Implementação de Jogo de Damas em Python Orientado a Objetos e em Python Funcional

Cristiano Medeiros Dalbem - 173362

INF01121 – Modelos de Linguagens de Programação

Instituto de Informática

Universidade Federal do Rio Grande do Sul

cmdalbem@inf.ufrgs.br

1 de dezembro de 2010

Sumário

1 Python

Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, de tipagem dinâmica e forte[1]. Ela tem um foco na legibilidade, característica que já é facilitada pelo fato de ser de alto nível. Mas ela se mostra preocupada com a redigibilidade de código, também, sendo que é muito rápido e fácil de programarmos nela. Por essas e outras razões ela é uma das linguagens mais utilizadas hoje em dia, seja pra pequenas aplicações do dia a dia até grandes projetos de grandes empresas

Um dos aspectos mais aparentes do estilo de programação, e que dá fama ao Python, é que não são necessárias chaves delimitando escopos (como é em C/C++ ou Java), ou marcadores especiais (como o *end* em Lua, ou *endif*/*done*/... em Shell Script): toda marcação de escopo é feita com o espaçamento da coluna onde está o código. Bem no início é estranho, mas logo logo nos acostumamos. A linguagem é bastante flexível quanto ao tipo de espaçamento, podendo ser qualquer coisa (desde que sempre a mesma coisa), e é possível também quebrar a linha no meio chamadas de funções (figura 1.1).

Outra característica que marca bastante a experiência da programação Python é sua reflexividade. Constantemente estarás programando python com um interpretador aberto ao lado, tanto pra testar algumas funções quanto para descobrir métodos dos objetos utilizando a função dir(). Também há outras diretias como o help() com algumas informações a mais sobre os métodos e o type() que diz o tipo do objeto.

Há ainda um último elemento que que fez toda diferença no desenvolvimento deste trabalho, que é as expressões List Comprehension. Esta é uma ferramenta pouco comum nas linguagens mais populares, e normalmente se encontra em linguagens puramente Funcionais (ou fortemente baseadas em). Apesar disso, é uma ferramenta muito interessante de ser usada em conjunto com outros paradigmas, mas exije um pouco de conhecimento e prática em programação funcional para que seja efetivamente bem utilizada. Nas últimas versões essa noção foi estendida para os Generators, que são uma forma poderosa de criar iteradores. Como não foram usados generators neste trabalho, não citarei mais eles. Exemplos de usos de list comprehension

estão nas figuras 1.2 e 3.4.

A única característica da linguagem que atrapalhou um pouco, mas talvez por pura falta de experiência com esse tipo de coisa, foi que não temos como verificar certos tipos de erros em tempo de compilação. Muitas vezes o programador é obrigado a rodar o programa e fazê-lo atingir a seção que se deseja testar. O maior número de erros que encontrei durante o projeto foi de tipagem errada, já que não existe verificação de tipos em tempo de compilação. Mas creio fortemente que isso é algo com que eu ainda vá me acostumar, e pessoalmente prefiro ainda a liberdade inigualável que a tipagem dinâmica nos oferece.

```
def willEat(dirx,diry):
    if self.isInBounds(piece.x+dirx,piece.y+diry) \
         and not self.hasPiece(piece.x+dirx,piece.y+diry) \
         and    ((piece.color=="white" and diry==1) or \
                (piece.color=="black" and diry==-1)):
        return 0
    elif (self.isInBounds(piece.x+dirx*2,piece.y+diry*2) and \
            self.hasPieceWithColor(piece.x+dirx,piece.y+diry,otherColor) and \
            not self.hasPiece(piece.x+dirx*2,piece.y+diry*2)):
        return 1
    else:
        return -1
```

Figura 1.1: Exemplo de como quebrar linhas de maneira arbitrária, demonstrando que o sistema de identação do Python não insere limitações, mas sim liberdades.

```
[x for x in range(1000) if x%3 == 0 and x%5 == 0]
```

Figura 1.2: List comprehension que retorna uma lista dos números de 0 a 999 que são múltiplos de 3 e de 5. Esse tipo de expressão de Python lembra muito a definição de conjuntos por compreensão na Matemática.

2 *Análises*

2.1 Características

2.1.1 Legibilidade e redigibilidade

Como já comentado, esses são fatores muito facilitados pela linguagem. Uma coisa interessante que notei ao desenvolver esse eu primeiro projeto em Python foi que, quando a estrutura imposta pela filosofia da linguagem deveria começar a nos atrapalhar, pecebe-se que é o próprio programador que está errado.

Um exemplo é quando o espaço vertical de um escopo ultrapassa o tamanho da tela, fazendo com que não enxerguemos mais o topo, então não sabemos onde identar as linhas de maneira que pertençam ao escopo certo. Ocorre que, se temos um trecho de código tão grande assim, então certamente ele não está atendendo a boas práticas de programação e, de fato, sempre consegui separar o código de um jeito melhor que ajudasse tanto na tarefa da identação quanto na facilidade de leitura deste.

Outra questão é quando temos uma classe com muitos métodos, ou uma coleção métodos muito extensos, gerando então um arquivo verticalmente extenso também. Isso poderia ser um problema pois não temos a possibilidade, como temos em C++, de escrever a implementação de métodos da classe em arquivos diferentes: em Python, todo código referente a classe deve ser escrito logo abaixo de sua declaração, e dentro do seu escopo de definição. Mas se chegarmos a ter problema com uma classe tão sobrecarregada, então está na hora de começarmos a pensar se não daria para dividí-la em subclasses, o que normalmente é uma ótima ideia.

2.1.2 Portabilidade

Python é uma linguagem interpretada, o que significa que sua compilação gera bytecode que é executado não diretamente na arquitetura do computador do usuário, mas, mas disso, é executado por uma máquina virtual. Isso permite que o código seja altamente portável e, em

conjunto com uma biblioteca gráfica igualmente portável, que é o caso da GTK+[3], temos então um programa que funciona em praticamente todos Sistemas Operacionais disponíveis.

2.1.3 Qualidade da solução

De maneira geral, classificaria como ótima a qualidade da solução em O.O. e mediana a qualidade da solução em Funcional. Este é um caso típico para programação O.O., onde temos uma organicidade inerente ao problema, e um estilo de hierarquia em árvore, isto é, um relacionamento unidirecional dos módulos (por exemplo: interface->tabuleiro->pecas). Já a programação funcional não foi tão poderosa nesta aplicação, e até atrapalhou bastante o desenvolvimento. Isso se dá ao estilo de algoritmos que são utilizados, que são iterações simples sobre estruturas de dados simples, com nada além de for e while básicos.

Acredito que a programação funcional tem seu valor para implementação de algoritmos mais complexos, principalmente os mais ligados à Matemática. Nisso se incluem estruturas de dados complexas como Grafos, onde recursão é uma palavra-chave. Eu não classifico pior a solução funcional porque ela foi híbrida, utilizando a estrutura da interface em O.O., senão ela teria atrapalhado ainda mais.

2.1.4 Confiabilidade

Ainda que seja um jogo simples eu tive que fazer bastantes testes durante o desenvolvido, em razão do que comentei anteriormente sobre a maior parte dos erros aparecerem em tempo de execução. Em Python não existem ponteiros então nos livramos dos desastrosos *segfaults* clássicos da programação C/C++, mas em compensação temos diversos erros que surgem durante a execução e que de fato derrubam o programa. Esses erros podem ser tanto de tipos (que realmente é algo que só pode ser avaliado durante a execução), mas podem ser erros de sintaxe também!

Para projetos maiores utilizando linguagens interpretadas acredito que tenha alto valor o uso de metodologias de testes, e é algo que estarei estudando para minhas próximas implementações em Python.

2.2 Tabela comparativa

Será utilizada um sistema de pontuação com valores de 0 a 5, onde 0 significa péssimo e 5 excelente.

Características	**Python O.O.**	**Python Funcional**
Simplicidade	5	3
Ortogonalidade	4	5
Estruturas de Controle	5	5
Tipos e estruturas de dados	5	5
Sintaxe	5	5
Suporte a abstração	2	2
Expressividade	4	3
Checagem de tipos	4	4
Restrições de *aliasing*	5	5
Tratamento de exceções	5	5
Legibilidade	24	23
Escritabilidade	30	28
Confiabilidade	44	42
Total	44	42

3 *Implementação*

3.1 Regras do jogo

O jogo de Damas tem uma infinidade de variações em suas regras. Então, além das regras padrão e óbvias, são convencioadas nesta implementação as seguintes:

- O jogo começa pelo jogador branco.
- Peças normais podem andar somente para frente, mas podem comer para trás.
- Damas podem andar e comer tanto para frente quanto para trás, e depois de comer uma peça ela a dama poderá parar qualquer casa depois daquela comida.
- Depois que um jogador "toca"(seleciona) uma peça, ele é obrigado a jogar com ela. O software não permitirá selecionar peças que não podem ser movidas.
- Comer é obrigatório. Quando tiver mais de uma opção, o jogador poderá escolher qual peça comer.
- Não há regras pré-definidas para o empate. Estas deverão ser combinadas entre os jogadores.

3.2 Arquitetura do Software

A estrura principal do software se dá pelos módulos:

GUI que carrega a estrutura da GTK e trata os eventos de cliques;

Screen que contém todos os métodos relacionados ao desenho do tabuleiro, das peças e outros efeitos indicadores dos estados do jogo, como jogadas obrigatórias, peça selecionada e jogadas possíveis;

Game é onde é analisado o tabuleiro e feitos os cálculos que dirão à interface o que são jogadas obrigatórias e o que são jogadas possíveis. Basicamente é onde é implementada a lógica do jogo, ou seja, suas regras. Aqui é diferenciada a implementação O.O. da funcional, que são duas classes que herdam da GameStructure, onde são armazenadas informações e métodos básicos comuns às duas implementações.

Há ainda algumas classes menores que auxiliam na organicidade e generalidade do código, que são:

Piece é uma peça do jogo. Ela sabe sua posição, a que jogador pertence, a cor que deve ser desenhada dependendo de seu estado e tem vetores que guardam as jogadas possíveis e as jogadas obrigatórias dela.

Player é um jogador. A classe Player é um padrão de jogador, que é herdado pela classe HumanPlayer e ComputerPlayer, que implementam esses dois tipos diferentes de jogadores.

Vale comentar que a estrutura da GTK, como posicionamento dos botões e elementos de GUI, além dos eventos que são disparados, todas essas informações estão no arquivo XML chamado "gui.glade", que é editável pelo Glade[2] - uma interface gráfica para criar interfaces gráficas GTK.

3.3 Requisitos de O.O.

3.3.1 Classes e Herança

A parte Orientada a Objetos do jogo utiliza várias classes. Há duas ocasiões de herança: da classe Player para HumanPlayer e ComputerPlayer, e da classe GameStructure para OOGame e funGame.

3.3.2 Polimorfismo

Já que não existe polimorfismo paramétrico em Python, apresentarei dois casos de polimorfismo: por inclusão (figura 3.2) e por sobrecarga (método *losePiece()* da figura 3.1).

```
class Player:
    def __init__(self,color):
        self.npieces = 12
        self.color = color

    def losePiece(self):
        self.npieces -= 1

class HumanPlayer(Player):
    def __init__(self,color):
        Player.__init__(self,color)
        self.state = "stopped"
        self.type = "Human"

class ComputerPlayer(Player):
    def __init__(self,color):
        Player.__init__(self,color)
        self.type = "Computer"
```

Figura 3.1: Classes que encapsulam os dados e métodos dos jogadores.

```
def newGame(self,playerTypes=0):
    colors = ["white","black"]

    if playerTypes!=0:
        for i in range(2):
            self.player[colors[i]] = newPlayer(playerTypes[i],colors[i])
    else:
        for i in range(2):
            self.player[colors[i]] = newPlayer(self.player[colors[i]].type,colors[i])
```

Figura 3.2: Exemplo de polimorfismo por sobrecarga. Esta função pode ser chamada tanto com newGame() quanto com newGame(["Human","Human"]), por exemplo.

3.4 Requisitos de Funcional

3.4.1 Funções Puras

Nem todas funções da seção Funcional do programa são funções puras. Isso é para que haja uma integração com a GUI escrita em O.O.. Na realidade com uma boa dose de trabalho e programação obscura poderia ser feito todo sistema de jogo em Funcional, comunicando-se com a GUI, mas optei por comprometer a "pureza"das funções e manter certa clareza e organicidade do código.

```
def checkWinner(self):
    return (self.player["white"].npieces == 0 and "black") or \
           (self.player["black"].npieces == 0 and "white") or \
           "nobody"
```

Figura 3.3: Exemplo de função pura que testa se já existe um jogador vencedor, retornando no caso afirmativo ou retornando *nobody* caso contrário.

```
self.pieces = dict([ ((x,y),Piece((lambda y: (y<5 and "white") or "black")(y),x,y))
                     for x in range(0,8)
                     for y in range(0,8)
                     if [(4,7),(3,0),(0,7),(1,6),(2,5),(7,2),(1,2),(6,7),
                         (7,6),(5,6),(5,0),(4,1),(2,7),(3,2),(3,6),(4,5),
                         (0,5),(2,1),(1,0),(6,5),(0,1),(7,0),(6,1),(5,2)]
                         .__contains__((x,y))
```

Figura 3.4: Exemplo de uso de listas, utilizando o recurso de list comprehension.

3.4.2 Manipulação de listas

3.4.3 Currying

Todos os usos de list comprehensions com expressões *lambda* podem ser vistos como ocasiões de currying.

3.4.4 Recursão

Ver figuras 3.5 e 3.6.

```
def calculateEatables(self,it):
    try:
        if next(it)!="noeat":
            return 1
        else:
            return self.calculateEatables(it)
    except StopIteration:
        return 0
```

Figura 3.5: Definição da função recursiva que percorre uma lista.

```
this.eatables = self.calculateEatables(this.movables.itervalues())
```

Figura 3.6: Chamada da função recursiva. Movables é um dicionário. Repare na chamada do método que retorna um iterador dos valores do dicionário.

Referências Bibliográficas

[1] Python (programming language) - Wikipedia - website: http://en.wikipedia.org/wiki/Python_(programming_language)

[2] Glade - A User Interface Designer - website: http://glade.gnome.org

[3] GTK+ - website: http://www.gtk.org