ActionScript for basic gaming

ActionScript 3.0 is the scripting language used to create interactivity and object movement in Adobe Flash Professional games.

The level of motion and interactivity can be simple, such as solving a jigsaw puzzle, or complex, such as guiding characters through a virtual world of danger in search of hidden treasures.

Whether you're an experienced game developer or just getting started, you'll notice several elements common to most Flash games. Simple elements might include timed player clicks or a character moving with the arrow keys. More complex elements might include characters who run and jump or maneuver to avoid flying objects. This guide begins with a few examples of Flash game types. You then use ActionScript to produce the common movements and interactions that are the building blocks of these games.

Note: This guide is intended for people who are new to ActionScript and game development but have a working knowledge of Adobe Flash Professional. You should be familiar with layers, the timeline, the library, setting properties, drawing shapes, and working with symbols and instances. You should also complete the guide "How to get started with ActionScript."

Timed player interaction games

The most basic example of a Flash game is one in which the player completes a task by clicking objects or target areas on the screen. You can use a timer to control how much time someone has to complete the task. You can also keep score by tracking how long each player takes to complete the same tasks. These games are ideal as educational games, especially for small children.

Some examples of games that include timed player interactions:

- *Eye Spy Games:* Locate (click) objects that are hidden (in plain sight), blended them in with the background artwork. This type of activity is usually timed.
- *Find the Difference (One of These Things Is Not Like the Other):* Identify (click) the differences between two photographs or artwork within the allowed time. Score can be kept by tracking how long it takes to find all the differences or by the number of differences the player identifies within the allowed time.
- *Memory/Recall:* The game board or virtual landscape includes several matching object pairs that are hidden from view. The player clicks to reveal two objects with each turn, attempting to find a match. With each turn, the player learns (and hopefully remembers) the location of each object pair. The objects can be playing cards or anything else that forms a match. For example, you could create a language game by matching English and Spanish words with the same meaning. Players can be timed to find all matches or can be given a fixed number of plays before the game ends.

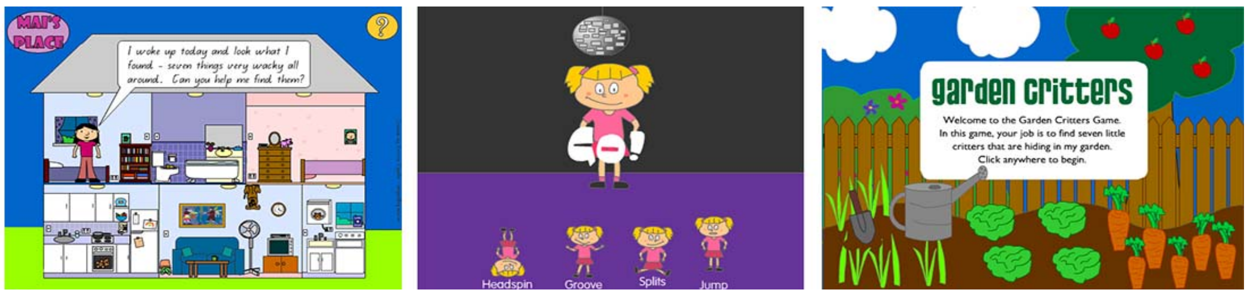

MAI'S PLACE
I woke up today and look what I found - seven things very wacky all around. Can you help me find them?
Headspin
Groove
Splits
Jump
garden critters
Welcome to the Garden Critters Game. In this game, your job is to find seven little critters that are hiding in my garden. Click anywhere to begin.

Figure 1 Timed player interaction games

Drag-and-drop games

A drag-and-drop game is one in which a player must drag objects on the screen and reposition them to complete a task. You can use drag-and-drop interactions to create a variety of games or applications, such as board games and puzzles. You can also use drag-and-drop interactions for online lessons, tests, and quizzes.

Some examples of drag-and-drop games:

- *Matching:* Players drag objects to their appropriate mates. Use this for games, online learning, and quizzes.
- *Design Your Own:* Give players the unfinished version of a product and let them complete it with a collection of objects you provide. The activity can include an empty room player get to furnish, a mannequin they get to dress, a car to which they can add trim features, and so on.
- *Puzzles:* Players drag from the collection of pieces and drop them in the puzzle frame to see where they fit.
- *Word Games:* Players drag from a tray of letters to form words on a game board.

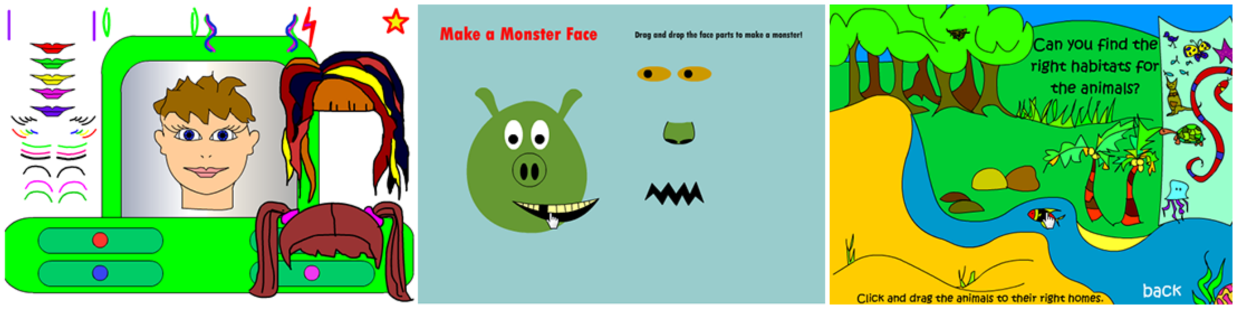

Make a Monster Face
Drag and drop the face parts to make a monster!
Can you find the right habitats for the animals?
Click and drag the animals to their right homes.
back

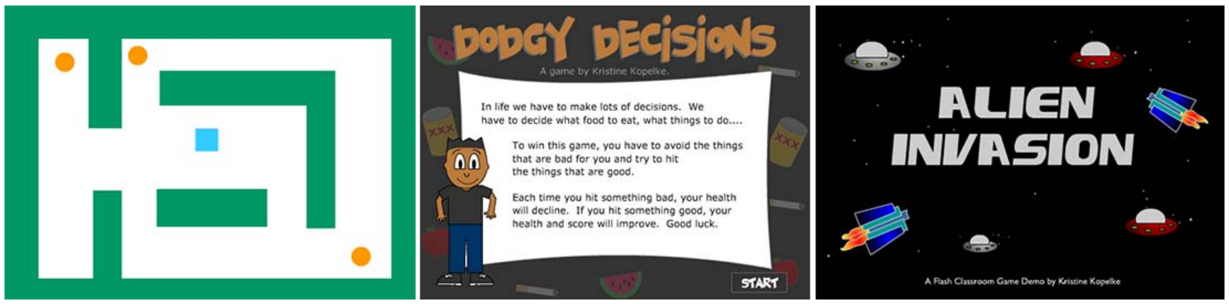
Figure 2 Drag-and-drop games

Character movement games

Character movement games involve using input controls to move one or more characters to accomplish a goal, such as following a path of discovery or interacting with objects on the screen. An example of a simple character movement game is one in which the player moves the character from point A to point B. As the character moves, the game might present the player with randomly appearing objects. Some objects are meant to be avoided. Other objects might represent positive interactions, such as gaining points or extending play.

Some examples of character movement games:

- *Adventure and Fantasy:* Players help characters run, jump, climb, swim, or fly along a journey or through a maze. They must avoid contact with dangerous objects or seek out positive targets and hidden treasures to gain bonus points or extra powers.
- *Shoot and Dodge:* A game that involves colliding with or shooting moving targets such as aliens or zombies. Players move a character to avoid being hit by hazards or intentionally colliding the character with positive targets. A classic video game called Asteroids is a good example of shooting and dodging.


DODGY DECISIONS
A game by Kristine Kopelke.

In life we have to make lots of decisions. We have to decide what food to eat, what things to do....

To win this game, you have to avoid the things that are bad for you and try to hit the things that are good.

Each time you hit something bad, your health will decline. If you hit something good, your health and score will improve. Good luck.

START
ALIEN INVASION
A Flash Classroom Game Demo by Kristine Kopelke

Figure 3 Character movement games

Using ActionScript 3.0 to create movement and interaction in games

Now that you're familiar with the popular game types, you can use ActionScript to produce the movement and interactions that are the building blocks of these games.

Creating objects and symbols for the game

Most games include artwork and other background objects that define how and where you interact with the game. For example, the goal of a game might be to escape a virtual room or building. Characters might be required to navigate a maze or avoid colliding with other objects on the screen.

One of the first steps to developing your game in Adobe Flash Professional is to design and place the artwork and other objects that appear in the game. To make the game objects interactive, they must first be converted to symbols and given a unique instance name in the Flash Properties panel.

Note: For information on creating and naming symbol instances, see the guide "Symbols, instances, and the library." For information on designing artwork and backgrounds in Flash, see the guide "How to draw and create shapes."

Using the keyboard to move characters and objects

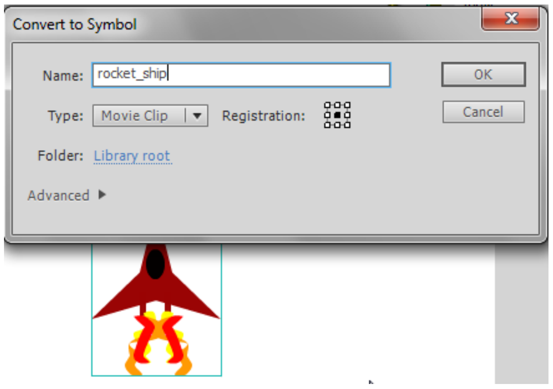
Players can interact with game characters and objects in several ways by using a variety of input devices. The most basic type of player interaction is to move objects and characters by pressing the arrow keys on the keyboard. Some games allow the character to rotate up to 360 degrees and move forward in the direction the character is pointing. This is a common movement for airplanes or spaceship characters.

The following steps show how to use ActionScript code to move a character in one direction (up) by using the Up Arrow key. After creating this simple movement, you edit the ActionScript code to include four-directional movement.

To move an object up by using the keyboard:

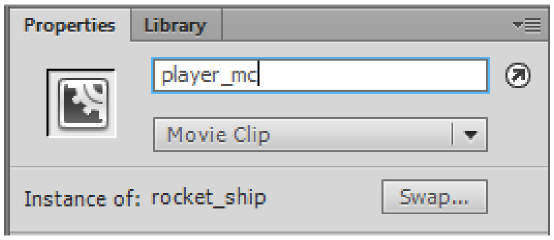
1. Start Flash and open a new blank document (ActionScript 3.0).
2. Select the artwork or sprite to use for the moving character. For this activity, you can also draw a simple shape to represent your moving character.
3. Convert the artwork or shape to a movie clip symbol. Set the symbol's registration point to the center of the object (**Figure 4**).
4. Make sure the movie clip symbol is selected on the Stage. In the Properties panel, give the character the instance name **player_mc** (**Figure 5**).

   At this point you have one symbol instance on the Stage. It exists in Frame 1 of Layer 1 in the timeline. It has the instance name player_mc.
5. In the Timeline panel, rename Layer 1 **Player** and add a new layer above it named **Actions**.



Name: rocket_ship
Type: Movie Clip
Registration:
Folder: Library root

Figure 4 Convert To Symbol dialog box



player_mc
Movie Clip
Instance of: rocket_ship

Figure 5 Properties panel

6. Select the blank keyframe in Frame 1 of the Actions layer.

   Next, you construct the ActionScript code to control the player_mc movie clip object.

7. Select Window > Actions to open the Actions panel, and make sure Script Assist is turned off.

8. Enter or copy and paste the following code to create the event listeners:

   ```
   stage.addEventListener(KeyboardEvent.KEY_DOWN, checkkeysdown);
   stage.addEventListener(KeyboardEvent.KEY_UP, checkkeysup);
   ```

   These first two lines of code set up the event listeners that "listen" for keys being pressed and released. Normally a game character will move when players press a key and stop when they release the key.

9. Enter or copy and paste the following code below the existing code to create the variables:

   ```
   var speed=10;
   var moveup=false;
   ```

   These variables keep track of the player's speed and whether the player is moving up or not.

10. Enter or copy and paste the following code below the existing code to create the function:

    ```
    function checkkeysdown(mykey:KeyboardEvent) {
    if (mykey.keyCode==Keyboard.UP) {
    moveup=true;
       }
    }
    ```

    This function deals with what happens when a key is pressed. It checks to see if the Up Arrow key is pressed and sets the variable *moveup* to true.

11. Enter or copy and paste the following code below the existing code to create another function:

    ```
    function checkkeysup(mykey:KeyboardEvent) {
    if(mykey.keyCode==Keyboard.UP) {
    moveup=false;
              }
    }
    ```

    This controls what happens when the Up Arrow key is released. It sets the variable *moveup* to false.

12. Enter or copy and paste the following code below the existing code to create the event listener:

    ```
    stage.addEventListener(Event.ENTER_FRAME, gameloop);
    function gameloop(evt:Event) {
    if (moveup==true) {
    player_mc.y-=speed;
         }
    }
    ```

    This section of code creates an event listener that runs continuously: the ENTER_FRAME listener. This causes the *gameloop* function to run, which moves the player's character.

13. Select Control > Test Movie > In Flash Professional.

14. Press the Up Arrow key to move the character.

15. Close the preview window.

To move objects in all four directions (optional):

You’ve learned how to move an object in one direction (up) by using ActionScript code. To make the characer move in other directions, you must create three more variables to track the other three movements (left, right, and downward) and additional code in each of the functions. The following is the final code used to move the character in all four directions. You can replace the code you have created so far with this entire example to move the character in all four directions.

```
stage.addEventListener(KeyboardEvent.KEY_DOWN, checkkeysdown);
stage.addEventListener(KeyboardEvent.KEY_UP, checkkeysup);
var speed=10;
var moveup=false;
var movedown=false;
var moveleft=false;
var moveright=false;
function checkkeysdown(mykey:KeyboardEvent) {
   if (mykey.keyCode==Keyboard.UP) {
   moveup=true;
   }
   if (mykey.keyCode==Keyboard.DOWN) {
   movedown=true;
   }
   if (mykey.keyCode==Keyboard.LEFT) {
   moveleft=true;
   }
   if (mykey.keyCode==Keyboard.RIGHT) {
   moveright=true;
   }
}
function checkkeysup(mykey:KeyboardEvent) {
   if (mykey.keyCode==Keyboard.UP) {
   moveup=false;
   }
   if (mykey.keyCode==Keyboard.DOWN) {
   movedown=false;
   }
   if (mykey.keyCode==Keyboard.LEFT) {
   moveleft=false;
   }
   if (mykey.keyCode==Keyboard.RIGHT) {
   moveright=false;
   }
}
```

```
stage.addEventListener(Event.ENTER_FRAME, gameloop);
function gameloop(evt:Event) {
   if (moveup==true) {
   player_mc.y-=speed;
   }
   if (movedown==true) {
   player_mc.y+=speed;
   }
   if (moveleft==true) {
   player_mc.x-=speed;
   }
   if (moveright==true) {
   player_mc.x+=speed;
   }
}
```

To create boundaries for a moving object (optional):

When playing the movie, you may have noticed the object moves beyond the Stage boundary and is gone. You can add additional code to constrain the object's movement to stay within the Stage boundary. To do this, edit the *gameloop* function as follows:

```
function gameloop(evt:Event) {
   if (moveup==true) {
      if (player_mc.y>0){
      player_mc.y-=speed;
   }
}
   if (movedown==true) {
      if (player_mc.y<400){
      player_mc.y+=speed;
   }
}
   if (moveleft==true) {
      if (player_mc.x>0){
      player_mc.x-=speed;
   }
}
   if (moveright==true) {
      if (player_mc.x<550){player_mc.x+=speed;
      }
   }
}
```

To move and rotate an object 360 degrees by using the keyboard:

1. Start Flash and open a new blank document (ActionScript 3.0).
2. Select the artwork or sprite to use for the character. You can draw a simple shape to represent your character.
3. Convert the artwork or shape to a movie clip symbol. Set the symbol's registration point to the center of the object.
4. Make sure the movie clip symbol is selected on the Stage. In the Properties panel, give the character the instance name **player_mc**.

   At this point you have one symbol instance on the Stage. It exists in Frame 1 of Layer 1 in the timeline. It has the instance name player_mc.
5. In the Timeline panel, rename Layer 1 **Player** and add a new layer above it named **Actions**.
6. Select the blank keyframe in Frame 1 of the Actions layer.

   Next, you construct the ActionScript code to control the player_mc movie clip object.
7. Select Window > Actions to open the Actions panel, and make sure Script Assist is turned off.
8. Enter or copy and paste the following code to add the variables:

   ```
   var rotateLeft=false;

   var rotateRight=false;
   ```

   These variables keep track of the direction of rotation. They are set to false at the start of the game. When the player presses the Right Arrow key, *rotateRight* will be set to true, and similarly *rotateLeft* is set to true when the Left Arrow key is pressed.
9. Enter or copy and paste the following code to detect keypresses and set the appropriate values:

   ```
   stage.addEventListener(KeyboardEvent.KEY_DOWN, startRotation);

   function startRotation(e:KeyboardEvent):void{

      if (e.keyCode==Keyboard.LEFT){

         rotateLeft=true;

      }

      if (e.keyCode==Keyboard.RIGHT){

      rotateRight=true;

      }

   }
   ```

   You must also detect when the keys are released and set the variables back to false.
10. Enter or copy and paste the following code to detect when keys are released:

   ```
   stage.addEventListener(KeyboardEvent.KEY_UP, stopRotation);

   function stopRotation(e:KeyboardEvent):void{

      if (e.keyCode==Keyboard.LEFT){

         rotateLeft=false;

      }

      if (e.keyCode==Keyboard.RIGHT){

         rotateRight=false;

      }

   }
   ```

So far, this code does not move the character. It only sets the variables to true or false as a result of keypresses.

11. Enter or copy and paste the following code to make the player rotate in the correct direction:

```
stage.addEventListener(Event.ENTER_FRAME, moveplayer);
function moveplayer(e:Event):void{
   if (rotateLeft==true){
      player_mc.rotation-=10;
   }
   if (rotateRight==true){
      player_mc.rotation+=10;
   }
}
```

The first line in the code adds an event listener, ENTER_FRAME, that runs constantly. This causes the function *moveplayer* to run constantly. The *mc_player* symbol then rotates in the appropriate direction by 10 degrees. Next you need to make the player character move forward. To do this you will use polar coordinates to plot a point in front of the character based on its angle of rotation. The character moves to the position of this new point. The polar coordinates method in ActionScript uses radians rather than degrees, so you must first convert the angle of the player to radians. You do this by multiplying the angle by pi divided by 180 (**Figure 6**). However, because the coordinates system in Flash is different from a normal coordinates system (it is rotated 90 degrees), you must first subtract 90 from the player's angle of rotation. Once you have calculated the angle in radians, you can plot a point in front of the player by using the *Point.polar()* method in ActionScript.

$$\text{radians} = \text{degrees} \times \frac{\pi}{180}$$

Figure 6 Formula used to convert the angle of the player to radians

12. Enter or copy and paste the following code to move the character forward in steps of 5 pixels.

Note: This code must be placed inside the *moveplayer* function, just before the last End bracket (}) in the code.

```
var radians=Math.PI/180 * (player_mc.rotation-90);
var newLocation=Point.polar(5, radians);
player_mc.x+=newLocation.x;
player_mc.y+=newLocation.y;
```

13. Select Control > Test Movie > In Flash Professional.

14. Use the arrow keys to move and rotate the character. The character moves forward continuously.

15. Close the preview window.

To create boundaries for a moving object (optional):

To prevent the character from moving offscreen, add the following code to the end of the *movieplayer* function, assuming a Stage size of 550 x 400:

```
if (player_mc.x>550){
player_mc.x=0;
}
if (player_mc.x<0){
player_mc.x=550;
}
if (player_mc.y>400){
player_mc.y=0;
}
if (player_mc.y<0){
player_mc.y=400;
}
```

Now if you let the player character move in one direction until it exits the Stage, it will re-enter the opposite side of the Stage.

To make a character jump:

1. Start Flash and open a new blank document (ActionScript 3.0).
2. Select the artwork or sprite to use for the jumping character.
3. Convert the artwork or shape to a movie clip symbol. Set the symbol's registration point to the center of the object (**Figure 7**).
4. Make sure the movie clip symbol is selected on the Stage. In the Properties panel, give the character the instance name **jumpman_mc** (**Figure 8**).

   At this point you have one symbol instance on the Stage. It exists in Frame 1 of Layer 1 in the timeline. It has the instance name jumpman_mc.

   Note: You might also want to add background objects, including moving objects the jumping man must avoid by jumping over. To learn how, refer to the steps on creating moving objects and adding collision detection later in this guide.
5. In the Timeline panel, rename Layer 1 **Player** and add a new layer above it named **Actions**.

   Next, you add animation to make the player jump.
6. Double-click the movie clip symbol on the Stage to open the symbol in editing mode.
7. Select the artwork or sprite you used to create the character symbol and convert that to another movie clip symbol. Name it anything you want.

   Note: The artwork needs to be a symbol to create the motion tween animation.
8. Insert a new frame (not a keyframe) in Frame 10 of the symbol's timeline.
9. Click Frame 1 and select Insert > Motion Tween.
10. Click in Frame 10 and drag the symbol upward to a position that is about twice its original height.
11. Insert a new frame (not a keyframe) in Frame 20.
12. Click in Frame 20 and drag the symbol back down to its starting position (landing position after jumping).

    Note: You can also use the position values in the Properties panel to match the exact position of the object in Frame 1.

    The symbol now includes animation of the character jumping (**Figure 9**).

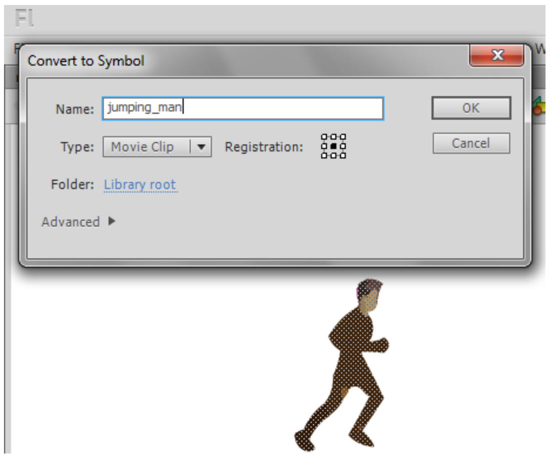
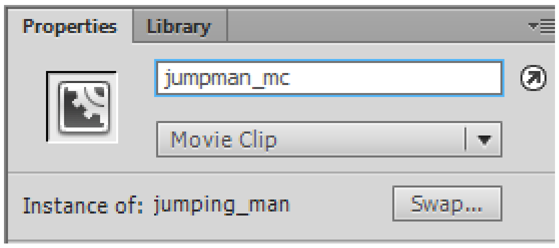

Convert to Symbol
Name: jumping_man
Type: Movie Clip
Registration:
Folder: Library root

Figure 7 Convert To Symbol dialog box



jumpman_mc
Movie Clip
Instance of: jumping_man

Figure 8 Properties panel

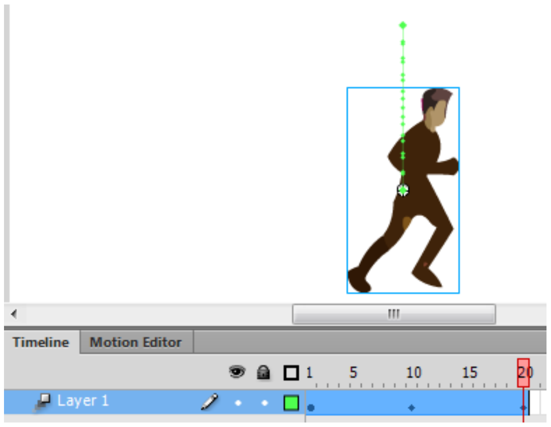


Figure 9 Symbol in editing mode

13. Play the animation in the Timeline panel to see the object jump.

    Right now the object will jump up and down continuously. You need to add a stop action to the symbol so the character remains still until it receives instructions to jump.

14. In the symbol's timeline, add a new layer named **Actions**.

15. Click the blank keyframe in Frame 1 of the Actions layer, open the Actions panel, and add the following stop action:

    ```
    stop();
    ```

    This tells the movie clip that the first thing it should do is stop. The jumping animation will not occur until the symbol receives instructions.

16. Close the Actions panel. Click Scene 1 to exit symbol editing mode and return to the main timeline.

    Now you can add the code to make the character jump.

17. Click in Frame 1 in the Actions layer in the main timeline.

18. Select Window > Actions to open the Actions panel, and make sure Script Assist is turned off.

19. Enter or copy and paste the following code to make the character jump:

    ```
    stage.addEventListener(KeyboardEvent.KEY_DOWN, jump);
    function jump(e:KeyboardEvent):void {
    jumpman_mc.play();
    }
    ```

    This code adds a new event listener that detects any keypress. When a key is pressed, the function *jump* runs and instructs the jumpman_mc movie clip symbol (the jumping animation) to play.

20. Close the Actions panel. Select Control > Test Movie > In Flash Professional.

    The character is initially stopped on the Stage. Press any key to make the character jump.

    You can add additional animation to the character by editing the symbol. For example, instead of using a static image, you can use an animated sprite that appears to be running, climbing, swimming, or flying. For information on creating animated sprites, see the guide "How to use sprite sheets."

    You can combine the jumping character with an animation of a moving background that makes the character appear to be moving forward, or place the character among other moving objects.

    If your character will jump or move to avoid other objects, you can add collision detection to detect when the character hits another object and provide instructions on what to do when a collision occurs.

21. Close the preview window.

Creating scrolling backgrounds and moving objects that collide with your characters.

To achieve the visual effect that a character is traveling across the screen, you create a moving or scrolling background. Design a repeating background image that is wider than the Stage and use a tween animation to scroll the background across the screen. To make the character appear to be moving to the right, your background needs to scroll to the left. To make the character travel to the left, scroll the background to the right.

To add more realism and excitement to the scene, you can create additional objects, such as obstacles or collectibles, and then use a *gameloop* function to move these objects continuously across the screen.

By adding collision detection, you can also control what happens when your player comes in contact with other objects on the screen. For example, you can create flying objects that threaten the health or life of the player character. The player can move, turn, or jump to avoid or cause a collision. You can include additional code that creates a consequence of avoiding or colliding with an object. For example, the consequence can be losing a player’s life or changing the player’s score (up or down).

To create a scrolling background:

1. Continue where you left off in the preceding activity.
2. Add a new layer to the main timeline and name it **Background**. Move the Background layer below all other layers in the Timeline panel.
3. Import or draw the artwork for your background. The artwork needs to be at least as wide as the Stage (**Figure 10**).
4. Select all of the artwork in the Background layer.

   Note: If your document includes other objects, be sure to select only the artwork in the Background layer.
5. Convert the background artwork to a new movie clip symbol named **Background**.
6. Double-click the Background symbol to open it in symbol editing mode.
7. Zoom out until you can see the entire Stage and the empty gray areas to the left and right of the Stage. A zoom level of 25% should work.
8. Copy and paste all of the background artwork and position a copy exactly to the right of the original artwork. Make a second copy and paste it to the left of the original artwork.

   You now have a background that repeats three times and extends well beyond the Stage boundaries (**Figure 11**).
9. Select all of the artwork (all three copies) and convert it to a new movie clip symbol. You can name it anything you want.
10. Position the symbol so the artwork aligns with the bottom and left edges of the Stage.

    This is the start point of the scrolling background motion tween (**Figure 12**).

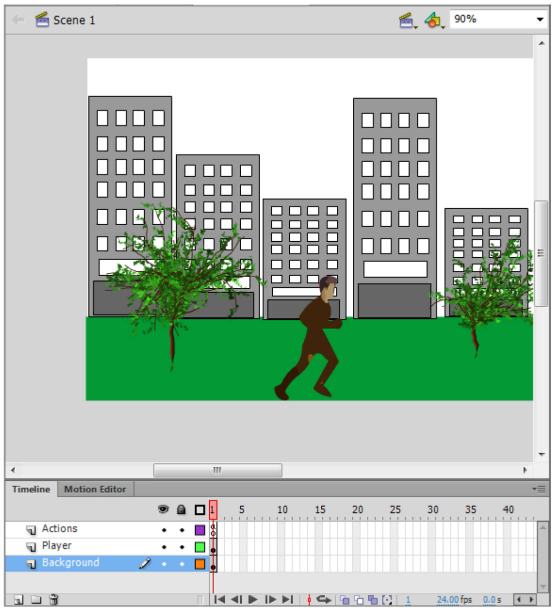


Figure 10 Artwork in the Background layer

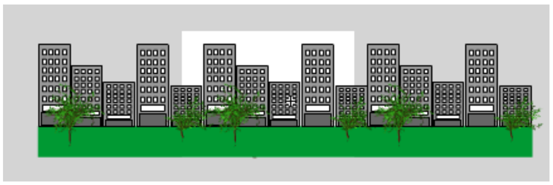


Figure 11 Repeating background artwork

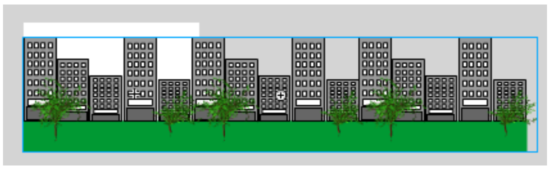


Figure 12 Start position of the motion tween

11. Add a frame (not a keyframe) farther down the timeline, such as in Frame 30.

    Note: The location of the end frame will determine how fast the background scrolls and, therefore, how fast the character appears to travel across the scene.

12. Click Frame 1 in the Background symbol's timeline and select Insert > Motion Tween.

13. Click in the end frame of the motion tween span and then Shift-drag the symbol to the left until its right edge aligns with the right edge of the Stage (**Figure 13**).

    Note: Holding down Shift as you drag locks the vertical position of the symbol. Make sure the Stage is covered during all frames of the motion tween and the background artwork moves horizontally but maintains the same vertical position throughout.

14. Play the motion tween in the Timeline panel.

15. Click Scene 1 to exit symbol editing mode and return to the main timeline.

16. Select Control > Test Movie > In Flash Professional.

    The repeated artwork in the background movie loops continuously, causing the background to scroll left. If your Stage includes a player character in another layer, the character appears to travel to the right.

    Note: If the background scrolls too fast or too slowly, open the Background symbol in editing mode and add or remove frames in the motion tween span.

17. Close the preview window.

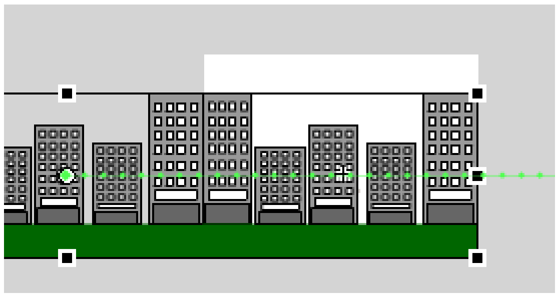


Figure 13 End position of the motion tween

To create a moving object:

1. Continue where you left off in the preceding activity.

2. Add a new layer to the main timeline and name it **Obstacle**. Make sure the Obstacle layer is above the Background and Player layers. Lock all layers except the Obstacle and Actions layers (**Figure 14**).

3. With the Obstacle layer selected, import or draw the artwork for the moving object.

   In this example, we created a yellow star the player must jump over to avoid collision (**Figure 15**).

4. Convert the object to a movie clip symbol. Select the instance of the object on the Stage and name it **obstacle_mc** in the Properties panel.

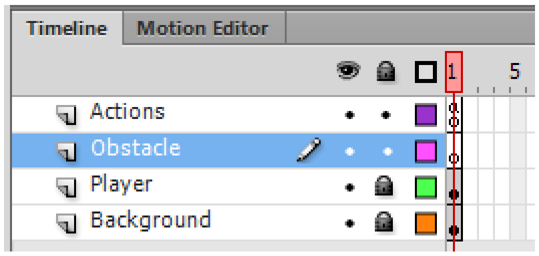


Figure 14 Obstacle layer in the Timeline panel

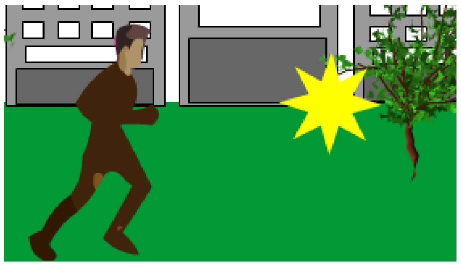


Figure 15 Star artwork for the moving object

5. **5.** Click in Frame 1 of the Actions layer, and select Window > Actions to open the Actions panel. Make sure Script Assist is turned off.

   Note: If your Actions panel already includes the code to make the player character jump, be sure to add all new code below this in the Actions panel.

   Next, you add the code to make the moving obstacle (in this case the star) move. This is done by repeatedly changing the object's position on the X axis. The star moves from right to left—that is, in a negative direction. You include an event listener to make the object move continuously. The event listener runs a function called *gameloop*.

6. **6.** Enter or copy and paste the following code to create the Enter Frame event listener and *gameloop* function:

```
stage.addEventListener(Event.ENTER_FRAME, gameloop);
   function gameloop(e:Event):void{
   obstacle_mc.x-=20;
}
```

   The line *obstacle_mc.x-=20;*changes the obstacle_mc symbol (star) on the X axis by 20 pixels, moving it in a negative direction (to the left).

7. **7.** Select Control > Test Movie > In Flash Professional.

   The moving object (obstacle_mc) moves from left to right and then disappears off the Stage to the left. What you really want is for yellow stars to continuously come at the player.

8. **8.** Close the preview window and return to Flash.

   You could make more yellow stars and move them too, but it's more efficient to reuse the same symbol. When the object moves off the screen, you can have it reappear by repositioning it.

9. **9.** Edit or copy and paste the three new lines in the following code so the *gameloop* function looks like this:

```
function gameloop(e:Event):void{
obstacle_mc.x-=20;
   if (obstacle_mc.x<-100){
   obstacle_mc.x=650;
   }
}
```

10. **10.** Close the Actions panel. Select Control > Test Movie > In Flash Professional.

    This time the object moves left off the screen and then re-enters on the right and repeats the move continuously. It appears that moving objects (stars) continually fly into the path of the player character.

    Right now the objects pass right through the player character as if it wasn't there. You can change that by adding ActionScript code to detect when one object collides with another and then giving Flash instructions on how the collision affects the objects.

11. **11.** Close the preview window.

Next, you add the code that gives the character player 20 lives. In Flash games, this is referred to as the player's *health*. Each time the player character collides with an obstacle (the moving stars), he loses health. After 20 collisions, the player's health is gone and the game is over.

To add collision detection:

1. Continue where you left off in the preceding activity.
2. Add a new layer to the main timeline and name it **Health**. Make sure the Health layer is above the Background, Player, and Obstacle layers. Lock all layers except the Health and Actions layers (**Figure 16**).
3. Click in Frame 1 of the Health layer.
4. Select the Text tool in the Tools panel.
5. In the Properties panel, select Classic Text in the Text Engine menu. Select Dynamic Text in the Text Type menu (**Figure 17**). Make the text Arial, Regular, 32 pt, and a color that stands out against the background.

   Note: The text style must be Regular or it will not show up.

   Dynamic text boxes can be filled in or updated by ActionScript. They are useful for displaying scores or lives in a game.
6. Drag to draw a text box in the upper-right corner of the Stage (**Figure 18**).
7. Make sure the text box is still selected. In the Properties panel, enter **health_txt** as the instance name (**Figure 19**).

   By giving the text box an instance name, you can refer to it in your ActionScript code.

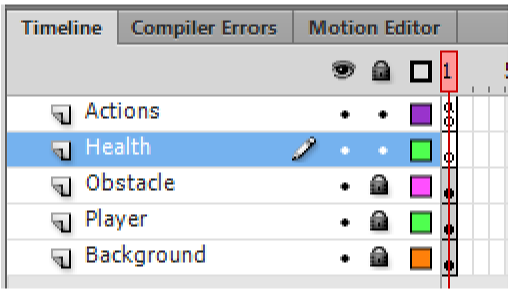
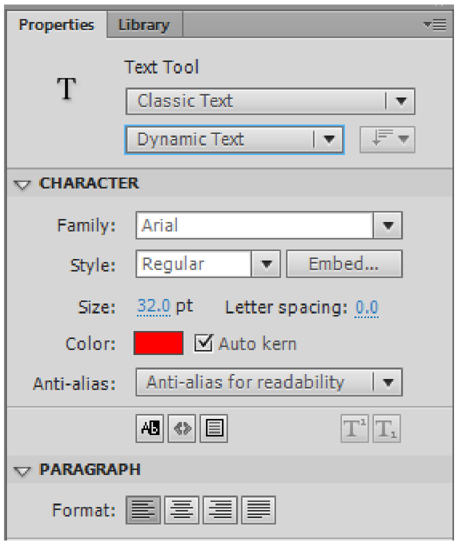


Figure 16 Health layer in the Timeline panel


Properties | Library

Text Tool
Classic Text
Dynamic Text

CHARACTER
Family: Arial
Style: Regular Embed...
Size: 32.0 pt Letter spacing: 0.0
Color: ☑ Auto kern
Anti-alias: Anti-alias for readability

PARAGRAPH
Format:

Figure 17 Properties panel, Text tool properties

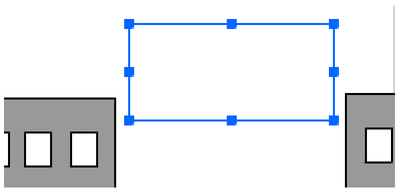
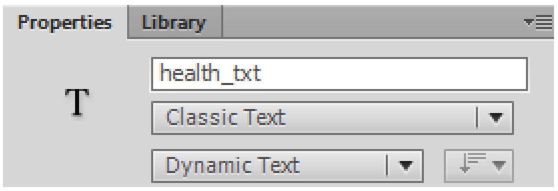


Figure 18 Dynamic text box


Properties | Library

health_txt
Classic Text
Dynamic Text

Figure 19 Text instance name in the Properties panel

8. Click in Frame 1 of the Actions layer and open the Actions panel. Place the insertion point before the first line of code in the Actions panel and press Enter (Windows) or Return (Mac OS) to add a blank line.
9. Enter or copy and paste the following code at the top of the Actions panel to establish the starting health of the player character:

   ```
   var hitObstacle:Boolean=false; // keeps track if obstacle is hit
   var health=20;
   health_txt.text=health.toString();
   ```

   The first line sets up a variable that remembers the player character's current health. At the start of the game, jumpman_mc has 20 lives.

   The next line displays the health of the character on the screen in the dynamic text box. The health is represented by a number, but a text box can display only a string of characters. The number has to be converted to a string, so you include *toString()* at the end of the code.
10. Locate the *gameloop* function you created in the preceding activity. Enter or copy and paste the new lines in the following code so the *gameloop* function looks like this:

    ```
    function gameloop(e:Event):void{
    obstacle_mc.x-=20;
       if (obstacle_mc.x<-100){
       obstacle_mc.x=650;
       }
       if (jumpman_mc.hitTestObject(obstacle_mc)) {
       if (hitObstacle==false){ //only subtract health if hitObstacle is false
          health--;
       }
      hitObstacle=true;
          health_txt.text=health.toString();
          }
       }
    ```

    The new lines of code check to see if jumpman_mc hits obstacle_mc. If so, the health changes in the text box. You use a special function called *hitTestObject* to test if one object collides with another. This is used a lot in games and can detect whether a player object has been hit by a bullet or a car or collides with any other object.
11. Close the Actions panel. Select Control > Test Movie > In Flash Professional.

    The player character's health begins at 20. Each time he collides with a moving obstacle (yellow stars), his health reduces by 1.

    Note: If each collision results in a health reduction greater than 1, that means Flash is having difficulty identifying the boundary of each object. Simple hit detection works best with rectangular shapes.

    You can also modify the code to increase the health each time the player collects (collides with) an object. To do this, add an additional *hitTestObject* function and change health– – to health++.
12. Close the preview window.

Next, you add a score board that credits players each time they achieve a goal. In this example, the goal is to avoid the moving objects (stars) created in the previous activities. So in addition to losing health when the character collides with an obstacle, they also score points when they avoid an obstacle.

To keep score:

1. Continue where you left off in the preceding activity.
2. Add a new layer to the main timeline and name it **Score**. Make sure the Score layer is above the Background, Player, Obstacle, and Health layers. Lock all layers except the Score and Actions layers (**Figure 20**).
3. Click in Frame 1 of the Score layer.
4. Select the Text tool in the Tools panel.
5. In the Properties panel, select Classic Text in the Text Engine menu. Select Dynamic Text in the Text Type menu. Make the text Arial, Regular, 32 pt, and a color that stands out against the background.

   Dynamic text boxes can be filled in or updated by ActionScript. They are useful for displaying scores or lives in a game.
6. Drag to draw a text box in the upper-left corner of the Stage (**Figure 21**).
7. Make sure the text box is still selected. In the Properties panel, enter **score_txt** as the instance name (**Figure 22**).

   By giving the text box an instance name, you can refer to it in your ActionScript code.
8. Click in Frame 1 of the Actions layer and select Window > Actions to open the Actions panel.

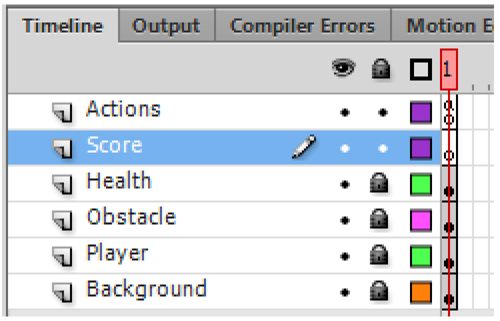


Figure 20 Score layer in the Timeline panel

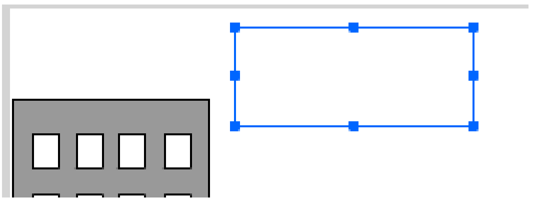
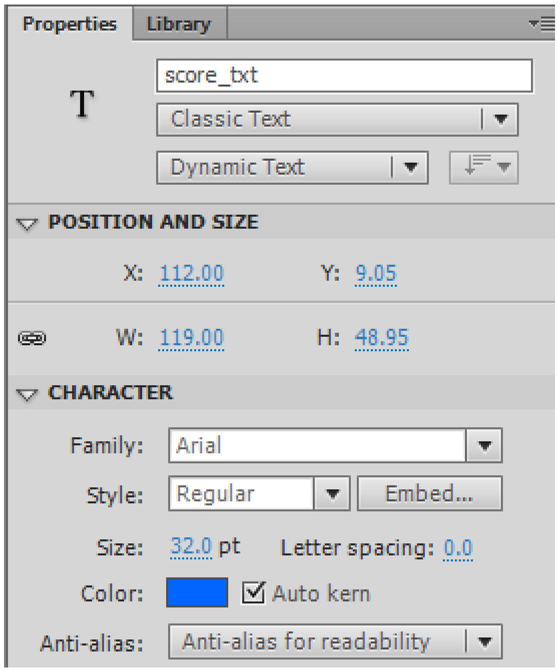


Figure 21 Dynamic text box



score_txt
Classic Text
Dynamic Text

POSITION AND SIZE
X: 112.00 Y: 9.05
W: 119.00 H: 48.95

CHARACTER
Family: Arial
Style: Regular
Size: 32.0 pt Letter spacing: 0.0
Color: ☑ Auto kern
Anti-alias: Anti-alias for readability

Figure 22 Properties panel, Text properties

9. Place the insertion point before the first line of code in the Actions panel and press Enter (Windows) or Return (Mac OS) to add a blank line.
10. Enter or paste the following code at the top of the Actions panel:

    ```
    var score=0;
    score_txt.text=score.toString();
    ```

    These two lines set up the score and place it on the screen. The first line creates a variable that remembers the current score. The second line displays the score in the text box on the screen.
11. In the Actions panel, scroll down to the *gameloop* function. Enter or copy and paste the two new lines in the following code so the *gameloop* function looks like this:

    ```
    function gameloop(e:Event):void{
    obstacle_mc.x-=20;
       if (obstacle_mc.x<-100){
       obstacle_mc.x=650;
       hitObstacle=false; // reset hitObstacle when obstacle moves off screen
       score++;
       score_txt.text=score.toString();
       }
       if (jumpman_mc.hitTestObject(obstacle_mc)) {
       if (hitObstacle==false){ // only subtract health if hitObstacle is false
          health--;
       }
      hitObstacle=true;
          health_txt.text=health.toString();
          }
       }
    ```

    The additional lines of code add 1 to the score (score++) each time the flying-star moving object (obstacle_mc) reaches the left side of the screen.
12. Close the Actions panel. Select Control > Test Movie > In Flash Professional.

    Each time the moving object reaches the left side of the screen, the score increases by 1. Notice the score increases even when the player collides with the obstacle. The player's score is directly related to how long she can maintain health.
13. Close the preview window.

The game should end when the player's health reaches zero. Next you add the code that does this.

To end the game:

1. Select Insert > Scene.

   A new blank scene and timeline appear. This is called Scene 2. Scene 1 still exists, but it's hidden from view.

2. Select the Text tool. In the Properties panel, change the text type to Static Text (**Figure 23**). Change the Family (font) to something different from Arial.

   Your static text should be from a different font family than you used for the dynamic text objects.

3. Create two text objects on the Stage that read GAME OVER and SCORE:.

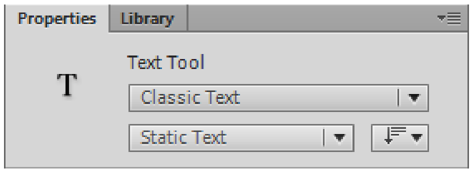
4. Add a new layer and name it **Final Score**.

5. On the Final Score layer, draw a third text box to the right of SCORE: and change the text type to Dynamic. Change the Family (font) to Arial and make sure the Style is Regular.

   Scene 2 now includes three text objects (**Figure 24**). The dynamic text box is where you'll display the player's final score after the game ends.

6. Select the dynamic text box and, in the Properties panel, enter **score_txt** as the instance name.

7. Add a new layer in the Scene 2 timeline and name it **Actions**.



Text Tool
Classic Text
Static Text

Figure 23 Properties panel, text properties

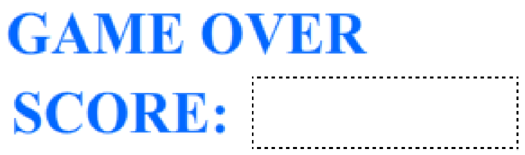


Figure 24 Static and dynamic text

8. Click in Frame 1 of the Actions layer and Select Window > Actions to open the Actions panel.

   The Actions panel is empty because this is a new scene.

9. Enter or copy and paste the following code to display the player's score in the dynamic text box in Scene 2.

   ```
   score_txt.text=score.toString();
   ```

10. Close the Actions panel.

11. Click the Edit Scene button (**Figure 25**) and choose Scene 1.

    Scene 2 closes and you return to Scene 1.

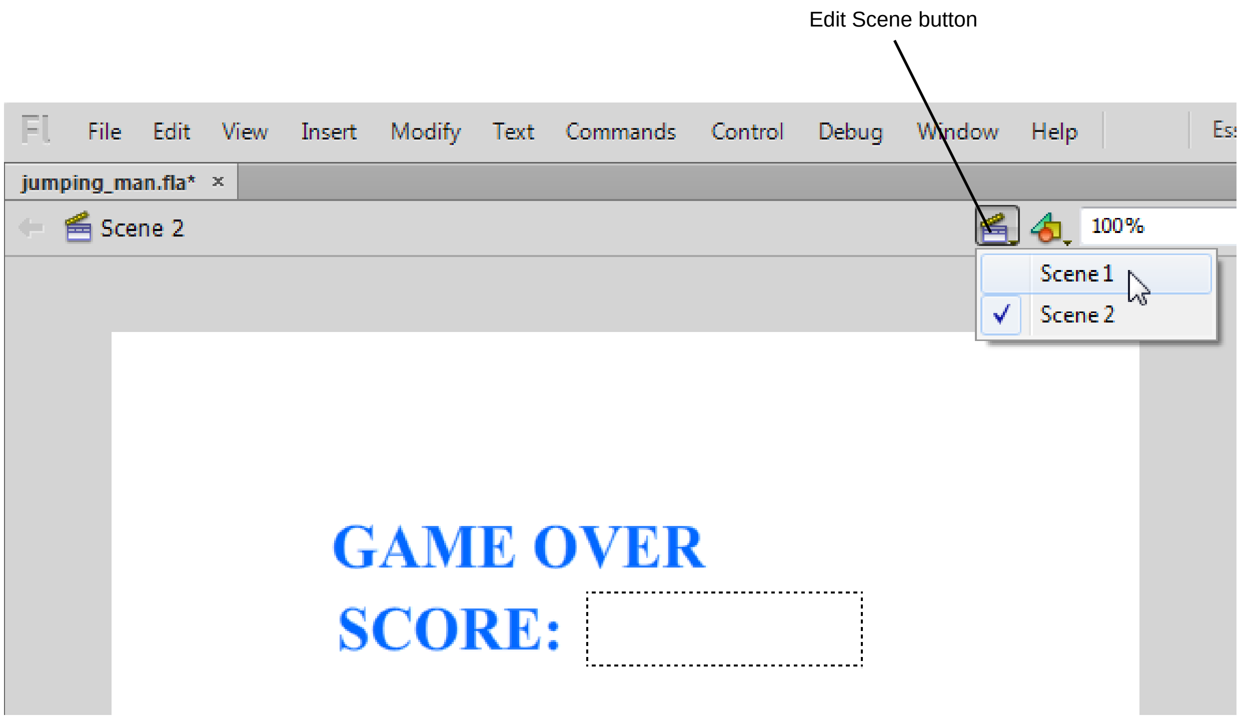


Figure 25 Scene 2

12. Click the Actions layer in the Timeline panel and open the Actions panel.

13. Insert a new blank line at Line 1 and enter or copy and paste the following code to prevent the Flash movie from looping between Scenes 1 and 2.

    ```
    stop();
    ```

    Next you add the code to end the game when the player reaches 0 health.

14. In the Actions panel, scroll down to the *gameloop* function. Enter or copy and paste the new lines in the following code so the *gameloop* function looks like this:

```
function gameloop(e:Event):void{
obstacle_mc.x-=20;
   if (obstacle_mc.x<-100){
   obstacle_mc.x=650;
   hitObstacle=false; // reset hitObstacle when obstacle moves off screen
   score++;
   score_txt.text=score.toString();
   }
   if (jumpman_mc.hitTestObject(obstacle_mc)) {
   if (hitObstacle==false){ // only subtract health if hitObstacle is false
      health--;
   }
  hitObstacle=true;
      health_txt.text=health.toString();
  if (health<=0){
      stage.removeEventListener(Event.ENTER_FRAME, gameloop);
      stage.removeEventListener(KeyboardEvent.KEY_DOWN, jump);
         gotoAndStop(1, "Scene 2");
      }
   }
}
```

The additional lines of code check to see if the jumpman_mc character's health is less than or equal to zero. If it is, Flash goes to and stops on Frame 1 of Scene 2.

The extra two lines of code prevent errors from appearing when the game ends. If you add event listeners, it is important that you remove them when the code is finished.

15. Close the Actions panel. Select Control > Test Movie > In Flash Professional.

Note: If your movie does not play properly, you may have accidentally removed the final right brace or added an extra brace when editing the gameloop function.

As long as the jumpman character avoids the moving obstacles, the score continues to increase. As soon as the player's health is depleted, the game ends and you are taken to the GAME OVER screen in Scene 2. The final score is shown.

Moving and dragging objects by using the mouse

You can use drag-and-drop interactions to create a variety of games or applications, including quizzes, jigsaw puzzles, and touch-screen navigation. Players move objects by dragging or directing them with the mouse pointer.

The following activity shows how to use a drag-and-drop interaction to create a simple game. The game involves dragging objects to design a face, but you can just as easily use drag-and-drop interactions to customize a product, build a puzzle, or match objects in a game or quiz.

Here is an example of a finished game (**Figure 26**). Players drag the objects on the right to create a face on the left.

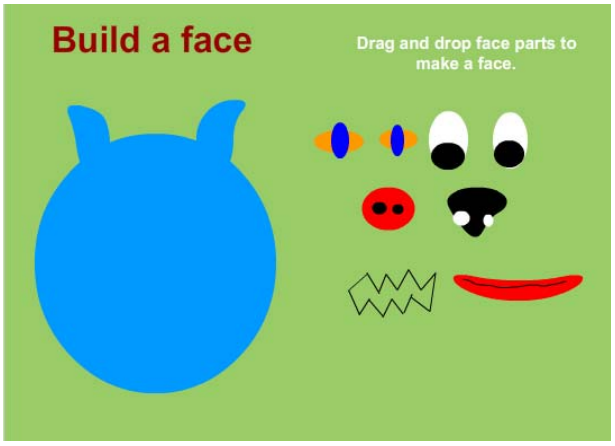


Figure 26 Drag-and-drop game example

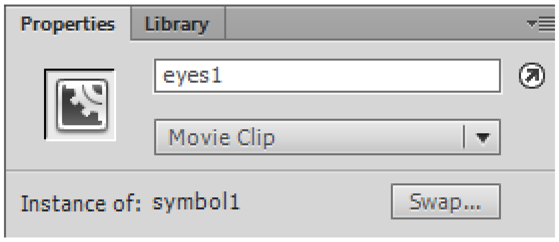
To create a drag-and drop interaction:

1. Start Flash and open a new blank document (ActionScript 3.0). In the Properties panel, select a background color for the document.
2. Select the Text tool and change the text type to Static Text. Enter some simple instructions for the game (**Figure 26**).
3. In the Timeline panel, rename Layer 1 **Head**.
4. Use the shape tools to draw a face shape that is about half the size of the Stage. Place it toward the left side of the Stage. Lock the Head layer in the Timeline panel.
5. Create a new layer named **Eyes 1** and draw a pair of eyes on the right side of the Stage.
6. Select the eyes and convert the drawing into a new movie clip symbol.

   The head on the left is the base object onto which players will drag objects. The Eyes 1 symbol is the first item in the collection of draggable objects (**Figure 27**).
7. Select the Eyes 1 symbol on the Stage. In the Properties panel, enter **eyes1** as the instance name (**Figure 28**).

   Next, you create the ActionScript code to make the eyes1 symbol draggable.

Figure 27 Head and Eyes 1



eyes1

Movie Clip

Instance of: symbol1

Figure 28 Properties panel, symbol instance name

8. Insert a new layer named **Actions**.
9. Select the Actions layer, open the Actions panel, and make sure Script Assist is turned off.
10. Enter or copy and paste the following code in the Actions panel:

    ```
    eyes1.addEventListener(MouseEvent.MOUSE_DOWN,pickup);
    ```

    This line of code adds an event listener to the eyes1 object. An event listener listens for a specific event. In this case we've told it to listen for a mouse button being pressed down (MouseEvent.MOUSE_DOWN). This line also tells Flash what it should do when it detects this event. It needs to run a function called pickup. You'll add the functions in a moment, but first add another event listener to detect when the mouse button is released, dropping the object.
11. Enter or copy and paste the next line of code below the first event listener:

    ```
    eyes1.addEventListener(MouseEvent.MOUSE_UP,drop);
    ```
12. Enter or copy and paste this additional code below the two event listeners for the eyes1 symbol:

    ```
    eyes1.buttonMode=true;
    ```

    This tells the eyes1 symbol to appear as a button that changes the appearance of the pointer when it's placed over the symbol. This tells the player it's a draggable object.

    Next, you add the two functions that tell Flash what to do when an object is picked up or dropped.
13. Enter or copy and paste this additional code below the first three lines of code:

    ```
    function pickup (event:MouseEvent):void{
       event.target.startDrag()
       }
    function drop (event:MouseEvent):void{
          event.target.stopDrag();
       }
    ```
14. Close the Actions panel. Select Control > Test Movie > In Flash Professional.

    You can drag the eyes and drop them anywhere on the screen.
15. Close the preview window.
16. Use the drawing tools to create additional objects for the face. For example, draw several different sets of eyes, noses, and mouths from which to choose. Put each object in its own layer in the Timeline panel.
17. Convert each object to a separate movie clip symbol and give each object a unique instance name on the Stage.

18. Afer you add some objects, change the code to include separate event listeners for each object.

    You don't have to add new functions. The same functions work for each object. The finished code will look similar to this example:

```
eyes1.addEventListener(MouseEvent.MOUSE_DOWN,pickup);
eyes1.addEventListener(MouseEvent.MOUSE_UP,drop);
eyes1.buttonMode=true;

function pickup (event:MouseEvent):void{
   event.target.startDrag();
}
function drop (event:MouseEvent):void{
   event.target.stopDrag();
}

eyes2.addEventListener(MouseEvent.MOUSE_DOWN,pickup);
eyes2.addEventListener(MouseEvent.MOUSE_UP,drop);
nose1.addEventListener(MouseEvent.MOUSE_DOWN,pickup);
nose1.addEventListener(MouseEvent.MOUSE_UP,drop);
nose2.addEventListener(MouseEvent.MOUSE_DOWN,pickup);
nose2.addEventListener(MouseEvent.MOUSE_UP,drop);
mouth1.addEventListener(MouseEvent.MOUSE_DOWN,pickup);
mouth1.addEventListener(MouseEvent.MOUSE_UP,drop);
mouth2.addEventListener(MouseEvent.MOUSE_DOWN,pickup);
mouth2.addEventListener(MouseEvent.MOUSE_UP,drop);
nose1.buttonMode=true;
eyes2.buttonMode=true;
nose2.buttonMode=true;
mouth1.buttonMode=true;
mouth2.buttonMode=true;
```

19. Save and test the movie.

    Note: This sample code assumes you created new symbols with instance names of eyes2, nose1, nose2, mouth1, and mouth2. If you did not, the sample code will return errors when you test your movie.

Adding timers

Timers are used in games for many purposes. For example, a timer can trigger events that happen at timed intervals or as a countdown to the end of the game.

In this activity, you create a simple countdown timer and then use the timer to end the game.

To control the allowed time to complete a game:

1. Start Flash and open a new blank document (ActionScript 3.0).
2. Rename Layer 1 **Text**.
3. Select the Text tool. In the Properties panel, make the font Arial, the style Regular, and the size 32. Select a color for the text that will represent a countdown clock on screen.
4. In the Properties panel, select Classic in the Text Engine menu and Dynamic Text in the Text Type menu.
5. In the Paragraph section of the Properties panel, set the text alignment to Align Right.

   You've set the properties for the countdown clock (**Figure 29**). Next you draw the dynamic text box on the Stage.
6. Using the Text tool, drag to draw the new dynamic text box on the screen where you want the countdown clock to appear. Make sure the text box is large enough to display the timer according to the font size you selected.
7. Select the dynamic text box on the Stage and give it the instance name **timetxt** in the Properties panel (**Figure 30**).
8. Insert a new layer and name it **Actions**.
9. Select the Actions layer and open the Actions panel. Make sure Script Assist is turned off.

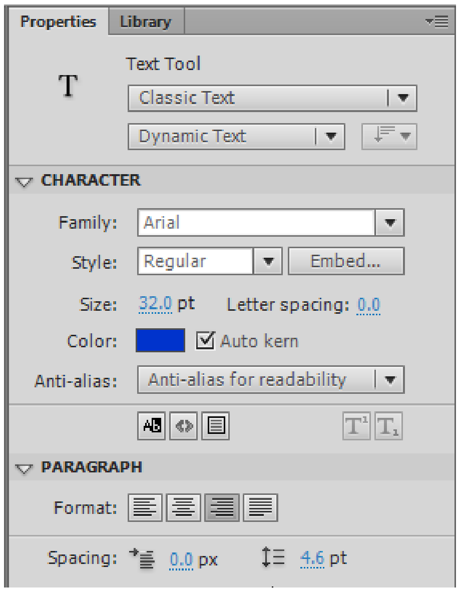

Text Tool
Classic Text
Dynamic Text
CHARACTER
Family: Arial
Style: Regular
Size: 32.0 pt Letter spacing: 0.0
Color: Auto kern
Anti-alias: Anti-alias for readability
PARAGRAPH
Format:
Spacing: 0.0 px 4.6 pt

Figure 29 Properties panel, Text properties

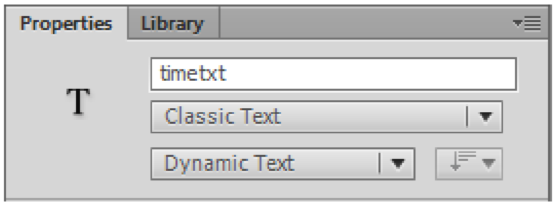

timetxt
Classic Text
Dynamic Text

Figure 30 Properties panel, instance name

10. Enter or copy and paste the following code:

```
stop();
var timeleft=10;
```

   This code creates a variable to keep track of the time left in the game before it ends. It is set to 10, but you can make this as long as you want.

   Next you display the time in the text box on the Stage.

11. Enter or copy and paste the next line of code:

```
timetxt.text=String(timeleft);
```

   This code takes the value of *timeleft*, converts it from a number into a string, and displays it in the *timetxt* text box.

   Next you create a timer object called *gameTimer.*

12. Enter or copy and paste the next line of code:

```
var gameTimer = new Timer(1000);
```

The timer is set to trigger every 1000 milliseconds, or 1 second.

13. Enter or copy and paste the next line of code:

```
gameTimer.addEventListener(TimerEvent.TIMER, countDown);
```

This event listener runs an event called countDown every second because the *gameTimer* has a value of 1000 milliseconds. Next, you create the *countdown* function.

14. Enter or copy and paste the next lines of code:

```
function countDown(e:TimerEvent):void{
timeleft--;
timetxt.text=String(timeleft);
}
```

The function decreases the time left by 1 and updates the information in the text box.

Finally, the timer has to start.

15. Enter or copy and paste the next line of code:

```
gameTimer.start();
```

16. Close the Actions panel. Select Control > Test Movie > In Flash Professional.

The timer text shows up on the screen and counts down backwards. The counter continues into negative numbers.

You need to tell the counter when to end the game and show the GAME OVER message.

To use a timer to end the game:

1. Select Insert > Scene.

Scene 2 appears with a blank timeline. Scene 1 still exists, but you don't see it.

2. Create a static text box in the center of the Stage that reads GAME OVER. Use a font that is different from the one you used for the countdown clock dynamic text box in Scene 1 (something other than Arial).

3. Click the Edit Scene button and choose Scene 1.

4. Select the Actions layer. Open the Actions panel and edit the *countDown* function as follows:

```
function countDown(e:TimerEvent):void{
timeleft--;
timetxt.text=String(timeleft);
   if (timeleft<0){
      gameTimer.stop();
      gameTimer.removeEventListener(TimerEvent.TIMER, countDown);
      gotoAndStop(1, "Scene 2");
   }
}
```

The additional code checks to see if *timeleft* is less than zero. If the time runs out, the timer stops, the event listener is removed, and the movie goes to Scene 2.

5. Close the Actions panel and select Control > Test Movie > In Flash Professional.

Now when the timer runs out, the GAME OVER message appears.

Add buttons to start and replay a game

In the previous activity, you created a timer to display a game-over screen when the game ends. The game-over screen is a good location for a Replay (play again) button. The Replay button includes simple instructions that tell Flash to go to and play the game's main scene timeline. You can also add a separate scene to use as the welcome screen. The welcome screen can include a Play button and instructions for the game.

To add a Replay button:

1. Continue where you left off in the preceding activity.

   Note: You can also create a new game, but we're using the previous file as an example because it already has a timer that takes players to the game-over screen when the game ends. You will use this screen to place the Replay button.

2. Click the Edit Scene button and choose Scene 2 to view the game-over screen (**Figure 31**).

3. Insert a new layer and name it **Replay**.

4. Select the Replay layer. Use the Text and shape tools to design the artwork for the Replay button (**Figure 32**). Do not use the same font used for the countdown clock dynamic text box in Scene 1.

5. Convert the button artwork to a button symbol.

   Note: For more information on designing buttons, see the guide "How to create a button symbol."

6. Select the Replay button on the Stage. In the Properties panel, give it the instance name **replay_btn** (**Figure 33**).

7. Insert a new layer and name it **Actions**.

8. Select the Actions layer and open the Actions panel.

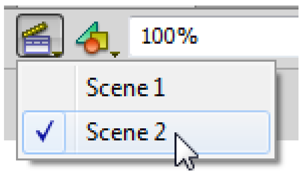


Figure 31 Edit Scene menu

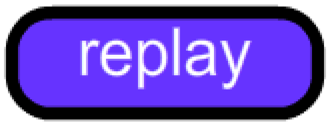


Figure 32 Artwork for the replay button

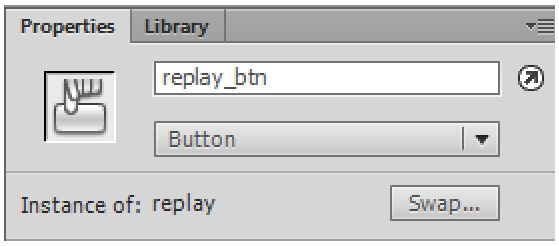


Figure 33 Properties panel, instance name

9. Enter the following code to tell Flash to go to and play Scene 1 when someone clicks the Replay button:

```
stop();
replay_btn.addEventListener(MouseEvent.CLICK,clickFunction);
function clickFunction(evt:MouseEvent):void {
gotoAndPlay(1,"Scene 1");
}
```

10. Close the Actions panel and select Control > Test Movie > In Flash Professional.

    The movie starts by counting down, using the timer you created in the previous activity. When the timer runs out, the game-over screen in Scene 2 appears. Clicking the Replay button takes you back to Scene 1, and the timer starts again.

11. Close the preview window.

To create a play button:

1. Continue where you left off in the preceding activity.
2. Select Insert > Scene to add a new scene to the document.
3. Select Window > Other Panels > Scene.

   The Scene panel appears (**Figure 34**). It includes the main scene of the game (Scene 1), the game-over screen (Scene 2), and the new scene for the welcome screen (Scene 3).
4. Double-click Scene 3 in the Scene panel and change its name to **welcome**.

   Note: If you rename Scene 1 or Scene 2, you'll also need to change the references to these scenes in your existing code. Changing names in the Scene panel does not update the code automatically.
5. Design the artwork for the Play button. Convert the artwork to a new button symbol and name the instance on the Stage **start_btn**.

   Note: As when creating your other static text, do not use the same font in your Play button that was used for the countdown clock dynamic text box in Scene 1.
6. In the welcome scene, insert a new layer and name it **Actions**.
7. Select the Actions layer and open the Actions panel.

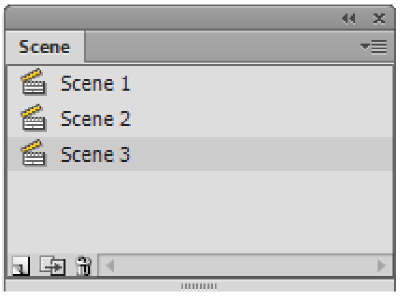


Figure 34 Scene panel

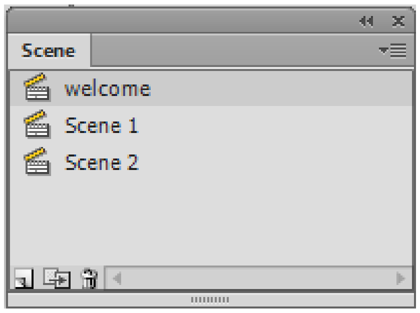


Figure 35 Scene panel

8. Enter the following code to tell Flash to go to and play Scene 1 to start the game when someone clicks the Play (start_btn) button:

```
stop();
start_btn.addEventListener(MouseEvent.CLICK,clickFunction2);
function clickFunction2(evt:MouseEvent):void {
gotoAndPlay(1,"Scene 1");
}
```

9. Close the Actions panel.
10. In the Scene panel, drag the Welcome scene to the top of the list to make it the scene that plays first (**Figure 35**).
11. Select Control > Test Movie > In Flash Professional.

    The movie starts on the Welcome screen. Clicking the Play button jumps to Scene 1 and starts the game (and the counter).
12. Close the preview window.