# Forest Fire Project.

For high school students mid-way through their first year of learning to write code. This is a unit capstone project, and students are expected to take several in-class work days to complete this project.

In the Forest Fire project, students focus on writing conditional statements (if, else if, and else) to add advanced behavior to the game. Students will also write compound Boolean expressions using *&&* and // operators.

Using conditional statements and compound Boolean expressions, students will be able to steer, accelerate, and decelerate the helicopter using keyboard input. They will also be able to make the helicopter launch bubbles of water. They will make objects wrap around the edges of the screen when their movement takes them across the border. Students will make flames appear at random locations on a constant timer. They will write code that makes flames disappear if they touch a bubble, and the helicopter change colors and eventually crash if it touches flames.

Webb's Depth of Knowledge:

- Level 1, Recall, is color coded in Yellow
- Level 2, Skills and Concepts, is in Blue
- Level 3, Strategic Thinking, is in Green
- Level 4, Extended Thinking, is in Pink


1. Download the code for the project "Forest Fire," and open it in Greenfoot. You should see a helicopter near the bottom of the screen. It doesn't do anything yet. Open the code editor for the class *Helicopter* and add the following code into *Helicopter's act()* method.

```
public void act()
{
    move(speed);
}
```

Click the run button and test your work. Your *Helicopter* should now move forward.

2. Our next step is to make the *Helicopter* turn when we press the arrow keys. Inside *Helicopter*, add a new method called *keyCommands()*. Inside your new *keyCommands()* method, write the following if statement.

```
if(Greenfoot.isKeyDown("left"))
{
    turn(-2);
}
```

Call the *keyCommands()* method inside *act()*. Test and see if you can make the *Helicopter* turn left when you press the left arrow key.

3. Add another if statement into *keyCommands()* that makes the *Helicopter* turn to the right when you press the right arrow key.

4. We want to make the *Helicopter* slow down when you press the down arrow key, but speed back up as soon as you let go of the down arrow key. To do this, we will need to use an **if statement** and an **else statement.** Add the following code to *keyCommands()*.

```
if(Greenfoot.isKeyDown("down"))
{
    speed = 1;
}

else
{
    speed = 2;
}
```

Test and make sure your helicopter slows down when you press the down arrow, and speeds back up when you let it go.

5. Now that we can make the *Helicopter* turn and slow down, we want to give the ability to wraparound -- that means, if it goes off of one edge of the screen, it will reappear on the other edge.

***insert video example***

Inside *Helicopter*, create a new method called *screenWrap()*, with an if statement that makes the *Helicopter* reappear on the right edge of the screen if it goes off of the left edge.

```
public void screenWrap()
{
    if(getX() < 10)
    {
        setLocation(590, getY());
    }
}
```

Call *screenWrap()* in *act()*, and test your new code. If you fly off of the left edge, do you reappear on the right edge?

6. Add a second if statement to *screenWrap()* that makes the *Helicopter* reappear on the bottom edge if it flies off of the top edge.

```
public void screenWrap()
{
    if(getX() < 10)
    {
        setLocation(590, getY());
    }

    if(getY() < 10)
    {
        setLocation(getX(), 390);
    }
}
```

7. Add two more if statements to *screenWrap()*, so that the *Helicopter* also has wraparound at the bottom edge and the right edge.

8. We're done with all of the movement code for our *Helicopter*. Now we want to animate the rotor blades, so it looks more like a real helicopter. We're going to use a class called *GifImage* that will also us to use animated gifs in our game.

Go to the main Greenfoot window, and right-click in the white on the right edge of the screen. Select "import class," and then import the class *GifImage*.

***insert video example***

9. Go back to the code editor for *Helicopter*. At the top of the class, declare a new class variable of type *GifImage* and name it *animation*.

```
public class Helicopter extends SmoothMover
{
    private GifImage animation;
    private int speed;
```

10. Inside *Helicopter's* constructor, write the code below to give *animation* a starting value using the picture "Copter.gif" (this code only works because the image file *Copter.gif* is already inside of our project's images folder).

```
public Helicopter()
{
    animation = new GifImage("Copter.gif");
    speed = 2;
}
```

11. Add the method *animate()* to the class *Helicopter*.

```
public void animate()
{
    GreenfootImage currentFrame = animation.getCurrentImage();
    setImage(currentFrame);
}
```

Call *animate()* inside the *act()* method. Test your code and see if it works.

12. We're done with the *Helicopter* for now. Open the code editor for the class *Flame*. We want to make the flames move in a random direction. At the top of the *Flame* class, declare a new *int* variable named *direction*.

```
public class Flame extends SmoothMover
{

    private int speed;
    private int direction;
```

13. Add the following code into *Flame*'s constructor to give *direction* a random value between 0 and 360.

```
public Flame()
{
    speed = 1;
    direction = Greenfoot.getRandomNumber(360);
}
```

14. Add the method *randomMovement()* to the *Flame* class.

```
public void randomMovement()
{
    setRotation(direction);
    move(speed);
    setRotation(0);
}
```

15. Call *randomMovement()* in the *Flame* class's *act()* method. Add several *Flame* test objects into the world. Do the flames all move in different random directions?

16. In the same way that we made the *Helicopter* wrap around the edges of the screen, add code to *Flame* to give our *Flame* objects wraparound. Don't forget to test your work.

17. In the same way that we animated the *Helicopter* using the image "Copter.gif", add code to *Flame* to animate it using the image "Flame.gif" (the image file "Flame.gif" is already inside your project's images folder).

18. Once you've got your *Flame* objects animated correctly, open the code editor for the world class *Forest*. We want to make the *Flame* objects appear onscreen on a timer.

Inside *Forest*, declare a new *int* variable named *flameTimer*. Inside *Forest*'s constructor, give *flameTimer* a starting value of *30*.

19. Add the following method *addFlame()* to *Forest*.

```
public void addFlame()
{
  addObject(new Flame(), 50, 50);

}
```

20. Add the following method *flameTimerCountdown()* to *Forest*. If *flameTimer* is greater than 0, it subtracts one from *flameTimer*. Otherwise, it resets *flameTimer* to zero and calls the *addFlame()* method. In this way, we can add a *Flame* object to the world every 30th time the *act()* method is called.

```
public void flameTimerCountdown()
{
    if(flameTimer > 0)
    {
        flameTimer = flameTimer - 1;
    }

    else
    {
        flameTimer = 75;
        addFlame();
    }
}
```

Call *flameTimerCountdown()* in the *Forest* class's *act()* method. Test your code. Does a new *Flame* object get added into the world every few seconds?

21. Right now, all of our flames are added to the top left of the screen. Let's change the code so that the flame is randomly added either to the top left or the top right. Change your *addFlame()* method to look like this:

```
public void addFlame()
{
    int location = Greenfoot.getRandomNumber(2);

    if(location == 0)
    {
        addObject(new Flame(), 50, 50);
    }

    else if(location == 1)
    {
        addObject(new Flame(), 550, 50);
    }

}
```

The method will now generate a random number between 0 and 2 (not including 2), and store it in an *int* variable called *location*. If *location* is equal to 0, we add the *Flame* object to the top left of the world. If *location* is instead equal to 1, we add the *Flame* object to the top right.

Test and make sure that half your flames appear in the top left, and half in the top right.

22. We're going to make more changes to *addFlame()*. We want the flame to be added in four separate locations: the top left, top right, bottom left, and bottom right. Start by making this change:

```
int location = Greenfoot.getRandomNumber(4);
```

Now we generate a random number between 0 and 4 (not including 4).

23. Add two more **else if statements** to the *addFlame()* method, so that there is a 25% chance that the flame appears in any of the four possible locations.

Test your code. Do *Flame* objects appear at random in the top left, bottom left, top right, and bottom right?

24. Our forest is catching on fire. We need a way to put the flames as they spread. Open the code editor for the class *Bubble*. In the same way we animated the *Helicopter* class with "Copter.gif" and the *Flame* class with "Flame.gif", add code to *Bubble* to animate it using "Bubble.gif" (the file "Bubble.gif" is already in your project's images folder).

Create some *Bubble* test objects and make sure they are animated correctly.

25. Inside *Bubble*, declare a new *int* variable named *speed*. In *Bubble*'s constructor, give *speed* a starting value of *3*.

26. Inside *Bubble*'s *act()* method, add the following code to make the *Bubble* move forward:

```
public void act()
{
    move(speed);
    animate();
}
```

27. We want our *Helicopter* to be able to make *Bubble* objects and add them into the world. Open the code editor for the class *Helicopter*. Add the method *makeBubble()* to *Helicopter*.

```
public void makeBubble()
{
    int direction = getRotation();

    Bubble b = new Bubble();
    b.setRotation(direction);
    getWorld().addObject(b, getX(), getY());
}
```

Do you see how the above code stores the rotation of the *Helicopter* in the variable *direction*, and then sets the *Bubble*'s rotation to the same value? This will make the *Bubbles* move in the same direction that the *Helicopter* is facing.

28. Inside *Helicopter*'s *keyCommands()* method, add another if statement that calls *makeBubble()* whenever you press "space". Test your code and see if it works.

29. Right now, if you hold down the space bar, you keep making more *Bubble* objects and adding them into the world. We only want to create one *Bubble* object every time you press the space bar.

At the top of *Helicopter*, declare a new *boolean* variable called *bubbleButton*. Inside *Helicopter*'s constructor, give *bubbleButton* a starting value of *false*.

30. Find the if statement you wrote in step 28. Change it to look like this:

```
if(Greenfoot.isKeyDown("space") && bubbleButton == false)
{
    makeBubble();
    bubbleButton = true;
}
```

31. Now add an **else if** statement underneath like this:

```
if(Greenfoot.isKeyDown("space") && bubbleButton == false)
{
    makeBubble();
    bubbleButton = true;
}

else if(!Greenfoot.isKeyDown("space") && bubbleButton == true)
{
   bubbleButton = false;
}
```

Test and see that you only make one *Bubble* every time you press the space bar. If it doesn't work, make sure that you didn't miss the *!* at the beginning of the **else if** statement's condition. A *!* means "NOT." The expression *!Greenfoot.isKeyDown("space")* checks to see if you are NOT currently pressing the space bar.

32. We want out *Bubble* objects to be able to put out an fires that they hit. Add the following method *hitFlame()* to the class *Bubble*.

```
public void hitFlame()
{
    Flame flame = (Flame) getOneIntersectingObject(Flame.class);
    if(flame != null)
    {
        World w = getWorld();
        w.removeObject(flame);
        w.removeObject(this);
    }

}
```

Call *hitFlame()* in the *Bubble* class's *act()* method. Test your code. If a *Bubble* object touches a *Flame()* object, both the *Bubble* and the *Flame* should be removed from the world.

33. We also want our *Bubble* objects to be removed from the world if they reach the edges of the screen. First, create a method *disappear()* that removes a *Bubble* from the world if it reaches the left edge.

```
public void disappear()
{
    if(getX() < 10)
    {
        World w = getWorld();
        w.removeObject(this);
    }
}
```

34. Our next step is a little different. We already have a method that can remove the *Bubble* from the world, *hitFlame()*. If we try to call *disappear()* after the the *Bubble* has already been removed from the world, then our program will crash. We need to make sure that we only call disappear in *act()* if it hasn't already been removed from the world. You can do that with an if statement, like this:

```
public void act()
{
    move(speed);
    animate();
    hitFlame();

    if(getWorld() != null)
    {
        disappear();
    }
}
```

Test your code.  Does you bubble disappear when it reaches the left edge?

35.  Go back to the method *disappear()*. Underneath the if statement, add three **else if statements** to remove the *Bubble* from the world if it reaches the right edge, top edge, or bottom edge.

Make sure you use else if statements and not if statements.

36. We are done with the *Bubble* class.  Now we're going back to the *Helicopter*.  We want the *Helicopter* to change colors from green to red if it gets hit by a *Flame*.

Open the code editor for the class *Helicopter*.  Declare a new *GifImage* variable named *redCopter*.

37. Inside *Helicopter*'s constructor, give *redCopter* a value using the file "RedCopter.gif":

```
public Helicopter()
{
    animation = new GifImage("Copter.gif");
    redCopter = new GifImage("RedCopter.gif");
    speed = 2;
}
```

38. When we test for equality inside an if statement's condition, if we are comparing an *int* or a *boolean*, we use the == and *!*= operators. However, if we are comparing *Strings* or other **objects**, we use the method *.equals()*. Inside *Helicopter*, add a new method *burned()* that uses *.equals()* to see if the current value of the *GifImage animation* is not equal to the *GifImage redCopter*.

```
public void burned()
{
    if(!animation.equals(redCopter))
    {
        animation = redCopter;
    }
}
```

39. Now add a method named *hitFlame()* that calls *burned()* if the *Helicopter* touches a *Flame* object.

```
public void hitFlame()
{
    Flame flame = (Flame) getOneIntersectingObject(Flame.class);
    if(flame != null)
    {
        getWorld().removeObject(flame);
        burned();
    }
}
```

Call *hitFlame()* in the *Helicopter* class's *act()* method. Test and see if the *Helicopter* turns red when it touches a *Flame* object.

40. Go back to the method *burned()*. Underneath the if statement, add an else statement that changes the world to *GameOver*.

Test your code. If you get hit by a flame the first time, the helicopter should turn red. The second time, you should see a game over screen.

41. We're almost done with our game. Currently, you can lose, but you can't win. We're going to add a time limit to the game. If you can last long enough, you win.

Open the code editor for the world class *Forest*. Declare a new *int* variable named *timeLeft* and a new *Bar* variable named *timeBar*.

42. Inside of *Forest*'s constructor, give *timeLeft* and *timeBar* starting values like this:

```
timeLeft = 2000;
timeBar = new Bar("Time Remaining", timeLeft, timeLeft);
addObject(timeBar, 150, 30);
```

43. Inside of *Forest*'s *act()* method, subtract one from *timeLeft* and set *timeBar*'s displayed value equal to the value of *timeLeft*.

```
public void act()
{
    flameTimerCountdown();
    timeLeft = timeLeft - 1;
    timeBar.setValue(timeLeft);
}
```

44. Add a new method to *Forest* called *timeLeftCountdown()*. Inside this method, if the value of *timeLeft* is greater than 0, subtract one from *timeLeft*. Otherwise, change the world to the victory screen.

```
public void timeLeftCountdown()
{
    if(timeLeft > 0)
    {
        timeLeft = timeLeft - 1;
    }

    else
    {
        Greenfoot.setWorld(new Victory());
    }
}
```

Call *timeLeftCountdown()* in *act()* and see if it works. If the bar makes it all the way down to zero, do you win the game? If not, find your error and fix it.