Terceiro Exercício-Programa - Parte B

BCC 2018 - MAC0110 - Entrega: até 25/06/2018 23:55 pelo PACA

Controlando o Mundo de Wumpus

Introdução

Neste derradeiro exercício-programa abandonaremos a perspectiva limitada das criaturas do Mundo de Wumpus e adotaremos uma postura demiúrgica em relação a ele. Talvez mais: seremos meta-demiurgos que definirão o próprio código que rege todo o universo. Ou talvez menos… vamos ver! O objetivo da parte B do EP3 será criar um sistema de gerenciamento de Mundos de Wumpus genéricos, capaz de lidar com múltiplas personagens de forma sincronizada, e que ofereça modos de visualização e acompanhamento das ações dessas personagens a partir da perspectiva de um observador externo (o usuário do sistema).

São 3 os ingredientes essenciais para atingir esse objetivo, que serão detalhados nas seções seguintes: o espelhamento das personagens, sua coreografia e a interface gráfica. As personagens-espelho são duplos das personagens implementadas na Parte A, porém são gerenciadas pelo mundo e refletem a realidade objetiva das personagens (em oposição às suas realidades subjetivas, ou ao que elas pensam ser a realidade). A coreografia se refere à articulação entre as percepções e ações de todas as personagens de forma sincronizada, garantindo o nexo causal das ações das personagens e a coerência na simulação do mundo. A interface gráfica permitirá a inspeção do mundo em diversos níveis de detalhamento e o controle da passagem do tempo; ela será feita em pygame, uma biblioteca Python que facilita o gerenciamento de elementos gráficos, sonoros e também de interação com teclado e mouse. Por ser essa a parte menos relacionada com o conteúdo das aulas, é fortemente recomendada a leitura antecipada de alguns textos adicionais (tutoriais) sugeridos na página 4.

Nossa implementação seguirá o paradigma de orientação a objetos, e estará centrada em torno de uma classe chamada MundoDeWumpus, que conterá as definições centrais (conteúdo do mundo), criará as personagens e coordenará toda a simulação (fornecendo percepções às personagens e coletando as intenções de ação), incluindo a interface gráfica. Toda a realidade será criada com uma única instanciação dessa classe, através de uma chamada da forma

```
m = MundoDeWumpus()
```

que criará um novo mundo, definindo um tamanho N para o toro, o conteúdo de cada sala, as posições e orientações de cada personagem, todos os aspectos visuais e sonoros dos elementos e ações do mundo, e todas as ações decorrentes das estratégias individuais. Que milagre uma única linha de código pode produzir!

Parte do código da Parte B já está implementada no módulo mundo.py que usamos na Parte A. Após a leitura cuidadosa deste enunciado, será conveniente estudar aquele código e procurar as correspondências do enunciado no código. Algumas coisas já estão prontas ou precisarão de mínimas alterações; outras demandarão um esforço um pouco maior. A partir desse ponto não existirá mais a personagem Dummy e o usuário já não determinará nenhum aspecto da simulação, mas poderá dar uma espiadinha no que acontece no canal do BBW, sem precisar de PPV.

As personagens-espelho

Cada personagem do Mundo de Wumpus foi implementada na Parte A como um módulo chamado personagemNUSP.py, aos quais teremos acesso para usar em nossa Parte B. Esses módulos essencialmente recebem informações do mundo (através das variáveis nFlechas e mundoCompartilhado) e definem as três funções principais realizadas por cada personagem: inicializa(), planejar() e agir(). No módulo mundo.py cada personagem será representada por um objeto de uma classe Personagem, que conterá um atributo modulo (de acesso àquelas informações), além de métodos para tratar cada uma das ações da personagem. O processamento das ações modificará atributos "espelho" posicao, orientacao e nFlechas, usados para representar a situação real da personagem no mundo, usando um sistema de coordenadas absoluto, fixo e desconhecido das próprias personagens. A necessidade dos atributos-espelho está associada ao fato de que os atributos do módulo são acessíveis para o código da personagem, e portanto poderiam ser modificados por ela de forma incondizente com a realidade, como no bug da atualização de posição contido na primeira versão da personagemNUSP; em outro exemplo, uma personagem poderia tentar se teletransportar para outra posição do mundo ou aumentar o seu número de flechas, o que não poderia ter nenhum efeito real (de um ponto de vista objetivo essas alterações seriam como devaneios da personagem). Assim os atributos-espelho representariam a realidade "objetiva" do Mundo de Wumpus, externa às representações (subjetivas) das personagens.

Os objetos da classe Personagem serão utilizados por um objeto da classe principal MundoDeWumpus, que possui o conhecimento completo do mundo e portanto pode processar todas as percepções e ações das personagens. Como exemplos, uma tentativa de movimentação da personagem p será processada pelo método p.andar(); se essa tentativa de andar for em direção a um muro, a ação será ignorada do ponto de vista da atualização da posição, mas deve gerar uma percepção de impacto para aquela personagem na próxima rodada, recebida através do método p.planejar(); uma tentativa de atirar só se concretizará se a variável espelho nFlechas (e não a variável do módulo) for positiva, e o método p.atirar() deve repassar a informação do valor atualizado da variável nFlechas para a variável interna ao módulo (que representa uma informação útil para a estratégia da personagem). Alguns códigos da classe Personagem serão idênticos a códigos que já foram escritos nos módulos personagemNUSP.py, e isso é normal: as atualizações de posição e orientação por exemplo seguem essencialmente as mesmas linhas, ainda que os sistemas de referência utilizados pelas personagens dos módulos e pelas personagens-espelho para representar posições e orientações sejam diferentes. A morte de uma personagem também deve ser tratada pela personagem-espelho, através do atributo p.estaviva e do método p.morrer().

Essa é uma parte da implementação que dependerá de poucas mudanças em relação ao que havia no mundo.py original. Você encontrará ali os atributos espelho e os métodos para processar as ações, além de receitas para identificar arquivos com nome personagem*.py, e como importar o módulo correspondente para um atributo do objeto. Como a Parte B não tem a personagem Dummy, não há necessidade de utilizar a subclasse PersonagemNUSP, que é herdada da classe Personagem; de fato você pode trazer o código dessa subclasse para a classe principal Personagem e eliminar as subclasses PersonagemNUSP e Dummy. Uma coisa que você verá no código de mundo.py é um vetor de funções, que permite a chamada dos métodos-espelho (ações) de forma indexada. Em Python qualquer coisa pode ser armazenada em variáveis e listas, incluindo funções, módulos e classes. Fiquem à vontade para usar esses recursos, mas lembrem-se sempre das palavras do tio Ben: *"with great power comes great responsibility!"*[1]

Todos os arquivos no diretório corrente contendo personagens do Mundo de Wumpus devem ser utilizados na simulação, e para isso você usará uma lista de objetos do tipo Personagem. Esta lista será usada para fazer a varredura das personagens, nos momentos de produzir as percepções e de chamar as funções planejar() e agir() das personagens na lista. A essa lista se refere o título da próxima seção.

---

[1] pensando bem, não lembro quem disse isso primeiro, mas não foi o tio Ben…

O império das almas

Para coordenar o fluxo de percepções e ações das personagens em uma coreografia coerente, o mundo manterá uma lista de objetos-personagens que lhe permitirá o acesso às funções e variáveis implementadas nos respectivos módulos e aos atributos-espelho e métodos-espelho. A inicialização desse império de almas se dará pela varredura do diretório em busca de arquivos com nome personagem*.py, e atribuirá aleatoriamente uma posição e orientação iniciais para cada personagem em uma sala livre, ou seja, evitando muros, poços e Wumpus.

A coreografia das personagens seguirá um ritmo determinado: a cada intervalo de tempo Δt o mundo percorrerá a lista obtendo as posições das personagens, produzindo as percepções correspondentes, invocando os métodos de planejamento de cada personagem, e em seguida executando os métodos de ação de cada personagem. O intervalo de tempo Δt controla a taxa de atualização da interface e reflete o tempo de espera entre varreduras sucessivas da lista de personagens; esse parâmetro deve ser inicializado com 1 segundo, e será controlado através da interface para produzir simulações mais rápidas (tecla “.”) ou mais lentas (tecla “,”), multiplicando/dividindo Δt pelo fator 1.5.

A sequência precisa de ações em cada rodada é muito importante: primeiramente todas as personagens devem receber suas percepções instantâneas, e depois todas as personagens devem agir simultaneamente. É claro que o código seguirá a ordem da lista de personagens e computará o que tiver que computar de forma sequencial, mas semanticamente esse processamento se refere a um instante único numa linha de tempo discreta (que avança em intervalos de Δt segundos). Isso evitará problemas de sincronização que inevitavelmente aconteceriam se houvesse um intercalamento entre percepções e ações de personagens diferentes. Por exemplo, se uma personagem A planejasse compartilhar com outra personagem B que foi percebida na mesma sala, e a personagem B saísse da sala antes de A realizar sua ação planejada, o compartilhamento seria impossível. Se um Wumpus vizinho morresse entre a percepção e a ação de uma personagem, ela ainda estaria sentindo um fedor que já não existia no momento da ação (impedindo-a por exemplo de explorar melhor seus arredores). Essas e outras situações complicadas do ponto de vista temporal desaparecem sob a hipótese de percepções e ações síncronas e instantâneas.

O controle da lista de personagens será feito através de uma classe ListaDePersonagens. Além de realizar as varreduras acima, essa classe também é responsável por garantir que ações inviáveis não sejam realizadas. Ao contrário da personagem controlada pelo usuário, nossas personagens produzem ações de forma autônoma, e poderiam acidentalmente decidir atirar uma flecha que não existe ou compartilhar quando não há mais ninguém na sala; tais ações serão ignoradas, e a personagem ficará sem ação naquela rodada.

Outra responsabilidade da classe ListaDePersonagens é realizar os compartilhamentos entre personagens. Lembrando que cada personagem representa o seu mundo conhecido através de um sistema de coordenadas individual, será necessário efetuar mudanças entre esses sistemas de coordenadas. O arquivo mundo.py traz um exemplo “engessado” dessa transformação; aquele código está amarrado à representação da personagemNUSP começando numa posição absoluta pré-definida (2,2) e olhando para a direita (0,1), sendo que ela representa sua própria situação como posicao=(0,0) e orientacao=(1,0), então certifiquem-se de entender a ideia para poder generalizá-la. Cada ação de solicitação de compartilhamento feita por uma personagem A deve identificar todas as demais personagens B≠A que estão na mesma sala, converter os sistemas de coordenadas destas personagens para o sistema de coordenadas usado por A (o que pode ser feito diretamente, ou em dois passos usando o sistema fixo do mundo como representação intermediária) e finalmente fazer a união (sem repetições) das listas que representam as anotações que cada personagem fez em cada sala.

Rumo à omnisciência

Como dito na introdução, todo o código parte da instanciação de um objeto da classe MundoDeWumpus. O construtor desse objeto deve realizar todas as etapas do processo de simulação do mundo, delegando subtarefas para objetos das classes pertinentes, como Personagem, ListaDePersonagens ou Interface (a ser descrita a seguir). Esse arquiteto universal deve realizar os seguintes passos: identificar as personagens, criar um mundo aleatório para elas, posicioná-las no mundo e iniciar o processo de simulação, alternando sequências de geração de percepções, planejamentos e ações, até que não haja mais Wumpus no mundo, ou até que não haja mais personagens, ou até que o usuário interrompa a simulação (com a tecla ESC ou fechando a janela).

O mundo criado deve ser de tamanho suficiente para abrigar personagens, Wumpus, muros, poços e ter suficientes casas livres para permitir a movimentação das personagens. As distribuições iniciais para as quantidades de salas com muros e poços será respectivamente de α=0.2 e β=0.2 (vezes o número de salas do mundo, que é N^2), e a relação entre as quantidades de Wumpus e personagens será de γ=0.9 (ou seja, o número de Wumpus será um pouco menor do que o número de flechas); a fração de salas livres será δ=0.5, de onde se pode deduzir que o tamanho do mundo será calculado como N=(γ*P/(1-α-β-δ))**0.5, onde P é o número de personagens. Use math.ceil() (arredondamento para cima) no cálculo de N e math.floor() (arredondamento para baixo) no cálculo do número de muros, poços e Wumpus, e gere um mundo que tenha exatamente o número especificado de salas de cada tipo (ou seja, crie um mundo com N**2 salas livres e depois sorteie floor(α*N**2) salas distintas e livres para abrigar muros, repetindo a operação para poços e Wumpus).

A interface gráfica deve ser produzida por um objeto da classe Interface, que será responsável por exibir em uma janela de MxM pixels (um atributo tela do objeto) uma perspectiva do mundo sob controle do usuário (usem M=800 como default). A janela conterá uma visualização de SxS salas (default S=5), inicialmente com a sala de posição absoluta (I0,J0)=(0,0) no canto superior esquerdo, e sempre usando o fato de que o mundo tem a geometria de um toro (o que permite inclusive visualizações com S>N). Todos os parâmetros mencionados poderão ser redefinidos pelo usuário através de parâmetros posicionais na chamada “python3 mundo.py M S α β γ δ”. Para isso, use sys.argv[] e trate os parâmetros como opcionais, ou seja, se a lista tiver menos que 6 argumentos então cada valor ausente deve ser configurado pelo valor default.

A interface responderá também a alguns comandos do teclado. Os comandos ‘-’ e ‘=’ servirão respectivamente para diminuir e aumentar o zoom (incrementar/decrementar S), e as setas direcionais servirão para trasladar a perspectiva redefinindo a sala que ocupa o canto superior esquerdo (use o padrão de rolagem chamado “natural”, em que a seta para cima faz o mundo “deslizar” para baixo e a seta da direita faz o mundo “deslizar” para a esquerda). Como mencionado anteriormente, os comandos ‘,’ e ‘.’ servirão respectivamente para desacelerar e acelerar a simulação (dividindo/multiplicando Δt por 1.5), e ‘ ’ (espaço) servirá para pausar a simulação.

Tanto o posicionamento de elementos gráficos (imagens usadas para representar salas livres, poços, muros, Wumpus, personagens, orientações e outros elementos que você quiser usar) quanto a interpretação dos comandos do teclado serão feitos usando a biblioteca pygame[2], que possui funções prontas para abrir janelas, posicionar/atualizar imagens na janela, receber comandos do teclado e tocar sons, entre muitas outras. Procurem na rede imagens e sons (de preferência mídias de uso livre[3]) para usar na interface, tais como figuras de personagens, poços, monstros, muros, sons de flechas, urros, impactos, etc.

---

[2] Leituras suplementares relacionadas ao pygame: Tutorial dr0id (simples e direto ao ponto), Outro tutorial, Mais tutoriais (talvez não precise), Exemplos que vêm instalados com pygame, Manipulação de imagens, Controles do teclado, e Sons.

[3] Alguns sites interessantes para isso são: Creative Commons, Flickr, 500px, SoundCloud, FreeSound, Free Music Archive e o Guia da biblioteca da escola de direito de Harvard para mídias livres.

Especificação

O objetivo desta especificação é definir, da forma mais enxuta possível, os elementos obrigatórios e opcionais da implementação que deve ser feita. Você pode escrever um código do zero ou usar o arquivo mundo.py como ponto de partida (mas nesse caso tenha muita atenção às diferenças de especificação). Questões relativas a detalhes ausentes na especificação poderão ser discutidas no fórum da disciplina.

Elementos obrigatórios (mínimos):

- Implementação da classe Personagem, contendo o atributo modulo e atributos-espelho (posicao, orientacao, nFlechas), métodos relativos às ações (andar, girarDireita, girarEsquerda, atirar e compartilhar) e métodos para montar percepções e desativar a personagem (perceber e morrer).
- Implementação da classe ListaDePersonagens, contendo o atributo lista (vetor de objetos da classe Personagem) e métodos processaPercepcoes, processaPlanejamentos, processaCompartilhamentos e processaAcoes (a separação aqui se deve ao fato de que os compartilhamentos envolvem múltiplas personagens).
- Implementação da classe Interface, contendo os atributos N, M, S, I0, J0, deltaT e tela (descritos acima), além de um vetor imagens contendo todos os elementos gráficos usados, e também métodos atualizaTela e processaComando. A interface deve obrigatoriamente representar através de imagens "intuitivas" as salas vazias, muros, poços, Wumpus e personagens (que podem ter todas a mesma imagem).
- Implementação da classe MundoDeWumpus, com atributos N, mundo, personagens (objeto da classe ListaDePersonagens) e interface (objeto da classe Interface), e métodos processaJogo e finalizaJogo.

Essas classes podem ser implementadas em um único arquivo mundo.py ou em arquivos separados (nesse caso o arquivo principal deve se chamar mundo.py). Os atributos e métodos acima são obrigatórios, mas você pode usá-los como quiser, e pode criar outros atributos e métodos como achar melhor para organizar seu código.

Elementos opcionais (bônus):

- **Elementos gráficos adicionais (+1.0 ponto):** Você ganhará +1.0 ponto de bônus se implementar corretamente o posicionamento de elementos gráficos que indiquem claramente as orientações das personagens (através de rotações ou setas) e suas intenções de ação (através de recursos visuais e/ou sonoros (mas cuidado que ninguém vai entender 60 personagens falando ao mesmo tempo)).
- **Traslados fracionários de perspectiva (+1.0 ponto):** A descrição original dos comandos associados às setas direcionais prevê incrementos inteiros dos atributos I0 e J0. Você ganhará um bônus de +1.0 ponto se implementar corretamente traslados fixos de 50 pixels (relativos à perspectiva atual); note que esses traslados correspondem a atualizações fracionárias dos atributos I0 e J0, de uma fração do tamanho da sala que depende do nível de zoom, e que essa mudança requer o posicionamento de algumas imagens fora da tela visível (para preencher todo o espaço da tela).
- **Efeitos sonoros (+1.0 ponto):** Você ganhará um bônus de +1.0 ponto se implementar corretamente o acionamento de efeitos sonoros condizentes com as seguintes situações: morte de um Wumpus, queda de personagem em poço, morte de personagem devorada por Wumpus, tiros de flecha, impactos e compartilhamentos.

Sendo essa a Parte B do EP3, as notas somadas do EP3A e do EP3B correspondem ao valor indicado como EP3 no critério de avaliação (veja as Informações Importantes no PACA). Você poderia interpretar isso como se cada uma das partes valesse 5.0 pontos, ou como se valessem 10.0 e EP3 = (EP3A+EP3B)/2, não faria nenhuma diferença. O que faz diferença é saber que os bônus acima se aplicam à primeira interpretação, ou seja, como se o EP3A valesse 5.0 e o EP3B valesse até 8.0 com todos os bônus (e a nota total do EP3 utilizada na média de EPs pode chegar a 13.0).

P.S.

Inevitavelmente haverá detalhes ausentes que merecerão discussões no fórum. Por exemplo, o requisito original de “sempre compartilhar quando há outra personagem na sala” impede as personagens de se moverem se tomado ao pé da letra, o que as paralisa; alguns alunos notaram isso e buscaram alternativas, outros não (e não estão errados pois o erro era da especificação). Neste momento não nos preocuparemos com isso, pois a simulação é completamente possível independentemente das estratégias das personagens (mas quem quiser mexer no código das personagens para evitar essa situação também pode). Outro exemplo: uma flecha pode matar outra personagem? Confesso não ter experiência direta e em primeira pessoa de sobrevivência em Mundos de Wumpus. Do ponto de vista de especificação há diversas soluções plausíveis, como dizer que as personagens usam armaduras impenetráveis e por isso as flechas só acertam Wumpus, ou decidir por sorteio se a flecha acerta alguém (e quem exatamente). Leiam o enunciado com atenção e mais de uma vez. O que não estiver especificado, inespecificado está. Participem do fórum e leiam as mensagens dos outros com atenção para não perdermos discussões importantes.

Last but not least… lembre-se sempre de:

- Ler as Instruções para Entrega de EPs no PACA;
- Não postar códigos no Fórum Geral, nem os compartilhar com colegas;
- Não deixar o EP para a última hora;
- Divertir-se programando!