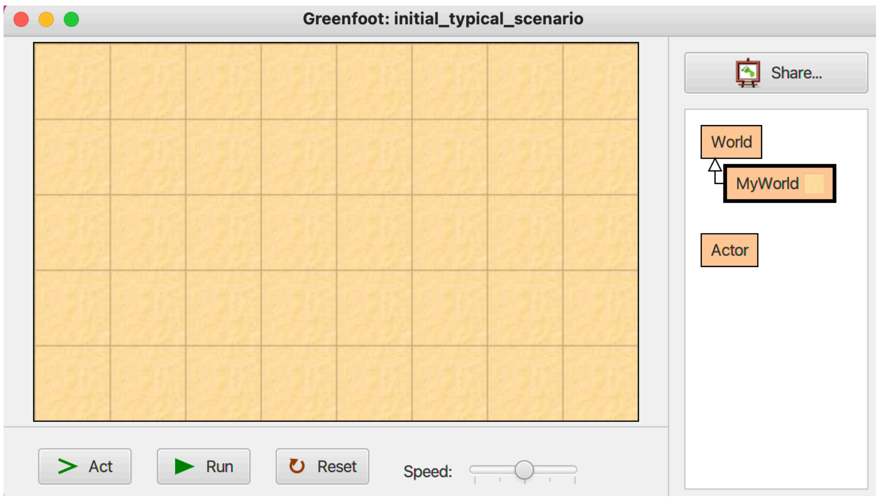
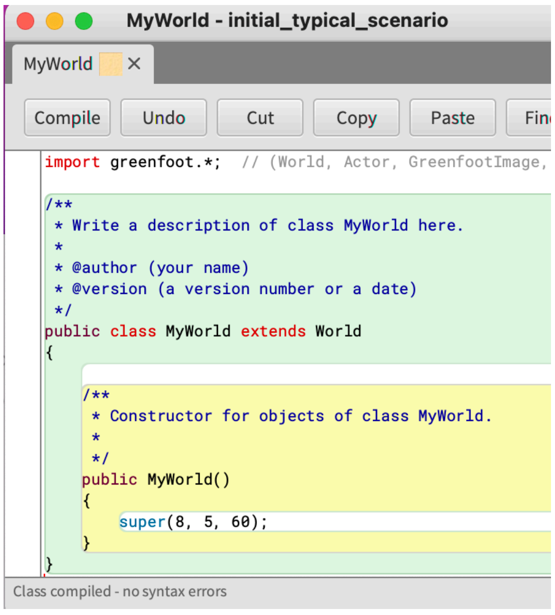
Greenfoot User Templates

francisco.guerra@ulpgc.es
Version 1.0

Test Driven Development with Greenfoot, which can be found in https://greenroom.greenfoot.org, contains the GreenfootUnitTestIDE-3.7.1.jar executable that adds the creation, editing and execution of unit tests in the Greenfoot IDE to perform the test-driven development in the learning process. Additionally, this latest version includes the user template configuration that the Greenfoot IDE applies to class creation, which is the focus of this document. Next, the need for and use of template configuration will be introduced through examples using the Greenfoot IDE.

Let´s create a new Java Project into Greenfoot tool. As usual, this project has only the World class ***MyWorld***. The size of the scenario has been limited to (8, 5, 60) so that the examples fit well in this document.

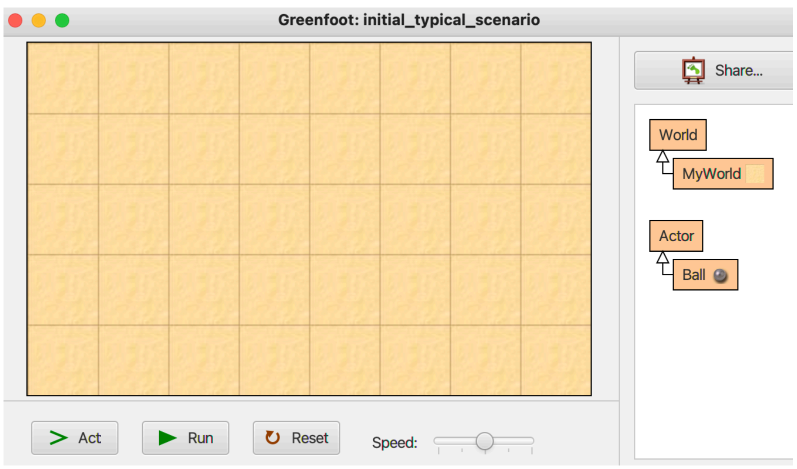


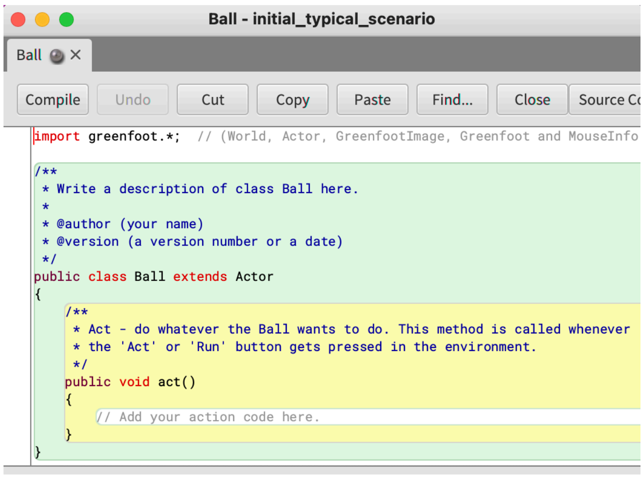


```
import greenfoot.*;  // (World, Actor, GreenfootImage,

/**
 * Write a description of class MyWorld here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class MyWorld extends World
{

    /**
     * Constructor for objects of class MyWorld.
     *
     */
    public MyWorld()
    {
        super(8, 5, 60);
    }
}
```

Class compiled - no syntax errors

Next, let´s create one Actor class, for example a ball.

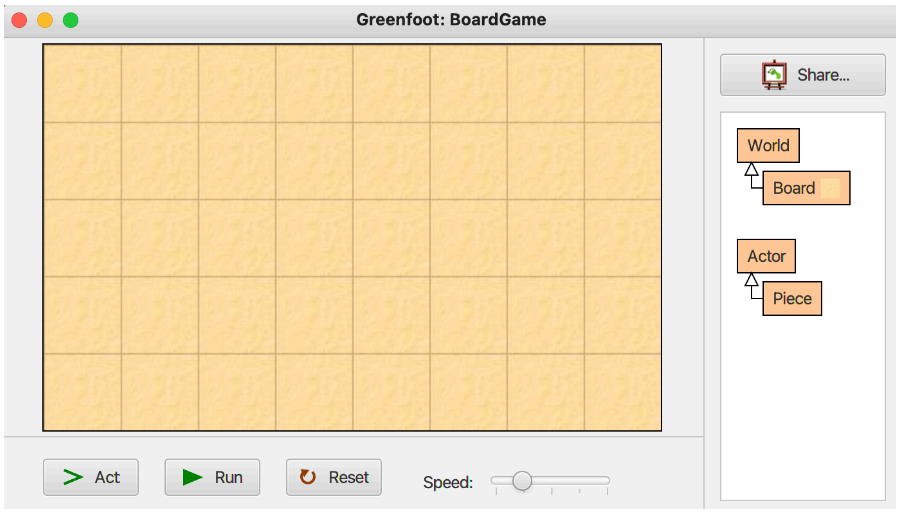




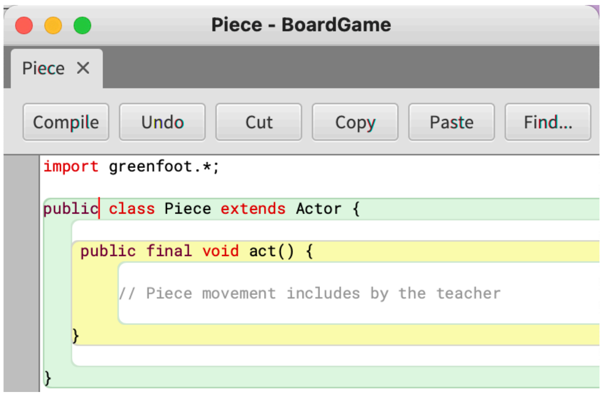
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo

/**
 * Write a description of class Ball here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Ball extends Actor
{
    /**
     * Act - do whatever the Ball wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

To add the MyWorld class or the Ball class, the Greenfoot IDE uses templates. These templates, which are applied to define the initial structure of an Actor or a World, help the user at the beginning of the programming of each class that is added to his project. In addition, to reduce the time of preparation of an exercise or to be able to apply it to new students, the teacher can add to the project one or several java classes that the students extend to create their classes with less effort. However, the template that is applied each time a student adds a new class does not always fit well with the class architecture of the project or the learning objectives sought in the exercise. To show the limitation in the application of the templates offered by Greenfoot, an exercise will be designed where control structures are practiced with beginning students.

Suppose you want to design a board game with a board and moving pieces to create one or more exercises in which students experiment with control structures. Initially, the teacher prepares a project that has an 8x6 square board (class ***Board***) and the class ***Piece***.

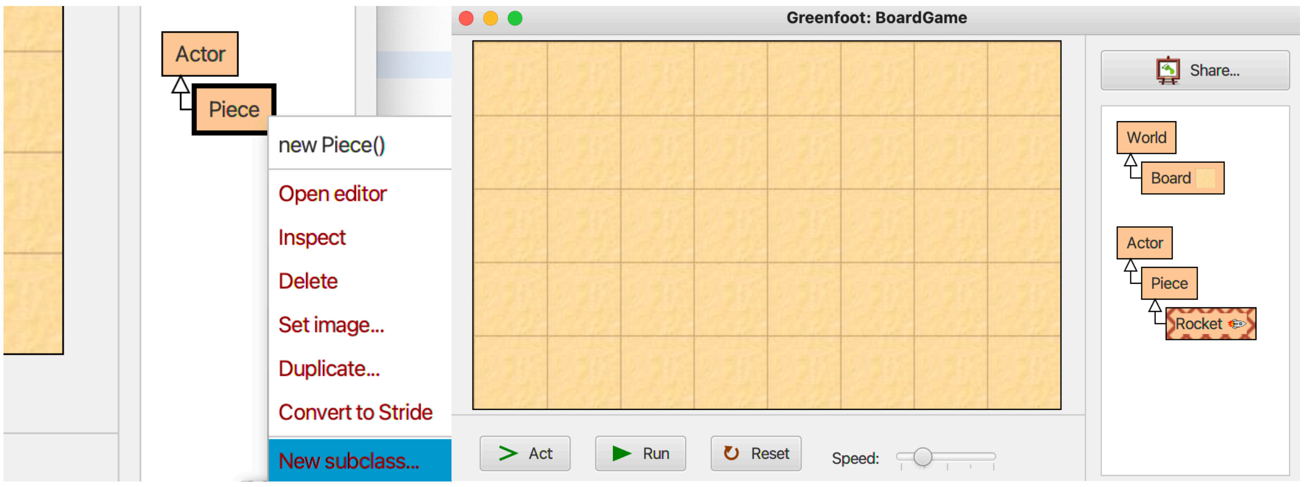


Since you want to motivate the students by showing the movement of the pieces they are programming, but at the same time the students do not have the necessary programming knowledge, the teacher adds the movement by programming the ***act()*** method of the ***Piece*** class and indicates that it is ***final*** to prevent students from substituting it when they add their pieces to the project.

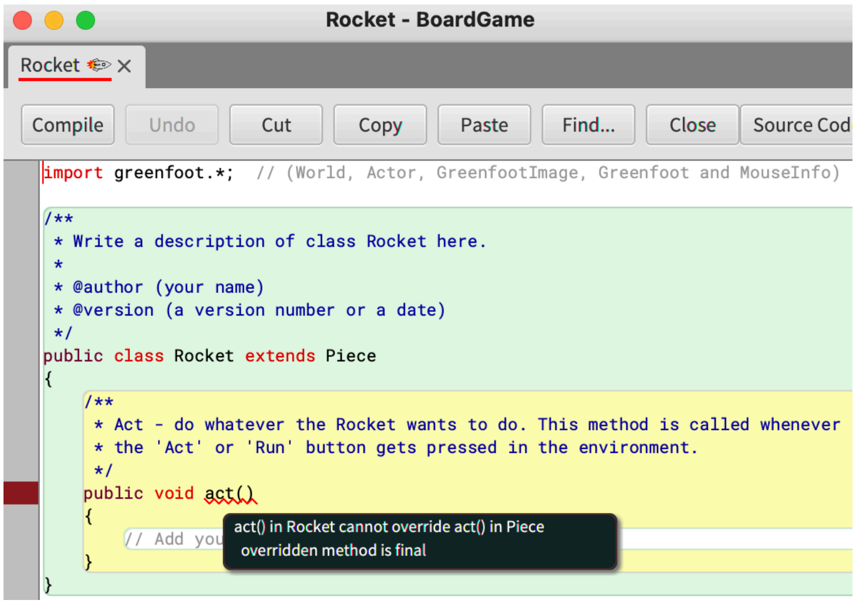


```
import greenfoot.*;

public class Piece extends Actor {

    public final void act() {
        // Piece movement includes by the teacher
    }
}
```

If a student decides to add the ***Rocket*** class to the board game to have pieces in the board game that represent rockets, he extends the ***Piece*** class that the teacher has given him with the Greenfoot project and that is associated with the control structures exercise.



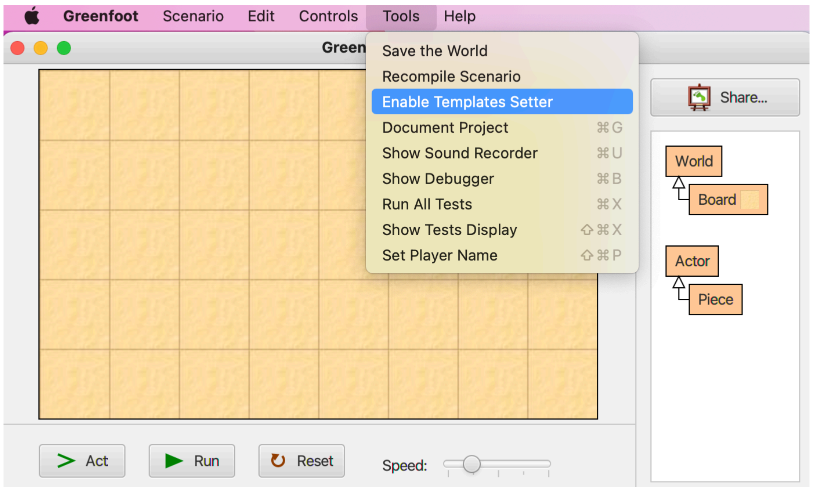
However, the student encounters an unexpected error: the Greenfoot IDE shows an error because the template used to create the ***Rocket*** class adds the ***act()*** method, and the teacher marked it as ***final*** in the Piece class to prevent this from happening.



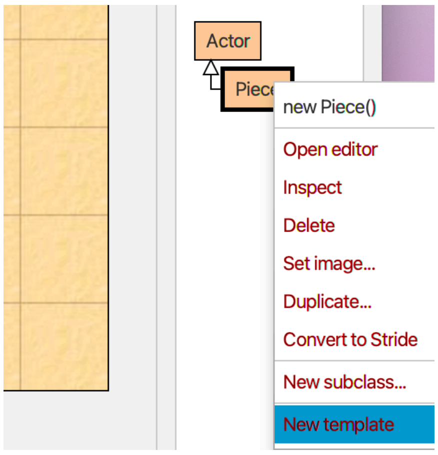
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Rocket here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Rocket extends Piece
{
    /**
     * Act - do whatever the Rocket wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add you
    }
}
```

act() in Rocket cannot override act() in Piece
overridden method is final

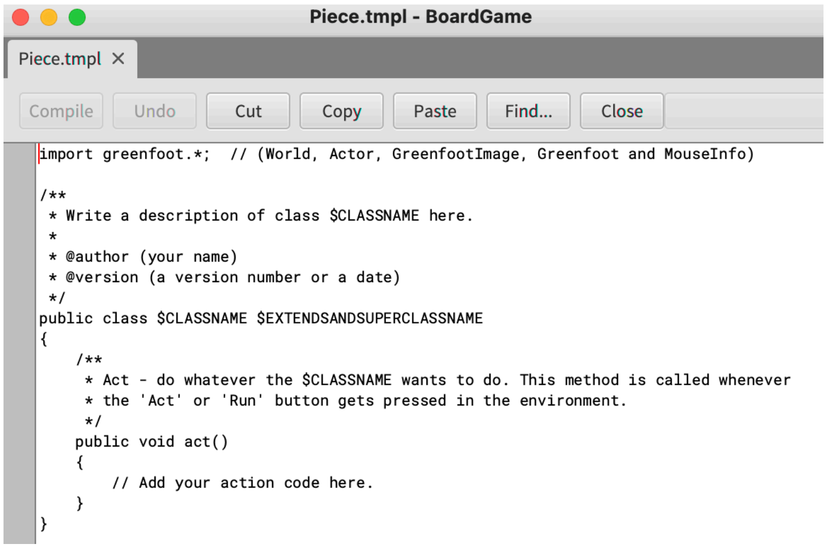
Difficult for a novice student to handle, this error occurs because the Greenfoot IDE uses a template that does not check that the inherited ***act()*** method is ***final***. The teacher can avoid generating this error in the student assignment if they can specify a custom template for the ***Piece*** class, that is, a template that does not include the ***act()*** method when a Student decides to create a new class by extending the ***Piece*** class. The modified interface of the Greenfoot IDE offers the possibility to enable the creation or modification of the templates used to create the classes.



When the template configurator is enabled, the Greenfoot IDE adds the ability to define a template to all classes. That is, if we select the ***Piece*** class, the operation of creating a ***new template*** is offered, which will be applied every time a class is created from the ***Piece*** class.



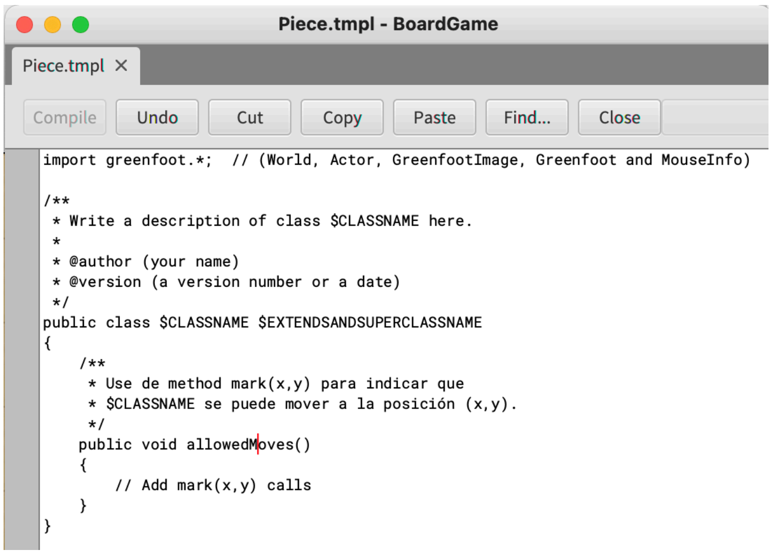
When the create a template operation is executed, the new template is started with the one used up to that moment to extend the class where this operation is executed. Since no templates have been defined in this project yet, the new template starts with the Greenfoot IDE's default template.



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

/**
 * Write a description of class $CLASSNAME here.
 * 
 * @author (your name) 
 * @version (a version number or a date)
 */
public class $CLASSNAME $EXTENDSANDSUPERCLASSNAME
{
    /**
     * Act - do whatever the $CLASSNAME wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act() 
    {
        // Add your action code here.
    }    
}
```

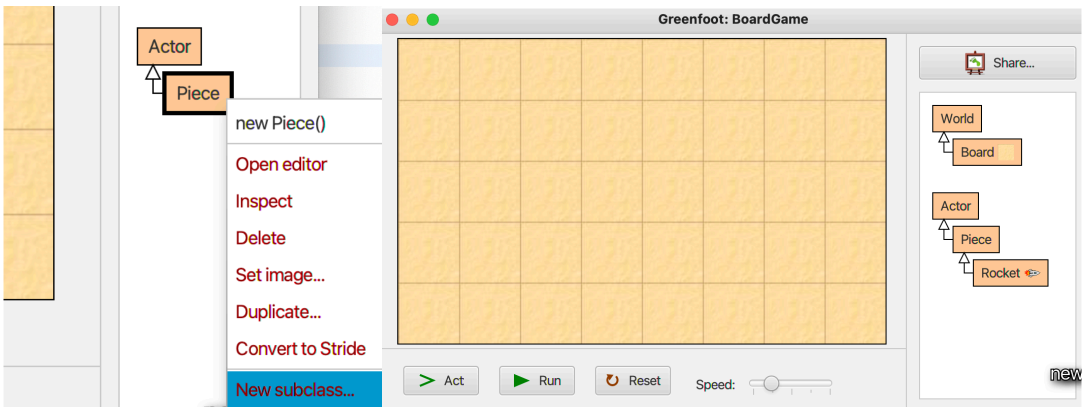
$CLASSNAME represents the name that the new class that is created will have, that is, the Greenfoot IDE will replace it with ***Rocket***. ***$EXTENDSANDSUPERCLASSNAME*** will be used by the Greenfoot IDE to put ***extends Piece***, that is to indicate that the new class is created from the ***Piece*** class.

The teacher should use the editor provided by the Greenfoot IDE to modify the template, removing or adding as necessary. In this example, we will assume that the students must program only the ***allowedMoves*** method. This method will be used by students to indicate the positions on the board to which a piece could be moved.

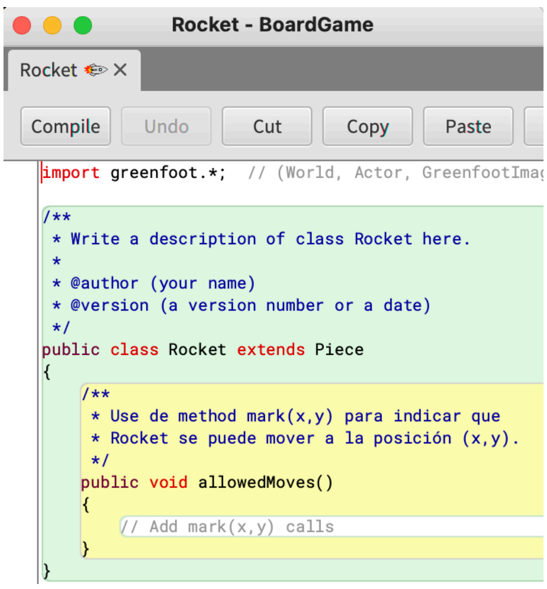


```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class $CLASSNAME here.
 * 
 * @author (your name) 
 * @version (a version number or a date)
 */
public class $CLASSNAME $EXTENDSANDSUPERCLASSNAME
{
    /**
     * Use de method mark(x,y) para indicar que
     * $CLASSNAME se puede mover a la posición (x,y).
     */
    public void allowedMoves() 
    {
        // Add mark(x,y) calls
    }    
}
```

Therefore, if the teacher adds the above template to the project he hands out to the students, when a student decides to add pieces representing rockets to the board game by extending the ***Piece*** class to create the ***Rocket*** class…

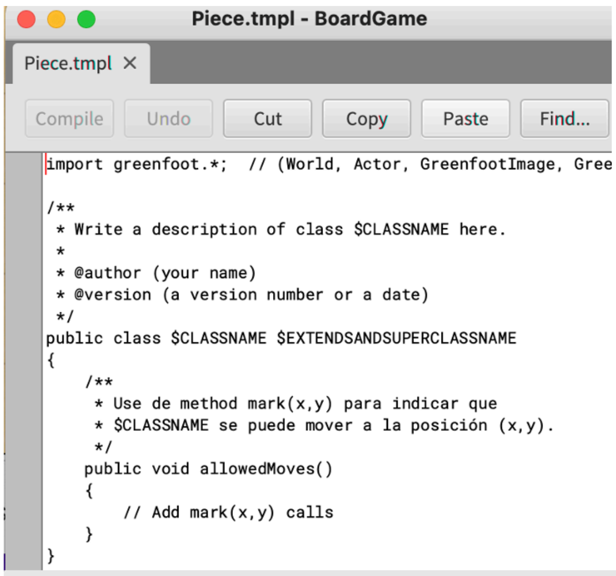


... the new class ***Rocket*** will be ready for the student to start their work.

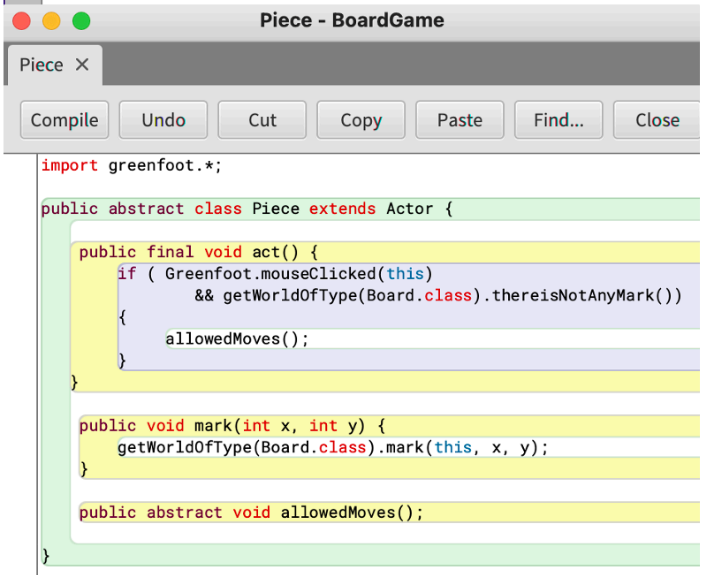


```
import greenfoot.*;  // (World, Actor, GreenfootImag

/**
 * Write a description of class Rocket here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Rocket extends Piece
{
    /**
     * Use de method mark(x,y) para indicar que
     * Rocket se puede mover a la posición (x,y).
     */
    public void allowedMoves()
    {
        // Add mark(x,y) calls
    }
}
```

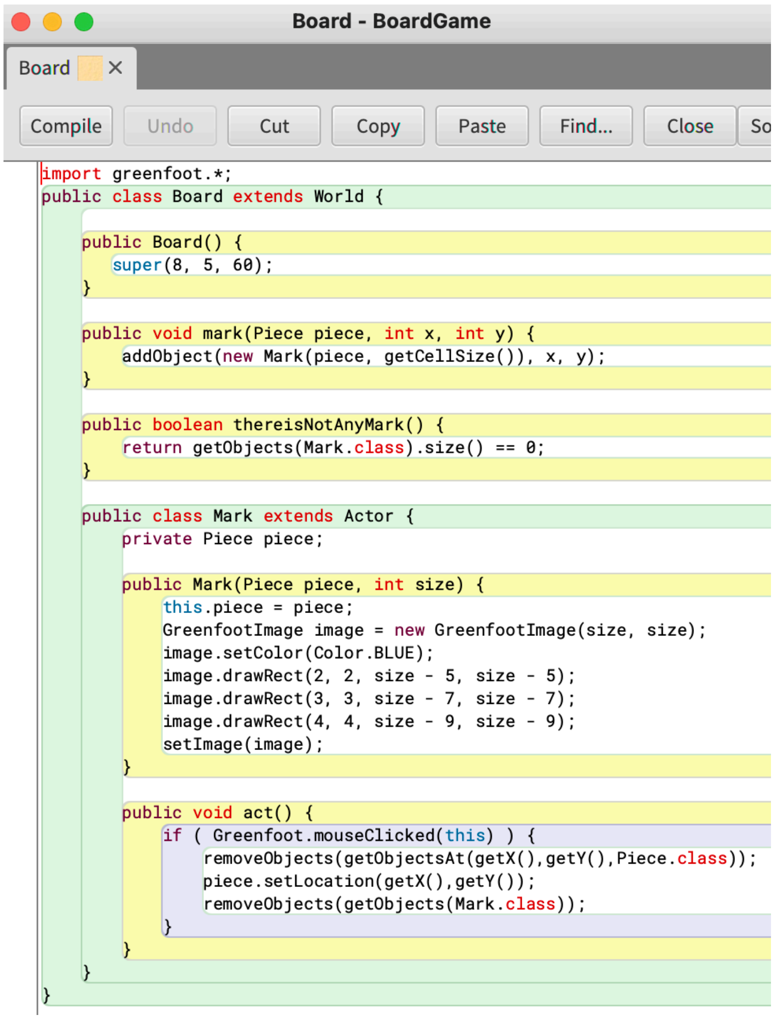
The teacher can prevent students from using the **Pieces** class to create pieces by improving the design of the exercise by marking the ***Piece*** class as ***abstract***. Below is a possible full version of the exercise.



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Gree

/**
 * Write a description of class $CLASSNAME here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class $CLASSNAME $EXTENDSANDSUPERCLASSNAME
{
    /**
     * Use de method mark(x,y) para indicar que
     * $CLASSNAME se puede mover a la posición (x,y).
     */
    public void allowedMoves()
    {
        // Add mark(x,y) calls
    }
}
```

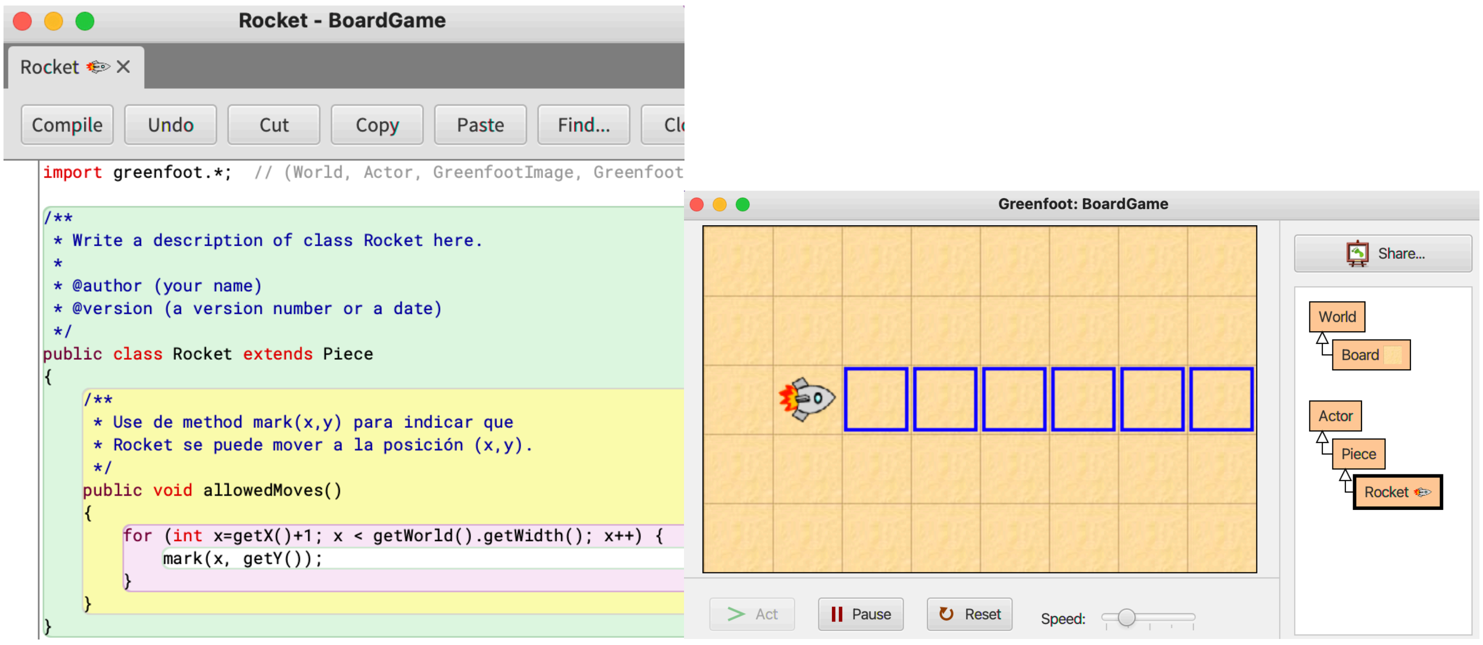


```
import greenfoot.*;

public abstract class Piece extends Actor {

    public final void act() {
        if ( Greenfoot.mouseClicked(this)
              && getWorldOfType(Board.class).thereisNotAnyMark())
        {
            allowedMoves();
        }
    }

    public void mark(int x, int y) {
        getWorldOfType(Board.class).mark(this, x, y);
    }

    public abstract void allowedMoves();

}
```



```
import greenfoot.*;
public class Board extends World {

    public Board() {
        super(8, 5, 60);
    }

    public void mark(Piece piece, int x, int y) {
        addObject(new Mark(piece, getCellSize()), x, y);
    }

    public boolean thereisNotAnyMark() {
        return getObjects(Mark.class).size() == 0;
    }

    public class Mark extends Actor {
        private Piece piece;

        public Mark(Piece piece, int size) {
            this.piece = piece;
            GreenfootImage image = new GreenfootImage(size, size);
            image.setColor(Color.BLUE);
            image.drawRect(2, 2, size - 5, size - 5);
            image.drawRect(3, 3, size - 7, size - 7);
            image.drawRect(4, 4, size - 9, size - 9);
            setImage(image);
        }

        public void act() {
            if ( Greenfoot.mouseClicked(this) ) {
                removeObjects(getObjectsAt(getX(),getY(),Piece.class));
                piece.setLocation(getX(),getY());
                removeObjects(getObjects(Mark.class));
            }
        }
    }
}
```

The behavior that the ***act()*** method defines for all pieces causes the ***allowedMoves()*** method to be called when a piece is clicked, and the behavior that the ***act()*** method defines for blue-ticked squares causes all blue-marked squares to be cleared, then it moves the selected piece to that square and finally removes all blue marks from the board. To practice the control structure for loop, students can have as an exercise programming the motion of a rocket that must be able to move from their position to any position to its right in a straight line, as shown below.



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

/**
 * Write a description of class Rocket here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Rocket extends Piece
{
    /**
     * Use de method mark(x,y) para indicar que
     * Rocket se puede mover a la posición (x,y).
     */
    public void allowedMoves()
    {
        for (int x=getX()+1; x < getWorld().getWidth(); x++) {
            mark(x, getY());
        }
    }
}
```

This exercise offers the possibility of many variations that have not been included in this document in order to make the explanation of the template use as simple as possible. At the **University of Las Palmas de Gran Canaria**, students use a variant of this scenario to practice control structures by programming a chess game.