero then the game starts again with an increased level and the score carried over. The WIN variable, later on, activates the “Congratulations” screen.

|  |
| --- |
| **The Editor – Draw sprites** |
| **DEF PROC Bullet**  **Fire = 1**  **Shot(1) = ShipX + getSpriteW (Ship) + 8**  **Shot(2) = ShipY + getSpriteH (Ship) / 2 - 10**  **Shot(3) = 1**  **ENDPROC**  **DEF PROC DrawEnemy**  **FOR eID = 0 TO EnemyMax CYCLE**  **IF Enemy(eID, 3) THEN**  **Enemy(eID, 1) = Enemy(eID, 1) - Enemy(eID, 6)**  **EY = Enemy(eID, 2) + COS (Enemy(eID, 1)) \* Enemy(eID, 4) \* 10**  **IF Enemy(eID, 1) > -getSpriteW(Rock(eID))\*2 AND Enemy(eID,1)<= gWidth THEN plotSprite (Enemy(eID, 0), Enemy(eID, 1), EY, 0)**  **IF Enemy(eID,1) <= -GetSpriteW(Rock(eID))\*2 THEN**  **hidesprite(Rock(eID))**  **Enemy(eID,3)=0**  **ENDIF**  **ENDIF**  **REPEAT**  **ENDPROC**  **DEF PROC DrawShip**  **plotSprite (Ship, ShipX, ShipY, ShipID)**  **Crash = spriteCollidePP (Ship, 2)**  **IF Crash > 0 AND Crash <= 64 THEN**  **lives = lives - 1**  **Enemy(Crash - 1, 3) = 0**  **hideSprite (Crash)**  **ShipX = 0**  **ShipY = gHeight / 2**  **ENDIF**  **IF Crash=68 then**  **lives=lives-1**  **ShipX = 0**  **ShipY = gHeight / 2**  **ENDIF**  **ENDPROC** |

These three procedures are responsible for working out bullet positions, checking various collisions and plotting rocks and the player’s ship to the screen.

Note; the highlighted line is one single line. What is does is figure out if a rock is visible on the screen by checking if it’s too far left or too far to the right. It only plots the sprite if it is in the viewable area.

The DrawShip procedure plots the player’s ship and checks to see if it has collided with either the rocks (sprite IDs 0 to 64) or the boss (sprite ID 68)

If it crashes into a rock the player’s ship is returned to its original position, lives are reduced by one and any rocks we may have crashed into are removed with the hideSprite command.



|  |
| --- |
| **The Editor – Player controls** |
| **DEF PROC CheckControls**  **ShipID = 1**  **UpKey = scanKeyboard (scanUp)**  **DownKey = scanKeyboard (scanDown)**  **LeftKey = scanKeyboard (scanLeft)**  **RightKey = scanKeyboard (scanRight)**  **SpaceKey = scanKeyboard (scanSpace)**  **IF SpaceKey AND NOT Fire THEN PROC Bullet**  **IF UpKey AND ShipY <= gHeight - 100 THEN**  **ShipY = ShipY + 8**  **ShipID = 2**  **ENDIF**  **IF DownKey AND ShipY >= 64 THEN**  **ShipY = ShipY - 8**  **ShipID = 0**  **ENDIF**  **IF LeftKey AND ShipX >= 0 THEN ShipX = ShipX - 8**  **IF RightKey AND ShipX <= gWidth / 2 THEN ShipX = ShipX + 4**  **ENDPROC** |

The Player controls procedure is very straightforward and we’ve covered most of it before.

Various keys are checked to see if they are being pressed and actions take place accordingly.

Notice the RightKey will only work if the player is not too far forward and it moves at half the number of pixels (4) than all the other directions (8). Moving slowly forward gives an extra sense of realism.

|  |
| --- |
| **The Editor – Add after END statement** |
| **DEF PROC killEverything**  **FOR num = 0 TO EnemyMax CYCLE**  **hideSprite (Rock(num))**  **REPEAT**  **hideSprite (Shot(0))**  **hideSprite (Ship)**  **IF bossActive THEN hideSprite(boss)**  **bossActive=0**  **CLS2**  **ENDPROC**  **DEF PROC wellDone**  **WAIT(1)**  **PROC killEverything**  **FOR delay = 0 to 300 CYCLE**  **INK=RND(15)+1**  **fontScale(5,5)**  **hvTab (tWidth/2 - LEN (Congrats$) / 2, tHeight / 2)**  **PRINT Congrats$**  **UPDATE**  **REPEAT**  **WIN=0**  **ENDPROC**  **DEF PROC gameOver**  **PROC killEverything**  **text$ = "GAME OVER"**  **fontScale (4, 4)**  **for num=0 to 100 CYCLE**  **hvTab (tWidth / 2 - LEN (text$) / 2, tHeight / 2)**  **INK = RND (15 + 1)**  **PRINT text$**  **UPDATE**  **REPEAT**  **IF score>hiScore THEN hiScore=score**  **clearKeyboard**  **WIN=0**  **lives=0**  **ENDPROC** |

Here’s a few smaller ones.  
  
killEverything is used to reset and hide all the active sprites. It is called before anything major happens like killing the boss and losing all your lives.

wellDone is only called when you have successfully killed the boss. It simply displays “Congratulations” in the middle of the screen.

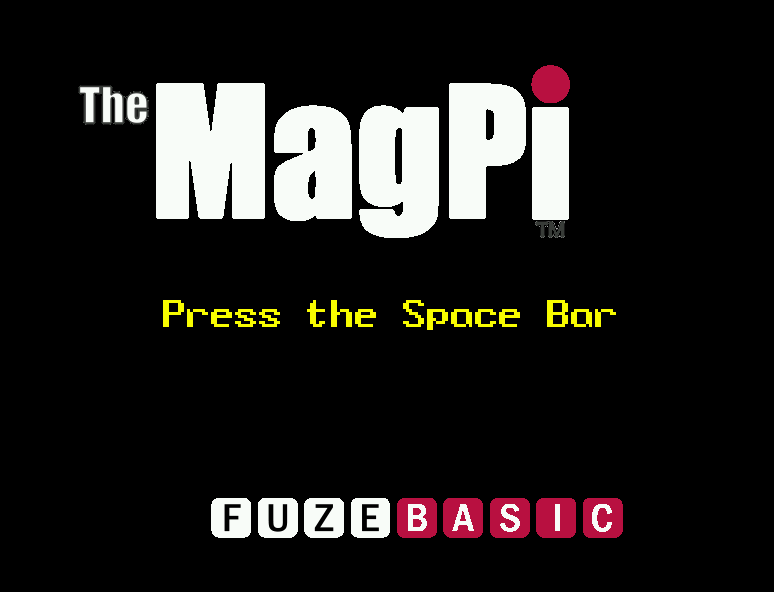


gameOver displays “GAME OVER” in the middle of the screen and resets a few ‘trigger’ variables. For example ‘lives’ is set to zero which is what controls the main program loop.

|  |
| --- |
| **The Editor – The Attract screen** |
| **DEF PROC attract**  **cls**  **t1X = gWidth / 2 - getSpriteW (title1) / 2**  **t2X = gWidth / 2 - getSpriteW (title2) / 2**  **t1Y = gHeight - getSpriteH (title1) \* 1.5**  **t2Y = getSpriteH (title2)**  **angle = 0**  **WHILE NOT scankeyboard(scanSpace) CYCLE**  **t1XX = t1X + 250 \* COS (angle)**  **t1YY = t1Y + 40 \* SIN (angle)**  **t2XX = t2X - 80 \* COS ( - angle)**  **t2YY = t2Y - 10 \* SIN ( - angle)**  **plotSprite (title1, t1XX, t1YY, 0)**  **plotSprite (title2, t2XX, t2YY, 0)**  **INK = RND (15 + 1)**  **fontScale (3, 3)**  **hvTab (tWidth / 2 - LEN (Press$) / 2, tHeight / 2)**  **PRINT Press$**  **UPDATE**  **angle = angle + 2**  **REPEAT**  **hideSprite (title1)**  **hideSprite (title2)**  **WAIT (.5)**  **CLS**  **clearKeyboard**  **ENDPROC** |

Every half decent game must have an attract screen. It just provides something to look at when the game isn’t running.

Lots of cosine and sine calculations here. The angle variable is increased to provide 360 degree movement. The X and Y positions of both images are calculated by taking a point of origin (t1x) then adding and multiplying a radius (250) by the cosine of the current angle.



The point of origin (t1X) never changes as it’s only the radius that is operated on.

This is repeated on t1Y, t2X and t2Y and then the two images are drawn as sprites.

“Press the Space Bar” is displayed in the middle of the screen using hvTab and PRINT Press$.

The whole thing is enclosed in a WHILE LOOP that checks for the Space Bar. Once the Space is pressed the loop ends, the sprites removed and the procedure returns.

|  |
| --- |
| **The Editor – Setup to execute once per session** |
| **DEF PROC SetupMain**  **HGR**  **hiScore=0**  **WIN=0**  **updateMode = 0**  **Warning$="Warning.. Huge Fruit Approaches!"**  **Press$="Press the Space Bar"**  **Congrats$="CONGRATULATIONS!"**  **Ship = newSprite (3)**  **loadSprite ("Player1.bmp", Ship, 0)**  **loadSprite ("Player2.bmp", Ship, 1)**  **loadSprite ("Player3.bmp", Ship, 2)**  **setSpriteTrans (Ship, 255, 0, 255)**  **EnemyMax = 63**  **DIM Enemy(EnemyMax, 6)**  **DIM Rock(EnemyMax)**  **FOR num = 0 TO EnemyMax CYCLE**  **Rock(num) = newSprite (1)**  **loadSprite ("BigRock.bmp", Rock(num), 0)**  **setSpriteTrans (Rock(num), 255, 0, 255)**  **REPEAT**  **DIM Shot(3)**  **Shot(0) = newSprite (1)**  **loadSprite ("Bullet.bmp", Shot(0), 0)**  **setSpriteTrans (Shot(0), 255, 0, 255)**  **title1 = newSprite (1)**  **loadSprite ("themagpi.bmp", title1, 0)**  **setSpriteTrans (title1, 255, 0, 255)**  **title2 = newSprite (1)**  **loadSprite ("fblogo.bmp", title2, 0)**  **setSpriteTrans (title2, 255, 0, 255)**  **shipPic = loadImage ("ship.bmp")**  **boss=newSprite (1)**  **loadSprite ("blackberry.bmp",boss,0)**  **setSpriteTrans (boss,255,0,255)**  **ENDPROC** |

There are two setup procedures. The first sets everything up that is required the very first time you run the game so things like text messages, all of the sprite graphics and major variables.

I think we have covered this before but just in case;

The ‘newSprite(#)’ command sets up a variable to store a sprite container ID. The # signifies how many sprites go into the container. For example;

Ship=newSprite(3) defines the sprite ‘Ship’ and gives it three slots for graphics. This is then referenced by plotSprite(Ship, X, Y, #) and is how we change the Ship graphic depending on if it’s moving up, down or just staying still.

loadSprite loads a sprite graphic into a container slot specified by the number at the end.

setSpriteTrans forces one colour to be transparent meaning it will never be displayed. This is important as even black would be displayed over a white background. I generally use bright pink as it is not often used for anything else.

The great thing about simple sprite controls like this is that it’s so easy to change the graphics by just changing a file name. You should probably go for similar sizes though and remember to set the transparent colour.

|  |
| --- |
| **The Editor – Setup to execute each time a game is played** |
| **DEF PROC SetupGame**  **IF NOT WIN THEN**  **lives=3**  **Level=1**  **score=0**  **ELSE**  **lives=lives+1**  **score=score+10000**  **ENDIF**  **ShipX = 0**  **ShipY = gHeight / 2**  **ShipID = 0**  **bossX=gWidth+getSpriteW(boss)**  **bossXX=10**  **bossY=gHeight/2+getSpriteH(boss)/2**  **bossYY=10**  **bossAng=0**  **bossHit=10**  **WIN=0**  **eID = 0**  **EnemyID = 0**  **EnemyX = 0**  **EnemyY = 0**  **EnemyActive = 1**  **EnemyVariation = 0**  **EnemyScore = 0**  **EnemySpeed = 0**  **EnemyCount = 0**  **RESTORE**  **UNTIL EnemyCount > EnemyMax CYCLE**  **READ EnemyX**  **READ EnemyY**  **READ EnemyVariation**  **READ EnemyScore**  **READ EnemySpeed**  **EnemyScore = EnemyScore \* EnemySpeed**  **DATA 1280, 100, 3, 50, 3**  **DATA 1280, 500, -3, 50, 3**  **DATA 4000, 366, 4, 50, 4**  **DATA 4000, 230, -4, 50, 4**  **DATA 7500, 100, 6, 50, 5**  **DATA 7500, 500, -6, 50, 5**  **DATA 11500, 400, 5, 50, 6**  **DATA 11500, 300, -5, 50, 6**  **FOR num = 0 TO 7 CYCLE**  **Enemy(EnemyCount + num, 0) = Rock(EnemyCount + num)**  **Enemy(EnemyCount + num, 1) = EnemyX + num \* getSpriteW (Rock(0))**  **Enemy(EnemyCount + num, 2) = EnemyY**  **Enemy(EnemyCount + num, 3) = EnemyActive**  **Enemy(EnemyCount + num, 4) = EnemyVariation**  **Enemy(EnemyCount + num, 5) = EnemyScore**  **Enemy(EnemyCount + num, 6) = EnemySpeed\*Level**  **REPEAT**  **EnemyCount = EnemyCount + 8**  **REPEAT**  **Fire = 0**  **bossActive=0**  **Warning=0**  **WarningCount=0**  **ENDPROC** |

Ok, this is it… the final furlong, the last mile, the big, oh just get on with it!

The last procedure is the game setup. This configures all the variables as required each time a level is played.

The first part checks to see if it is a new game or a new level.

Everything thereafter configures game variables to start a new level.

The DATA section resets all the enemy rocks to their default position.

The boss is reset and then the procedure returns to the main loop.

Once you have typed all of the listing in, or copied it if you were being lazy… you can attempt to **RUN** it with **F3**.

Now, the chances of this working perfectly first time are slim. Errors usually creep in so expect some debugging.

Generally errors are typing mistakes, misspellings and or wrong capitalisation. You have to be very thorough.

Once you get it up and running, play the game. I’ve managed to get to the level five boss, can you?

The great thing about a listing like this is that you get to experiment, which is exactly what your next steps should be. Most of the variable names make sense so you should be able to change most things.

However if you’re up to the challenge then why not add some sound, explosions, more animation and perhaps different enemies and bosses for later levels. *And* let’s not forget the PowerUp!

The MagPi game is a decent little game start but there’s plenty of scope to improve, change and experiment.

This just leaves me to say, it has been an absolute privilege to work on this project and I have thoroughly enjoyed doing it. I only hope I have not gone on too long as it would be great to be invited back for more. I hope you’ve enjoyed it too and at the very least gained a better understanding of BASIC and now appreciate just how great a language it can be.

My name is Jon Silvera, please feel free to contact me at [contact@fuze.co.uk](mailto:contact@fuze.co.uk); I’ll be happy to try and help if you have any questions and I’d also love to see any projects you develop with FUZE BASIC. Happy coding folks, bye for now, teamFUZE.