Universidade Federal do Ceará

Campus de Sobral – Engenharia da Computação

Técnicas de Programação – Trabalho Dirigido 05 – 18/12/2015

Prof. Iális Cavalcante

Entrega: dia 13/01/2016.

Este trabalho dirigido está voltado para implementação em dois novos ambientes na disciplina: **Greenfoot** e **Robocode**. O primeiro é disponibilizado pela própria Oracle como uma ferramenta didática (desenvolvido em University of Kent e Deakin University) para estudo da linguagem de programação Java; enquanto o segundo é um ambiente que une programação e entretenimento, projetado por Mathew A. Nelson e seus demais colaboradores, em um jogo de batalha de robôs. Atente para as questões a seguir, siga as instruções e comece o desenvolvimento!

Questão 1

Greenfoot (http://www.greenfoot.org/)

Observe a figura que segue:

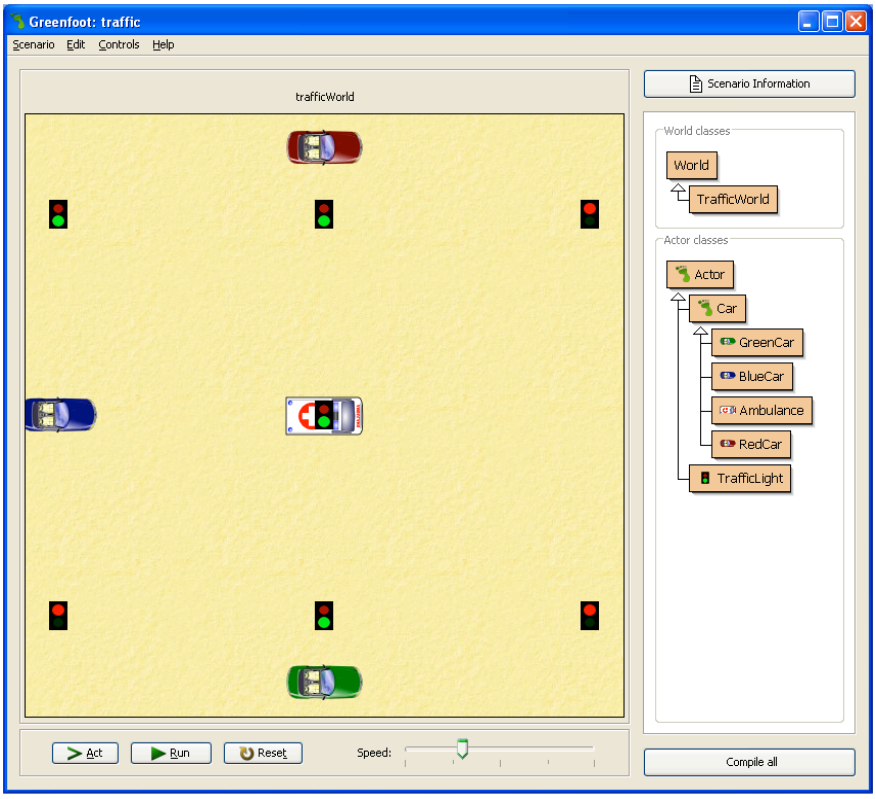

Figura 1. Tela principal a ser projetada.

Para recordar, acompanhe material complementar (Introdução ao Greenfoot) disponível em no SIGAA.

Use o cenário **wombats2** como base: crie suas novas classes e compare com esta implementação já pronta! Crie um cenário chamado *traffic* acessando *Scenario* > *New*:

- Crie uma classe chamada **TrafficWorld** (compare com **WombatWorld**) que tenha o código a seguir:

```
import greenfoot.*;

import java.util.Random;

/**
 * Uma simulação de tráfego com carros e semáforos.
 *
 * Define o layout dos objetos no mundo.
*/

public class TrafficWorld extends World
{
   private int x;
   private Random randomizer = new Random();

   /**
    * Construtor para objetos da classe TrafficWorld
    *
    * Cria um novo mundo com células 8x8 e com uma célula maior de 60x60 pixels
    */
   public TrafficWorld()
   {
      super(9, 9, 65);
      setBackground("sand.jpg");
   }

   /**
    * Povoar o mundo com um cenário fixo de carros e semáforos.
    */
   public void populate()
   {
      RedCar rc = new RedCar();
      addObject(rc, 4, 0); // posiciona o objeto por [coluna,linha]

      GreenCar gc = new GreenCar();
      addObject(gc, 4, 8);

      BlueCar bc = new BlueCar();
```

```
        addObject(bc, 0, 4);

        Ambulance ambulance = new Ambulance();
        addObject(ambulance, 4, 4);

        TrafficLight tl1 = new TrafficLight();
        addObject(tl1, 0, 1);

        TrafficLight tl2 = new TrafficLight();
        addObject(tl2, 8, 1);

        TrafficLight tl3 = new TrafficLight();
        addObject(tl3, 0, 7);

        TrafficLight tl4 = new TrafficLight();
        addObject(tl4, 8, 7);

        TrafficLight tl5 = new TrafficLight();
        addObject(tl5, 4, 1);

        TrafficLight tl6 = new TrafficLight();
        addObject(tl6, 4, 7);

        TrafficLight tl7 = new TrafficLight();
        addObject(tl7, 4, 4);
    }

    /**
     * Liga e desliga as luzes no semáforo apresentado.
     *
     * @param  numLights   numero de semáfotos que são inseridos no mundo
     */
    public void randomLights(int numLights)
    {
        for(int i=0; i<numLights; i++) {
            TrafficLight tl = new TrafficLight();
            int x = Greenfoot.getRandomNumber(getWidth());
            int y = Greenfoot.getRandomNumber(getHeight());
            addObject(tl, x, y);
        }
    }
}
```

- Crie uma classe chamada **Car** que tenha o código a seguir:

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

/**
 * Car – uma classe para representar carros.
```

```
*/

public class Car extends Actor
{
   private static final int EAST = 0;
   private static final int WEST = 1;
   private static final int NORTH = 2;
   private static final int SOUTH = 3;

   private int direction;

   private GreenfootImage carRight;
   private GreenfootImage carLeft;

   /**
    * Construtor para objetos de classe Car
    */
   public Car()
   {
      carRight  = getImage();
      carLeft = new GreenfootImage(getImage());
      carLeft.mirrorHorizontally();

      setDirection(EAST);
   }

   /**
    * Indica qual direção o carro se movimenta.
    */
   public void act()
   {
      // descreva este código
   }

   /**
    * Checa se há um semáforo na mesma célula em que o carro está.
    */
   public boolean foundTrafficLight()
   {
      // descreva este código
   }

   /**
    * Verifica o semáforo e decide o que o carro deve fazer.
    */
   public void verifyTrafficLight()
   {
```

```
        // descreva este código
    }

    /**
     * Move uma célula à frente na direção atual.
     */
    public void move()
    {
        // descreva este código
    }

    /**
     * Testa se pode mover adiante.
     * Retorna true se pode mover, false em caso contrário.
     */
    public boolean canMove()
    {
        // descreva este código
    }

    /**
     * Toma uma direção aleatória.
     */
    public void turnRandom()
    {
        // descreva este código
    }

    /**
     * Direciona pra esquerda.
     */
    public void turnLeft()
    {
        // descreva este código
    }

    /**
     * Modifica a direção a ser seguida.
     */
    public void setDirection(int direction)
    {
        // descreva este código
    }
}
```

- Crie uma classe chamada **BlueCar** (compare com **Wombat**) que tenha o código a seguir:

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

```
/**
 * BlueCar – uma classe para representar qualquer carro azul.
 *
 */

public class BlueCar extends Car
{

   /**
    * Construtor para objetos da classe BlueCar
    */
   public BlueCar()
   {
   }

   /**
    * Indica qual direção o carro azul se movimenta.
    */
   public void act()
   {
      // descreva este código
   }

   /**
    * Verifica o semáforo e decide o que o carro azul pode fazer. Dobrar à direita, preferencialmente!
    */
   public void verifyTrafficLight()
   {
      // descreva este código
   }

}
```

- Crie uma classe chamada **Ambulance** (compare com **Wombat**) que tenha o código a seguir:

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

/**
 * Ambulance – uma classe para representas as ambulâncias.
 *
 */

public class Ambulance extends Car
{

   /**
```

```
    * Construtor para objetos da classe Ambulance
    */
   public Ambulance()
   {
   }

   /**
    * Indica qual direção a ambulância se movimenta.
    */
   public void act()
   {
      // descreva este código
   }

   /**
    * Não verifica o semáforo e a ambulância segue seu caminho.
    */
   public void verifyTrafficLight()
   {
      // nenhum semáforo deve ser considerado! – prioridade no tráfego para a ambulância...
      // descreva este código
   }
}
```

- Crie uma classe chamada **TrafficLight** (compare com **Rock** e **Leaf**) que tenha o código a seguir:

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

/**
 * TrafficLight - uma classe para representar semáforos.
 *
 */

public class TrafficLight extends Actor
{
   private int counter;
   private boolean freePassage;

   /**
    * Construtor para objetos da classe TrafficLight
    */
   public TrafficLight()
   {
      setCounter(0);
      /* Math.round(Math.random()*1) - returns 0 or 1
       * if equals to 0 -> red light
       * if equals to 1 -> green light
```

```
     */
    if(Math.round(Math.random()*1) == 0)
    {
       setFreePassage(false);
       setImage("semaphore-closed.gif");
    }
    else
    {
       setFreePassage(true);
       setImage("semaphore-opened.gif");
    }
 }

 /**
  * Liga/desliga o semáforo.
  */
 public void act()
 {
    setCounter(getCounter()+1);
    if(getCounter()%20 == 0) // atualiza o status do semáforo para cada vinte rodadas
    {
       if(getFreePassage())
       {
          setFreePassage(false);
          setImage("semaphore-closed.gif");
       }
       else
       {
          setFreePassage(true);
          setImage("semaphore-opened.gif");
       }
    }
 }

 /**
  * Modifica o valor do contador que é relacionado ao semáforo.
  *
  * @param  value   novo valor do contador
  */
 public void setCounter(int value)
 {
    this.counter = (value >= 0 ? value : 0);
 }

 /**
  * Recupera o valor do contador que é relacionado ao semáforo.
  *
```

```
     * @return     o valor atual do atributo contador
     */
    public int getCounter()
    {
        return this.counter;
    }

    /**
     * Modifica o valor de freePassage que indica quando um carro pode passar direto.
     *
     * @param  status   novo status do semáforo
     */
    public void setFreePassage(boolean status)
    {
        this.freePassage = status;
    }

    /**
     * Recupera o status do semáforo.
     *
     * @return     status do semáforo
     */
    public boolean getFreePassage()
    {
        return this.freePassage;
    }
}
```

A cada método que pede a descrição do código (texto em vermelho), você deverá desenvolver o código que corresponda ao objetivo descrito no comentário de seu código. Para ajudar neste objetivo, estude o funcionamento das classes referenciadas do cenário **wombats2**. Com isso você terá boa parte deste novo cenário desenvolvida.

Após a implementação dessas classes, associa cada uma delas no ambiente à sua imagem representativa (ver diretório *images*). Clique em *Compile all* para configurar todas as classes de acordo com sua implementação e corriga possíveis erros que venham a aparecer (notar que a cada correção/modificação em qualquer classe, é exigido compilar novamente o cenário).

Clique com o botão direito no nome *trafficLight* (objeto do mundo, nome que aparece acima do cenário) e selecione o método *populate()*.Para ver a ação de cada objeto por rodada, clique em *Act*. Mas se quiser que todos atuem de forma constante para ver o desenrolar do seu projeto de cenário, clique em *Run* para executá-lo.

Como teste para sua implementação neste ambiente, crie duas novas classes: **GreenCar** e **RedCar** que devem, respectivamente, andar no sentido contrário e se mover para a esquerda. Associe cada nova classe nova à sua imagem correspondente e veja a execução do cenário após essas modificações.

Questão 2
Robocode

Para saber um pouco mais sobre o ambiente **Robocode** e como desenvolver seus protótipos no mesmo, acompanhe material complementar (Curso para Jovens Programadores) disponível no SIGAA.

Inicie o programa **Robocode**. Verifique no menu *Battle* a opção *Open*. Abra então o arquivo *sample* disponível e inicie a escolha dos robôs que participarão da batalha a ser iniciada: selecione com o mouse no campo *Robots* (e veja sua descrição no texto que surge abaixo da tela) e clique em *Add* quando for um robô de interesse. Quando quiser iniciar a batalha, clique no botão *Start Battle* (quer coisa mais intuitiva que essa?). Então é só ver o resultado da batalha de robôs iniciada!

Agora é a sua vez de preparar um robô para a batalha! Na tela inicial do **Robocode**, acesse o menu *Robot* e entre na opção *Editor*. Dentro do editor de robôs, siga para o menu *File* > *New* > *Robot*. Nomeie o seu robô e o pacote a qual ele deve pertencer. Para ajudar na sua implementação acesse: *File* > *Open*. Abra o código de quaisquer robôs já disponíveis nesse ambiente e utilize os códigos visualizados no seu desenvolvimento.

Crie um robô diferenciado dos que já existem (com características novas ou mesclando comportamentos já existentes em outros robôs) e envie até o prazo definido. Seu protótipo será desafiado pelos robôs desenvolvidos pelos seus colegas de turma. Boa sorte!
