Disciplina: Programação com Interfaces Gráficas
AD1 2^o semestre de 2020.

AD1: Simulação de Doença

1 Informações Gerais

Objetivos: Praticar conceitos importantes de Programação Orientada a Objetos (OO) aprendidos em Programação com Interfaces Gráficas, praticar bons estilos de documentação usando o Doxygen [1], criar código robusto que funcione de acordo com a especificação do trabalho e criar código de teste com o *unittest* [2] para testar a correção do programa. Neste trabalho, você usará os seguintes conceitos:

1. Classes e interfaces
2. Herança
3. Sobreposição
4. Listas aninhadas
5. Exceções:
   (a) ValueError, lançada tipicamente quando os valores dos parâmetros estão nos intervalos errados.
   (b) RuntimeError, lançada tipicamente quando uma referência nula é encontrada onde não deveria ser.
   (c) SyntaxError, que é usada quando a aplicação estiver em um estado que não deveria estar.

[1] `http://www.doxygen.nl/manual/doxygen_usage.html`
[2] `https://docs.python-guide.org/writing/tests/`

(d) ValueError, lançada quando um valor não estiver no formato esperado. Por exemplo, o programa espera um inteiro, mas recebe um float no lugar.

(e) IOError, lançada quando houver um problema na abertura, leitura ou escrita em um arquivo.

6. Classes básicas de Python: Object, String, List.

7. Tipos primitivos: int, float, boolean, etc.

Importante: Comece o trabalho cedo. Aqui estão as etapas recomendadas para lidar com a tarefa, para que ela não seja muito esmagadora. Não se deixe intimidar por uma descrição longa, pois ela serve para ajudá-lo a realizar a tarefa.

1. Leia do início até o final da Seção 2 para obter uma visão geral do problema e de como as classes estão relacionadas. Pare e pense se entendeu esta seção ou não. Se não, leia-a novamente.

2. Leia a Seção 3 e o código Python fornecido parcialmente e entenda os métodos a serem implementados e pense na maneira de implementá-los. Há sugestões disponíveis nessa seção.

3. Leia a Seção 4 e tente entender a saída do programa dado um arquivo de entrada particular.

4. Implemente cada classe. Comece com as classes World e Actor. Certifique-se que todo método valide seus parâmetros de entrada e lance a exceção apropriada em cada caso inválido. Implemente as classes Disease e MyWorld. Desenvolva código de teste com o *unittest* [3], para as classes, a medida que as implementa.

5. Implemente o Simulador. Use o arquivo de configuração e verifique se obteve a mesma saída.

Verifique se a saída está no formato correto, com os pontos decimais e a ortografia (*spell*) apropriados, e tente com outros arquivos de configuração para verificar se o seu código gera resultados consistentes. Verifique se os nomes das classes, as assinaturas dos métodos (nome do método, parâmetros, tipos e ordem dos parâmetros) e os tipos de retorno estão de acordo com a especificação. Nomes de variáveis podem ser escolhidos por você, mas selecione nomes significativos para facilitar a depuração.

Sugestão: é uma boa prática manter privadas as variáveis de classe e instância de objeto, e fornecer métodos públicos (*getters* e *setters*) para ler e modificar os valores das variáveis privadas [4].

[3] `https://www.tutorialspoint.com/unittest_framework/unittest_framework_overview.htm`

[4] Python não possui o conceito de variável privada. No entanto, dentro de uma classe, um identificador como `__spam` é textualmente alterado para `_classname__spam`, onde `classname` é o nome da classe com os *underscore(s)* removidos.

2 Descrição Geral do Problema

Ciência da Computação, Engenharia da Computação e Engenharia de Software são campos que contribuíram para avanços em outras disciplinas das ciências e engenharia. Programas de computador são usados para modelar e prever o clima, controlar robôs para aplicações de vigilância, prever interações medicamentosas, prever o crescimento do câncer, prever a propagação de uma determinada doença (por exemplo, vírus) em todo o mundo etc.

Nesta tarefa, solicitamos que você escreva um simulador simplificado para calcular a força de uma doença, depois de ter sido plantada em um mundo virtual, e a força acumulada de todas as doenças neste mundo.

Por simplicidade, assumimos que a doença não pode se mover, mas fica mais forte após cada unidade de tempo (por exemplo, um dia), quando estiver na região do mundo com a temperatura adequada para o seu crescimento. Inicialmente, a força de uma doença vale 1. Após cada unidade de tempo decorrida, sua força é multiplicada pela sua taxa de crescimento para essa faixa de temperatura (por exemplo, 2 para a temperatura entre, digamos, 15 e 25 unidades de temperatura), se a doença estiver localizada na região do mundo com a temperatura média dentro desse intervalo.

O mundo simulado é dividido em quatro regiões iguais e não sobrepostas (quadrantes), e cada quadrante possui a mesma temperatura média. Em cada unidade de tempo, o programa relata a força acumulada de todas as doenças no mundo simulado. Você deve implementar cinco classes, e a Fig. 1 mostra como essas classes estão relacionadas.

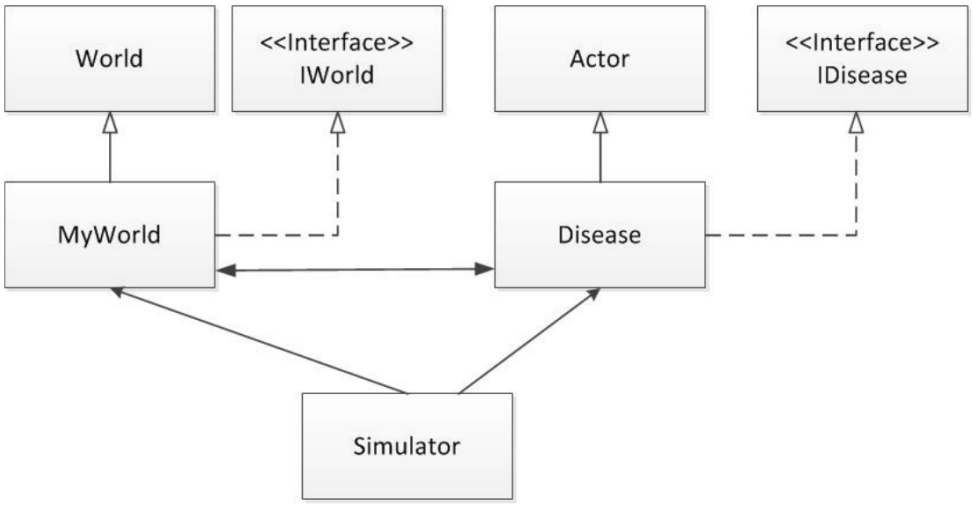

Figura 1: Diagrama de classes simplificado, onde as classes são mostradas em retângulos. Interfaces são marcadas com o termo $<< Interface >>$. As setas sólidas brancas indicam uma única hierarquia de herança (ou a relação de superclasse-subclasse). As setas sólidas pretas indicam interações entre classes. O Simulador chama os métodos da classe MyWorld e da classe Disease. A classe MyWorld e a classe Disease chamam os métodos uma da outra.

A classe MyWorld é uma subclasse da classe World e também implementa os métodos

especificados na interface IWorld [5]. Na terminologia Python, isso significa que "MyWorld deriva de World e de IWorld". A classe Disease é uma subclasse da classe Actor e implementa os métodos especificados na interface IDisease.

O Simulador possui um método principal que instancia primeiro um Objeto MyWorld, que lê um arquivo de configuração "simulation.config". Ele instancia objetos Disease e adiciona esses objetos a locais diferentes no objeto MyWorld. Ele define a temperatura em cada quadrante do seu mundo.

O Simulador entra em loop, chamando o método `act()` do objeto MyWorld, para relatar a força acumulada das doenças em todo o mundo nessa unidade de tempo e chamando o método `act()` de cada objeto Disease para calcular a nova força da doença após o término da unidade de tempo.

O Simulador também instancia um objeto World e adiciona dois objetos Actor neste mundo. Em seguida, ele executa uma simulação semelhante. Você pode ver a diferença nos resultados da simulação entre o uso de objetos MyWorld e World. Em seguida, vamos descrever os detalhes de cada classe.

3 Especificação dos Métodos

3.1 class World

```
##
 # Construct a new world.
 # The world is represented by a two-dimensional array
 # of cells with the specified width and height.
 #
def __init__(self, worldWidth, worldHeight):
```

Veja a Fig. 2. Uma célula pode manter no máximo cinco objetos Actor.

Há várias maneiras de manter objetos Actor no mundo, por exemplo, usando as classes do Python Collections [6]. Pedimos que seja implementada apenas uma forma específica, descrita a seguir.

Requisitos de Implementação

```
##
 #  Use a three-dimensional array of Actor objects (three nested lists).
 #  The maximum number of elements in the 1st dimension
 #  is equal to worldHeight;
 #  the maximum number of elements in the 2nd dimension
 #  is equal to worldWidth;
```

[5] Infelizmente, Python não possui interface ou, pelo menos, não totalmente embutida na linguagem. Empregue, para isso, a classe base abstrata do Python, ou, graciosamente, ABC. Funcionalmente, as classes base abstratas permitem definir uma classe com métodos abstratos, que todas as subclasses devem implementar para serem inicializadas.

[6] `https://docs.python.org/3/library/collections.html`

```
 #  and the maximum number of elements in the 3rd dimension is 5.
 #  After you instantiate the array of Actor objects,
 #  initialize each array element to None.
 #
self.__grid = []

##
 #  This method does not do anything.
 #  Leave the body of the method blank.
 #
def act(self):
    pass

##
 #  Add the Actor object at the cell (x,y) of the
 #  world, after the most recently added object for that cell,
 #  if there are less than five objects in this cell.
 #
def addObject(self, object, x, y):
```

Implementação

Após a referência do objeto Actor ser armazenada na grade, certifique-se de que o objeto adicionado saiba que está neste mundo e que está nesta célula. Pense em quais métodos da classe Actor chamar.

Comentário: A verificação da validade de cada parâmetro de entrada de um método antes de usá-lo é uma boa prática de programação. Permite uma depuração mais fácil, especialmente em algoritmos recursivos. Os usuários do seu método também conhecem as causas do erro e podem verificar rapidamente o método chamado.

```
## Return the height of the world in number of cells.
def getHeight(self):

## Return the width of the world in number of cells.
def getWidth(self):

## Return the total number of objects in the world.
def numberOfObjects(self):

## Return an array of Actor objects in a list of
 # Python objects.
def getObjects(self):
```

Implementação

Para cada linha e cada coluna, adicione uma referência de objeto não nula à lista de objetos. Por exemplo, na Fig. 2, a lista de objetos retornada possui referências aos objetos 0, 4, 2, 1, e 3 nessa ordem.

```
##
 #  Assign aGrid to grid.
 #  Set the height of the grid to len(aGrid).
 #  Set the width of the grid to len(aGrid[0]).
 #  Set the number of Actor objects to len(aGrid[0][0]).
 #
def setGrid(self,aGrid):
```

Veja o código padrão fornecido para os casos que lançam exceções.

Verificação:

1. Após haver implementado essa classe corretamente, você deve ter entendido melhor como acessar cada elemento em uma matriz tridimensional.

3.2 class Actor

```
##
 #  Construct a new Actor object.
 #
 #  It sets the initial values for its member variables.
 #  It sets the unique ID for the object and initializes
 #  the reference to the World object to which this
 #  Actor object belongs to None.
 #  The ID of the first Actor object is 0.
 #  The ID gets incremented by one each time a new
 #  Actor object is created.
 #  Next, it sets the iteration counter to zero and
 #  initializes the location of the object to cell (0,0).
 #
def Actor(self):

##
 #  You may use a static member variable to assign
 #  a unique ID to each object. A static member variable is
 #  shared/seen by all objects in the same class.
 #
__ID = 0

##
 #  Print on screen in the format:
 #  "Iteration <ID>: Actor <Actor ID>."
 #  The <ID> is replaced by the current iteration number.
 #  <Actor ID> is replaced by the unique ID of the Actor object
 #  that performs the act() method. For instance, the
 #  actor with ID 1 shows the following result on the output
 #  screen after its act() method has been called twice.
 #    Iteration 0: Actor 1
 #    Iteration 1: Actor 1
 #
def act(self):
```

```
## Set the location of this object at the cell(x,y).
def setLocation(self, x, y):

## The Actor object remembers that it belongs to
 # the given World object.
def addedToWorld(self, world):

## Return the reference to the World object to
 # which this Actor belongs.
def getWorld(self):

## Return the x-coordinate of the object's current cell.
 #
 #  See the provided template code for cases to throw exceptions.
 #
def getX(self):

## Return the y-coordinate of the object's current cell.
 #
 #  See the provided template code for cases to throw exceptions.
 #
def getY(self):
```

Verificação:

2. Ao implementar esta classe, você entende a diferença entre uma variável estática [7] e uma variável não estática, melhor?

3.3 class IDisease(ABC)

```
##
 #  Set the growth condition of a Disease object to gRate.
 #  The value of gRate gets multiplied to the current
 #  disease strength, only when the disease is located in
 #  the world region, with the average temperature in between
 #  the values of lTemp and hTemp.
 #
@abstractmethod
def setGrowthCondition(self, lTemp, hTemp, gRate): pass

##
 #  Return the disease strength of the object
 #  that implements this interface.
 #
@abstractmethod
def getStrength(self): pass
```

[7] Variáveis de classe ou estáticas são as variáveis que pertencem à classe e não aos objetos, e que são compartilhadas entre os todos objetos da classe. Todas as variáveis às quais são atribuídos valores na declaração da classe são variáveis de classe. Já as variáveis às quais são atribuídos valores dentro dos métodos da classe (com o prefixo *self.*) são variáveis de instância.

Nota: `@abstractmethod` é um *decorator* de Python, uma função que recebe uma outra função como argumento e estende o seu comportamento sem modificá-la explicitamente.

3.4 class IWorld(ABC)

```
##
 #  Prepare the world. Open a text file named
 #  "simulation.config" in the current path
 #  (directly under the project directory).
 #  Parse the configuration file for the number
 #  of Disease objects, the cell locations of these objects,
 #  the growth rates, and the temperature ranges associated
 #  with individual growth rates.
 #  Read Section 4 on the content of the configuration file
 #  before reading the rest.
 #
@abstractmethod
def prepare(self): pass
```

Sugestão: Pode-se usar *readlines* para ler o conteúdo do arquivo para uma lista. Há alguma informação para auxiliá-lo na Tabela 1.

```
inputFile = open("./simulation.config")
lines = inputFile.readlines()
...
# read the next line into a String object
line = lines[0]
...
# split the string at "="; sArr is a list of String
sArr = line.split("=")
...
# check whether the string on the left hand side is NumDiseases
sArr[0] == "NumDiseases"
```

Tabela 1: Interpretação do arquivo de configuração.

Conhecendo a palavra-chave, chame o método correspondente para processar o lado direito de cada linha, de acordo com o valor da palavra-chave.

Se o arquivo simulation.config não for encontrado, ou se houver um erro, tal como, não haver informação suficiente em cada linha para executar a simulação, imprima a última mensagem na tela "Terminating the program." e finalize o programa. Este método deve ser o único método a chamar `sys.exit(-1)` para finalizar o programa elegantemente.

```
##
 #  Set the temperature of the region of the world
```

```
 #  to the value of temp.
 #  The quadID indicates the region.
 #  The valid value is between [0, 3].
 #  Any value of float is accepted for temp.
 #
@abstractmethod
def setTemp(self, quadID, temp): pass

##
 #  Return the temperature of the world region with
 #  the ID of quadID.
 #  The valid value is between zero and three inclusive.
 #
@abstractmethod
def getTemp(self, quadID): pass

##
 #  Create Disease objects; the number of the objects equals
 #  to the value passed in numDisStr.
 #  Return a list of object references to the created
 #  Disease objects.
 #
 #  An example of a valid numDisStr is below.
 #
 #  Ex: "2"
 #
 #  If numDisStr is None or it cannot be converted to
 #  a positive integer, print a message on screen
 #  "Check the NumDiseases line in simulation.config."
 #  and return None.
 #
 #  No exceptions are thrown.
 #
@abstractmethod
def initDiseases(self, numDisStr): pass

##
 #  Set the temperature for each quadrant of the MyWorld
 #  according to the value of the tempStr.
 #  An example of tempStr is below.
 #  The region temperatures for regions 0, 1, 2, and 3
 #  are 12, 20, 50, and 100, respectively.
 #
 #  Return 0 for a successful initialization of
 #  the quadrant temperatures. No exceptions are thrown.
 #
 #  Ex: "12;20;50;100"
 #
 #  If tempStr is empty or not in the correct format
 #  or does not have all the temperatures of all
 #  the regions, print on screen
 #  "Check the Temperature line in simulation.config."
 #  and return -1.
 #
```

```
@abstractmethod
def initTemps(self, tempStr): pass

##
 #  Add each Disease object into the MyWorld object
 #  implementing this method according to the information
 #  in locationStr.
 #
 #  An example of a locationStr is "200,200;400,480".
 #  This means that the first Disease is planted at cell
 #  (200,200) and the second Disease is at cell (400, 480).
 #
 #  If the locationStr is empty or not in the correct format
 #  or does not have all the cell coordinates of all the
 #  Disease objects, print on screen
 #  "Check the Locations line in simulation.config"
 #  and return -1.
 #
 #  Return 0 for a successful initialization
 #  of the Disease locations. No exceptions are thrown.
 #
@abstractmethod
def initLocations(self, locationsStr, diseaseArr): pass

##
 #  Set the lower bound and upper bound temperature
 #  and the growth rate for each disease according
 #  to the input growthStr.
 #  An example of a valid string for two Disease objects is:
 #
 #  Ex: "10.0,15.0,2.0;10.0,13.0,3.0"
 #
 #  If growthStr is empty or not in the correct format
 #  or does not have all the growth for all the Disease objects
 #  in the Disease array, print on screen
 #  "Check the DiseasesGrowth line in simulation.config."
 #  and return -1.
 #
 #  Return 0 for a successful
 #  initialization of the Disease growth conditions.
 #  No exceptions are thrown.
 #
@abstractmethod
def initGrowthConditions(self, growthStr, diseaseArr): pass

## Return the list of objects in
 # the class implementing this interface.
@abstractmethod
def getObjects(self): pass

## Return the total disease strength of all the diseases
 # in the class implementing this interface.
@abstractmethod
def getSumStrength(self): pass
```

Verificação:

3. Observe que os métodos `initDiseases, initGrowthConditions`, `initTemp` e `prepare` não lançam qualquer exceção, mas identificam a linha incorreta no arquivo de entrada. Nós tentamos ajudar o usuário do programa a entender melhor o que ele fez de errado. Na prática, devemos fornecer mais detalhes sobre a causa do problema para cada linha de entrada incorreta, mas para facilitar a pontuação, mantivemos a simplicidade.

3.5 class MyWorld (World,IWorld)

Essa classe tem seu construtor padrão, métodos herdados da classe World, e os métodos especificados na interface IWorld. Você pode introduzir outros métodos privados, mas não deve haver outros métodos públicos.

```
##
 #  Call the constructor of the World class with the
 #  width and height of 720 and 640 cells, respectively.
 #
 #  Initialize a list to keep the average temperature
 #  of each world region (quadrant).
 #
 #  Call the prepare() method.
 #
def __init__(self):

##
 #  This method overrides the act() method in the World class.
 #  This method prints:
 #
 #  "Iteration <ITRID>: World disease strength is <WorldDisease>"
 #  where <ITRID> is replaced by the current iteration number and
 #  <WorldDisease> is replaced by the returned value of
 #  getSumStrength() in 2 decimal places.
 #  An example is below.
 #
 #  Iteration 0: World disease strength is 2.00
 #  Iteration 1: World disease strength is 3.00
 #
def act(self):
```

3.6 class Disease (Actor,IDisease)

Esta classe possui seu construtor padrão, métodos herdados da classe Actor, e os métodos especificados em IDisease.

```
##
 #  Call its superclass's default constructor.
```

```
 #
 #  Initialize the lower bound and the upper bound
 #  temperatures for the growth rate to 0.
 #
 #  Set the growth rate to 0.
 #
 #  Set the disease strength to 1.
 #
def __init__(self):

##
 #  This method overrides the act() method in the Actor class.
 #  Check whether the object is in the region where the region
 #  temperature is within the lower bound and the upper bound
 #  temperatures for the object's growth rate. If it is
 #  the case, multiply its strength with the growth rate.
 #
def act(self):
```

Veja o padrão fornecido para esta classe.

3.7 Simulador

```
##
 #  This is the main method that
 #  sets up a virtual world and simulates the growth of
 #  the diseases in the world.
 #  If the number of iterations is given in the command line
 #  argument, run the simulation for that many number
 #  of iterations.
 #  Otherwise, use the default number of iterations: 5.
 #
def main(args=None):
```

Pseudo código dado no algoritmo 1.

Sugestão: Observe se os resultados da segunda simulação, com os objetos World e Actor, são diferentes daqueles usando os objetos MyWorld e Disease.

Nota: É possível passar argumentos na linha de comando do programa, usando o módulo *getopt* [8]. Neste trabalho, você pode passar o número de iterações para rodar a simulação.

A Fig. 2 apresenta um exemplo de um objeto MyWorld com quatro regiões (quadrantes). Há três objetos Disease (com IDs 0, 2, e 4) na região 0 (Quadrant ID=0). Há um objeto

[8] `http://orion.lcg.ufrj.br/python/ADs/AD2_2020-1.pdf`

Algorithm 1: Algoritmo para simular o crescimento de doenças num mundo fictício.

```
1  Simulator ;
   Input  : simulation.config and number of iterations.
   Output: Total strength at each step.
2  numItr = 5
3  if len(args) > 1 then
4  |  numItr = int(args[1])
5  end
6  Print on screen "Simulation of MyWorld"
7  MyWorld() - create new object
8  for i in range(numItr) do
9  |  call the act() method of the MyWorld object
10 |  call the getObjects() method to get all objects in the MyWorld object
11 |  for each object in the Object array do
12 |  |  call the act() method of that object
13 |  end
14 end
15 Print on screen "Simulation of World."
16 World(100,100) - create new object
17 Add two Actor objects into the world at the location (10,10) and (90,90)
18 for i in range(numItr) do
19 |  call the act() method of the World object
20 |  call the getObjects() method to get all objects in the World object
21 |  for each of the object in the Object array do
22 |  |  call the act() method of that object
23 |  end
24 end
```

Disease com ID 1 na região 2 e um com ID 3 na região 3.

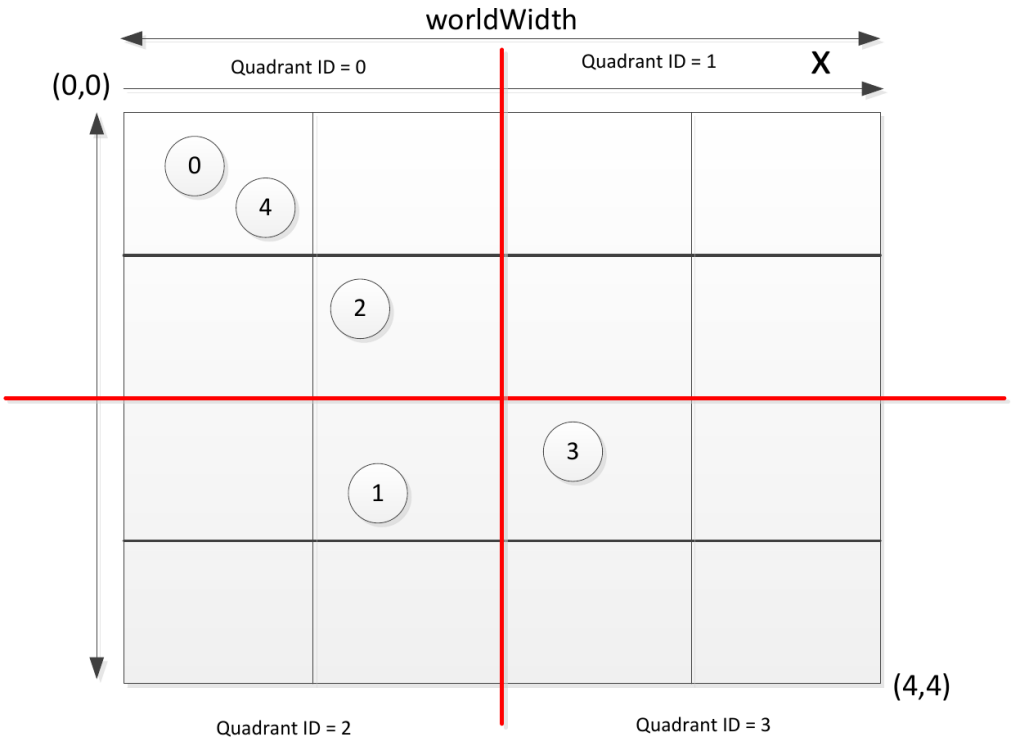

Figura 2: Um objeto MyWorld 4×4 com 16 células. Os objetos Disease, com seus ID únicos, são representados por círculos. Cada região/quadrante do mundo possui um único ID de quadrante.

4 Conteúdo do simulation.config

```
NumDiseases=2
Locations=200,200;400,480
DiseasesGrowth=10.0,15.0,2.0;10.0,13.0,3.0
Temperature=12;20;50;100
```

Tabela 2: Exemplo do simulation.config.

O simulation.config é um arquivo texto que contém informações importantes para a nossa simulação. Na prática, é comum encontrar programas que leem a entrada de arquivos, processam-as e geram uma saída. Nós aproveitamos essa oportunidade para praticar isso.

Para cada linha deste arquivo, o lado esquerdo do "=" é uma palavra-chave que não diferencia maiúsculas de minúsculas. O lado direito do "=" lista um ou mais valores, dependendo da palavra-chave no lado esquerdo.

Por exemplo, na Tabela 2, a primeira linha indica que o número de objetos Disease é 2. A segunda linha indica que a localização do primeiro objeto Disease está na célula (200,200) e o segundo objeto está na célula (400,480). O Simulador instancia um objeto MyWorld e adiciona dois objetos Disease nos locais especificados.

A terceira linha indica o limite inferior, o limite superior e a taxa de crescimento para cada objeto Disease, separados por ponto e vírgula. Por exemplo, na tabela 2, o primeiro objeto Disease cresce a uma taxa de 2.0, somente quando está numa região com temperatura entre 10.0 e 15.0. O segundo objeto Disease cresce à taxa de 3.0, quando está localizado numa região com o temperatura entre 10.0 e 13.0. Qualquer valor do tipo float é aceito para a temperatura. Não há restrição quanto ao número de casas decimais para os valores de temperatura.

A quarta linha indica as temperaturas das regiões do mundo, começando na região 0, até a região 3. Neste exemplo, a região 0 tem a temperatura de 12; a região 1 tem a temperatura de 20; região 2 tem a temperatura de 50. A última região tem a temperatura de 100.

Dada a configuração acima, e o tamanho fixo do mundo de 720 em largura e 640 em altura, o resultado da simulação na tela é mostrado na Tabela 3.

Existem dois objetos Disease adicionados a um objeto MyWorld. As coordenadas das células dos dois objetos Disease indicam que eles estão nas regiões 0 e 3, respectivamente. As temperaturas da região 0 e 3 são 12 e 100, respectivamente. Somente o primeiro objeto Disease na região 0 cresce na taxa de 2.0 por iteração, depois de imprimir sua força. Isso ocorre porque a temperatura 12 da região está entre 10 e 15. A força inicial para cada objeto Disease é 1.

Portanto, na Iteração 0, temos 2 como a soma das forças das doenças, que vale 1, para cada objeto. A Disease 0 na região 0 multiplica sua força pela taxa de crescimento. Na Iteração 1, a força das doenças no mundo é 2 + 1. Na Iteração 2, a força das doenças no mundo é 4 + 1 = 5. Na Iteração 3, a força das doenças é 8 + 1 = 9. Na última iteração, a força das doenças é 16 + 1 = 17.

A simulação do objeto MyWorld acabou. A simulação do Objeto World inicia. Adicionamos dois objetos Actor ao objeto World. Porque a referência do objeto, em cada elemento, na lista de objetos retornados, refere-se ao objeto Actor, ele usa o método `act()` do objeto Actor. O resultado é o ID do objeto Actor em cada iteração.

Nós fornecemos código padrão [9] para World.py, Actor.py, Disease.py, IDisease.py e IWorld.py, para ajudá-lo a começar. Complete as partes omitidas nesses arquivos e inclua comentários adicionais, se necessário. Pedimos que MyWorld.py e Simulator.py sejam escritos do zero. Certifique-se de fornecer comentários Doxygen (@author, @param, @throws, e @return) nessas classes.

Documentação

Todos os métodos públicos devem ter comentários Doxygen seguindo o formato Doxygen padrão, incluindo descrições de cada parâmetro (@param) e exceções (@throws). Quaisquer métodos privados devem ter comentários ao estilo Doxygen ou normal, explicando o que os métodos fazem. Seu nome deve aparecer após o @autor em cada classe.

[9] `http://orion.lcg.ufrj.br/python/ADs/arquivos-AD1-2020-2.tar.gz`

```
Simulation of MyWorld
Iteration 0: World disease strength is 2.00
Iteration 1: World disease strength is 3.00
Iteration 2: World disease strength is 5.00
Iteration 3: World disease strength is 9.00
Iteration 4: World disease strength is 17.00

Simulation of World
Iteration 0: Actor 2
Iteration 0: Actor 3
Iteration 1: Actor 2
Iteration 1: Actor 3
Iteration 2: Actor 2
Iteration 2: Actor 3
Iteration 3: Actor 2
Iteration 3: Actor 3
Iteration 4: Actor 2
Iteration 4: Actor 3
```

Tabela 3: A saída da simulação, dado o conteúdo do arquivo de configuração mostrado na Tabela 2, e o número padrão de iterações (cinco).

Submissão

Escreva suas classes, bem como as classes de teste com o *unittest* no diretório AD1. Coloque todos os arquivos Python (.py) no diretório AD1 no formato zip. Nomeie seu arquivo zip FirstName_LastName.zip em que FirstName é seu primeiro nome e LastName é o seu sobrenome. Lembre-se de salvar a estrutura de diretórios (por exemplo, entregue o arquivo zip. Não entregue seus arquivos .pyc).

As classes de teste com o *unittest* a serem enviadas são WorldTest, ActorTest, e DiseaseTest. Você é encorajado a escrever casos de teste com o *unittest* para outras classes, mas elas não são necessárias para o envio.

Certifique-se de acessar a plataforma, e verificar quanto a política de envio tardio, para evitar perder os prazos.

Comece o trabalho cedo!