F. THE GAME

Part1: Player Movement and Fire

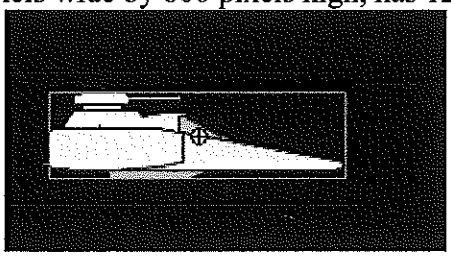
Introduction

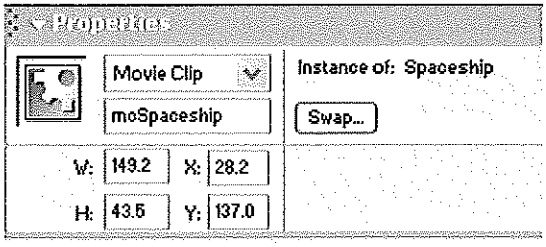
This part of the exercise covers controlling the movement of the player with the keyboard and firing weapons.

Step 1: Adding the Spaceship to the Movie

Create a new layer and rename it *Spaceship*. Click in frame 4 on this layer and insert a blank keyframe (F6). Draw your spaceship or import one you have created in another program. Select the

spaceship and then use **Convert to Symbol** (or F8) and make your spaceship a movie clip. Call the movie clip instance *mcSpaceship*. Place the spaceship on the left part of the stage. The stage is at 800 pixels wide by 600 pixels high, has 12 frames per second and a background colour of black..





Step 2: Moving the Spaceship

We need to detect which key or keys are being pressed using `Key.isDown` and then adjust the spaceship x-position and y-postion accordingly. The code will be carried out in **clip events**, like earlier exercises.

Explanation

A **clip event** is something that happens to a movie clip. Two of the most useful movie clip events are `load` and `enterFrame` which occur when a **movie clip** is loaded and when it enters a frame. The code looks like this

```
onClipEvent(load){

// code goes here
}
```

Any code contained between the curly brackets will be run when the movie clip is loaded or first appears on the stage. The `load` event is useful in games for initialising variables and defining `functions`, anything you want to do once at the start of a game - like setting the score to zero. The `enterFrame` event occurs every time the **movie clip** enters a new frame. Code you want to happen over and over again (eg detecting a collision) should be included in an `enterFrame` event.

Adding the Code

Add the code to the spaceship instance.

```
onClipEvent(load){
     moveSpeed=10;
}
onClipEvent (enterFrame){
     if (Key.isDown(Key.RIGHT)){
          this._x+=moveSpeed;
     }
     if (Key.isDown(Key.LEFT)){
          this._x-=moveSpeed;
     }
     if (Key.isDown(Key.DOWN)){
          this._y+=moveSpeed;
     }
```

```
    if(Key.isDown(Key.UP)){
        this._y-=moveSpeed;
    }
}
```

Explanation
This sets a variable called `moveSpeed` to 10 when the spaceship is first loaded. This variable will control the number of pixels the player moves. By changing this one number, it changes the speed in all directions.

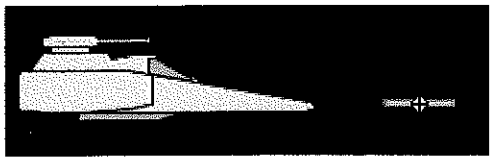
The arrow keys control screen movement of the spaceship. The code is within an `enterFrame` clip event, which means that every time the movie clip enters a new frame the code is run. Effectively the code is run over and over again in a loop unless we remove or stop the movie clip.

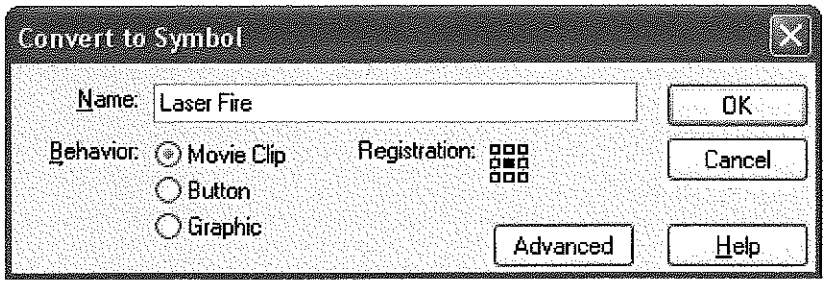
Copy of spaceship code is located at *j:\Mmedia34\Flash\Save the Earth\Part1 Step 2*

Step 3: Adding the Laser Beam

What we will do is create a **movie clip** that looks like spaceship laser fire. Whenever the *space bar* is pressed we will duplicate this **movie clip**, set its location to the same location as the spaceship and then, within a loop, increase the **movie clips** x-position until it goes off the screen. We increase the horizontal position so the laser fire moves from left to right – the direction the spaceship is flying.

Add a layer and call it *Laser Fire*. Insert a blank keyframe (F6) in frame 4. Draw some laser fire eg a couple of blue lines, a thin blue rectangle. **Group** the laser fire, then convert it to a **movie clip symbol** naming it *Laser Fire* and call the instance *mcLaser*.





Adding the code
Select the spaceship **movie clip** and modify the code for `onClipEvent(load){` by adding the two lines shown.

```
onClipEvent(load){
    moveSpeed=10;
    _root.laser._visible=false;
    laserCounter=1;
}
```

Explanation
The first of these new lines makes the laser invisible when the spaceship first loads. This will be the source laser clip. Whenever the user presses the space bar this **movie clip** will duplicate to create a new laser fire. This is the original **movie clip** and what gets fired across the screen every time the space bar is pressed is a 'duplicate'.

The second line is setting a counter variable equal to one. This is explained later.

Adding the code
Add the following code into the spaceship `clipEvent`. The following code goes directly after the line `clipEvent` line.

```
if (Key.isDown(Key.SPACE)) {
     laserCounter++;
     _root.mcLaser.duplicateMovieClip("Laser"+laserCounter,
     laserCounter);
    _root["Laser"+laserCounter]._visible=true;
}
```

One line of code

Explanation
This code is run when the space bar is pressed. The code does three things. It adds one to the `laserCounter` variable, it duplicates the *Laser Fire* **movie clip** and makes the new, duplicated clip, visible. The `laserCounter++;` adds one to the variable `laserCounter`. The ++ means 'increase by one', so the line is the same as `laserCounter=laserCounter+1`.

The second line duplicates the **movie clip** and gives this new duplicated **movie clip** the name 'Laser' plus the value of the laser counter variable and sets the **movie clips** depth to the value of the laser count variable. So if `_root.laser.count` is equal to 4 then the new **movie clip** will have the name `laser4` and the depth of 4.

The third line of code makes the new duplicated **movie clip** visible. It uses array style referencing for objects and is especially useful if you want to reference a dynamically created object name.

Now add this code to the Laser Fire **movie clip** instance.

```
onClipEvent (load) {
     laserMoveSpeed=20;
     this._y=_root.mcSpaceship._y+15;
     this._x=_root.mcSpaceship._x+85;
}
```

Explanation
The `onClipEvent(load)` code is associated with the *Laser Fire* **movie clip** so it will be run when the laser **movie clip** first appears on the stage. The code is also run whenever the laser movie clip is duplicated. When you duplicate a **movie clip** you duplicate the graphics, frames and all code associated with the **movie clip**. Each new duplicated **movie clip** has its own copy of the original **movie clips** code. And each duplicated **movie clip** runs its copy of the `onClipEvent(load)` code when its loaded or duplicated.

The first line sets a new variable called `laserMoveSpeed` equal to 20. This will be the number of pixels that the laser moves per frame ie the speed of the laser. The second line sets the y-position of the laser (or duplicated copy of the laser) to the same y-position as the spaceship the +15 is used to adjust the horizontal position of where the laser starts. The third line sets the x-position of the laser (or duplicated copy of the laser) to the same x-position as the spaceship plus 85. We add 85 because the x-position is for the centre of the spaceship and this make the laser appear at the front of my spaceship. On your spaceship you may wish to adjust this number. You may also wish to change the x- and y-positions using different numbers.

Adding the code
Now add this code under the code you just entered to the *Laser Fire* **movie clip**.

```
onClipEvent (enterFrame) {
     this._x+=laserMoveSpeed;
     if (this._x>800){
          this.removeMovieClip();
     }
}
```

Explanation
This code moves the laser movie clip (or duplicated copy of the laser) to the right of the screen. When the x-position gets greater than 800 it removes itself. The first line sets the x-position of the laser movie clip (or duplicated copy of the laser) to the current x-position plus `laserMoveSpeed`. As we set `laserMoveSpeed` to 20 this means that the x-position increases by 20 every frame.

The next lines are an `if` statement, which simple removes the laser movie clip if its x-position is greater than 800.

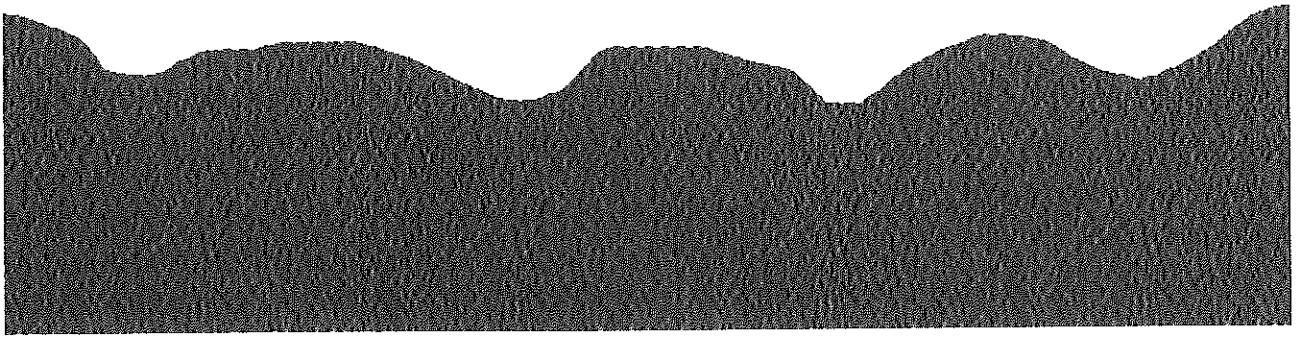
Copies of spaceship and laser code is located at *j:\Mmedia34\Flash\Save the Earth\Part1 Step 3*

Part 2 – Scrolling Background

Introduction
This part of the exercise will cover adding scrolling backgrounds.

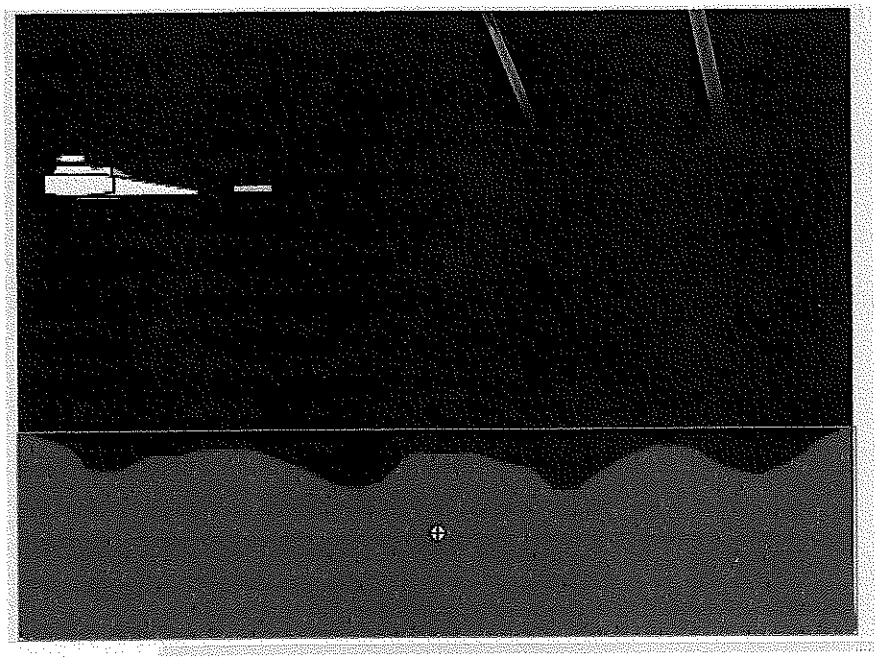
Step 1: Adding the Ground
The first thing we are going to do is to produce some ground that will scroll past as the spaceship flys over it.

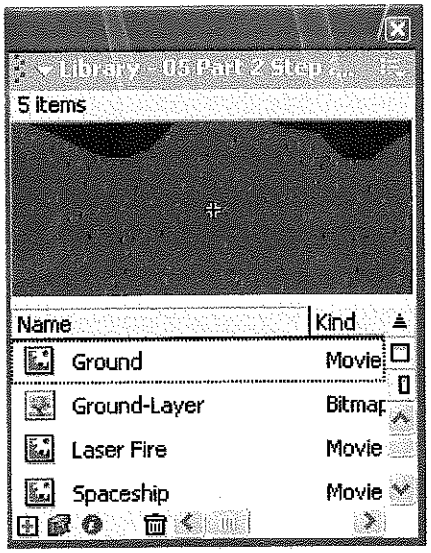
In *Photoshop* create an interesting graphic for the ground. As the stage in *Flash* was set to 800 pixels wide and 600 pixels high, the ground needs to be 800 pixels wide and an appropriate height, with a resolution of 72dpi. Make sure that that the ground will tile horizontally - so ensure that the ground starts and ends at the same height at each end of the graphic. Make the ground appear hilly. Safe as a GIF file.



Example of ground

Import your ground graphic into your Flash **library**. Create a new layer and call it *Ground*. Insert a blank keyframe (F6) and place the graphic from the library on to the stage. Convert the ground to a **movie clip** called *Ground*. Select the *Ground* **movie clip** and name the instance name to *Ground* (as shown below).





Step 2: Scrolling the Ground

The ground will scroll to the left of the screen, which is achieved by gradually reducing the x-coordinate of the ground movie clip.

We need two copies of the ground **movie clip** placed next to each other to achieve a smooth scrolling effect. The second copy is sometimes referred to as a **buffer**. This ensures that the ground always covers the **stage**. Instead of moving both of these **movie clips** to create our scrolling, what we will do is put them inside another **movie clip** (which we will call *mainGround*) and then we can move this parent **movie clip** instead of the two ground **movie clips**. In other words the *mainGround* movie clip will contain the two copies of the ground and we will just move *mainGround* instead of moving the two grounds individually.

The first thing we need to is to put the *Ground* instance inside another new movie clip.

Select the *Ground* **movie clip** and select **Convert to Symbol**. Give it the name *mainGround* and choose **movie clip** as the behaviour. Call the **Instance** *mainGround.*

We now have the *Ground* **movieclip** inside of the *mainGround* **movie clip**. We will now duplicate the *Ground* **movie clip** within the *mainGround* **movie clip**. Then we will be able to move the two ground movie clips at once by just moving the parent **mainGround** movie clip. This will be less work for the Flash player - as it only has to move one movie clip not two and hence will improve the speed of the scrolling. In building flash games the speed of the flash player is very important, so where-ever possible you should reduce the amount of code or movie clip manipulation that flash has to do.

Adding the code
We are now going to add in the `clipEvent` code for the *mainGround* movie clip. Select the *mainGround* and add the following code:

```
onClipEvent (load) {
     Ground.duplicateMovieClip("Ground2", 1000);
     Ground2._x = Ground._x+Ground._width;
     groundStartx = this._x;
     groundSpeed=10;
}
```

Explanation
This code is contained within a load ClipEvent, so it will be run when the *mainGround* is first loaded. The second line of the code makes a duplicate copy of the *Ground* instance of the movie clip and gives it the name *Gground2* and a depth of 1000.

The third line sets the x-position of *Gground2* to the x-position of the original ground plus the width of the ground. This has the effect of locating *Gground2* exactly to the right of the first ground movie clip.

The fourth line creates a new variable called `groundStartx` which is equal to the x-position of the the *mainGround* **movie clip**. The purpose of this variable is to store the start location of the *mainGround* **movie clip** - we will use this variable later.

The fifth line `groundSpeed=10;` just sets up a new variable called `groundSpeed` with a fixed value of 10. This will be the number of pixels the ground moves per frame.

Adding the code
Now add the following code , below the previous.

```
onClipEvent (enterFrame) {
     this._x-=groundSpeed;
     if (this._x<= (groundStartx-Ground._width)){
          this._x=groundStartx-groundSpeed;
     }
}
```

Explanation
This is contained within an `enterFrame onClipEvent` so it will be run over and over again. Remember that the code within an `enterFrame` clipEvent is run everytime the movieclip enters a new frame. The *mainGround* **movie clip** has just one frame - but it is still constantly entering that one frame.

The second line: `this._x-=groundSpeed;` reduces the x-coordinate of the ground by the value of `groundSpeed`. As we set `groundSpeed` to 10, then the code moves the *mainGround* to the left by 10 pixels.

The third line checks to see if the *mainGround* has moved so far to the left that the first ground **movie clip** is completely off the stage, if this is true then the fourth line sets the x-coordintate of the *mainGround* such that it is positioned back where it started.
`this._x` is the current x-coordinate of *mainGround*. `Ground._width` is the width of the ground movie clip. `groundStartx` is the variable we set up earlier. When the game starts the first ground exactly fills the stage. So this variable stores the x-coordinate at which the first ground fills the stage.

So `groundStartx - Ground._width` is the start x-coordinate minus the width of the ground. When the first ground is completely to the left of the stage its x-coordinate will be equal to its start x-coordinate minus its width. At this point the second ground will be exactly filling the stage. When we reset the *mainGround* back to its start position on the stage we have to also subtract the `groundSpeed` because the *mainGround* still needs to be moving left by the amount specified in `groundSpeed`.

You can now test the scrolling ground.

Copies of the spaceship, laser and mainGround code is located at *j:\Mmedia34\Flash\Save the Earth\Part2 Step 2*

Step 3: Modifying the scrolling

In most space games the ground doesn't just keep scrolling. What happens is that the player starts on the far left of the screen, when they get about a third of the way to the right of the screen they stop moving right and the ground starts scrolling. This is also what happens in a lot of platform games. This is what will happen in this game.

Most of the code for this will be in the spaceship **movie clip** events. As the spaceship location will determine when the scrolling starts and stops.

Adding the code - when to start scrolling
Select the spaceship and within the `onClipEvent(load)` code, under the line `laserCounter=1;` type the following:

```
scrollx=_root.mainGround.ground._width/3;
scrollStart=false;
```

Explanation
The first line sets up a new variable, `scrollx`. This variable will be the x-coordinate at which the spaceship stops moving right and the ground starts scrolling. We've made it one third of the width of the ground.

The second line sets up another new variable called `scrollStart`. This variable will be set to `true` if the ground should be scrolling and `false` if the ground isn't scrolling.

Adding the code
If the spaceship has gone past `scrollx` then it will stop moving right and the ground will start scrolling. The code for the right arrow will have to be changed.

Replace the code for the right arrow movement of the spaceship with the following:

```
if (Key.isDown(Key.RIGHT)) {
     if (this._x<scrollx){
          this._x+=moveSpeed;
     } else {
     scrollStart=true;
 }
```

Explanation
What we have done is introduce an if statement. If `this._x` (the current x-coordinate of the spaceship) is less than `scrollx` then the spaceship x-coordinate is increased - hence it moves right. However if that isn't the case, then we set variable `scrollStart` to `true` and don't change the spaceship x-coordinate.

Adding the code - When to stop scrolling
Now, we to need add in a whole new bit of `onClipEvent` code, dealing with a whole new event. Underneath all of the clipEvent code for the spaceship (after the last }) type the following:

```
onClipEvent (keyUp) {
     if (Key.getCode() == Key.RIGHT) {
          scrollStart=false;
     }
}
```

Explanation
This code introduces a new `clipEvent`, namely `keyUp`. This event occurs whenever the player takes their finger off a key. Its useful here because what we want is for the ground to scroll while you are holding the right arrow key down, but to stop scrolling when you take your finger off the right arrow key.

The method `Key.getCode()` gives the last key that was released. So the `if` statement see's if the last key that was released was the right key and if that's true then it sets our variable `scrollStart` to `false`.

All we need is to add some code so that the ground actually stops scrolling when `scrollStart` is `false`. Select the *mainGround* **movie clip**. Modify the mainGround code to read:

```
onClipEvent (enterFrame) {
     if (_root.mcSpaceship.scrollStart==true){
          this._x-=groundSpeed;
          if (this._x<= (groundStartx-Ground._width)){
               this._x=groundStartx-groundSpeed;
          }
     }
}
```

==========

Explanation
An `if` statement has been added so that the code to move the *mainGround* only happens if the variable `scrollStart` is `true`.

==========

Finally, set the frame rate of your flash file to 25fps. A frame rate of 25fps should make the scrolling look quite smooth. Test your movie, you should now find that the ground scrolls when you reach a third of the way across the stage.

Copies of the spaceship, laser and mainGround code is located at *j:\Mmedia34\Flash\Save the Earth\Part2 Step 3*

Step 4: Parallax scrolling

This game only has one scrolling background so far. To improve the look of the game a second scrolling background will be added. Your second background could be some stars or distant mountains. If you set the `scrollSpeed` for this background to a slower amount you will get an effect known as parallax scrolling - it gives a feeling of depth to the game.

Try adding in the second scrolling background yourself. You will follow the same steps and use the exact same code as you did for the ground. The only difference is that the graphic will be something different (eg: Stars) and that the movie clip instance names will be different (eg: Stars and mainStars). And the set the speed to a slower speed such as 4.

The code for you mainStars movie clip should be something like (assuming you have named your movie as **Stars**)

```
onClipEvent (load) {
     Stars.duplicateMovieClip("Stars2", 1000);
     Stars2._x = Stars._x+Stars._width;
     starsStartx = this._x;
     starsSpeed=4;
}
onClipEvent (enterFrame) {
     if (_root.mcSpaceship.scrollStart){
          this._x-=starsSpeed;
          if (this._x<= (starsStartx-Stars._width)){
               this._x=starsStartx-starsSpeed;
```

```
            }
        }
}
```

Copies of the spaceship, laser, mainStars and mainGround code is located at
j:\Mmedia34\Flash\Save the Earth\Part2 Step 4

Part 3 - Meteors and collisions

Introduction

This part of the exercise will cover meteors and collisions.

Step 1: Creating Meteors

First create a meteor to fire at. Easy to do in Photoshop or Paint.

For example 

On the **Main Timeline** create a new layer and call it *Meteor*.

Import the meteor into the **library**. On *Meteor* layer insert the meteor. Select the meteor and Convert it to a **movie clip** symbol, calling it *Meteor*. Call the instance *Meteor1*.

The meteors

The game is going to follow a typical space shooting game structure. The meteors are going to move across the stage from right to left. The player will need to either dodge or shoot the meteors. If the player shoots a meteor it will explode. If a meteor collides with the player the game will be over.

We will use `duplicateMovieCip` to create multiple meteors. However if all the meteors started at the same location the game would be very easy and boring. We need to introduce a random element to the game. So all the meteors will start at the same x-position (to the right of the stage) but will each have a random y-position. We will also make the meteor speed random, to add an extra challenge to the game.

To write code for setting up a random start position and speed for the meteors. Functions are going to be used.

What are Functions?

Functions let you group together lines of code. Often in game programming you want the same code to be run in different parts of your game. For example, you might want to increase the players score and play an animation if they collect a bonus. You might want the exact same thing to happen if they complete a level. You could write the code for collecting the bonus and then copy and paste it to the part of the game that handles completing a level. OR you could write a function to increase the score and play the animation and call the function whenever you wanted the score increase and animation to happen. The code for a function looks like this:

```
function scoreBonus(){
     // code goes here
}
```

This is known as **defining the function** - it defines what the function is called and what code makes up the function. **scoreBonus** is the functions name. Any code contained between the curly brackets will be run when the function is called. '**Calling a function**' means running a function and is done in your code simply by typing the function name. So the code `scoreBonus();` will call the function resulting in the functions code being run.

Note: Defining the function doesn't actually run the functions code. The code is only run when the function is called.

Adding the code - a reset function

We are going to create a function to set up the start location and speed for the meteors.
Select the meteor **movie clip**. Type the following code:

```
onClipEvent (load) {
     function reset(){
          this._x=800;
          this._y=Math.random()*500+1;
          meteorSpeed=Math.random()*20+10;
          this.gotoAndStop(1)
     }
     reset();
}
```

Explanation

The code defines the function called `reset` and then runs this function. The code is all within a `load` clip event so it will be run when the movie clip first loads.

The line `function reset(){` defines the reset function. The next three lines are the reset functions code.

The line `this._x=800;` just sets the x-coordinate of the meteor to 800 (just to the right of the stage).

`Math.random` generates numbers between 0 and 1.

The line `this._y=Math.random()*500+1;` sets the y-coordinate of the meteor to a random number between 0 and 499 ie from the top of the screen to five hundred pixels down (100pixels above the bottom). `meteorSpeed=random()*20+10;` sets a new variable called `meteorSpeed` to a value random between 10 and 30. This will be the number of pixels that the meteor moves per frame.

`this.gotoAndStop(1)` goes to the first frame of the meteor movie clip – it is only one frame at the moment, but you will be adding more frames as the meteor explodes.

Finally the line `reset();` calls the function - which runs the code we defined above. We could have just written the three lines of code and left out the function definition altogether. Well in the next step you will see that we also want to call the `reset` function elsewhere in our code.

Step 2: Making the meteors move

Underneath the code from the previous step type the following code for the meteors `enterFrame` clip event:

```
onClipEvent (enterFrame) {
     if (_root.mcSpaceship.scrollStart){
          this._x-=meteorSpeed+_root.mainGround.groundSpeed;
     } else {
          this._x-=meteorSpeed;
     }
     if (this._x<-10) {
          reset();
     }
     if (this.hitTest( _root.mcSpaceship ) ){
     _root.gotoAndStop(6);
     }
}
```

Explanation

This code does two things; it moves the meteor across the stage by reducing its x-coordinate and it resets the meteor if it has moved off the left edge of the stage. The first if statement checks to see if our variable `scrollStart` is `true`.

If it is `true` then the meteors x-coordinate is decreased by `meteorSpeed+_root.mainGround.groundSpeed` otherwise it is decreased by just `meteorSpeed`. So if the ground isn't scrolling the meteor will be moved left by the random number assigned to `meteorSpeed`. If the ground is scrolling the meteor's x-coordinate will be decreased by `meteorSpeed` plus the speed at which the ground is scrolling. We need to do this to ensure the movement looks realistic. If the ground started scrolling and the meteor just kept moving at the same speed it wouldn't look right.

The second if statement checks to see if the meteor **movie clip** has moved off the left of the stage (ie it's x-coordinate is less than -10). If this is true then the `reset` function is called. The `reset` function will reposition the movie clip on the right of the stage and assign a new random speed and y-coordinate. If you test your Flash file you should now have a meteor moving across the stage and resetting itself after it moves off the left side of the stage.

Adding extra meteors

One meteor isn't going to be a very challenging game, `duplicateMovieClip` will be used to add some more meteors. We will put the code to duplicate the meteor in a new layer on the main timeline.

Adding the code
On frame 4 of the Actions layer insert a blank keyframe (F6). Add the following code in this frame:

```
numMeteor=5;
for (i=2; i<=numMeteor; i++){
     Meteor1.duplicateMovieClip( "Meteor"+i, i+100 );
}
```

Explanation
The first line creates a new variable called `numMeteor` and sets it to 5. This variable will be the number of meteors on the stage at any one point in time, if you want to make the game harder just increase this number.

The next three lines are a `for` loop. The loop duplicates the *meteor1* **movie clip** instance.

For loops are used when you want to repeat some code a set number of times. For loops use a counter, in this case its called `i`, which is typically increased by one each time *Flash* goes through the loop.
The line `for (i=2; i<=numMeteor; i++){` does a number of things.
The `i=2` sets the variable `i` equal to 2 when the loop starts the first time. The `i++` increases `i` by one everytime the loop is repeated.The `i<=numMeteor` keeps looping while `i` is less than or equal to `numMeteor`. It stops looping when *i* is greater than `numMeteor`.

The result of this is that four duplicates of *Meteor1* will be created, called *Meteor2*, *Meteor3*, *Meteor4* and *Meteor5*. If you increase the value of `numMeteor` then the number of duplicate meteors will increase accordingly.

Add a blank frame to the *Ground*, *Spaceship*, *Laser Fire*, *Meteor* and *Stars* (or whatever you made instead of Stars) layers in frame 5 (F5) and a blank keyframe (F6) on the *Actions* layer

In the blank keyframe on the Actions layer add the following code:

```
stop();
```

Explanation
This stops the main timeline. Even though the **main timeline** is stopped the laser, spaceship, stars, meteors and ground **movieclips** are still playing. The timelines of **movie clips** run independent of the main timeline, they will only stop if you specifically tell each one of them to stop.

If you test your flash file now you should have five meteors flying across the screen.

Copies of the main timeline, meteor, spaceship, laser, mainStars and mainGround code is located at *j:\Mmedia34\Flash\Save the Earth\Part 3 Step 2*

Step 3: Collision detection

We want to detect when a laser hits an meteor and consequently make the meteor explode and disappear. Similarly we want to detect when a meteor collides with the players spaceship, at which point the game will end.

The spaceship, meteors and lasers are all **movie clips**. So we will use the `hitTest`.

Adding the code

For the code to detect the collision between the lasers and the meteors open the laser code and change the enterFrame onClipEvent to the following:.

```
onClipEvent (enterFrame) {
     this._x+=laserMoveSpeed;
     if (this._x>800){
          this.removeMovieClip();
     }
     for (i=1; i<=_root.numMeteor; i++){
          if (this.hitTest( _root["Meteor"+i])){
               _root.Score+=100;
               _root["Meteor"+i].gotoAndPlay(2);
          }
     }
}
```

Explanation

This code checks to see if this laser movie clip is colliding with any of the meteors. It uses a `for` loop to check if this laser is colliding with each one of the meteor **movie clips**. If this is `true` it increases the score by 100 and tells the associated meteor **movie clip** to goto and play its frame 2 (not made yet).

The first line sets up a `for` loop – `for` loops explained earlier. The second line is an `if` test. It checks to see if `this` **movie clip** (ie the laser) is colliding with a meteor **movie clip** specified by the code `_root["Meteor'+i]`. As explained earlier, this is an array style referencing for a **movie clip**. So when `i` is equal to 1 then `_root.["Meteor'+i]` will evaluate to `_root.Meteor1` when `i` is 2 it will evaluate to `_root.meteor2` and so on.

If the `hitTest` is true then the next two lines are run. These lines deal with things to be set up in the next step.

The line `_root.Score+=100;` increases a variable called `Score` by 100. We haven't set up this variable yet, we will do it in the next step and you will see how easy it is to add a score to your game.

The other line `_root["Meteor"+i].gotoAndPlay(2);` tells the meteor that has collided with the laser, to goto its frame 2. What we are going to do is put an explosion animation in the meteor **movie clip** at frame 2.

The original laser - tidying things up
Modify the laser code as follows:

```
onClipEvent (enterFrame) {
     if (this._name<>"mcLaser"){
          this._x+=laserMoveSpeed;
          if (this._x>800){
               this.removeMovieClip();
          }
          for (i=1; i<=_root.numMeteor; i++){
               if (this.hitTest( _root["Meteor"+i])){
                    _root.score+=100;
                    _root["Meteor"+i].gotoAndPlay(2);
               }
          }
     }
}
```

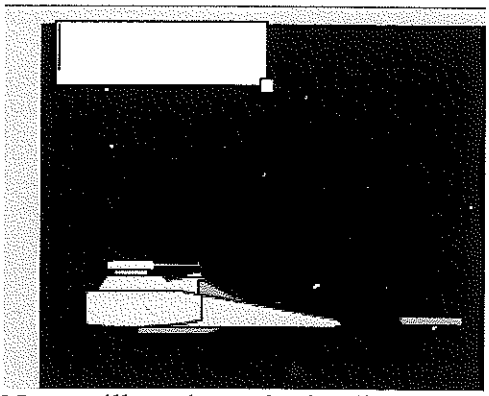
Explanation
The original laser **movie clip** instance (called *mcLaser*) is never removed. This is because the `removeMovieClip` method only applies to **movie clips** that were created using `duplicateMovieClip`. As the original laser was drawn by us it can't be removed using `removeMovieClip`. – which we don't want to do anyway. If we removed the original laser we wouldn't be able to duplicate new lasers from the original.

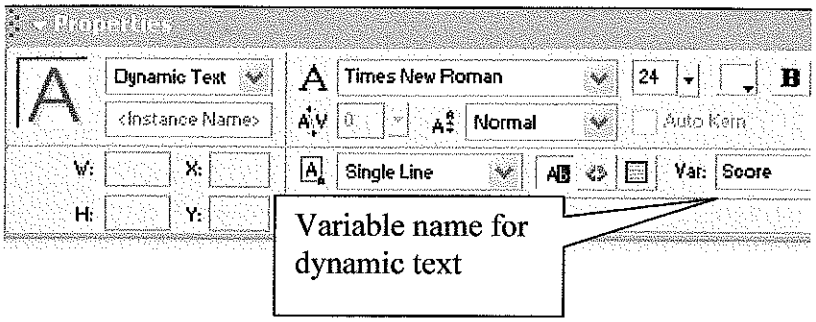
The added `if` statement makes the **movie clip** check which version of the laser it is. If it is the original version (the one with the name *mcLaser*) then the code to move, remove and check for collisions will not be run. The <> means not equal to.

Step 4: Scores and explosions

We are now going to add in a score display, some graphics to show the meteor exploding and some collision detection code which will detect if the player's spaceship has hit a meteor.

Score Display
On the **main timeline** create a new layer called *Score*. Insert a new blank keyframe in frame 4. On this layer create a dynamic text box, you can put it anywhere on the screen. Stretch the text box out so it is wide enough to display the score (as shown below). Change the text from **Static Text** to **Dynamic Text**, with an appropriate colour font – you can type some numbers so you can see what it will look like (these numbers will not show in the finished game).





Give the text's variable name (Var) the name `Score`.

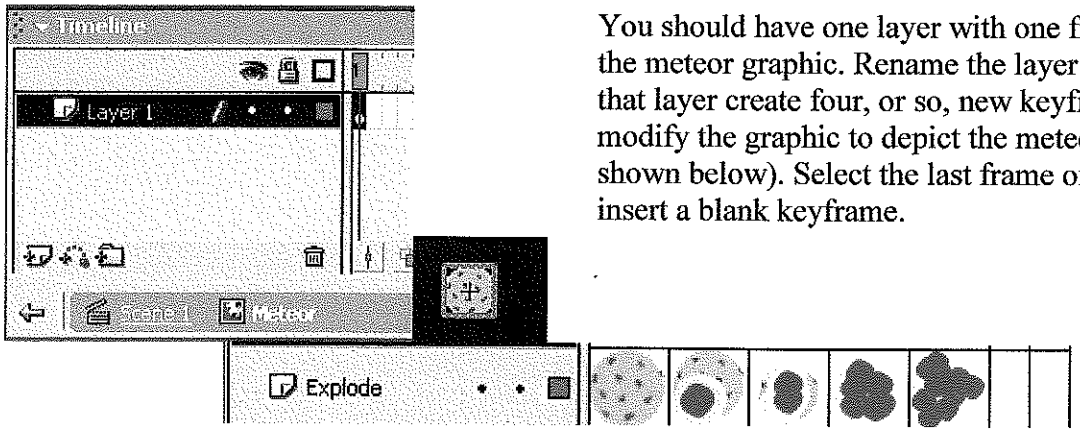
Now, still on the main timeline select the fourth frame of the *Action* layer and add the following code underneath the existing code:

```
Score=0;
```

Click in frame 5 of the Score layer and insert a frame (F5).

Explanation
This simply sets up the score variable and sets it to zero at the start of the game.

Exploding meteors!
Now we want to add in an explosion animation for the meteor **movie clip** when it is hit by a laser. Open the *Meteor* **movie clip** (Double click the *Meteor1* **movie clip** on the stage).



You should have one layer with one frame containing the meteor graphic. Rename the layer *Explode*. On that layer create four, or so, new keyframes and modify the graphic to depict the meteor exploding (as shown below). Select the last frame of this layer and insert a blank keyframe.

Create a new layer and call it *Actions*. Select the first frame of this layer and type the following code:

```
stop();
```

Now insert a blank keyframe on the last frame of the *Actions* layer (above the blank key frame on the *Explode* layer) Type the following code in this frame:

```
stop();
```

Explanation
We stop the meteor **movie clip** on frame one because we don't want the explosion animation to play until the meteor is hit by a laser.

We have previously added the code to play the animation from frame 2 when the laser collides with the meteor **movie clip**.

We stop the **movie clip** after the explosion on the last frame because there are no graphics on this frame. The **movie clip** still exists and will still keep moving across the stage until it reaches the left hand side and is reset. However as there are no graphics being displayed Flash won't detect collisions between the movie clip and the spaceship or the lasers once the explosion animation has finished.

Step 5: Detecting a collision between the spaceship and the meteor

We will add some code into the *Meteor1* **movie clip** instance to test if it is colliding with the player's spaceship and if this is true then main **timeline** will move to a game over section.

Player / meteor collision detection and resetting after exploding

Modify the Meteor1 movie clip instance code to the following:

```
onClipEvent (load) {
     function reset(){
          this._x=800;
          this._y=Math.random()*500-100;
          meteorSpeed=Math.random()*20+10;
          this.gotoAndStop(1)
     }
     reset();
}
onClipEvent (enterFrame) {
     if (_root.mcSpaceship.scrollStart){
          this._x-=meteorSpeed+_root.mainGround.groundSpeed;
     } else {
          this._x-=meteorSpeed;
     }
     if (this._x<-10) {
          reset();
     }
     if (this.hitTest( _root.mcSpaceship ) ){
     _root.gotoAndStop (3);
     }
}
```

Explanation
This is an `if` statement that checks if `this` movie clip (ie the meteor) is colliding with the spaceship. If this is true, then the **main timeline** is instructed to goto and stop at frame 6.

We have set up the meteor **movie clip** to play an explosion animation and stop on its empty frame if it is hit by a laser. When it moves off the left edge of the stage and is reset, it effectively becomes a new meteor. It is the same **movie clip**, but seems like a new meteor emerging from the right side of the stage. So we reset it back to the first frame by using `this.gotoAndStop(1);` as the last line of the reset function.

Game Over
You may want to do an animation sequence for the end of the game, but to keep this exercise as simple as possible our end sequence will be one frame with the message Game Over, a play again button and the final score.

On the **main timeline** you will add an extra frame to the end of every layer of the game frames as instructed. So you will have a 6 frame movie.

Insert a blank keyframe (F6) on frame 6 of the *Meteor*, *Laser*, *Spaceship* and *Actions* layers.

The main timeline stops on frame 5 while the game is playing. When the game is over it will move to frame 6. We inserted blank keyframes on the laser, spaceship and meteor layers because we don't want these **movie clip** still visible at the end of the game.

On the **main timeline** add in a new layer called *Game Over* and insert a blank keyframe on the sixth frame of this new layer.

On the *Game Over* layer in the new keyframe add some text on the stage saying game over. It doesn't matter what font or size or colour - just pick something you like.

Create a button that allows you to play again by retuning to frame 4 on the main time line.

Test the finished game.

Copies of the main timeline, meteor, spaceship, laser, mainStars and mainGround code is located at *j:\Mmedia34\Flash\Save the Earth\Final*

G. EXTENSIONS/IMPROVEMENTS/EXPERIMENTATION

While we have built a simple space flying game, many of the core elements of the game could be used in a variety of action games. Almost all action games involve player movement, meteor movement, collision detection and scoring, which were all implemented in this game.

Some suggestions on how the game could be improved:
To keep the exercise from being too long, the game was kept as simple as possible. However there are quite a few improvements that could be made to the game.

- Make the game harder by increasing the speed of the meteors or the number of meteors or limiting the laser fire.
- If you noticed the game performing slowly or if movement was a little jerky, this would be because the graphics and stage in the game are quite large. Try reducing the size of the stage and game graphics.
- At the moment the player can move off the top, bottom and left of the stage. Limit the spaceship to the stage.

- Add in player lives so instead of the game ending when the player collides with a meteor, the player just loses a life. You will need to add in a lives variable and reduce this by one when the collision occurs. When the lives is equal to zero the game will be over. It is similar to the Scoring feature.
- Add sound effects eg laser firing, meteor exploding.
- The lasers currently keep on travelling even when they hits a meteor. You could change this by using the same code that is used to remove a laser when it goes off the stage to remove the laser when it hits a meteor.

References

David Doull Building Games in Flash 5 Parts 1, 2 and 3 from *www.flashkit.com*
Game source and text developed from these three tutorials.