Rock Dodger Project.

For beginning students in their first two weeks of learning to write code.

In this project, students use getter methods and setter methods to allow separate code objects to communicate with each other, reading and changing other objects' data values.

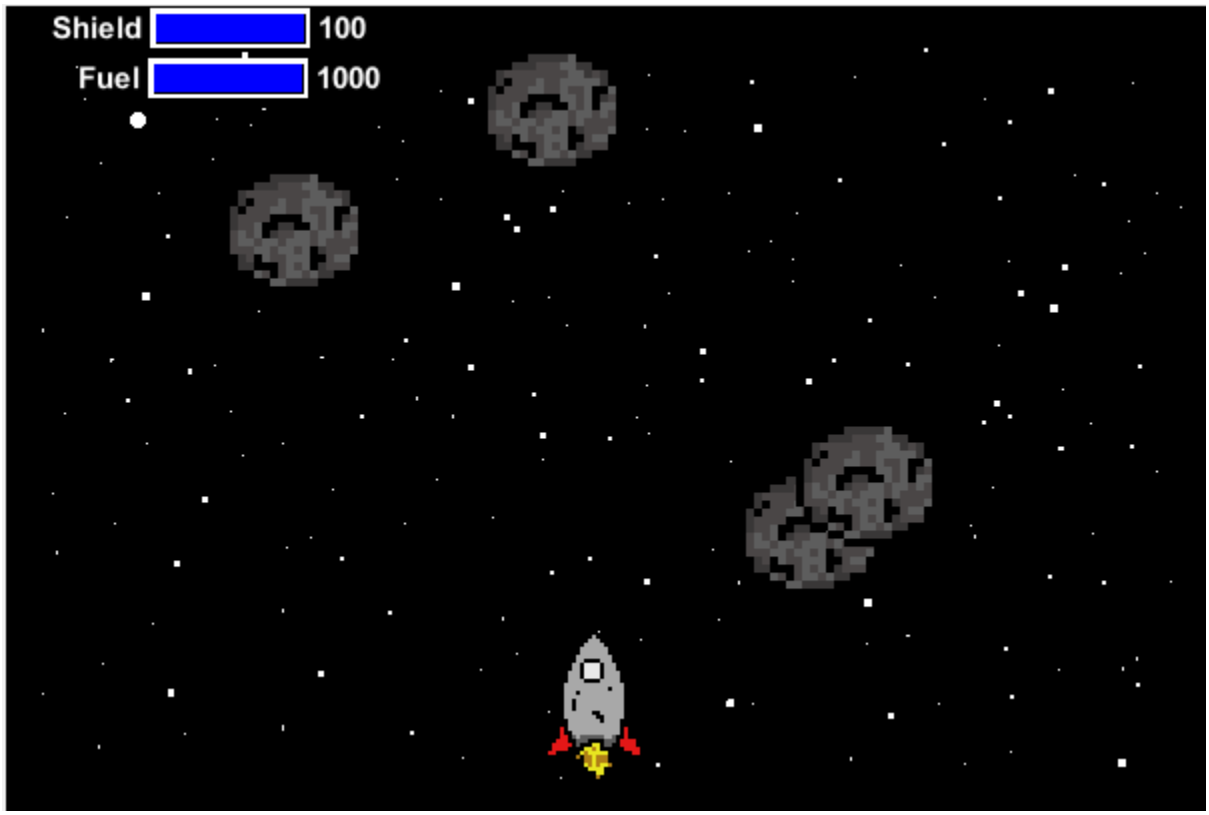
By applying getter and setter methods, students can make a SpaceShip's fuel and shield change as it interacts with other entities on screen, and display the values of the fuel and shield using animated status bars.

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

Rock Dodger

Objectives: Making objects communicate using getter and setter methods



1. Open up the scenario RockDodger in Greenfoot. Press the run button, and the press the right and left arrow key on the keyboard to see how RockDodger is played.

We have two *Bars* up at the top of RockDodger, one for our shield, and one for our fuel. The shield bar is supposed to go down when we touch a rock, and the fuel bar is supposed to go down when we move. That's not happening yet. These next instructions will walk through how to make it happen.

2. Open the code editor for the class *SpaceShip*. *SpaceShip* has two important *int* **variables**, shield and fuel, and unfortunately, neither variable has a **getter method** or a **setter method** yet. Write a getter and a setter for both variables. Remember, getter methods allow us to retrieve a variable's value, and setters allow us to change a variable's value.

3. Now go to the class Rock. Find the method hitRocket

```
public void hitRocket()
{
    SpaceShip ship = (SpaceShip)  getOneIntersectingObject(SpaceShip.class);
    if(ship != null)
    {


        World world = getWorld();
        world.removeObject(this);
    }
}
```

Change it to look like this:

```
public void hitRocket()
{
    SpaceShip ship = (SpaceShip)  getOneIntersectingObject(SpaceShip.class);
    if(ship != null)
    {
        int currentShield = ship.getShield() - 10;
        ship.setShield(currentShield);


        World world = getWorld();
        world.removeObject(this);
    }
}
```

You might have to change our two new lines if you named your getter and your setter methods differently. Their purpose is to take the *SpaceShip's shield* **variable** down by 10 every time you get hit by a *Rock*.

4. Next go into the class RockWorld. Find the **method** *act*. Inside this method, add two lines of code:

```
public void act()
{
    if(randomChance(1))
    {
        addRock();
    }

    int shield = ship.getShield();
    shieldBar.setValue(shield);
}
```

This will cause our Shield Bar to update every time a change occurs onscreen.

5. Go back to the class SpaceShip. Find the method moveLeft. Inside this method, add the following:

```
fuel = fuel - 1;
```

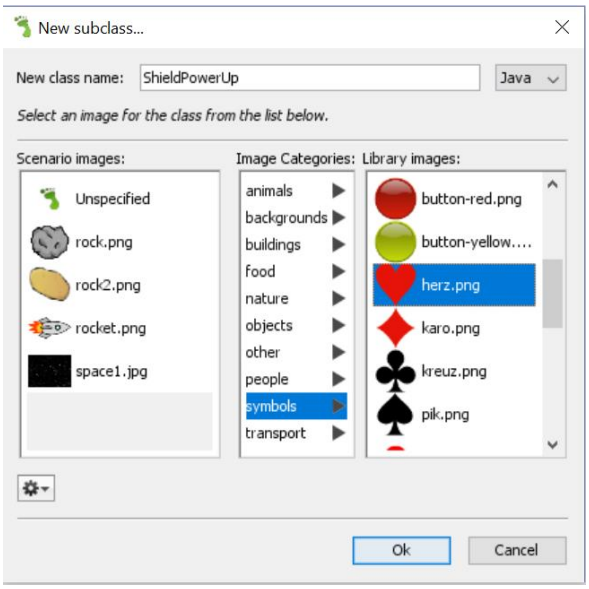
This will make us lose fuel while moving to the left.

6. Now make the *SpaceShip* lose fuel while moving to the right.

7. Go back to *RockWorld* and figure out how to make the fuel *Bar* update.

Great job! Now let's get a little more challenging.

8. On the main Greenfoot screen, right-click on the class *Actor*, and choose: New subclass. Name your new subclass *ShieldPowerUp*, and give it any picture you would like.



9. Look at the *Rock* class and see how the code makes it move downward when it's on the screen. Make your *ShieldPowerUp* behave the same way.

10. Right now, at any given moment, there is a 1 percent chance that a *Rock* is added to the top of the screen inside *RockWorld*. See if you can figure out how to add a *ShieldPowerUp* at a 1 percent chance.

11. Now see if you can make a *ShieldPowerUp* increase the *SpaceShip's* shield by 20 every time you touch one.

12. Add a *FuelPowerUp* that increases the *SpaceShip's* fuel when you touch it.

We've made a game! Using getter methods and setter methods, our *SpaceShip* object can communicate with *Rock* objects, *Bar* objects, and *FuelPowerUp* objects. The objects can read each other's variable values, and also make changes to those values.