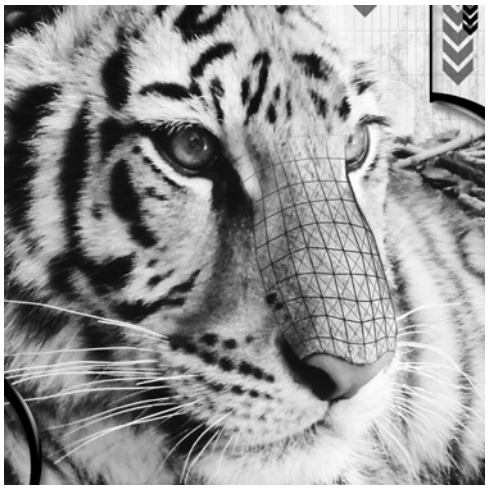

Chapter 14

Advanced Player Control Systems

In this bonus chapter we'll take a detailed look at how to design these popular types of game character control systems:

- A mouse-controlled platform game character.
- A keyboard controlled space ship.
- A car for an overhead driving game
- A tank that can rotate and fire bullets from a turret.

All these player control systems are based on the techniques we've already covered, so there are no new skills to learn. However, it can be complex, time consuming and difficult to work out exactly how to put all the pieces together. To help you out, you'll find all the code to build these systems in the next few sections. Feel free to rip the code apart, adapt it, and improve upon it for your own games. You'll find all the working code in the chapter 10 folder of the book's source files.

A mouse controlled platform character

Open the project folder `MouseControlledPlatformGame` for a working example of how to control a platform game character with the mouse. Move the mouse left and right to move the character, and click to jump (Figure 14-1).

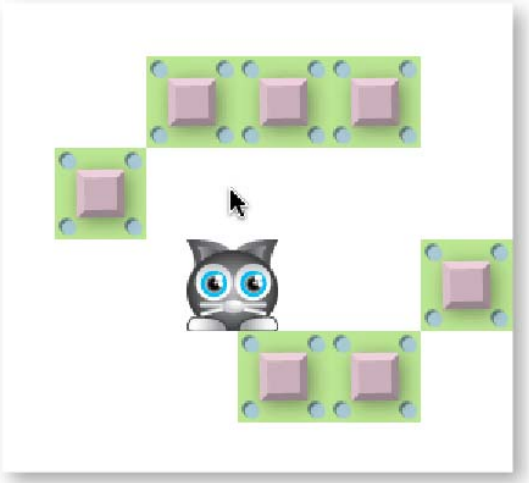

Figure 14-1. Use the mouse to move the cat around the game world.

The bulk of this code is identical to the **LoopingThroughBoxes** example we looked at in chapter 9, except for these two details:

1. The cat's vx value is worked out using this formula in the `enterFrameHandler`:

```
_character.vx
  = (stage.mouseX - (_character.x + _character.width / 2)) * 0.2;
```

It calculates the difference between the cat's center position and the mouse, then multiplies it by 0.2, which is the easing value. This code also doesn't need to apply acceleration or friction, because that's handled by the easing effect itself

2. A `mouseDownHandler` makes the cat jump if the mouse is clicked.

```
private function mouseDownHandler(event:MouseEvent):void
{
  if(_character.isOnGround)
  {
    _character.vy += _character.jumpForce;
    _character.gravity = 0.3;
    _character.isOnGround = false;
  }
}
```

And those are the only changes you need to make. There's less code, and the code is simpler, than the original code we used to move the mouse with the keyboard.

Keyboard controlled spaceship

The **Spaceship** project folder contains a working example of a keyboard controlled rocket ship. Use the left and right arrow keys to rotate the ship, use the up arrow key to make it move, and press the space key to fire missiles. Figure 14-2 shows how this looks.

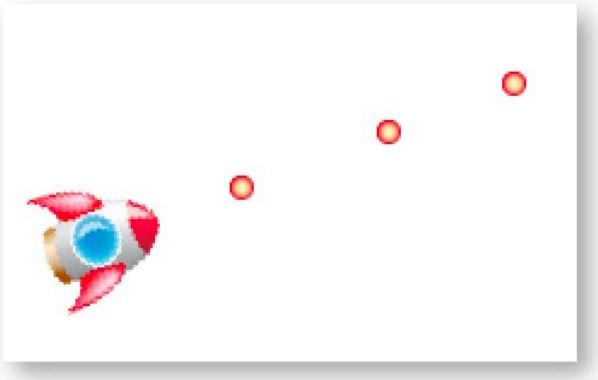

Figure 14-2. A keyboard controlled spaceship.

There are two model classes: `Bullet` and `Rocket`. The `Bullet` class embeds and centers the bullet image and includes the vx and vy properties that the bullet needs to move.

```
package
{
```

```
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class Bullet extends Sprite
  {
    //Embed the bullet image
    [Embed(source="../images/bullet.png")]
    private var BulletImage:Class;

    //Private properties
    private var _bulletImage:DisplayObject = new BulletImage();

    //Public properties
    public var vx:Number = 0;
    public var vy:Number = 0;

    public function Bullet()
    {
      this.addChild(_bulletImage);

      //Center the image
      _bulletImage.x = -(_bulletImage.width / 2);
      _bulletImage.y = -(_bulletImage.height / 2);
    }
  }
}
```

The Rocket class embeds the rocket image and contains some custom properties that the rocket needs to move. Here's the Rocket class

```
package
{
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class Rocket extends Sprite
  {
    //Embed the image
    [Embed(source="../images/rocket.png")]
    private var RocketImage:Class;

    //Private properties
    private var _rocketImage:DisplayObject = new RocketImage();

    //Public properties
    public var vx:Number = 0;
    public var vy:Number = 0;
    public var speed:Number = 0;
    public var acceleration_X:Number = 0;
    public var acceleration_Y:Number = 0;
    public var speedLimit:Number = 5;
    public var friction:Number = 0.96;
    public var rotationSpeed:Number = 0;

    public function Rocket()
    {
      //Display the image in this class
      this.addChild(_rocketImage);
      _rocketImage.x -= _rocketImage.width / 2;
      _rocketImage.y -= _rocketImage.height / 2;
    }
  }
}
```

The Spaceship application class makes the rocket move and fires the bullets. You'll see that it's a mix-and-match collection of all the physics based motion techniques we've been using.

```
package
{
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.events.KeyboardEvent;
  import flash.ui.Keyboard;

  [SWF(width="550", height="400",
  backgroundColor="#FFFFFF", frameRate="60")]

  public class Spaceship extends Sprite
  {
    //Private  properties
    private var _rocket:Rocket = new Rocket();
    private var _bullets:Array = new Array();
    private var _angle:Number;

    public function Spaceship()
    {
      stage.addChild(_rocket);
      _rocket.x = 275;
      _rocket.y = 175

      stage.addEventListener
        (KeyboardEvent.KEY_DOWN, keyDownHandler);
      stage.addEventListener
        (KeyboardEvent.KEY_UP, keyUpHandler);
      stage.addEventListener
        (Event.ENTER_FRAME, enterFrameHandler);
    }
    public function enterFrameHandler(event:Event):void
    {
      //Calculate the angle of rotation
      _angle = _rocket.rotation  * (Math.PI / 180);

      //Figure out the acceleration based on the angle
      _rocket.acceleration_X = Math.cos(_angle) * _rocket.speed;
      _rocket.acceleration_Y = Math.sin(_angle) * _rocket.speed;

      //Add the acceleration to the velocity
      _rocket.vx += _rocket.acceleration_X;
      _rocket.vy += _rocket.acceleration_Y;

      //Add friction
      _rocket.vx *= _rocket.friction;
      _rocket.vy *= _rocket.friction;

      //Force the rocket's velocity to zero
      //after it falls below 0.1
      if(Math.abs(_rocket.vx) < 0.1
      && Math.abs(_rocket.vy) < 0.1)
      {
        _rocket.vx = 0;
        _rocket.vy = 0;
      }

      //Move and rotate the rocket
      _rocket.x += _rocket.vx;
      _rocket.y += _rocket.vy;
      _rocket.rotation += _rocket.rotationSpeed;

      //Move the bullets
```

```
  for(var i:int = 0; i < _bullets.length; i++)
  {
    var bullet:Bullet = _bullets[i];
    bullet.x += bullet.vx;
    bullet.y += bullet.vy;

    //check the bullet's stage boundaries
    if (bullet.y < 0
    || bullet.x < 0
    || bullet.x > stage.stageWidth
    || bullet.y > stage.stageHeight)
        {
      //Remove the bullet from the stage
      stage.removeChild(bullet);
      bullet = null;

      //Remove the bullet from the _bulletsarray
      _bullets.splice(i,1);

      //Reduce the loop counter
      //by one to compensate
      //for the removed bullet
      i--;
    }
  }
}
public function keyDownHandler(event:KeyboardEvent):void
{
  if (event.keyCode == Keyboard.LEFT)
  {
    _rocket.rotationSpeed = -10;
  }
  else if (event.keyCode == Keyboard.RIGHT)
  {
    _rocket.rotationSpeed = 10;
  }
  else if (event.keyCode == Keyboard.UP)
  {
    _rocket.speed = 0.2;
    _rocket.friction = 1;
  }
  else if (event.keyCode == Keyboard.SPACE)
  {
    //Create a bullet and add it to the stage
    var bullet:Bullet = new Bullet();
    stage.addChild(bullet);

    //Set the bullet's starting position
    var radius:int = 30;
    bullet.x = _rocket.x + (radius * Math.cos(_angle));
    bullet.y = _rocket.y + (radius * Math.sin(_angle));

    //Set the bullet's velocity based
    //on the angle
    bullet.vx = Math.cos(_angle) * 7 + _rocket.vx;
    bullet.vy = Math.sin(_angle) * 7 + _rocket.vy;

    //Push the bullet into the _bullets array
    _bullets.push(bullet);
  }
}
public function keyUpHandler(event:KeyboardEvent):void
{
  if (event.keyCode == Keyboard.UP)
```

```
      {
        _rocket.speed = 0;
        _rocket.friction = 0.96;
      }
      if (event.keyCode == Keyboard.LEFT
      || event.keyCode == Keyboard.RIGHT)
      {
        _rocket.rotationSpeed = 0;
      }
    }
  }
}
```

When the player presses the arrow keys, the code sets the rocket's `rotationSpeed` property to either 10 or -10. The rocket rotates by adding this value to its `rotation` property
`_rocket.rotation += _rocket.rotationSpeed;`

If `rotationSpeed` is 10, the rocket will rotate 10 degrees to the right. If it's -10, it will rotate 10 degrees to the left.

The `enterFrameHandler` figures out the angle based on the rocket's current rotation value.
`_angle = _rocket.rotation  * (Math.PI / 180);`

The rocket's acceleration is based on its angle, multiplied by its `speed` property.
```
_rocket.acceleration_X = Math.cos(_angle) * _rocket.speed;
_rocket.acceleration_Y = Math.sin(_angle) * _rocket.speed;
```

The rocket's speed is set to 0.2 when the up key is pressed, and then set back to zero when the up key is released.

When the player presses the space key, a bullet is added to the stage. Its starting point is the rocket's nose, which is found by adding 30 to the rocket's current angle.
```
var radius:int = 30;
bullet.x = _rocket.x + (radius * Math.cos(_angle));
bullet.y = _rocket.y + (radius * Math.sin(_angle));
```

The bullet velocity is found by multiplying the angle by 7, and then adding the rocket's own velocity. This will make the bullet move at 7 pixels per frame, plus whatever speed the rocket is currently moving at.
```
bullet.vx = Math.cos(_angle) * 7 + _rocket.vx;
bullet.vy = Math.sin(_angle) * 7 + _rocket.vy;
```

A `for` loop in the `enterFrameHandler` makes the bullets move by looping through all the bullets in the `_bullets` array and adding their velocities to their current x and y positions.
```
bullet.x += bullet.vx;
bullet.y += bullet.vy;
```

This code is identical to the code that makes the stars move in the **`ButtonFairy`** project.

Overhead car driving system

A little twist to the spaceship code gives us a car that you can drive around the stage as you can see in Figure 14-3. You'll find this working example in the **`DrivingSystem`** project folder.

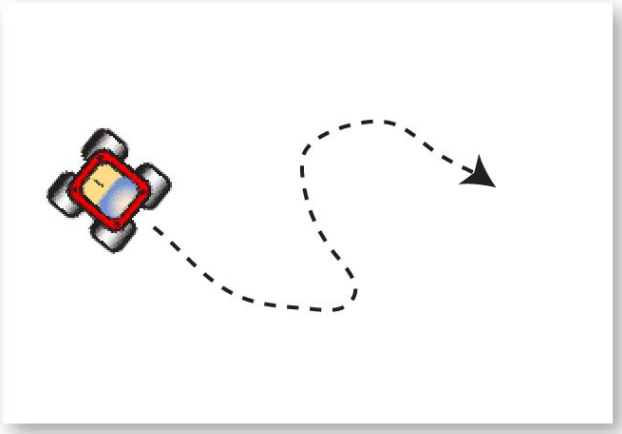

Figure 14-3. Drive a car around the stage.

The biggest difference between the spaceship and the car is that you can change the car's direction even if it's not accelerating. This is done by setting a Boolean variable called accelerate to `true` or `false`, and using that value to determine whether or not to move the car. This `accelerate` variable is a property of the `Car` class, listed below.

```
package
{
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class Car extends Sprite
  {
    //Embed the image
    [Embed(source="../images/car.png")]
    private var CarImage:Class;

    //Private properties
    private var _carImage:DisplayObject = new CarImage();

    //Public properties
    public var vx:Number = 0;
    public var vy:Number = 0;
    public var speed:Number = 0;
    public var accelerate:Boolean = false;
    public var acceleration_X:Number = 0;
    public var acceleration_Y:Number = 0;
    public var speedLimit:Number = 5;
    public var friction:Number = 0.97;
    public var rotationSpeed:Number = 0;

    public function Car()
    {
      //Display the image in this class
      this.addChild(_carImage);
      _carImage.x -= _carImage.width / 2;
      _carImage.y -= _carImage.height / 2;
    }
  }
}
```

Apart from this one new `accelerate` property, the `Car` class is identical to the `Rocket` class in the previous example.

Here's the `DrivingSystem` application class that makes the car work. Notice how the keyboard event handlers use an optional `switch` statement to figure out which key is being pressed.

```
package
{
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.events.KeyboardEvent;
  import flash.ui.Keyboard;

  [SWF(width="550", height="400",
  backgroundColor="#FFFFFF", frameRate="60")]

  public class DrivingSystem extends Sprite
  {
    //Private properties
    private var _car:Car = new Car();
```

```
private var _angle:Number;

public function DrivingSystem()
{
  stage.addChild(_car);
  _car.x = 275;
  _car.y = 175

  stage.addEventListener
    (KeyboardEvent.KEY_DOWN, keyDownHandler);
  stage.addEventListener
    (KeyboardEvent.KEY_UP, keyUpHandler);
  stage.addEventListener
    (Event.ENTER_FRAME, enterFrameHandler);
}
public function enterFrameHandler(event:Event):void
{
  //Increase the car's speed if the up key is being pressed
  if(_car.accelerate)
  {
     _car.speed += 0.1;

     //Add some optional drag
    _car.speed *= _car.friction;
  }
  else
  {
    //Add friction to the speed if the car is
    //not accelerating
    _car.speed *= _car.friction;
  }
  //Calculate the acceleration based on the angle of rotation
  _angle = _car.rotation  * (Math.PI / 180);
  _car.acceleration_X = _car.speed * Math.cos(_angle);
  _car.acceleration_Y = _car.speed * Math.sin(_angle);

  //Update the car's velocity
  _car.vx = _car.acceleration_X;
  _car.vy = _car.acceleration_Y;

  //Force the car’s velocity to zero
  //it falls below 0.1
  if(Math.abs(_car.vx) < 0.1
  && Math.abs(_car.vy) < 0.1)
  {
    _car.vx = 0;
    _car.vy = 0;
  }

  //Move and rotate the car
  _car.x += _car.vx;
  _car.y += _car.vy;
  _car.rotation += _car.rotationSpeed;
}
public function keyDownHandler(event:KeyboardEvent):void
{
  switch (event.keyCode)
  {
    case Keyboard.LEFT:
      _car.rotationSpeed = -3;
      break;

    case Keyboard.RIGHT:
      _car.rotationSpeed = 3;
```

```
            break;

          case Keyboard.UP:
            _car.accelerate = true;
        }
      }
      public function keyUpHandler(event:KeyboardEvent):void
      {
        switch (event.keyCode)
        {
          case Keyboard.LEFT:
            _car.rotationSpeed = 0;
            break;

          case Keyboard.RIGHT:
            _car.rotationSpeed = 0;
            break;

          case Keyboard.UP:
            _car.accelerate = false;
        }
      }
    }
}
```

When the up key is pressed, accelerate becomes `true`. When it's released, it becomes `false`. The `enterFrameHandler` uses this value to determine whether or not to move the car.

```
if(_car.accelerate)
{
  _car.speed += 0.1;

  //Add some optional drag
  _car.speed *= _car.friction;
}
else
{
  //Add friction to the speed if the car is
  //not accelerating
        _car.speed *= _car.friction;
}
```

If the car isn't accelerating, friction is added to gradually slow it down.

A tank with a rotating turret

The most complex example in this chapter is a tank with a rotating turret that fires bullets, as shown in Figure 14-4. The turret always points to the mouse, and you can fire bullets either by pressing the space key or clicking the left mouse button. What makes this work is a combination of all the little techniques we've looked at in this chapter. In a new twist, the turret is actually a sub-object inside the tank class, so our code needs to covert its local coordinates to global, like we did with the frog's eyes in chapter 9.

Figure 14-4. Drive the tank with the arrow keys and rotate the turret to fire bullets with the mouse.

This project uses three classes: `Bullet`, `Tank`, and the application class, `TankSystem`. The `Bullet` class is the same one from the **`Spaceship`** project. The `Tank` class embeds two images: an image of the tank's body and the image of the turret. Both images are centered in the class. Here's the `Tank` class that does this.

```
package
{
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class Tank extends Sprite
  {
    //Embed the images
    [Embed(source="../images/body.png")]
    private var BodyImage:Class;
    [Embed(source="../images/turret.png")]
    private var TurretImage:Class;

    //Private properties
    private var _bodyImage:DisplayObject = new BodyImage();
    public var _turretImage:DisplayObject = new TurretImage();
    public var turret:Sprite = new Sprite();

    //Public properties
    public var vx:Number = 0;
    public var vy:Number = 0;
    public var speed:Number = 0;
    public var accelerate:Boolean = false;
    public var acceleration_X:Number = 0;
    public var acceleration_Y:Number = 0;
    public var speedLimit:Number = 5;
    public var friction:Number = 0.97;
    public var rotationSpeed:Number = 0;

    public function Tank()
    {
      //Display and center the tank body image
      this.addChild(_bodyImage);
      _bodyImage.x -= _bodyImage.width / 2;
      _bodyImage.y -= _bodyImage.height / 2;

      //Add the turretImage to the turret sprite
      turret.addChild(_turretImage);

      //Center the turret image inside  the turret sprite
      _turretImage.x -= _turretImage.width / 2;
      _turretImage.y -= _turretImage.height / 2;

      //Add the turret sprite
      this.addChild(turret);
    }
  }
}
```

The turret image is centered and contained inside a Sprite called `turret`. This is important so that the application class can accurately target and rotate this sub-object, and you'll see how it does this next.

Here's the `TankSystem` application class that makes all this work. I'll explain how the turret is accurately rotated and the bullets correctly fired in the pages ahead.

```
package
{
  import flash.display.Sprite;
  import flash.events.Event;
```

```
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;
import flash.events.MouseEvent;
import flash.geom.Point;

[SWF(width="550", height="400",
backgroundColor="#FFFFFF", frameRate="60")]

public class TankSystem extends Sprite
{
  //Properties
  private var _tank:Tank = new Tank();
  private var _tankAngle:Number;
  private var _turretAngle:Number;
  private var _bullets:Array = new Array();

  public function TankSystem()
  {
    stage.addChild(_tank);
    _tank.x = 275;
    _tank.y = 175

    stage.addEventListener
      (KeyboardEvent.KEY_DOWN, keyDownHandler);
    stage.addEventListener
      (KeyboardEvent.KEY_UP, keyUpHandler);
    stage.addEventListener
      (Event.ENTER_FRAME, enterFrameHandler);
    stage.addEventListener
      (MouseEvent.MOUSE_DOWN, mouseDownHandler);
  }
  public function enterFrameHandler(event:Event):void
  {
    //THE TANK
    //Increase the tank's speed if the up key is being pressed
    if(_tank.accelerate)
    {
      _tank.speed += 0.1;

      //Add some optional drag
      _tank.speed *= _tank.friction;
    }
    else
    {
      //Add friction to the speed if the tank is
      //not accelerating
      _tank.speed *= _tank.friction;
    }
    //Calculate the acceleration based on the angle of rotation
    _tankAngle = _tank.rotation  * (Math.PI / 180);
    _tank.acceleration_X = _tank.speed * Math.cos(_tankAngle);
    _tank.acceleration_Y = _tank.speed * Math.sin(_tankAngle);

    //Update the tank's velocity
    _tank.vx = _tank.acceleration_X;
    _tank.vy = _tank.acceleration_Y;

    //Force the tank’s velocity to zero
    //it falls below 0.1
    if(Math.abs(_tank.vx) < 0.1
    && Math.abs(_tank.vy) < 0.1)
    {
      _tank.vx = 0;
      _tank.vy = 0;
```

```
  }

  //Move and rotate the tank
  _tank.x += _tank.vx;
  _tank.y += _tank.vy;
  _tank.rotation += _tank.rotationSpeed;

  //THE TURRET
  //Convert the turret's points from local to global coordinates
  var turretPoint:Point
    = new Point(_tank.turret.x, _tank.turret.y);
  var turretPoint_X:Number
    = _tank.localToGlobal(turretPoint).x;
  var turretPoint_Y:Number
    = _tank.localToGlobal(turretPoint).y;

  //Find the turret's angle based on the
  //position of the mouse
  _turretAngle
    = Math.atan2
    (
      turretPoint_Y - stage.mouseY,
      turretPoint_X - stage.mouseX
     )
     * (180/Math.PI);

  //Compensate for the rotation of the tank
  _turretAngle -= _tank.rotation;

  //Rotate the turret towards the mouse
  _tank.turret.rotation = _turretAngle;

  //MOVE THE BULLETS
  //Move the bullets
  for(var i:int = 0; i < _bullets.length; i++)
  {
    var bullet:Bullet = _bullets[i];
    bullet.x += bullet.vx;
    bullet.y += bullet.vy;

    //Check the bullet's stage boundaries
    if (bullet.y < 0
    || bullet.x < 0
    || bullet.x > stage.stageWidth
    || bullet.y > stage.stageHeight)
    {
      //Remove the bullet from the stage
      stage.removeChild(bullet);
      bullet = null;

      //Remove the bullet from the _bulletsarray
      _bullets.splice(i,1);

      //Reduce the loop counter
      //by one to compensate
      //for the removed bullet
      i--;
    }
  }
}
private function fireBullet():void
{
  //Create a bullet and add it to the stage
  var bullet:Bullet = new Bullet();
```

```
    stage.addChild(bullet);

    //Set the bullet's starting position
    //1. Set the radius
    var radius:int = -30;

    //2. Add the tank and turrets rotation values together
    //and convert the result to radians
    var angle:Number
      = (_tank.rotation + _tank.turret.rotation)  / (180/Math.PI);

    //3. Position the bullet
    bullet.x = _tank.x + (radius * Math.cos(angle));
    bullet.y = _tank.y + (radius * Math.sin(angle));

    //Set the bullet's velocity based on the angle
    bullet.vx = Math.cos(angle) * -7 + _tank.vx;
    bullet.vy = Math.sin(angle) * -7 + _tank.vy;


    //Push the bullet into the _bullets array
    _bullets.push(bullet);;
  }
  private function mouseDownHandler(event:MouseEvent):void
  {
    fireBullet();
  }
  private function keyDownHandler(event:KeyboardEvent):void
  {
    switch (event.keyCode)
    {
      case Keyboard.LEFT:
        _tank.rotationSpeed = -3;
        break;

      case Keyboard.RIGHT:
        _tank.rotationSpeed = 3;
        break;

      case Keyboard.UP:
        _tank.accelerate = true;
        break;

      case Keyboard.SPACE:
        fireBullet();
    }
  }
  private function keyUpHandler(event:KeyboardEvent):void
  {
    switch (event.keyCode)
    {
      case Keyboard.LEFT:
        _tank.rotationSpeed = 0;
        break;

      case Keyboard.RIGHT:
        _tank.rotationSpeed = 0;
        break;

      case Keyboard.UP:
        _tank.accelerate = false;
          }
  }
}
```

```
}
```

To accurately rotate the turret to the mouse's position, the code first converts the point of the tank's `turret` sub-object to global coordinates.

```
var turretPoint:Point
        = new Point(_tank.turret.x, _tank.turret.y);
var turretPoint_X:Number
        = _tank.localToGlobal(turretPoint).x;
var turretPoint_Y:Number
        = _tank.localToGlobal(turretPoint).y;
```

It then uses these new global points to figure out the angle between the mouse and the turret.

```
_turretAngle
        = Math.atan2
        (
                turretPoint_Y - stage.mouseY,
                turretPoint_X - stage.mouseX
        )
        * (180/Math.PI);
```

To correctly rotate the turret, the angle then has to compensate for the tank body's rotation.

```
_turretAngle -= _tank.rotation;
```

And once that's done, the code can use the `_turretAngle` value to accurately rotate the turret.

```
_tank.turret.rotation = _turretAngle;
```

The code that adds the bullets to the stage also needs to compensate for the rotation of the tank's body before it can accurately plot the bullets on the end of the turret when they're fired. The code finds the correct angle by adding the turret and tank body rotations together, and then coverts that number to radians.

```
var angle:Number
  = (_tank.rotation + _tank.turret.rotation)  / (180/Math.PI);
```

This number has to be in radians to that it can be used by the `Math.sin` and `Math.cos` functions in the next two lines of code:

```
bullet.x = _tank.x + (radius * Math.cos(angle));
bullet.y = _tank.y + (radius * Math.sin(angle));
```

This code plots the bullets on the nose of the turret. (The radius value is 30). The bullet velocities are then calculated in exactly the same way as they were calculated in the Spaceship example.

```
bullet.vx = Math.cos(angle) * -7 + _tank.vx;
bullet.vy = Math.sin(angle) * -7 + _tank.vy;
```

I've provided this example as so that you can see what a really, really complex player control system looks like and that if you ever have to make something like this, you have this as a reference. However, this code would be much easier to write and manage if you didn't have to jump through all the hoops needed to convert points and compensate for main object/sub-object rotations. If I had created the tank's body and the turret as individual objects in the main application class, they'd both occupy the same coordinate space and the code would be much more straightforward. Make your life easier by avoiding the need to program sub-objects if you can, but, if you have to, you now know how.