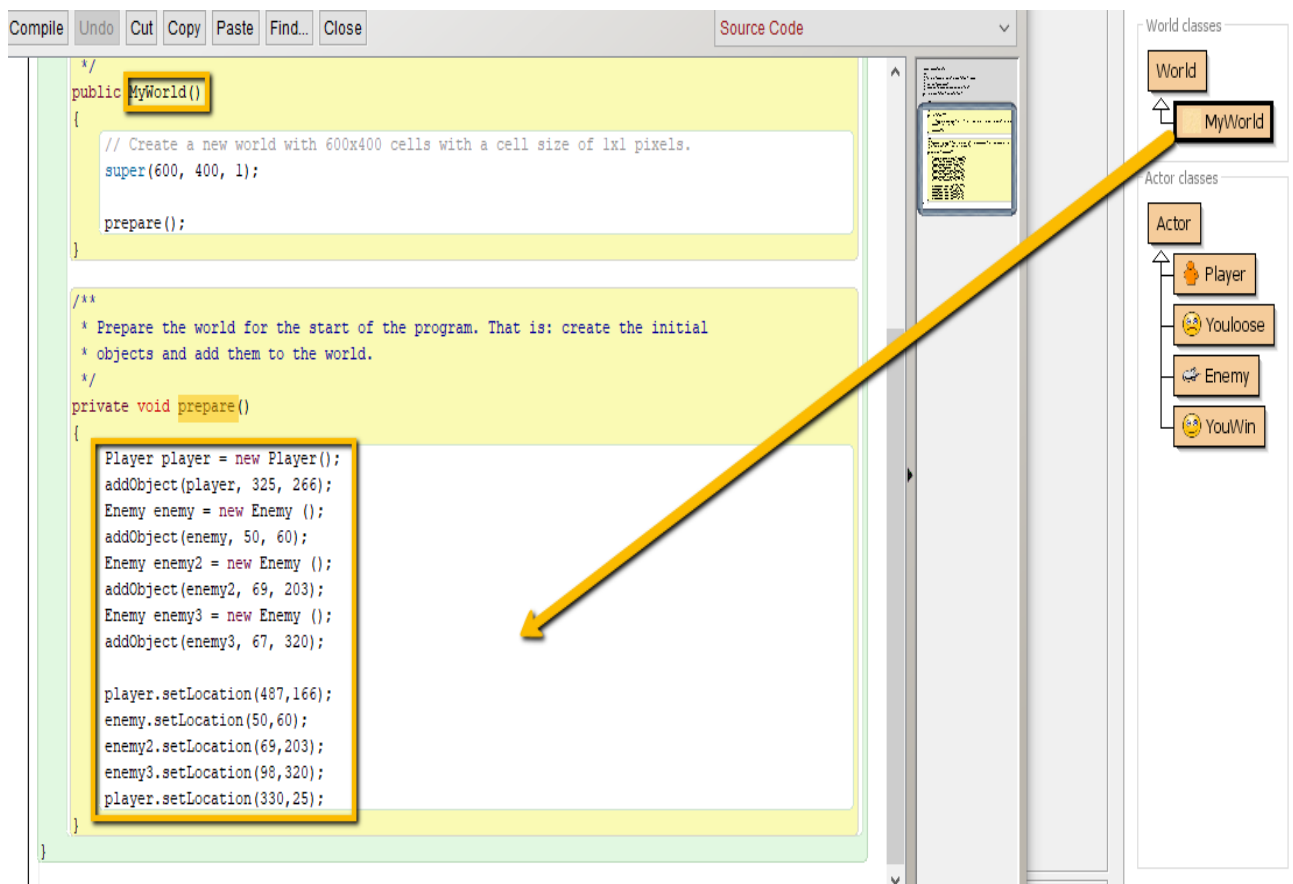
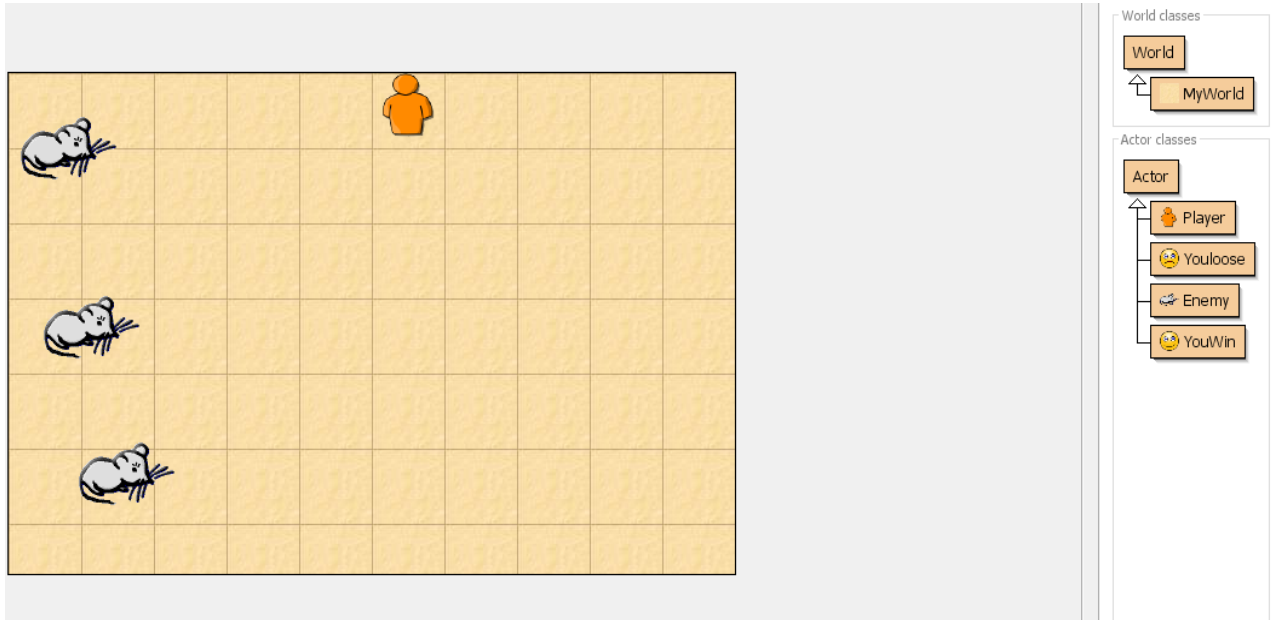


Άσκηση 3^η

Δημιουργήστε τις κλάσεις και τις υποκλάσεις όπως απεικονίζονται παρακάτω, γράψτε τους κώδικες στους αντίστοιχους editor και «τρέξτε» το πρόγραμμα....



```
import greenfoot.*;

/**
 * Write a description of class Youloose here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Youloose extends Actor
{
    /**
     * Act - do whatever the Youloose 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.
    }
}
```

Δεν γράφουμε τίποτα ως εντολή. Το δηλώσαμε μόνο ως υποκλάση

```
graph TD
    Actor --> Player
    Actor --> Youloose
    Actor --> Enemy
    Actor --> YouWin
```

```
import greenfoot.*;

/**
 * Write a description of class YouWin here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class YouWin extends Actor
{
    /**
     * Act - do whatever the YouWin 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.
    }
}
```

Δεν γράφουμε τίποτα ως εντολή. Απλά το δηλώσαμε ως υποκλάση

```
graph TD
    Actor --> Player
    Actor --> Youloose
    Actor --> Enemy
    Actor --> YouWin
```

Προγραμματισμός με Java στο GreenFoot

The screenshot shows the GreenFoot IDE with the `Player` class code. The `int speed = 2;` line is highlighted in red. The `act()` method is highlighted in yellow, and the `moveAround()` method is highlighted in blue. The `moveAround()` method contains logic for turning and changing speed based on key presses. To the right, a class hierarchy diagram shows `World` and `MyWorld` as world classes, and `Actor`, `Player`, `Youlose`, `Enemy`, and `YouWin` as actor classes. Arrows point from the `Player` class in the diagram to the `act()` and `moveAround()` methods in the code.

```
public class Player extends Actor
{
    int speed = 2;
    /**
     * Act - do whatever the Player wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        moveAround();
        hitEnemy();
        youWin();
    }
    public void moveAround()
    {
        move (speed);
        if (Greenfoot.isKeyDown("right"))
        {
            turn (4);
        }
        if (Greenfoot.isKeyDown("left"))
        {
            turn (-4);
        }
        if (Greenfoot.isKeyDown("space"))
        {
            speed = speed + 2;
        }
        else if (Greenfoot.isKeyDown("v"))
        {
            speed = 1;
        }
    }
}
```

The screenshot shows the `hitEnemy()` and `youWin()` methods of the `Player` class. The `hitEnemy()` method is highlighted in yellow and contains logic to add a `Youlose` object and remove the `Player` object when an enemy is hit. The `youWin()` method is highlighted in blue and contains logic to add a `YouWin` object and remove the `Player` object when the player reaches the end of the world. To the right, a class hierarchy diagram shows `Player`, `Youlose`, `Enemy`, and `YouWin` as actor classes. Arrows point from the `Youlose` and `YouWin` classes in the diagram to the corresponding object creation lines in the `hitEnemy()` and `youWin()` methods.

```
public void hitEnemy()
{
    if (isTouching(Enemy.class))
    {
        getWorld().addObject(new Youlose(), 400, 300);
        getWorld().removeObject(this);
        Greenfoot.stop();
    }
}
public void youWin()
{
    if (getY() >= 390)
    {
        getWorld().addObject(new YouWin(), 400, 300);
        getWorld().removeObject(this);
        Greenfoot.stop();
    }
}
```

```
public class Enemy extends Actor
{
    /**
     * Act - do whatever the Enemy wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        moveAround();
    }
}
```

The UML class diagram on the right shows a hierarchy where 'Actor' is the superclass for 'Player', 'Youlose', 'Enemy', and 'YouWin'. A yellow arrow points from the 'Enemy' class in the diagram to the 'act()' method in the code editor.

```
public void moveAround()
{
    move(4);
    if (getX() >= 599)
    {
        setLocation(0, getY());
    }
    else if (getX() <= 1)
    {
        setLocation(599, getY());
    }
    else if (getY() >= 390 )
    {
        setLocation(getX(), 0);
    }
    else if (getY() < 1 )
    {
        setLocation(getX(), 390);
    }
    if (Greenfoot.isKeyDown("q"))
    {
        turn(-1);
    }
    if (Greenfoot.isKeyDown("w"))
    {
        turn(1);
    }
}
}
```

The UML class diagram on the right shows 'World' as a superclass for 'MyWorld'. Below it, the 'Actor classes' section shows a hierarchy where 'Actor' is the superclass for 'Player', 'Youlose', 'Enemy', and 'YouWin'. A yellow arrow points from the 'Enemy' class in the diagram to the 'moveAround()' method in the code editor.