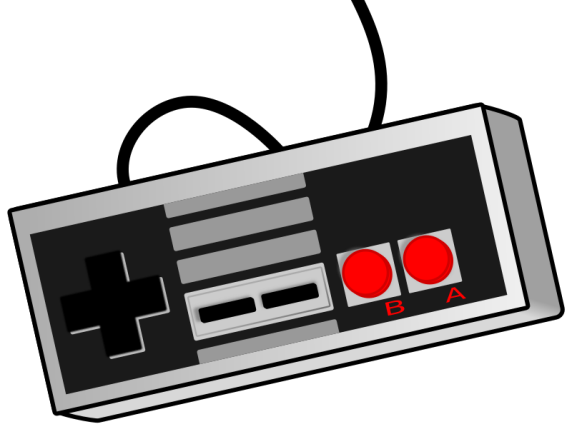


Lab_9

Greenfoot Simulation (Part 2)

420-141-VA - GAME PROGRAMMING 1 - VANIER COLLEGE

Step 3: Open Greenfoot Simulation

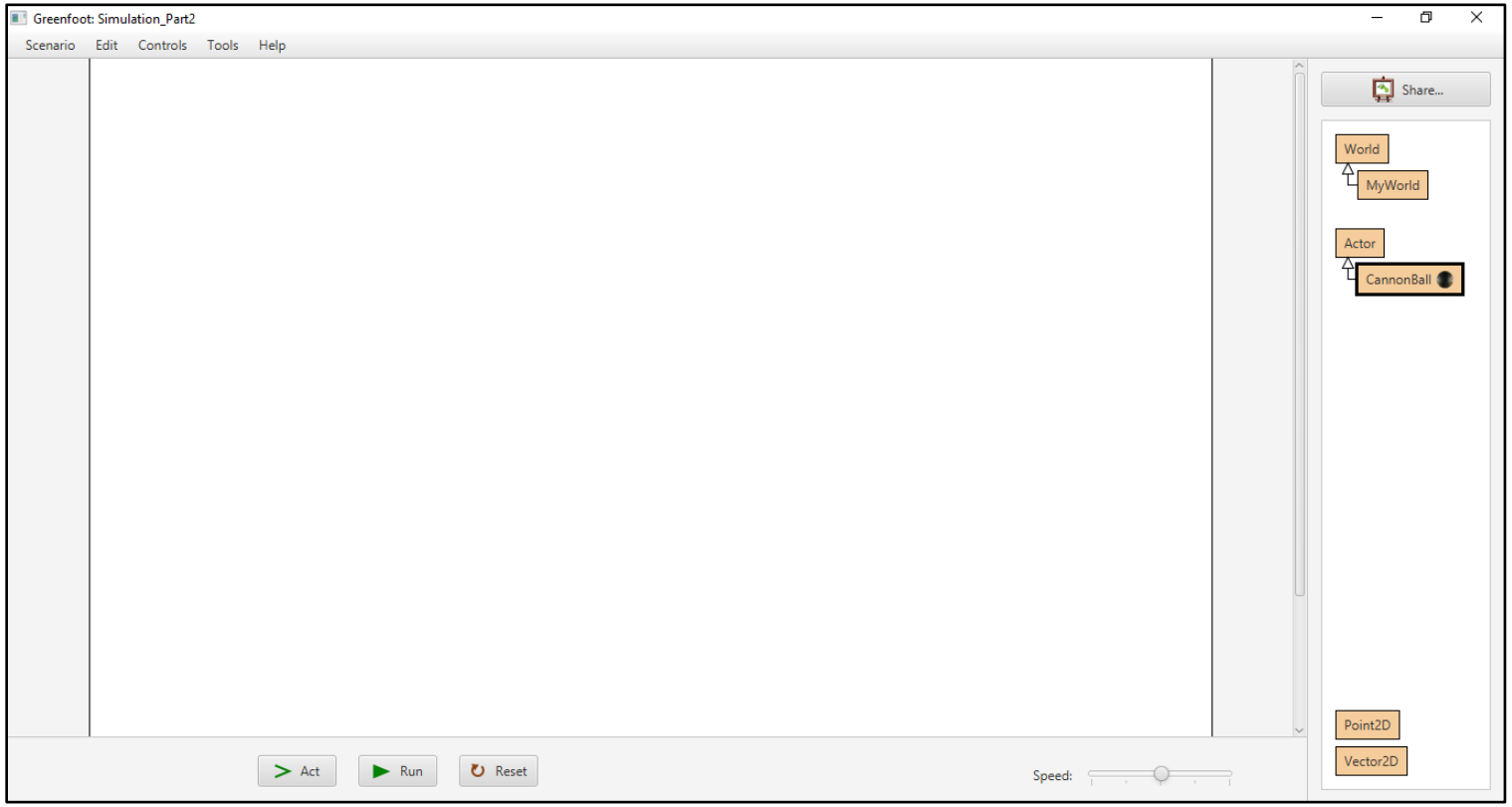


Figure 7

Step 4: Enhance Greenfoot World

Instantiate 4 Cannon balls at the top of the world

- Run the scenario (**Figure 8a**)

Remove the Cannon balls from the world

Create new Actor classes

- Create a Cannon class with the cannon image
- Create a Target class with the target image

Setup the scene to include a cannon and multiple targets

- See image on the right for reference (**Figure 8b**)

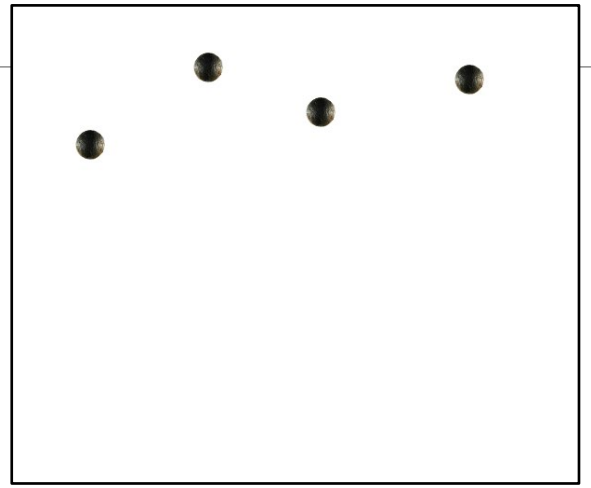


Figure 8a

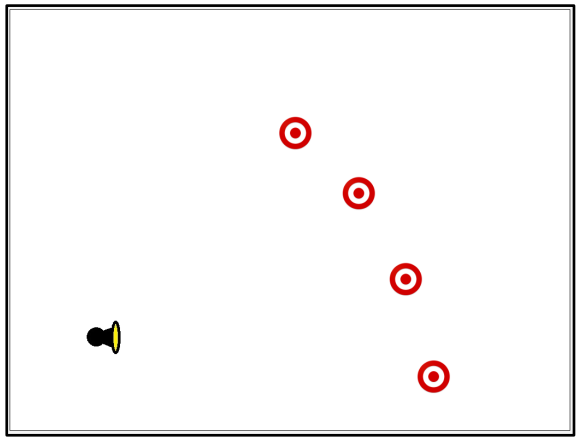
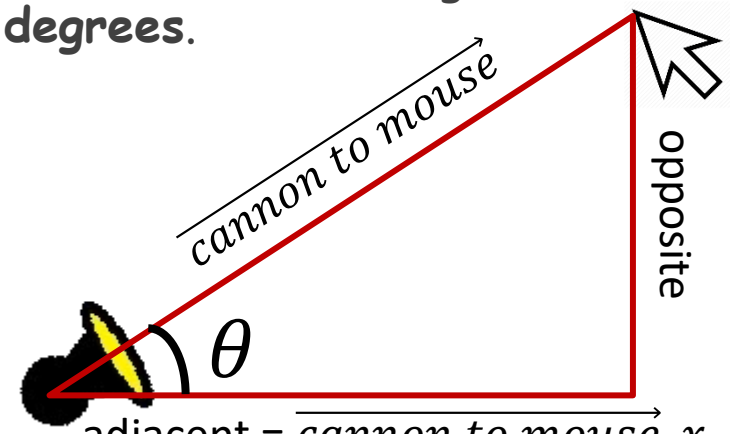


Figure 8b

Step 5: Making the Cannon Rotate towards Mouse

Using Trigonometry, we calculate the angle for the cannon to always face towards the mouse.

Java uses **radians** for angles, Greenfoot uses **degrees**.



adjacent = $\overrightarrow{cannon\ to\ mouse}.x$

$\tan\theta = opposite/adjacent$

$\theta = \tan^{-1}(\frac{opposite}{adjacent})$

```
public class Cannon extends Actor
{
    public void act()
    {
        MouseInfo mouse = Greenfoot.getMouseInfo();

        if (mouse != null)
        {
            Vector2D cannonToMouse = new Vector2D(mouse.getX() - getX(),
                                                  mouse.getY() - getY());

            double adjacent = cannonToMouse.getX();
            double opposite = cannonToMouse.getY();

            double angleRadians = Math.atan2(opposite, adjacent);
            double angleDegrees = Math.toDegrees(angleRadians);

            setRotation((int) angleDegrees);
        }
    }
}
```

Step 6: Making the Cannon shoot Cannon balls

The code for aligning the cannon can be refactored in the method **alignWithVector()**:

```
public void alignWithVector(Vector2D v)
{
    double adjacent = v.getX();
    double opposite = v.getY();

    double angleRadians = Math.atan2(opposite, adjacent);
    double angleDegrees = Math.toDegrees(angleRadians);

    setRotation((int) angleDegrees);
}
```

Then, we instantiate Cannon Balls towards the mouse when the mouse is clicked. The velocity can be set by normalizing the vector and multiplying with a constant.

```
private static final double CANNON_BALL_VELOCITY = 1500.0;

public void act()
{
    MouseInfo mouse = Greenfoot.getMouseInfo();

    if (mouse != null)
    {
        Vector2D cannonToMouse = new Vector2D(mouse.getX() - getX(),
                                              mouse.getY() - getY());
        alignWithVector(cannonToMouse);

        if (Greenfoot.mouseClicked(null))
        {
            cannonToMouse.normalize();
            cannonToMouse = Vector2D.multiply(cannonToMouse, CANNON_BALL_VELOCITY);

            CannonBall newBall = new CannonBall();
            newBall.setVelocity(cannonToMouse);
            getWorld().addObject(newBall, getX(), getY());
        }
    }
}
```

Step 7: Commit your changes and push to github (1)

Try to remember the following commands (see Figure_9a and Figure_9b)

- **git add ***
- **git status** /*check that files are there*/
- **git commit -m "Add cannon rotation and shoot balls in world"**

(if you need to enter your name and email, just use the setup commands and commit again)

And then push your commit to Github

- **git push**

  /* Push the files to your repository, if you don't do this step, your files will not be saved online */

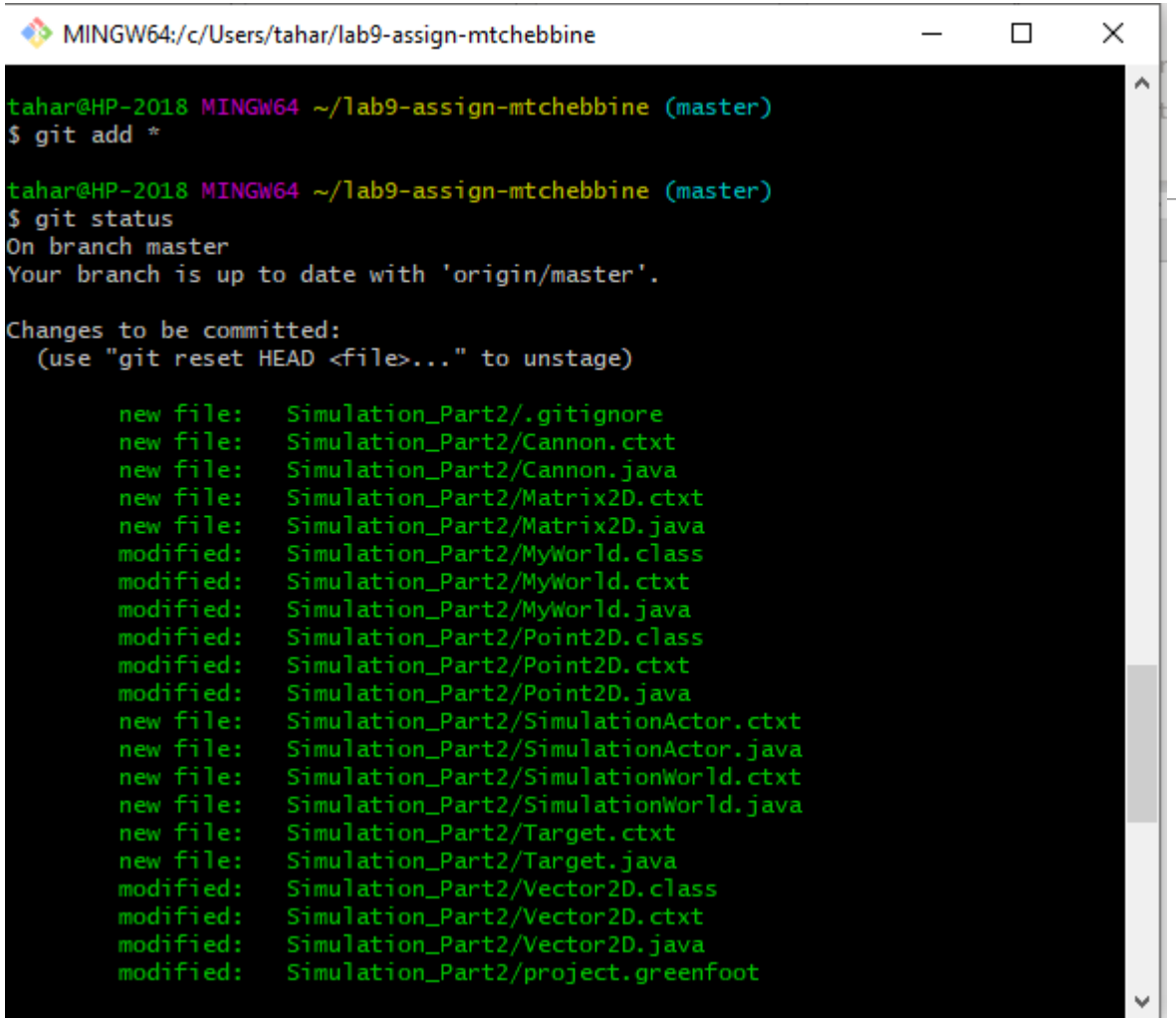


Figure 9a

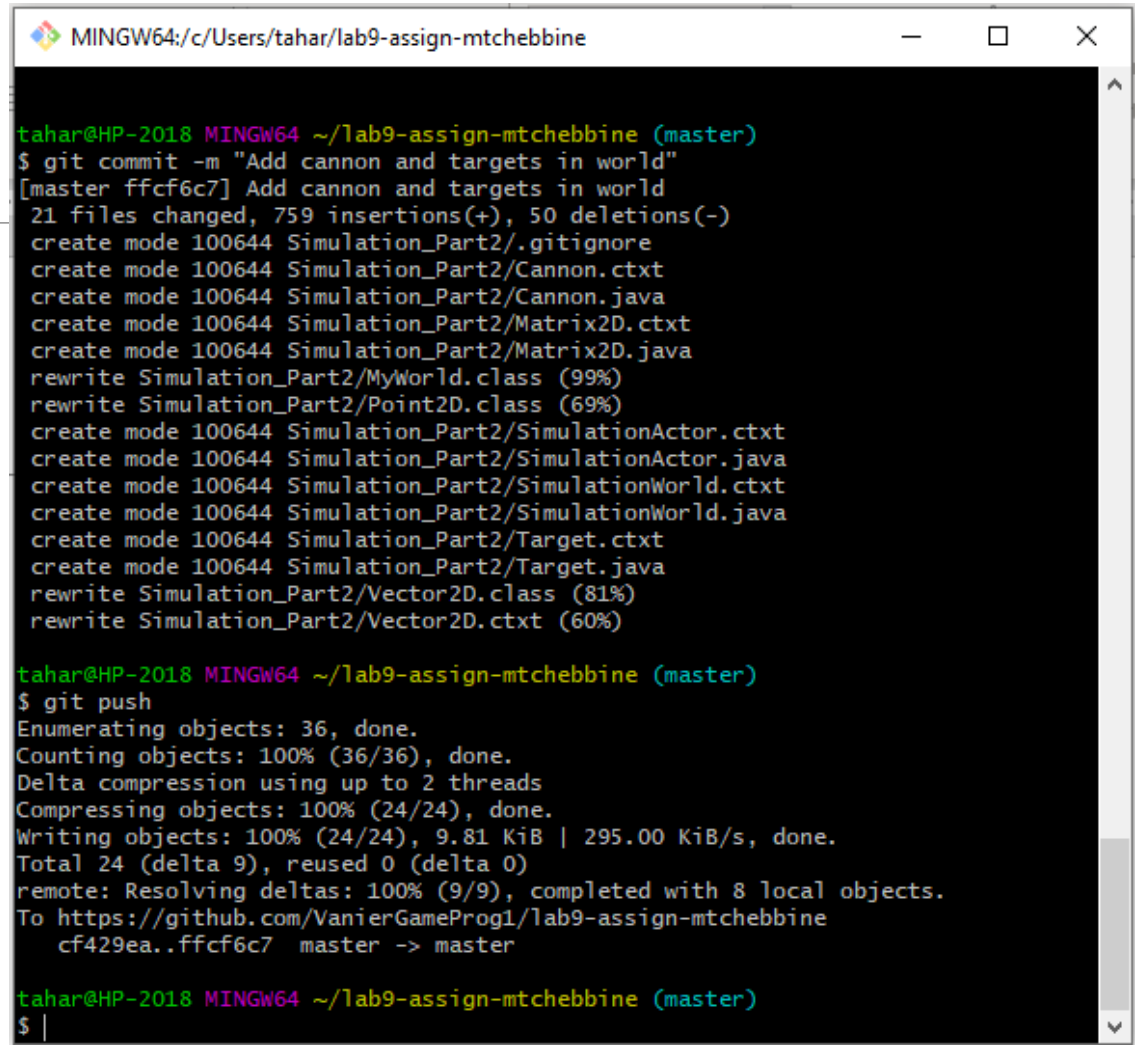
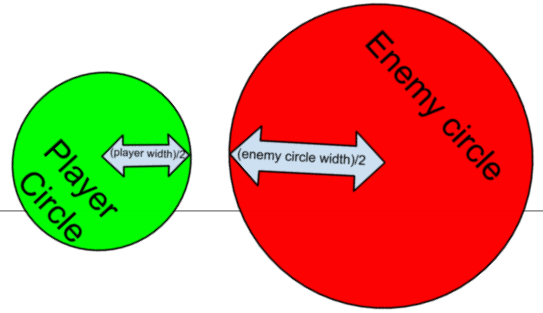


Figure 9b

Projectiles and Collisions



Projectiles at constant speed

- Instantiate a projectile at an initial position
- Set the velocity vector towards a destination (v = destination – initial)
- Set the magnitude of your vector to be the speed you need (normalize then multiply by speed)

Detecting collision between 2 circles of arbitrary position (Vector2 c_1 and c_2) and radius (float r_1 and r_2)

- Calculate distance between 2 centers (c_1 and c_2)

$$d = \sqrt{(c_2.x - c_1.x)^2 + (c_2.y - c_1.y)^2}$$

- if ($d < r_1 + r_2$), then the 2 circles collide

You can get projectiles collision circle approximately by setting the radius to half the size of the sprite

Every frame, detect collision between each pair of circles that can collide

Step 8: Collisions for targets and cannon balls

Targets can check for collision with each Cannon balls.

1. Retrieve the list of cannon balls
2. For each cannon ball, calculate the distance with target
3. Test if the distance is less than the target's radius + ball's radius
4. When collision is detected, replace target image

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

public class Target extends Actor
{
    public void act()
    {
        // Detect intersection with each cannon ball in the World
        List<CannonBall> cannonBalls = getWorld().getObjects(CannonBall.class);

        for (int i=0; i < cannonBalls.size(); i++)
        {
            CannonBall ball = cannonBalls.get(i);
            Vector2D targetToBall = new Vector2D(ball.getX() - getX(), ball.getY() - getY());
            double distance = targetToBall.magnitude();

            if (distance < getImage().getHeight() / 2 + ball.getImage().getHeight() / 2)
            {
                setImage(new GreenfootImage("targetDestroyed.png"));
            }
        }
    }
}
```

You need to import this for List

Step 9: Run Greenfoot Simulation

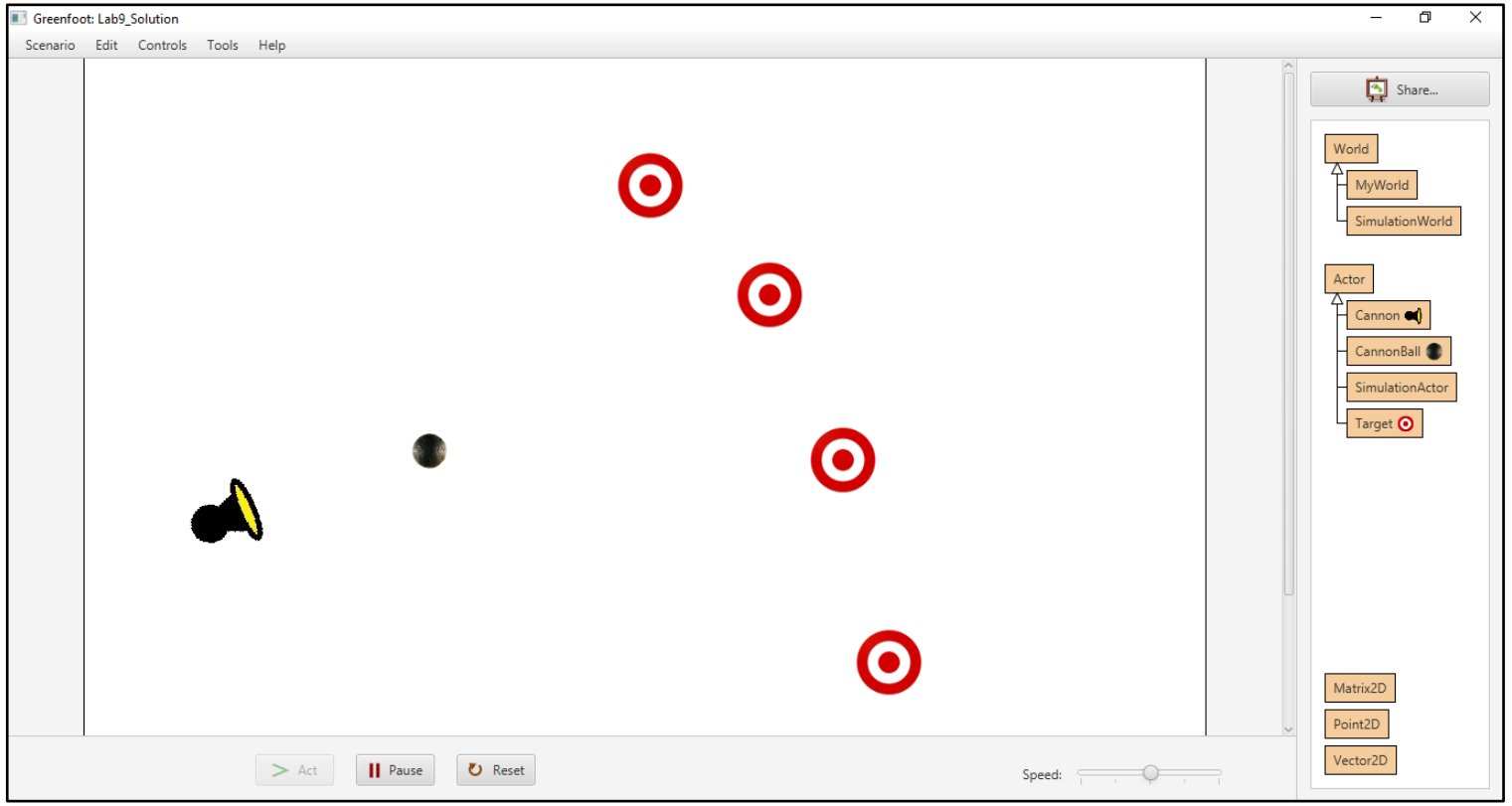


Figure 11

Step 10: Commit your changes and push to github (2)

Try to remember the following commands

- **git add ***
- **git status** /*check that files are there*/
- **git commit -m "Add targets in world"**

(if you need to enter your name and email, just use the setup commands and commit again)

And then push your commit to Github

- **git push**

  /* Push the files to your repository, if you don't do this step, your files will not be saved online */

-

Double-check Github.com for your commit