Disciplina: Desenvolvimento Web I
AD1 1^o semestre de 2021.
Professor: Paulo Roma

AD1: Bejeweled

1 Informações Gerais

Essa tarefa deve ser feita apenas por você. Se precisar de ajuda, consulte seu instrutor ou um dos tutores.

No entanto, como nenhum código de teste precisa ser entregue como parte desta tarefa, você pode compartilhar qualquer código de teste que escrever e/ou publicá-lo na plataforma.

Inicie o trabalho tão logo quanto possível e tire suas dúvidas cedo. Ler uma especificação técnica, como este documento, não é como ler uma história, do começo ao final feliz. Você provavelmente terá que ler tudo várias vezes antes de começar a clicar. É possível começar a escrever e testar o código antes de entender completamente todos os detalhes, no entanto, a seção 4 oferece alguns conselhos úteis.

1.1 Se Tiver Perguntas

Para dúvidas, consulte as páginas da plataforma e clique na pasta AD1. Se sua pergunta não for respondida, crie uma nova postagem com a sua questão. Tente indicar claramente a pergunta ou o tópico no título da sua postagem, e anexe a etiqueta AD1. Mas lembre-se, não poste nenhum código fonte para as classes a serem entregues. Não há problema em postar o código-fonte para exemplos de Python que não estão sendo entregues.

Se você tiver uma pergunta que absolutamente não possa ser feita sem mostrar parte do seu código-fonte, torne a postagem "privada"para que somente os instrutores e os tutores possam vê-la. Certifique-se de ter formulado uma pergunta específica; solicitações vagas do tipo "Leia todo o meu código e me diga o que há de errado com ele" geralmente serão ignoradas.

Obviamente, os instrutores e os tutores estão sempre disponíveis para ajudá-lo. Consulte a tabela de atendimento para encontrar um horário conveniente para você. Nós faremos o nosso melhor para responder a cada pergunta com cuidado, a menos de realmente escrever o código para você, mas seria injusto para a equipe analisar completamente sua tarefa em detalhe antes dela ser entregue.

Quaisquer postagens dos instrutores na plataforma rotuladas como "Esclarecimento Oficial" são consideradas parte da especificação, e você pode perder pontos se ignorá-las. Essas postagens sempre serão colocadas na seção Anúncios do curso (prometemos que nenhum esclarecimento oficial será postado antes de 24 horas da data de entrega).

2 Introdução

Para esta tarefa, você implementará a lógica de um videogame simples, baseado no jogo "Bejeweled".

Quase todo o seu código estará na classe GameImpl, descrita abaixo. Há uma classe associada, chamada BasicGenerator, que também deve ser implementada. Os objetivos deste trabalho são:

- Praticar o uso de matrizes, e Listas.
- Praticar a usar o conceito de interfaces[1] em Python.
- Ter a chance de trabalhar com uma aplicação que interage com uma GUI (*Graphical User Interface*).

2.1 Resumo

Bejeweled® é um jogo criado pela PopCap Games[2] em 2001. Várias versões estão disponíveis online e em dispositivos como iPhone, iPad e Android. O jogo consiste em uma grade (geralmente 8x8) de "jóias" de diferentes formas e cores. O objetivo é trocar as posições de duas jóias para que haja três do mesmo tipo, em uma linha ou coluna. Quando isso ocorre, as três jóias são excluídas, as jóias acima caem para preencher os lugares vazios e novas jóias são adicionadas ao topo. Na Wikipedia[3] há uma visão geral do jogo original (existem vários sites onde é possível jogar online).

Nossa versão será um pouco simplificada, como pode ser visto na Figura 1. O jogo opera sobre uma uma grade de ícones coloridos. Uma jogada consiste em trocar dois ícones de posição para formar uma "sequência" definida como três ou mais ícones idênticos consecutivos. Você pode clicar e arrastar um ícone para um de seus vizinhos para fazer uma jogada. Se a mudança for bem sucedida (ou seja, pelo menos uma sequência for formada), os ícones na sequência (três ou mais) desaparecem. Em seguida, cada coluna colapsa, ou seja, quaisquer ícones que estão acima das células vazias caem para tomar seus lugares, e novos ícones aleatórios caem do topo para preencher as células vazias da grade. Um certo número de pontos é adicionado à pontuação para cada sequência formada.

As interfaces e classes principais com as quais você precisa se familiarizar estão resumidas brevemente abaixo. Tudo está totalmente implementado no código fornecido, exceto GameImpl e BasicGenerator. Veja a documentação gerada pelo Doxygen[4] para obter detalhes sobre os métodos e construtores.

```
class Icon(ABC):
    """An Icon represents an icon that occupies one cell of the grid.
```

[1] Infelizmente, Python não possui interface ou, pelo menos, não totalmente embutida na linguagem. Emprega-se, 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.

[2] http://www.popcap.com

[3] http://en.wikipedia.org/wiki/Bejeweled

[4] http://orion.lcg.ufrj.br/python/ADs/AD1_2021-1_refman.pdf

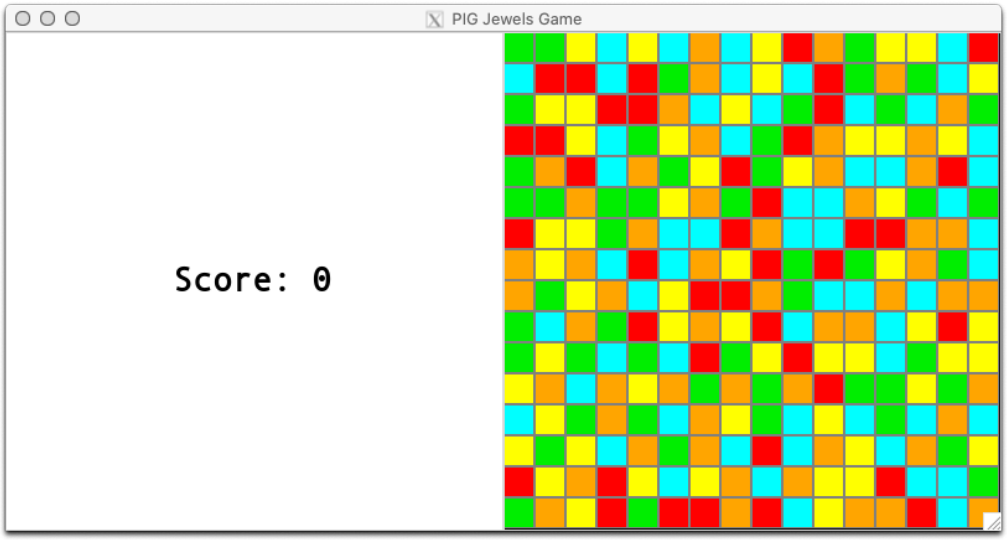

Figura 1: Diagramação do Bejeweled.

```
    Each icon is identified by its "type", which is just an integer."""

class Cell(object):
    """A Cell encapsulates a grid position and its icon, and optionally may
    include information about the icon's previous location in the grid."""

class IGenerator(ABC):
    """A generator is used to create new icons to populate the grid.
    An instance of IGenerator must be supplied when constructing
    a GameImpl."""

class BasicIcon(Icon):
    """A simple implementation of the Icon interface."""

class BasicGenerator(IGenerator):
    """An implementation of the IGenerator interface whose generate()
    method generates random instances of BasicIcon.
    This class is to be implemented by you."""

class IGame(ABC):
    """This interface specifies the methods used by clients to interact
    with the game. Methods are provided for selecting moves, determining
    which cells are part of a run and triggering collapse of the cells
    in runs, and getting the score."""

class GameImpl(IGame):
    """Concrete implementation of the IGame interface.
    This is where most of your work will be done.
    This class is to be implemented by you, starting from the skeleton
    in the sample code."""
```

Além disso, existem três classes que compõem a interface do usuário: GamePanel, Score-

Panel e GameMain. Elas estão localizadas no arquivo GamePanelGL.py. Existe um arquivo GamePanel.py implementado apenas com tkinter, mas com uma execução muito lenta (apenas um quadro por segundo (*fps*), na maioria das vezes). A versão eficiente foi implementada por meio do PyOpenGL[5].

Esse código é um pouco complexo, baseado no conceito de um *Timer*, que esquedula tarefas para serem executadas após um certo intervalo de tempo, e está fora do escopo deste curso. Portanto, você está liberado de lê-lo e entendê-lo, embora possa ser interessante!

3 O Código Fornecido

Repare que o código fornecido não funcionará até que seja criada a classe BasicGenerator (ou pelo menos o seu esqueleto). Não é absolutamente necessário o uso de uma IDE, como o Eclipse[6], Eric[7], ou PyCharm[8] no desenvolvimento deste projeto. Basta um editor de texto qualquer. No entanto, aqui está a receita de bolo para importar o código no Eclipse:

1. Baixe o arquivo zip para um local conveniente.
2. No Eclipse, vá para File → Import → General → Existing Projects into Workspace
3. Click o radio button para "Select archive file"
4. Use o botão Browse e importe o arquivo zip
5. Click Finish

Se necessário, pode-se descomprimir o arquivo (num local fora do seu *workspace*), criar os pacotes, e arrastar o código fonte para os pacotes.

3.1 Implementando a Classe BasicGenerator

Você deve escrever a classe BasicGenerator, que implementa a interface IGenerator, e possui dois métodos. O método `generate()` é usado pelo jogo sempre que novos ícones são necessários para preencher a grade, e o método `initialize()` é usado para inicializar a grade quando a instância do GameImpl é construída pela primeira vez (no esqueleto do código para o GameImpl é possível ver esse método sendo usado).

Uma instância do classe BasicGenerator deve ser fornecida ao construir um GameImpl. Olhando-se o começo do método `create()` da classe GameMain, pode-se ver um exemplo de como esse construtor é chamado com uma instância do BasicGenerator.

Para obter informações gerais sobre os métodos necessários, consulte a documentação da Interface IGenerator gerada pelo Doxygen. Sua implementação do construtor do BasicGenerator também deve incluir duas inicializações diferentes e deve satisfazer alguns requisitos

[5] `http://pyopengl.sourceforge.net`

[6] `https://cloud.google.com/appengine/docs/standard/python/tools/setting-up-eclipse#sdk-paths-dialog`

[7] `https://eric-ide.python-projects.org`

[8] `https://www.jetbrains.com/pycharm/`

comportamentais específicos, descritos na documentação gerada pelo Doxygen para o BasicGenerator.

Observe que há um requisito específico de que o método `initialize()` inicialize a matriz 2D, de modo que nenhuma célula adjacente contenha ícones do mesmo tipo (nós não desejamos que a grade inicial possua várias sequências já formadas). Você pode garantir isso da maneira que quiser. Por exemplo, para começar, é possível atribuir ícones em um padrão quadriculado xadrez de dois tipos. Não é necessário usar randomização neste método. Idealmente, no entanto, você pode descobrir uma maneira de escolher aleatoriamente um tipo para o ícone de cada célula, excluindo os tipos dos ícones das células adjacentes.

O método `generate()`, por outro lado, é necessário para retornar tipos de ícones de forma uniformemente aleatória.

3.2 Implementando a Classe GameImpl

Para implementar esta classe, trabalhe no esqueleto[9] que está presente no arquivo GameImpl.py. Consulte a documentação online Doxygen do GameImpl, para obter detalhes sobre os métodos (se você estiver lendo o código fonte, observe que as diretrizes Doxygen para a maioria dos métodos estão realmente localizadas no arquivo IGame.py, que especifica a interface).

Do ponto de vista de um cliente (como a GUI), a ordem em que os métodos principais são chamados é a seguinte:

```
def select(self,cells):
    """Selects two icons to be swapped, and returns True if the move
    can be made."""

def findRuns(self,doMarkAndUpdateScore):
    """Identifies all runs in the current grid and returns a list of Cells
    representing their positions."""

def collapseColumn(self):
    """Collapses one column (icons move toward the bottom to fill
    null cells) and returns a list of Cells representing the moved
    icons in their new positions."""

def fillColumn(self):
    """Fills any null cells at the top of a column with newly generated
    icons, and returns a list of Cells representing the new icons in
    their new positions."""
```

A classe GameImpl implementa a interface IGame, que é como os clientes (como a GUI) interagem com o seu jogo. As classes no arquivo GamePanelGL.py não contêm referências para a classe GameImpl, apenas para a interface IGame. Dessa forma, a mesma interface com o usuário pode ser usada em implementações de jogos semelhantes, que podem ter regras diferentes para movimentos, sobre o que constitui uma sequência ou quanto a pontuação.

[9] `http://orion.lcg.ufrj.br/python/ADs/arquivos-AD1-2021-1.tar.gz`

3.3 Sugestões para Implementar `select()`

Observe que o método `findRuns()` possui um parâmetro. Se este parâmetro for False, significa que o método deve encontrar todas as sequências, mas não deve realmente mudar a grade ou a pontuação. Se `True`, todas as sequências serão substituídas por `Nones` na grade e a pontuação é atualizada. Para determinar se um movimento é permitido no método `select()`, deve-se verificar se uma sequência é criada pelo movimento. Isso é fácil: basta trocar os ícones, chamar `findRuns(False)` e verificar se uma das células especificadas está na mesma posição que uma das células retornadas (caso contrário, lembre-se de trocar os ícones do jeito que estavam!).

Nota sobre testes e o método `setIcon()`

Em uso normal, a classe GameImpl deve atualizar sua própria grade e um cliente acessaria a grade usando apenas o método `getIcon()`. Para desenvolvimento e teste a interface também especifica um método `setIcon()` para definir configurações específicas de valores de ícone nas células da grade. Isso possibilita o desenvolvimento de casos de teste que inicializam a grade com valores conhecidos e chamar métodos como `findRuns()` ou `collapseColumns()` individualmente. Esse é o mecanismo principal usado para testar seu código. Em particular, seus métodos devem sempre funcionar conforme especificado com qualquer que seja a grade atual, quer sejam invocados ou não, na ordem especificada acima (`select, findRuns, collapseColumn, fillColumn`).

Há dois métodos `__str__` e `toString` que foram implementados para você no esqueleto da classe GameImpl que serão úteis para testes, por exemplo, escrevendo:

```
print(myGame)
```

para ver o conteúdo da grade no jogo myGame na forma textual. De maneira similar, para examinar o conteúdo de uma lista de células, escreva:

```
print(myGame.toString(lcells))
```

As células são exibidas usando o formato `[(row, col) icon]` caso a linha e a linha anterior sejam idênticas, ou `[(row, col) icon (previous row)]` se a linha e a linha anterior forem diferentes.

3.4 A Interface Gráfica - GUI

A classe GameImpl, como será implementada, pode ser desenvolvida e testada fora de qualquer interface do usuário. A interface do usuário é fornecida para que você possa realmente jogar, depois de fazer o jogo funcionar. Lembre-se, no entanto, que ainda é necessário prestar atenção à especificação, quanto ao que as classes GameImpl e BasicGenerator devem fazer, pois elas serão testadas e pontuadas de acordo com a especificação, e não jogando o jogo!

Para iniciar a GUI, basta executar a classe principal GameMain. É possível também executá-la no modo de depuração, para testar e definir pontos de interrupção (*break points*)

no seu código. No entanto, o código da GUI é complexo e geralmente esta é uma maneira de testar difícil e frustrante. Nós recomendamos testar os métodos da classe GameImpl individualmente, usando `setIcon, getIcon`, e/ou `toString` para iniciar e verificar a grade.

4 Sugestões para Começar

1. Crie um esqueleto para BasicGenerator e faça o projeto inteiro "compilar".

2. Implemente o método `initialize()` do BasicGenerator. Como ponto de partida, pode-se preencher a grade com ícones alternados de tipo 0 e 1. Crie uma classe de teste e adicione o seguinte código ao método principal:

```
def main(args):
    gen = BasicGenerator(5)
    testGame = GameImpl(4, 4, gen)
    print(testGame)
```

   Deve aparecer algo como:

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

3. Implemente `getIcon, setIcon, getWidth e getHeight` (uma linha cada). Neste ponto tente rodar a UI. Deve aparecer a grade e ser possível clicar sobre ela, para selecionar as células, mas nada mais acontecerá. Tente adicionar um `print` ao método `select()` para certificar-se de que ele está sendo chamado.

4. Depois, tente `fillColumn`. Casos de teste podem ser criados usando `setIcon` para atribuir alguns `Nones` ao topo de uma coluna.[10] A lista retornada pode ser impressa usando-se `print`.

5. Implemente `collapseCells`; teste da mesma maneira.

6. Implemente `findRuns`. Comece assumindo que o parâmetro é `False` e não se preocupe com a pontuação neste momento.

7. Após `findRuns` estar funcionando, adicione um teste no final, para atribuir todas as posições apropriadas com `None`, se o parâmetro for `True`.

8. Uma vez que `findRuns` esteja funcionado, implemente `select`. Neste ponto, você deve ser capaz de jogar com a interface gráfica!

[10] O método `toString` exibe `Nones` (nulls) usando o caractere "*".

5 Estilo e Documentação

Mais ou menos 15 a 20% dos pontos serão para documentação e estilo.

1) Você escreveu alguns loops com lógica complexa na classe GameImpl. Esperamos ver alguma explicação do que foi feito, na forma de comentários (estilo #). Lembre-se de que os comentários internos sempre precedem o código que descrevem, e devem ser identados no mesmo nível.[11]

2) Você não precisa fornecer diretrizes adicionais para o Doxygen para os métodos públicos de GameImpl, pois todos estão documentados na interface. A convenção para as classes que implementam uma interface é que as diretivas do Doxygen são necessárias apenas para métodos cujo comportamento seja diferente do indicado na documentação da interface; caso contrário, a ferramenta Doxygen apenas os copia da interface. Veja a documentação gerada pelo Doxygen para o método `select` no GameMain para ter um exemplo.

3) Caso contrário, as expectativas quanto a documentação são as mesmas das tarefas anteriores; veja abaixo.

   – Cada classe, método, e variável de instância, quer pública ou privada, deve ter um comentário ao estilo Doxygen. Classes devem incluir a etiqueta @author, e métodos devem incluir as etiquetas @param e @return quando apropriado.

     * Para BasicGenerator, você pode copiar e colar a partir da descrição dos métodos online se quiser.
     * Não esqueça de adicionar uma etiqueta @author à classe GameImpl.

     Todos os nomes de variáveis devem ser significativos (ou seja, nomeados de acordo com o valor que armazenam). Use variáveis de instância apenas para o estado "permanente" do objeto. Use variáveis locais para cálculos temporários dentro dos métodos. Todas as variáveis de instância devem ser privadas. Use um estilo consistente para identação e formatação.

   – Perceba que é possível fazer com que o Eclipse use o estilo de formatação da sua preferência e então usar Ctrl-Shift-F para formatar o código. Para brincar com as preferências de formatação, vá para Eclipse → Preferences → PyDev → Code Style → Code Formatter e crie o seu "profile" de formatação.

5.1 O que Entregar

Envie na plataforma um arquivo zip que inclua, pelo menos, seus arquivos GameImpl.py e BasicGenerarator.py. Os arquivos devem estar localizados dentro de um diretório chamado AD1. Normalmente, a coisa mais simples a fazer é clicar com o botão direito do mouse sobre o diretório do seu projeto e selecionar "Enviar para → compactado/arquivo zip". É perfeitamente aceitável incluir os arquivos python restantes relacionados à tarefa.

[11] Uma boa regra geral é: se tiver de pensar por mais de um minuto para descobrir como fazer algo, provavelmente deveria ser incluído um comentário, explicando o raciocínio.

Após criar o arquivo zip, verifique-o com cuidado. Extraia os arquivos em um diretório temporário vazio, e olhe para eles antes de enviar. Eles são arquivos .py? Todos os arquivos necessários estão presentes no arquivo comprimido? Eles estão no diretório correto? São as versões mais recentes e que funcionam do seu código?

Lembre-se de VERIFICAR sua submissão após enviá-la na plataforma, procurando no seu histórico. Você perderá pontos se a sua submissão estiver incorreta. Se você não tiver certeza de como fazer essas coisas, consulte o documento "Submissão de Tarefa", que pode ser encontrado na página de Tarefas da turma ou em um link na plataforma.