TASK 9a: KEYBOARD CONTROL

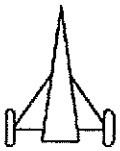
EXERCISE 1: HORIZONTAL AND VERTICAL MOVEMENT

For this exercise:
Copy the file Basic *Spaceship.fla* to your h: drive from *j:\Mmedia34\Flash\Movement*.

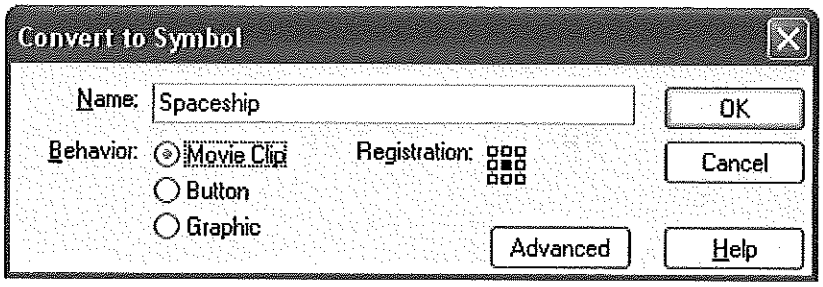
This file consists of one frame containing a spaceship that was drawn in Adobe Illustrator (from two triangles and two round cornered rectangles) and imported into Flash.

a) Horizontal Movement

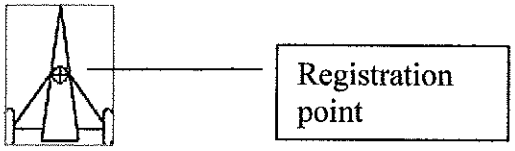
Open the file and save it as *Spaceship Exercise 1a.fla*.



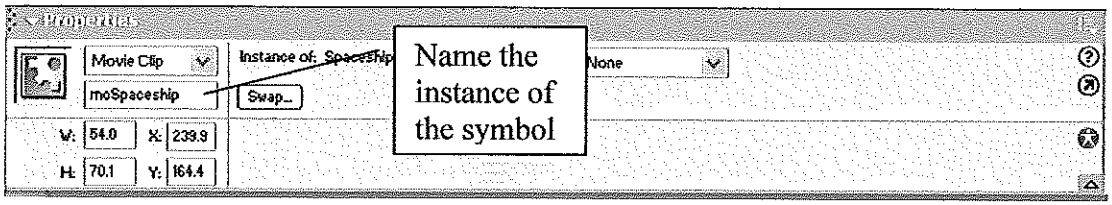
Select the spaceship by clicking on it, then select **Convert to Symbol** as follows:



Call the symbol *Spaceship* and make it a **Movie Clip**. It is now in the **library**.



The spaceship is now a symbol and has a registration point.



Name this **instance** something meaningful – such as *mcSpaceship*. You don't have to have *mc* in front of Spaceship – it just is a signal that Spaceship is a movie clip. It makes things easier to track down.

Check that the **symbol** is in your **library**.

Select the **symbol**. You are now going to add actionscript script to the **movie clip**. To do this select the **Actions** panel (if it is not visible F9 to open it or use the **Windows** menu).
Enter the following script (use either **Expert Mode** or **Normal Mode** – your choice):

```
onClipEvent (enterFrame) {
    if (Key.isDown(Key.LEFT)) {
        _x -= 3;
    }
    if (Key.isDown(Key.RIGHT)) {
        _x += 3;
    }
}
```

Once completed test the movie. This gives sideways movement of the spaceship using the left and right arrow keys. Experiment with these values. Save the file with the settings above.

Explanations

Script	Explanation
`onClipEvent (enterFrame)`	This allows script to be associated with a **movie clip** instance (in this case *mcSpaceship*). The brackets `()` contains an event. In this case `enterFrame`. `enterFrame` causes the movie clips frame rate to trigger it continuously. So the movie clip can be continuously moved about the stage – even though the Flash movie itself only has one frame. The curly bracket `{` is the start of the actions that will then occur, this matches up with the curly bracket `}` on the very last line. The `onClipEvent` is not indented, nor is the last `}`, indicating they go together There are other events that can be used. The event load means that the loading of the movie clip triggers the event.

`if (condition) {`
`  some action`
`}`
	The `if` command carries out an action(s) if a particular condition is met. The condition is in rounded brackets ie `(` and `)` – these are the argument of the function. The curly bracket `{` indicates the start of the actions that will occur when the condition is met, the first `}` at the same indentation as the if indicates the end of the actions.
	
	In the script used. `Key.isDown(Key.LEFT)` is the first condition. This condition is met if the left arrow key is pressed. If the left arrow key is pressed `_x -= 3` moves the movie clip 3 pixels to the left.
	
	Similarly, `Key.isDown(Key.RIGHT)` is checking for the right arrow key to be pressed. If it is, `_x += 3` moves the movie clip 3 pixels to the right.

Script	Explanation
_x	This is an objects horizontal position on stage. _x += 3. _x is a variable (ie it can change value). += 3 adds 3 to the previous value ie it moves the object 3 pixels horizontally to the right. -= 3 subtracts 3 from the previous value ie it moves the object 3 pixels horizontally to the left.
_y	This is an objects vertical position on stage – it is included here for completeness. y += 3. _y is a variable (ie it can change value). += 3 adds 3 to the previous value ie it moves the object 3 pixels vertically up. -= 3 subtracts 3 from the previous value ie it moves the object 3 pixels vertically down.

b) Horizontal and Vertical Movement

Now save the file as *Spaceship Exercise 1b.fla.*
Modify the script as follows:

```
onClipEvent (enterFrame) {
    if (Key.isDown(Key.LEFT)) {
        _x -= 3;
    }
    if (Key.isDown(Key.RIGHT)) {
        _x += 3;
    }
    if (Key.isDown(Key.UP)) {
        _y -= 3;
    }
    if (Key.isDown(Key.DOWN)) {
        _y += 3;
    }
}
```

Experiment by modifying the amount you move each key press. Move the spaceship around the screen, see what happens at the edge of the screen. Save the file.

c) Screen Wrapping

Screen wrapping is where you move off one side of the stage and appear on the opposite side eg you move off the top of the **stage** and you come back on from the bottom of the **stage**; you move off the right of the **stage** and you come back on from the left of the **stage**.

Save the file as *Spaceship Exercise 1c.fla.* The **stage** in this movie is 400 pixels high and 550 pixels wide.

Modify the script as follows:

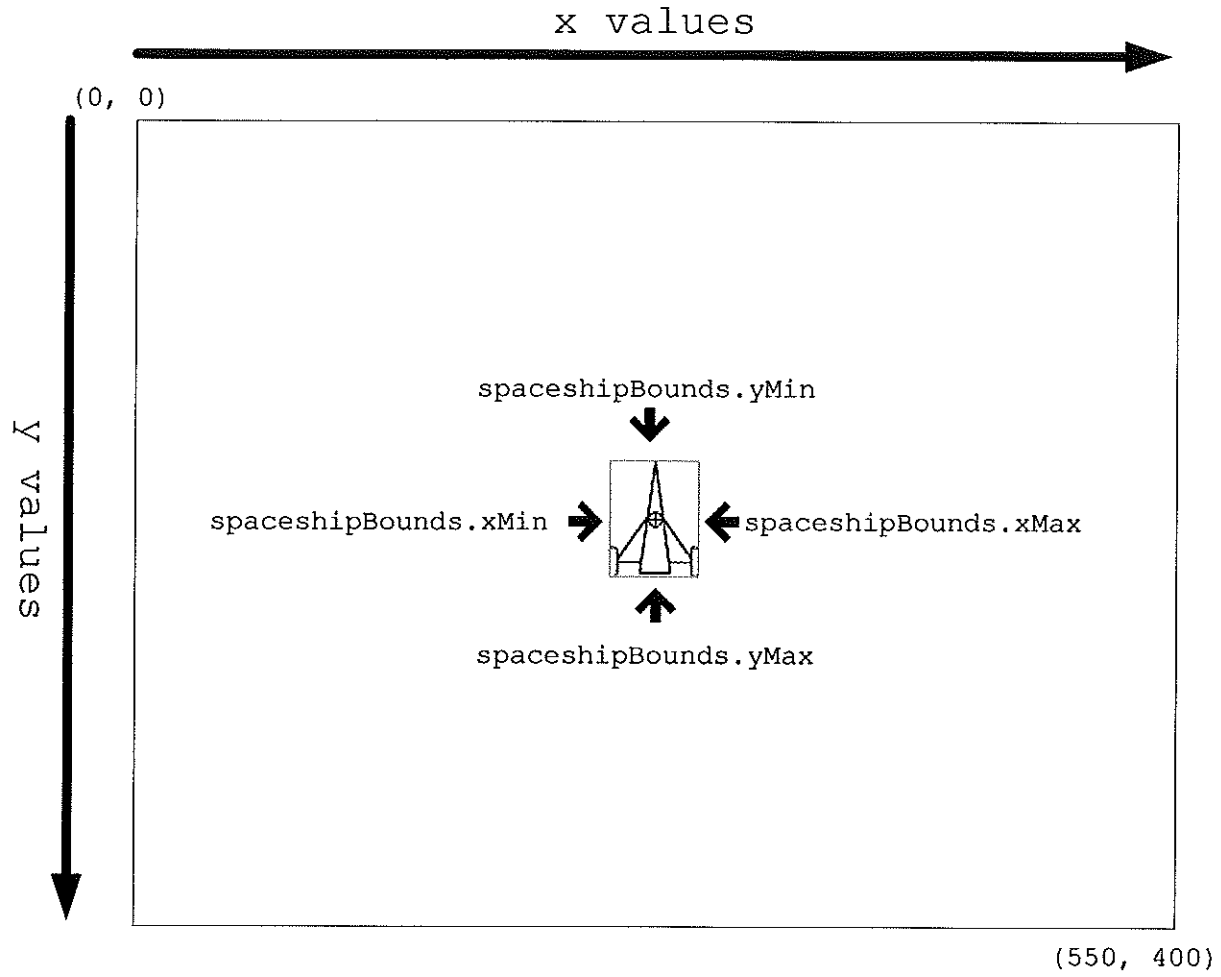
```
onClipEvent (enterFrame) {
      if (Key.isDown(Key.LEFT)) {
            _x -= 3;
      }
      if (Key.isDown(Key.RIGHT)) {
            _x += 3;
      }
      if (Key.isDown(Key.UP)) {
            _y -= 3;
      }
      if (Key.isDown(Key.DOWN)) {
            _y += 3;
      }
      // screen wrapping
      spaceshipBounds = this.getBounds(_root);
      if (spaceshipBounds.yMax < 0) {
            _y = 400;
      }
      if (spaceshipBounds.yMin > 400) {
            _y=0;
      }
      if (spaceshipBounds.xMax < 0) {
            _x=550;
      }
      if (spaceshipBounds.xMin > 550) {
            _x=0;
      }
}
```

Test the movie. Move the spaceship around the screen – including on and off **stage**.

Explanations

Script	Explanation
`spaceshipBounds = this.getBounds(_root);`	`spaceshipBounds` is the name of a variable I have made up. It is being used to find the boundaries of the movie clip ie the extremes horizontally and vertically. So I have used a name that is 'meaningful'. `this.getBounds(_root)` is the command that finds the position of these extremes of the movie clip on the stage. `this` simply means that it is the **instance** of the **movie clip**, that has the script attached to, that is being referred to. `getBounds(_root)` collects the minimum and maximum x and y values of the movieclip from the stage. The values are called `xMin`, `xMax`, `yMin` and `yMax`. The `_root` refers to the main timeline ie the position on stage.

Script	Explanation
`spaceshipBounds.yMax`	This is the lowest position of the spaceship on the stage.
`spaceshipBounds.yMin`	This is the highest position of the spaceship on the stage.
`spaceshipBounds.xMax`	This is the position furthest to the right of the spaceship on the stage.
`spaceshipBounds.xMin`	This is the position furthest to the left of the spaceship on the stage.



The top left hand corner of the stage is `(0, 0)` and the bottom right hand corner `(550, 400)`. The `if` statements, such as:

```
if (spaceshipBounds.yMax < 0) {
    _y = 400;
}
```

test to see if the **movie clip** is moving off the edge of the screen and if it is it sends it back on from the opposite side. In the script above, if the lowest point of the **movie clip** (`spaceshipBounds.yMax`) has moved over the top of the stage (`< 0`). Then its `_y` position is set to the bottom of the screen.

Stages can be different sizes – either by being set at different sizes or by being resized. Remember a Flash movie is published as a Shockwave file (swf) which is a vector graphic and can be rescaled. To avoid this being a potential problem you can produce script that works in any situation – this is a general solution.

Convert your script to a general solution. Modify the screen wrapping script as follows:

```
spaceshipBounds = this.getBounds(_root);
if (spaceshipBounds.yMax < 0) {
     _y = Stage.height;
}
if (spaceshipBounds.yMin > Stage.height) {
     _y=0;
}
if (spaceshipBounds.xMax < 0) {
     _x=Stage.width;
}
if (spaceshipBounds.xMin > Stage.width) {
     _x=0;
}
```

Save the file and test the movie.

Keep *Spaceship Exercise 1c.fla* open. Open a new Flash file. Using the **Stage properties** set it to something like 600 high by 800 wide – don't use the defaults of 400 high and 550 wide.

Using the **Window menu** show *Spaceship Exercise 1c.fla*. Click on the spaceship symbol on the stage. Select copy. Close the *Spaceship Exercise 1c.fla* file. In your new Flash document paste the symbol on the stage. Now test the movie. Close the file when finished.

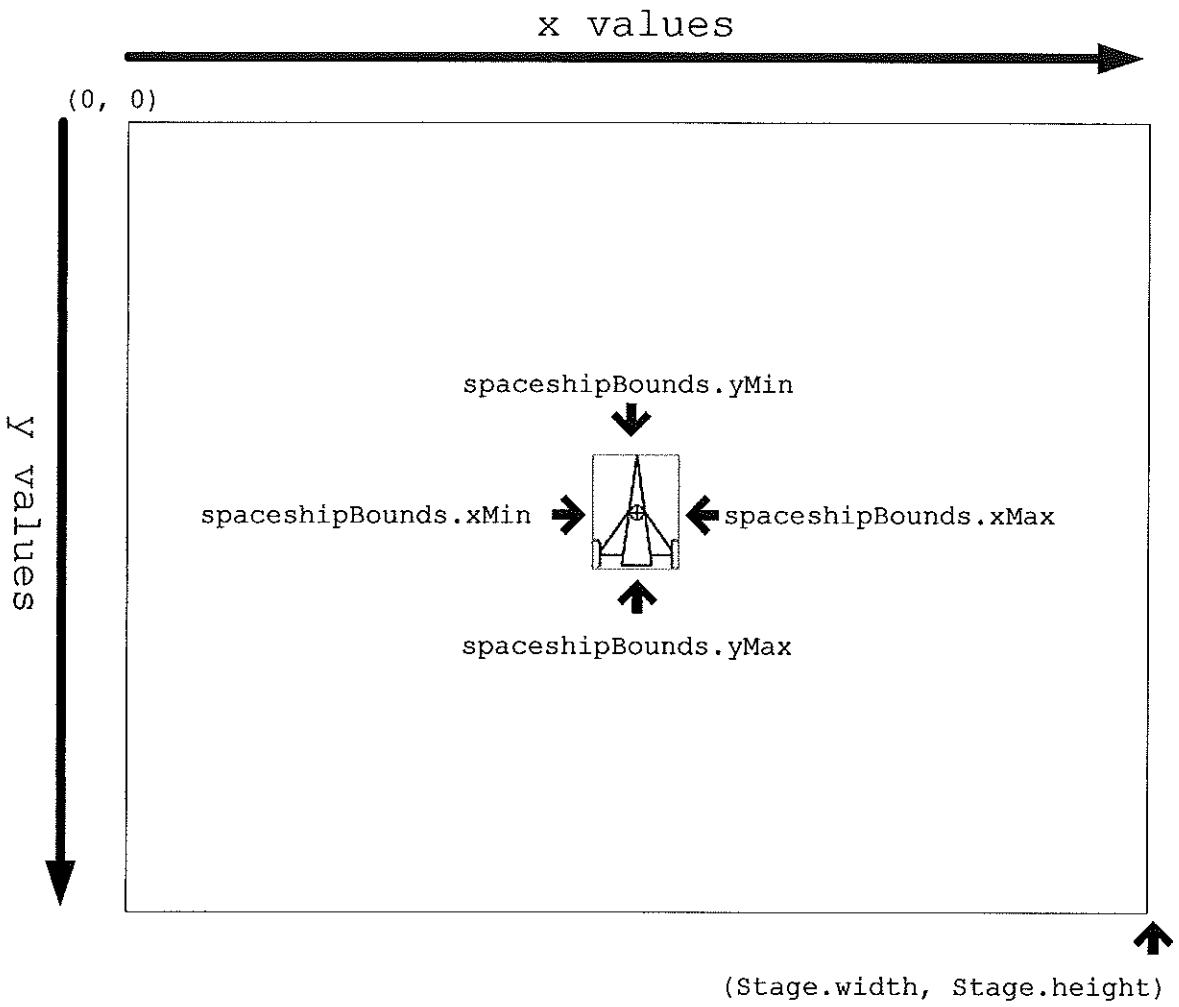
Note:

1) When you copied the movie clip, the actionscript script was also copied.
2) The general solution showed it worked on different stage sizes and needed no modification of script.

Explanations

Script	Explanation
`Stage.width`	The pixel width of the stage.
`Stage.height`	The pixel width of the stage.

(See diagram next page.)



d) Restricting movement to the stage

Open the file *Spaceship Exercise 1b.fla* and save it as *Spaceship Exercise 1d.fla*. This script will limit the spaceship to the stage only. It will only be able to move up to the edge and no further. Display the current script for the movie clip. Highlight it and delete it. Add the new script as follows:

```
onClipEvent (enterFrame) {
     spaceshipBounds = this.getBounds(_root);
     if ((Key.isDown(Key.LEFT)) and (spaceshipBounds.xMin >= 0)){
          _x -= 3;
     }
     if ((Key.isDown(Key.RIGHT)) and (spaceshipBounds.xMax <= 550)){
          _x += 3;
     }
     if ((Key.isDown(Key.UP)) and (spaceshipBounds.yMin >= 0)){
          _y -= 3;
     }
     if ((Key.isDown(Key.DOWN)) and (spaceshipBounds.yMax <= 400)){
          _y += 3;
     }
}
```

Test the movie to see the result. Check all boundaries. This script is specific for a stage that is 400 pixels high and 550 pixels wide. Save the file.

To produce a general solution do the following:

1. Replace the script `spaceshipBounds.xMax <= 550` with `spaceshipBounds.xMax <= Stage.width`
2. Replace the script `spaceshipBounds.yMax` <= 400 with `spaceshipBounds.yMax <= Stage.height`

Explanation

The `if` statements have two conditions that must be met before the actions are taken. The word `and` means that **both** conditions must be met – if the word `or` was used, then **either** condition **or both** conditions must be met for the action to be taken. For example:

```
if ((Key.isDown(Key.LEFT)) and (spaceshipBounds.xMin >= 0)){
```

For the if statement to be met the left arrow has to be presses and the spaceship must be to the right of the left edge of the stage.

---

e) Examples of Flash games with this style of movement

The games listed below are open source games downloaded from the internet. Copy them to your h: drive from *j:\Mmedia34\Flash\Movement\Ex 1 Examples*. You only need to copy the shockwave (swf) file and run this by double clicking on the file – the source file (fla) is also included if you wish to look at the script for the whole game. The games have similar movement controls to those you have created. Test them out.

Game	Files	Source	Comments
Space Shooter 3	SpaceShooter3.fla SpaceShooter3.swf	www.flashkit.com	Use all for arrow keys and spacebar to fire. Written with Flash 5.
Air Fox	airfox.fla airfox.swf	www.flashkit.com	Use all for arrow keys and spacebar to fire. Written with Flash 5.
Car Game	map_004.fla map_004.swf	www.actionscripts.org	Use left arrow and right arrow to move. Up arrow to speed up and down arrow to slow down. Written with Flash MX.
Random Maze	random_maze.fla random_maze.swf	www.actionscripts.org	Use arrow keys to move in required direction. Written with Flash MX.

EXERCISE 2: MOVEMENT, ROTATION AND ACCELERATION

This exercise will develop a system to move the spaceship in all directions – not just left, right, backwards and forwards – by rotating the spaceship.

a) Basic model using arrow keys

Open the file *Spaceship Exercise 1a.fla*.and save it *Spaceship Exercise 2a.fla*. Open the **Actions** panel for the actionscript associated with the symbol and delete it all.

Enter the following script (Note: After the // are comments explaining what the script does, it is not part of the script):

```
onClipEvent (enterFrame) {
      // makes the spaceship go forward by increasing speed by 1
      if (Key.isDown(Key.UP)) {
            speed += 1;
      }
      // makes the spaceship go backwards by decreasing speed by 1
      if (Key.isDown(Key.DOWN)) {
            speed -= 1;
      }
      // tells the spaceship to slow down after the speed of 20 is
      reached
      if (Math.abs(speed)>20) {
            speed *= .7;
      }
      // you can change the rotation of the spaceship - this
      rotates 15 degrees
      if (Key.isDown(Key.LEFT)) {
            _rotation -= 15;
      }
      if (Key.isDown(Key.RIGHT)) {
            _rotation += 15;
      }
      // Calculating the new position of the spaceship
      xPos = Math.sin(_rotation*(Math.PI/180))*speed;
      yPos = Math.cos(_rotation*(Math.PI/180))*speed*-1;
      _x += xPos;
      _y += yPos;
}
```

The up arrow sends the spaceship forward and the down arrow sends it backwards. The left arrow rotates the spaceship 15° to the left and the right arrow rotates the spaceship 15° to the right. The spaceship has a maximum speed of 20 units and is slowed down automatically if it is reached.

Test the movie and experiment with the movement – try different values (eg speed change, rotation). Save the file with the above values.

Explanation

Script	Explanation
`_rotation`	The number of degrees an object is rotated. 0 is the original position. Positive values rotate clockwise and negative values rotate anticlockwise. `_rotation -= 15` rotates 15 degrees anticlockwise. `_rotation += 15` rotates 15 degrees clockwise.
`Math.abs(speed)>20`	`Math` indicates the math **object** is being used. `abs` is the **method** is being used. `abs` stands for absolute value, it gives the size of something the absolute value of 20 is 20, the absolute value of -20 is 20. When the `if` statement condition is met then `speed *= .7` is the action carried out. The speed is multiplied by 0.7 ie it is 0.7 or 70% of what it was.

```
xPos = Math.sin(_rotation*(Math.PI/180))*speed
yPos = Math.cos(_rotation*(Math.PI/180))*speed*-1
```

The two line of code are working out the position of the spaceship taking into account its direction. It uses trigonometry from the unit circle – part of Mathematical Methods and Specialist Mathematics. These values are then given to _x and _y.

b) Adding a stop function

Open *Spaceship Exercise 2a.fla* and save it as *Spaceship Exercise 2b.fla*. In this activity you will be adding the ability to stop immediately to the spaceship by pressing the spacebar. After the rotation script and before the calculation of position script, add the following:

```
// Making spaceship stop immediately
if (Key.isDown(Key.SPACE)){
     speed=0;
}
```

Test the new movie. Save the file.

c) Screen wrapping

Open *Spaceship Exercise 2b.fla* and save it as *Spaceship Exercise 2c.fla*. In this activity you will be adding screen wrapping. It will be a general solution, so will work on any sized **stage**. Add this script before the last } – on the last line of script.

```
// screen wrapping
spaceshipBounds = this.getBounds(_root);
if (spaceshipBounds.yMax < 0) {
     _y = Stage.height;
}
if (spaceshipBounds.yMin > Stage.height) {
     _y=0;
}
if (spaceshipBounds.xMax < 0) {
     _x=Stage.width;
}
if (spaceshipBounds.xMin > Stage.width) {
     _x=0;
}
```

Test the movie to see how it works. Save the file.

d) Modified movement

Open *Spaceship Exercise 2c.fla* and save it as *Spaceship Exercise 2d.fla*. Delete the script and add the following (you can edit your script if you wish, as much of the script is the same).

```
onClipEvent (enterFrame) {
     // makes the spaceship go forward by increasing speed by 1
     and calculates new position
     if (Key.isDown(Key.UP)) {
          speed += 1;
          xPos = Math.sin(_rotation*(Math.PI/180))*speed;
          yPos = Math.cos(_rotation*(Math.PI/180))*speed*-1;
     } else {
          //slows if key is not pressed
          xPos *= 0.97;
          yPos *= 0.97;
     }
     // you can change the rotation of the spaceship - this
     rotates 15 degrees
     if (Key.isDown(Key.LEFT)) {
          _rotation -= 15;
     }
     if (Key.isDown(Key.RIGHT)) {
          _rotation += 15;
     }
     // Making spaceship stop immediately
     if (Key.isDown(Key.DOWN)){
          xPos = 0;
          yPos = 0
     }
     // tells the spaceship to stay at 20 if reached
     if (Math.abs(speed)>20) {
          speed *= 1;
     }
```

```
        // Calculating the new position of the spaceship
        _x += xPos;
        _y += yPos;
        // screen wrapping
        spaceshipBounds = this.getBounds(_root);
        if (spaceshipBounds.yMax < 0) {
            _y = Stage.height;
        }
        if (spaceshipBounds.yMin > Stage.height) {
            _y=0;
        }
        if (spaceshipBounds.xMax < 0) {
            _x=Stage.width;
        }
        if (spaceshipBounds.xMin > Stage.width) {
            _x=0;
        }
}
```

==========

Explanation

```
        if (Key.isDown(Key.UP)) {
            speed += 1;
            xPos = Math.sin(_rotation*(Math.PI/180))*speed;
            yPos = Math.cos(_rotation*(Math.PI/180))*speed*-1;
        } else {
            //slows if key is not pressed
            xPos *= 0.97;
            yPos *= 0.97;
        }
```

The `if` statement has been modified to have an `else` component. If the up key is pressed, the speed is increased and a new position is calculated – but if the up key is not pressed then the `else` actions are used (this slows the spaceship).

e) Examples of Flash games with this style of movement

The games listed below are open source games downloaded from the internet. Copy them to your h: drive from *j:\Mmedia34\Flash\Movement\Ex 2 Examples*. You only need to copy the shockwave (swf) file and run this by double clicking on the file – the source file (fla) is also included if you wish to look at the script for the whole game. The games have similar movement controls to those you have created. Test them out.

Game	Files	Source	Comments
Car Game	cargame.fla cargame.swf	www.flashkit.com	Use arrow keys. Not Written with Flash MX.
Metioroids	index.fla index.swf	www.actionscripts.org	Uses arrow keys ans space bar. Written with Flash 5.