

# Platform Jumper: Running, Falling, Jumping

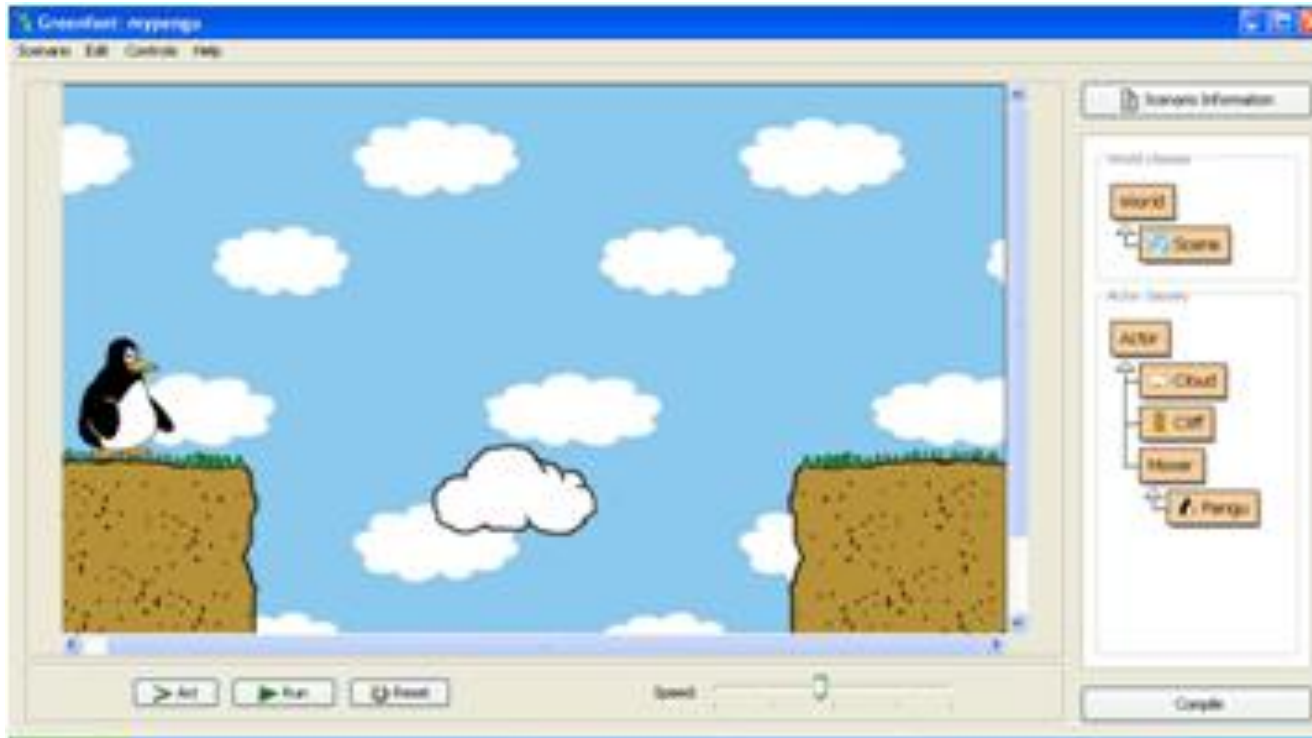
With thanks to Michael Kolling

# Pengu!



We will be implementing a small segment of a platform game. In this game, the player controls a character that has to move from one area on the screen to another, while overcoming obstacles such as a gap in the ground

# Pengu!



In this scenario, there are two pieces of ground on either side of the screen, and the penguin can get across by jumping onto a moving cloud. We will implement the running, jumping and falling behavior

# Creating Movement

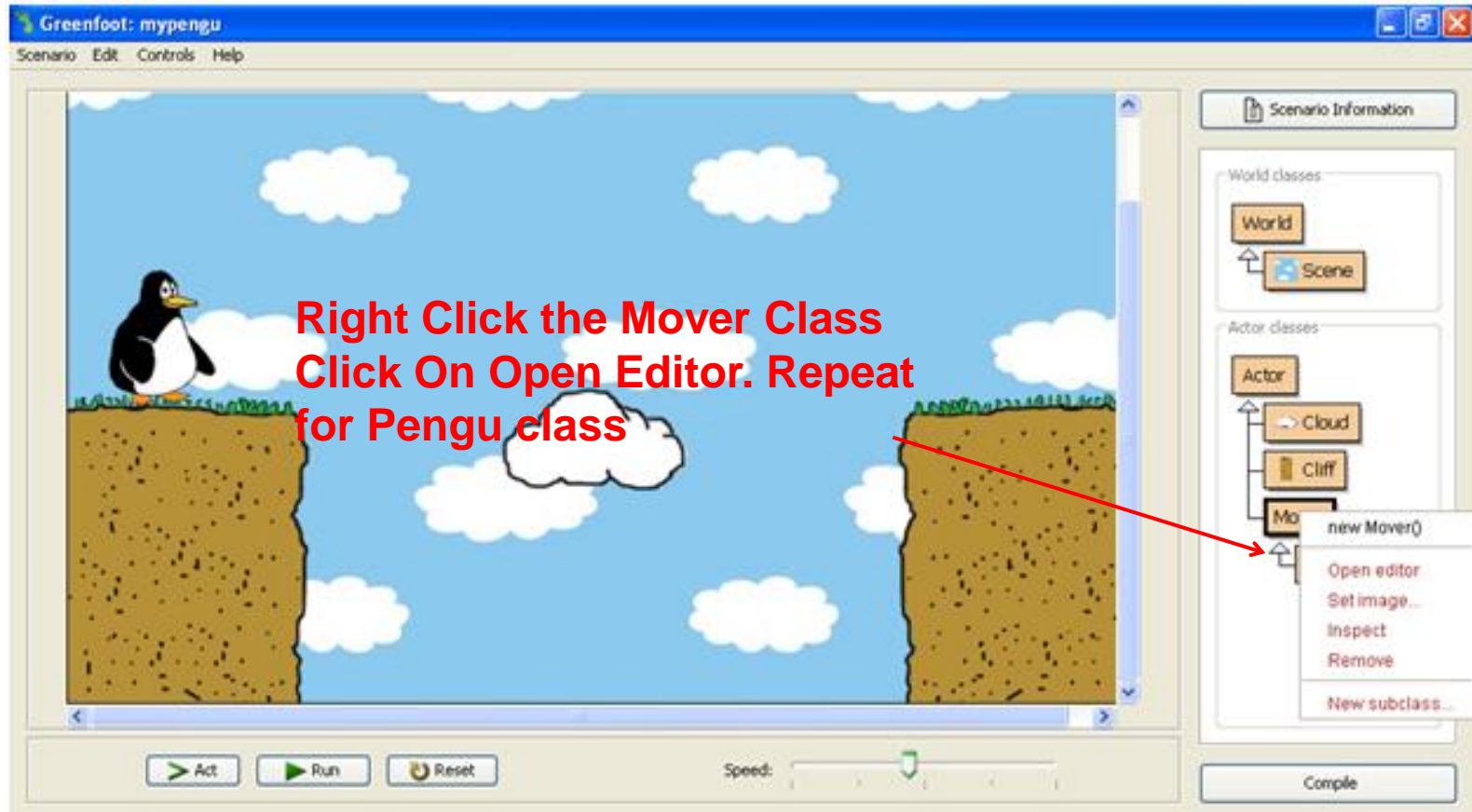
**Click Run. Notice that Pengu can move left/right but does not fall when he leaves ground**

The screenshot shows the Greenfoot IDE window titled "Greenfoot: mypengu". The main canvas displays a penguin character standing on a cliff edge. A red arrow points from the text to the "Run" button in the bottom control panel. The right-hand side of the IDE shows a class hierarchy:

- World classes: World, Scene (inherits from World)
- Actor classes: Actor, Cloud, Cliff, Mover (inherits from Actor), Pengu (inherits from Mover)

At the bottom of the IDE, there are buttons for "Act", "Run", and "Reset", along with a "Speed" slider and a "Compile" button.

# Let's Make Some Changes



# Surveying the Code

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * The class Mover provides some basic movement methods. Use this as a superclass
 * for other actors that should be able to move left and right, jump up and fall
 * down.
 */
public class Mover extends Actor
{
    private static final int speed = 7; // running speed (pixels per second)

    public void moveRight()
    {
        setLocation ( getX() + speed, getY() );
    }

    public void moveLeft()
    {
        setLocation ( getX() - speed, getY() );
    }
}
```

Class compiled - no syntax errors

**When Run is Clicked, there is only left/ right movement without Falling or jumping ability**

**This is because no method exists in the source code of the Mover class for falling or jumping.**

```
public class Pengu extends Mover
{
    /**
     * Check keyboard input and act accordingly
     */
    public void act()
    {
        checkKeys();
    }

    private void checkKeys()
    {
        if (Greenfoot.isKeyDown("left") )
        {
            setImage("pengu-left.png");
            moveLeft();
        }
        if (Greenfoot.isKeyDown("right") )
        {
            setImage("pengu-right.png");
            moveRight();
        }
    }
}
```

Class compiled - no syntax errors

**This method would need to be called in the Pengu class**

# Making Pengu fall

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)
```

```
/**
```

```
* The class Mover provides some basic movement methods. Use this as a superclass  
* for other actors that should be able to move left and right, jump up and fall  
* down.
```

```
*/
```

```
public class Mover extends Actor  
{  
)
```

```
private static final int speed = 7; // running speed (sideways)
```

```
private int vSpeed = 5; // vertical speed
```

```
public void moveRight()  
{  
    setLocation ( getX() , getY() );  
}
```

**Open the Mover class and  
create a new variable  
called vSpeed**



# Making Pengu fall continued

```
public void moveLeft()
{
    setLocation ( getX(), getY() );
}

public void fall()
{
    setLocation ( getX(), getY() + vSpeed);
}
```

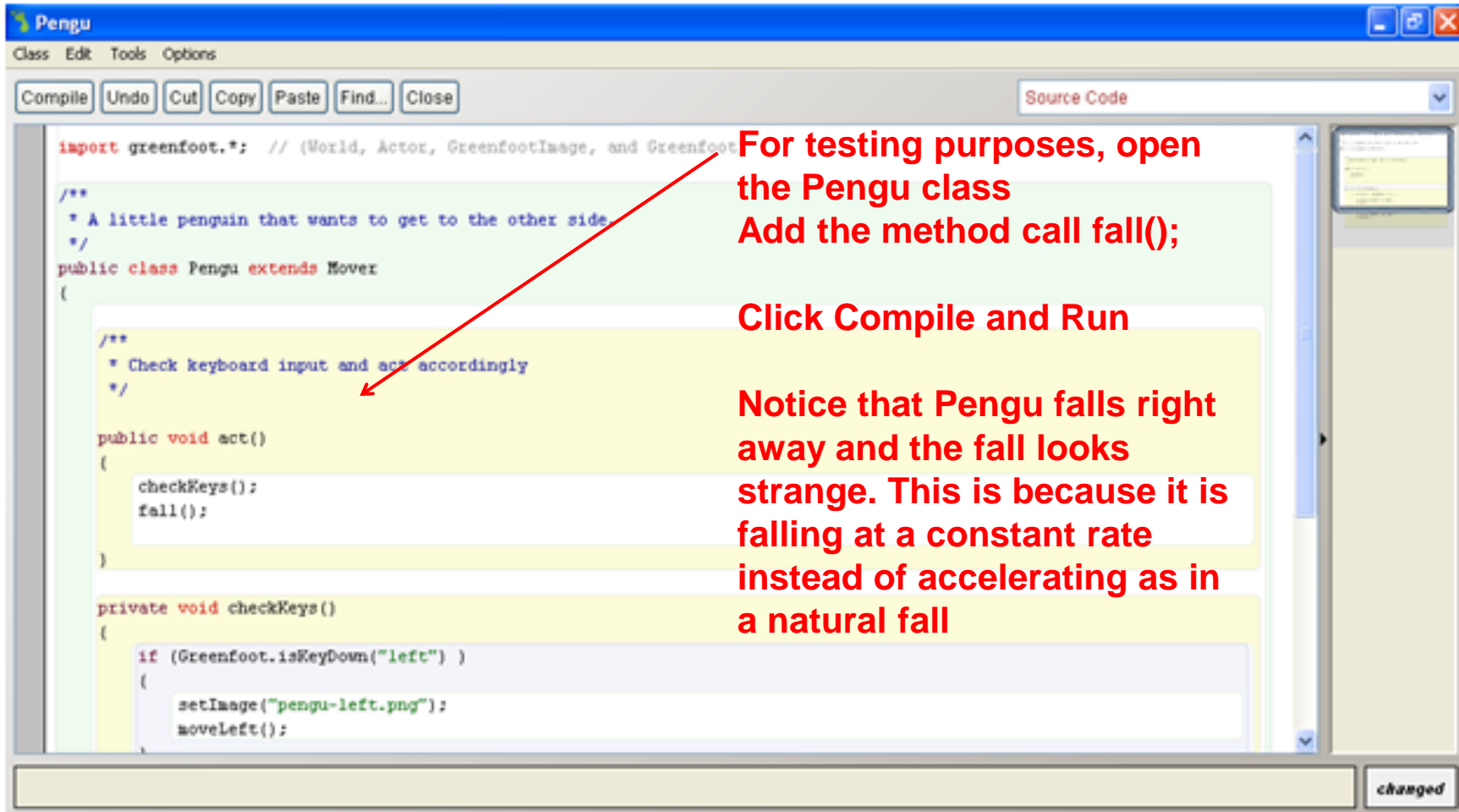
**In the Mover class,  
Create a new method, fall();**

**This will create movement  
downward as we change the y-  
coordinate but leave the x-  
coordinate unchanged**

**Hit Compile button to save**



# Almost Ready To Go...



```
import greenfoot.*; // (World, Actor, GreenfootImage, and GreenfootClicker)

/**
 * A little penguin that wants to get to the other side.
 */
public class Pengu extends Mover
{
    /**
     * Check keyboard input and act accordingly
     */

    public void act()
    {
        checkKeys();
        fall();
    }

    private void checkKeys()
    {
        if (Greenfoot.isKeyDown("left") )
        {
            setImage("pengu-left.png");
            moveLeft();
        }
    }
}
```

**For testing purposes, open the Pengu class**  
**Add the method call fall();**

**Click Compile and Run**

**Notice that Pengu falls right away and the fall looks strange. This is because it is falling at a constant rate instead of accelerating as in a natural fall**

changed

# Watch Pengu fall!

**Pengu falls off the cliff at a constant rate!**

**But right now, he falls even if he's on ground**

Greenfoot: pengu  
Scenario Edit Controls Help

Scenario Information

World classes:  
World  
Scene

Actor classes:  
Actor  
Cloud  
Cliff  
Mover  
Pengu

Act Run Reset Speed: [Slider] Compile

# Creating acceleration during fall

```
public class Mover extends Actor
{
    private static final int speed = 7;           // running speed (sideways)
    private int vSpeed = 0;                       // current vertical speed
    private static final int acceleration = 2;    // down (gravity)

    public void moveRight()
    {
        setLocation ( getX() + speed, getY() );
    }

    public void moveLeft()
    {
        setLocation ( getX() - speed, getY() );
    }
    public void setVSpeed(int speed)
    {
        vSpeed = speed;
    }

    public void fall()
    {
        setLocation ( getX(), getY() + vSpeed);

        vSpeed = vSpeed + acceleration;
    }
}
```

**For a more realistic fall, we need acceleration to simulate gravity ( vSpeed needs to increase during the fall)**

**Open the Mover class**

**Initialize the variable vSpeed to 0 and create method setVSpeed**

**Create a new variable for acceleration**

**Add the following code to the fall() method**

**Hit Compile button and run.**

# Preventing falling when on ground

```
public class Mover extends Actor
{
    private static final int acceleration = 2;    // down (gravity)
    private static final int speed = 7;         // running speed (sideways)

    private int vSpeed = 0;                    // current vertical speed

    public void moveRight()
    {
        setLocation ( getX() + speed, getY() );
    }

    public void moveLeft()
    {
        setLocation ( getX() - speed, getY() );
    }
}
```

```
public boolean onGround()
{
    Object under = getObjectAtOffset(0, getImage().getHeight()/2-8 , null);
    return under != null;
}
```

**Pengu currently falls even if he is standing on ground**

**We need to check if he's on ground, ( check to see if there is an object immediately under our object of type Pengu) and only fall if he's not**

**We can do this using a boolean method called onGround. A boolean tests to see if a condition holds true or not.**

**Open the Mover class, create the method and implement it as follows**

**Hit Compile to save**



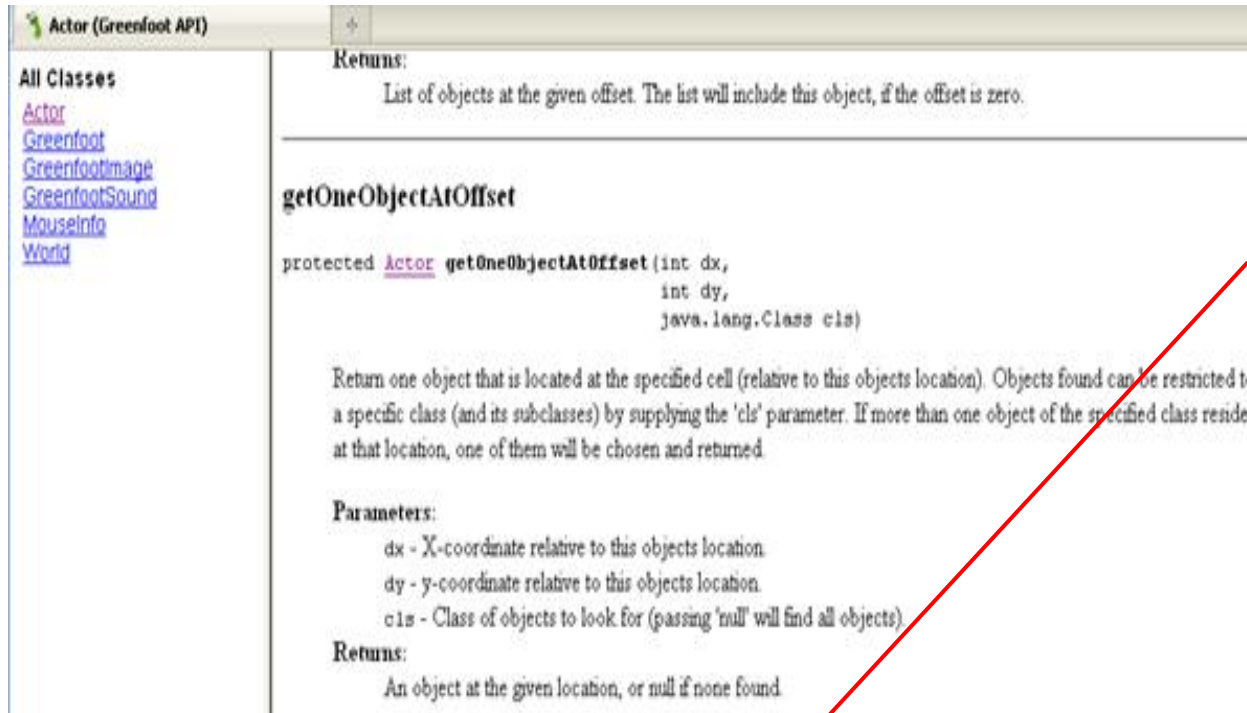
# Preventing falling when on ground continued

**Hit Compile  
Invoke onGround()  
by right clicking on  
Pengu to see that  
what you did works!**

**The boolean should  
return true if Pengu  
is on ground and  
false otherwise**

The screenshot shows the Greenfoot IDE with a penguin character on a cliff. A context menu is open over the penguin, listing methods such as `void fall()`, `void moveLeft()`, `void moveRight()`, `boolean onGround()`, and `void setSpeed(int speed)`. A 'Method Result' dialog box is open, showing that the `onGround()` method returned `true`. The IDE interface includes a menu bar (Scenario, Edit, Controls, Help), a main workspace, a class hierarchy on the right (World classes: World, Scene; Actor classes: Actor, Cloud, Cliff, Mover, Pengu), and control buttons at the bottom (Act, Run, Reset, Speed slider, Compile).

# Preventing falling when on ground continued



The screenshot shows the Greenfoot API documentation for the Actor class. On the left, there is a sidebar titled "All Classes" with links to Actor, Greenfoot, GreenfootImage, GreenfootSound, MouseInfo, and World. The main content area shows the "Returns:" section for the Actor class, which states: "List of objects at the given offset. The list will include this object, if the offset is zero." Below this is the "getOneObjectAtOffset" method signature: `protected Actor getOneObjectAtOffset(int dx, int dy, java.lang.Class cls)`. The description of the method states: "Return one object that is located at the specified cell (relative to this objects location). Objects found can be restricted to a specific class (and its subclasses) by supplying the 'cls' parameter. If more than one object of the specified class resides at that location, one of them will be chosen and returned." The "Parameters:" section lists: `dx` - X-coordinate relative to this objects location, `dy` - y-coordinate relative to this objects location, and `cls` - Class of objects to look for (passing 'null' will find all objects). The "Returns:" section states: "An object at the given location, or null if none found."

## Greenfoot API: Code Explained

`getOneObjectAtOffset()` is a method inherited from the Actor class used to retrieve an object at a given offset from our location. In this case, our x, y coordinates are 0, `getImage().getHeight()/2-8` as this is the centerpoint directly under our image

Method returns true if there was an object of any type, [this is what the null meant] at the offset point

```
public boolean onGround()
{
    Object under = getOneObjectAtOffset(0, getImage().getHeight()/2-8, null);
    return under != null;
}
```

# Preventing falling when on ground continued

```
public class Pengu extends Mover
{

public void act()
{
    checkKeys();
    checkFall();
}

private void checkKeys()
{
    if (Greenfoot.isKeyDown("left"))
    {
        setImage("pengu-left.png");
        moveLeft();
    }
    if (Greenfoot.isKeyDown("right"))
    {
        setImage("pengu-right.png");
        moveRight();
    }
}

private void checkFall()
{
    if (onGround()) {
        setVSpeed(0);
    }
    else {
        fall();
    }
}
}
```

Now we need to create code that will result in our character falling only if we are not on ground

Open the Pengu class

Delete the method fall() and replace with a new method called checkFall()

Implement the method checkfall in the body of the code

If the character is on ground, this stops the fall as vertical speed will be set to zero else the character will fall

Hit Compile and run to test!

# Pengu falls!

The screenshot displays the Greenfoot IDE interface for a project named "pengu". The main window shows a 2D scene with a blue sky, white clouds, and two brown cliffs. A penguin is standing on the left cliff, and a speech bubble containing a cloud icon is positioned between the cliffs. The right sidebar contains a "Scenario Information" panel with two sections: "World classes" showing a hierarchy where "Scene" inherits from "World", and "Actor classes" showing a hierarchy where "Pengu" inherits from "Mover", which inherits from "Cliff", which inherits from "Cloud", which inherits from "Actor". At the bottom of the IDE, there are buttons for "Act", "Run", and "Reset", along with a "Speed" slider. The Windows taskbar at the bottom shows the Start button and several open applications: Microsoft PowerPoint, the "pengu" folder, two Java(TM) Platform instances, Document1 - Microsoft Word, and an untitled Paint window. The system clock indicates the time is 12:47 PM.



# Getting Pengu to Jump!

```
public class Pengu extends Mover
{
    public void act()
    {
        checkKeys();
        checkFall();
    }

    private void checkKeys()
    {
        if (Greenfoot.isKeyDown("left" )
        {
            setImage("pengu-left.png");
            moveLeft();
        }
        if (Greenfoot.isKeyDown("right" )
        {
            setImage("pengu-right.png");
            moveRight();
        }
        if (Greenfoot.isKeyDown("space" )
        {
            if (onGround())
                jump();
        }
    }
}
```

**Now we need to create code that will let our character jump!**

**First we need to add a key that will control the jump**

**Open the Pengu class and add this code**

**Pengu will jump when you press the space key if he is on ground**



# Getting Pengu to Jump Continued!

```
public class Pengu extends Mover
{
    public void act()
    {
        checkKeys();
        checkFall();
    }

    private void checkKeys()
    {
        if (Greenfoot.isKeyDown("left") )
        {
            setImage("pengu-left.png");
            moveLeft();
        }
        if (Greenfoot.isKeyDown("right") )
        {
            setImage("pengu-right.png");
            moveRight();
        }
        if (Greenfoot.isKeyDown("space") )
        {
            if (onGround())
                jump();
        }
    }
}

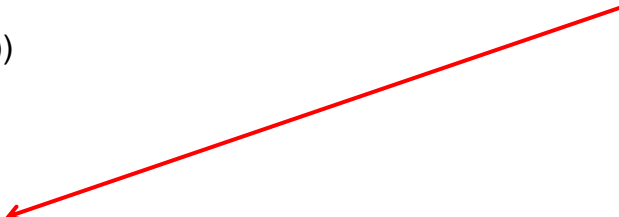
private void jump()
{
    setVSpeed(-16);
    fall();
}
```

**Now we need to implement the jump method. Jumping is similar to falling but with upwards movement**

**So we start with a negative vertical speed because the vertical speed will decrease until standstill**

**Open the Pengu class and add this code**

**Hit Compile and run to test it out!**



# Getting Pengu to Jump Continued!

```
public class Pengu extends Mover
{
    private static final int jumpStrength = 16;
    public void act()
    {
        checkKeys();
        checkFall();
    }

    private void checkKeys()
    {
        if (Greenfoot.isKeyDown("left") )
        {
            setImage("pengu-left.png");
            moveLeft();
        }
        if (Greenfoot.isKeyDown("right") )
        {
            setImage("pengu-right.png");
            moveRight();
        }
        if (Greenfoot.isKeyDown("space") )
        {
            if (onGround())
                jump();
        }
    }
    private void jump()
    {
        setVSpeed(-jumpStrength);
        fall();
    }
}
```

**Now lets use a variable in the Pengu class for the vertical speed when jumping because this will make it easier to find and change later!**

**Lets call it jumpStrength**

**Don't forget to also update the jump method with the new variable name!**

**Test out different jumpStrengths by changing the value.**

**Compile and run!**

# Running, falling, jumping!

**Pengu should now be able to run, fall and jump correctly!**

Greenfoot: pengu  
Scenario Edit Controls Help

Scenario Information

World classes

- World
- Scene

Actor classes

- Actor
  - Cloud
  - Cliff
  - Mover
    - Pengu

Complete

Act Run Reset Speed: [slider]

# Creating a moving platform

The screenshot shows the Greenfoot IDE interface. The main window displays a scene with a penguin on a cliff on the left, a gap, a cloud in the middle, and another cliff on the right. A red arrow points from the text 'We can create a moving platform that Pengu can surf on like the cloud!!' to the cloud. The interface includes a menu bar (Scenario, Edit, Controls, Help), a toolbar (Act, Run, Reset), a speed slider, and a right-hand panel with 'Scenario Information', 'World classes' (World, Scene), and 'Actor classes' (Actor, Cloud, Cliff, Mover, Pengu). A 'Compile' button is at the bottom right.

**We need a way for Pengu to get from side to side without falling!**

**We can create a moving platform that Pengu can surf on like the cloud!!**

# Moving Platforms!

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)
```

```
/**  
 * A cloud that moves back and forth between two defined  
 points.  
 */  
public class Cloud extends Actor  
{  
    private int speed = 4;  
  
    /**  
     * Move in the direction we are currently moving in.  
     Turn if we reach a turning point.  
     */  
    public void act()  
    {  
        setLocation ( getX() + speed, getY() );  
  
        Actor actor = getOneIntersectingObject(null);  
        if(actor != null) {  
            actor.setLocation ( actor.getX() + speed, actor.getY() );  
        }  
    }  
}
```

**Open the Cloud class**

**Before we get the cloud moving from point to point, we need to make sure that when Pengu jumps on the cloud, the Cloud and Pengu move in the same direction together.**

**Add this code to do this!**

# Moving Platforms!

```
public class Cloud extends Actor
{
    private int speed = 4;
    private int leftTurn = 270;
    private int rightTurn = 480;

    /**
     * Move in the direction we are currently moving in. Turn if we reach a turning point.
     */
    public void act()
    {
        setLocation ( getX() + speed, getY() );

        Actor actor = getOneIntersectingObject(null);
        if(actor != null) {
            actor.setLocation ( actor.getX() + speed, actor.getY() );
        }

        if (atTurningPoint()) {
            speed = -speed;
        }
    }

    /**
     * Test if we are at one of the turning points.
     */
    public boolean atTurningPoint()
    {
        return (getX() <= leftTurn || getX() >= rightTurn);
    }
}
```

Now we need the cloud to move back and forth

To get the cloud to do this we need to tell the cloud to change directions when it gets to a specific point

To do this, first add the leftTurn and the rightTurn variables. These variables specify the x coordinate at which the cloud should change directions

Now create a boolean method that decides if we're at a turning point. The method will return true as soon as we reach either the left turning point or the right turning point

Now write the code that tells the cloud to change directions if it is at a turning point. Using the negative of speed reverses the movement

Compile and run!





# Ending the game after a fall!

```
public boolean onGround()
{
    Object under = getOneObjectAtOffset(0, getImage().getHeight()/2-8 , null);
    return under != null;
}

public void setVSpeed(int speed)
{
    vSpeed = speed;
}

public void fall()
{
    setLocation ( getX(), getY() + vSpeed);
    vSpeed = vSpeed + acceleration;
    if ( atBottom() )
        gameEnd();
}

private boolean atBottom()
{
    return getY() >= getWorld().getHeight() - 2;
}

private void gameEnd()
{
    Greenfoot.stop();
}
}
```

**We want our game to be over if our character falls off and dies!**

**Open the Mover class**

**Create the method atBottom. This will decide if Pengu has fallen**

**Create the method gameEnd. This stops running Greenfoot**

**Call these methods in the fall method**