Universidade Federal do Rio Grande do Sul
Departamento de Informática Aplicada
INF01121 - Modelos de Linguagens de Programação
Orientador: Prof. Dr. Leandro Krug Wives

Elyphoot
Trabalho Final - Relatório

Germano de Mello Andersson – 137719.
Marius Fontes – 172308.
Raphael Lopes Baldi – 143756.

Porto Alegre, 21 de Junho de 2012.

Sumário

Apresentação do Problema

Elifoot é um jogo de futebol para PC da década de 90, estilo manager, que fez muito sucesso entre os brasileiros. Decidimos implementá-lo pois os componentes do grupo pertencem ao grupo de jovens que usufruíram de muitas horas diante deste clássico.

O Elifoot

O Elifoot simula o campeonato brasileiro, dividido em 4 divisões com 8 equipes. O fluxo do jogo é o seguinte: o usuário assume aleatoriamente um time da 4ª divisão, organiza a escalação e a formação de seu time e acompanha os jogos do seu campeonato. Cada campeonato possui rodadas de turno e returno, onde todos times de uma divisão jogam entre si. Ao final de uma temporada, os 2 primeiros colocados sobem para a divisão seguinte, enquanto que os 2 últimos são rebaixados. Uma nova temporada é então iniciada.

Requisitos do Projeto

Funcionais

- Sistema multiusuário.
- Gerenciar 4 campeonatos, separados em divisões, com rebaixamento.
- Fornecer um time da série D para o usuário gerenciar, aleatoriamente.
- Exibir tabela de classificação dos 4 campeonatos, rodada a rodada.
- Permitir a gerência da escalação do seu time, alterando formação e jogadores escalados para próxima partida.
- Exibir, minuto a minuto, o placar parcial de todos jogos de uma rodada.
- Permitir que o usuário interrompa a rodada e retome-a posteriormente, seguindo de onde abandonou o jogo.

Não Funcionais

- Utilização de princípios de programação Orientada a Objetos.
- Utilização de princípios de programação Funcional.

Requisitos Implementados

De todos os requisitos que levantamos o único que não foi implementado refere-se ao gerenciamento da equipe (troca de formação e escalação). Todos os demais apresentam-se de forma completamente funcional na jogo apresentado.

Apresentação das Tecnologias Utilizadas

Linguagens

Optamos pela linguagem Python, versão 2.6. O motivo da escolha entre as disponíveis foi o interesse dos integrantes do grupo em desenvolver conhecimento sobre esta linguagem.

Para maior riqueza na interface disponibilizada via browser, utilizamos a linguagem JavaScript com manipulação de dados via JSON (AJAX).

Frameworks

Utilizamos o framework Django, versão 1.4. Ele é um framework web para Python que utiliza o modelo de desenvolvimento MVC. É um dos frameworks com maior destaque no principal motor de busca da internet.

Bibliotecas

- jQuery, versão 1.7.2.
- jQuery Effects.

Banco de Dados

- SQLite, versão 3.

Ferramentas

Para o ambiente de desenvolvimento, optamos por cada integrante utilizar a sua ferramenta preferida. As IDEs utilizadas foram: Eclipse, Vim e TextWrangler.

Como repositório de código utilizamos o Subversion, através do serviço gratuito oferecido pelo Google. A página do nosso projeto está disponível em http://code.google.com/p/elyphoot/.

Apresentação da Linguagem Python 2

Python é uma linguagem desenvolvida com o intuito de trazer a experiência de escrita de um código mais limpo e simples, inclusive em detrimento de outras características como performance. Foi criada por Guido Van Rossum, no final da década de 80. Atualmente é mantida pela organização sem fins lucrativos Python Software Foundation.

Implementações

A implementação mais popular do Python, utilizada em nosso software, é a CPython, desenvolvida em linguagem C. Existem outras implementações, mencionadas na documentação oficial, como Jython (desenvolvida em Java) e PyPy (desenvolvida na própria linguagem Python).

Notação

A descrição de análise léxica e sintática utiliza uma gramática BNF modificada. Encontramos referência a gramática de uma versão legada no link: http://docs.python.org/release/2.5.2/ref/grammar.txt.

Construções

Python é uma linguagem completa em relação a construtores básicos (estruturas de seleção, repetição, construção de classes, subrotinas, escopo, etc.).

Indentação

A indentação é recurso obrigatório na linguagem, fazendo o papel de delimitador de blocos. É mais uma característica que ajuda na legibilidade e redigibilidade do código.

Tradução

Apesar de Python ser conhecida como uma linguagem interpretada, também é possível compila-lá. É importante salientar que o ganho de performance de código compilado se dá apenas em tempo de carga, pois já está em formato bytecode. O código interpretado também é transformado em bytecode, porém tal construção acontece a cada execução dos módulos do programa.

Sistema de Tipos

Python é uma linguagem de tipagem dinâmica forte. Isso significa que, durante a execução, uma variável possui um único tipo, é permitido, porém, que esta expressão altere seu tipo, implicitamente. Tal característica induz o a utilização de polimorfismo de inclusão, dando ao código alto potencial de reuso.

Orientação a objetos

Provê todos os mecanismos básicos de uma linguagem orientada à objetos. Permite herança múltipla, motivo pelo qual não é necessário o conceito de interfaces utilizado em Java ou C#, por exemplo. Implementa encapsulamento da classe permitindo que atributos e métodos sejam declarados como privados através do prefixo '__'.

Programação Funcional

Python permite a utilização de programação funcional, fornecendo manipulação de listas de forma pura, construções do tipo lambda, funções parciais (currying), suporte a iteração de diversos tipos de dados, além de outros conceitos. Apesar de Python ser reconhecida como uma linguagem com funcionalidades de programação funcional, é importante salientar que, além de permitir programação funcional não pura, a documentação oficial da linguagem informa que em alguns casos Python utiliza mecanismos procedurais internamente para executar expressões de notação funcional.

Biblioteca padrão

Um dos grandes méritos de Python é a extensa biblioteca padrão da linguagem, oferecendo serviços das mais variadas áreas de desenvolvimento (criptografia, interface com sistema operacional, sockets de rede, acesso a dados na internet, etc.). Tal riqueza faz de Python uma linguagem constantemente utilizada para integração de sistemas.

Implementação

A seguir detalhamos a implementação, focando nos itens solicitados pelo professor e justificando aqueles que não foram completamente atendidos.

Classes, Atributos e Métodos

O diagrama simplificado das classes da nossa aplicação é o que segue:

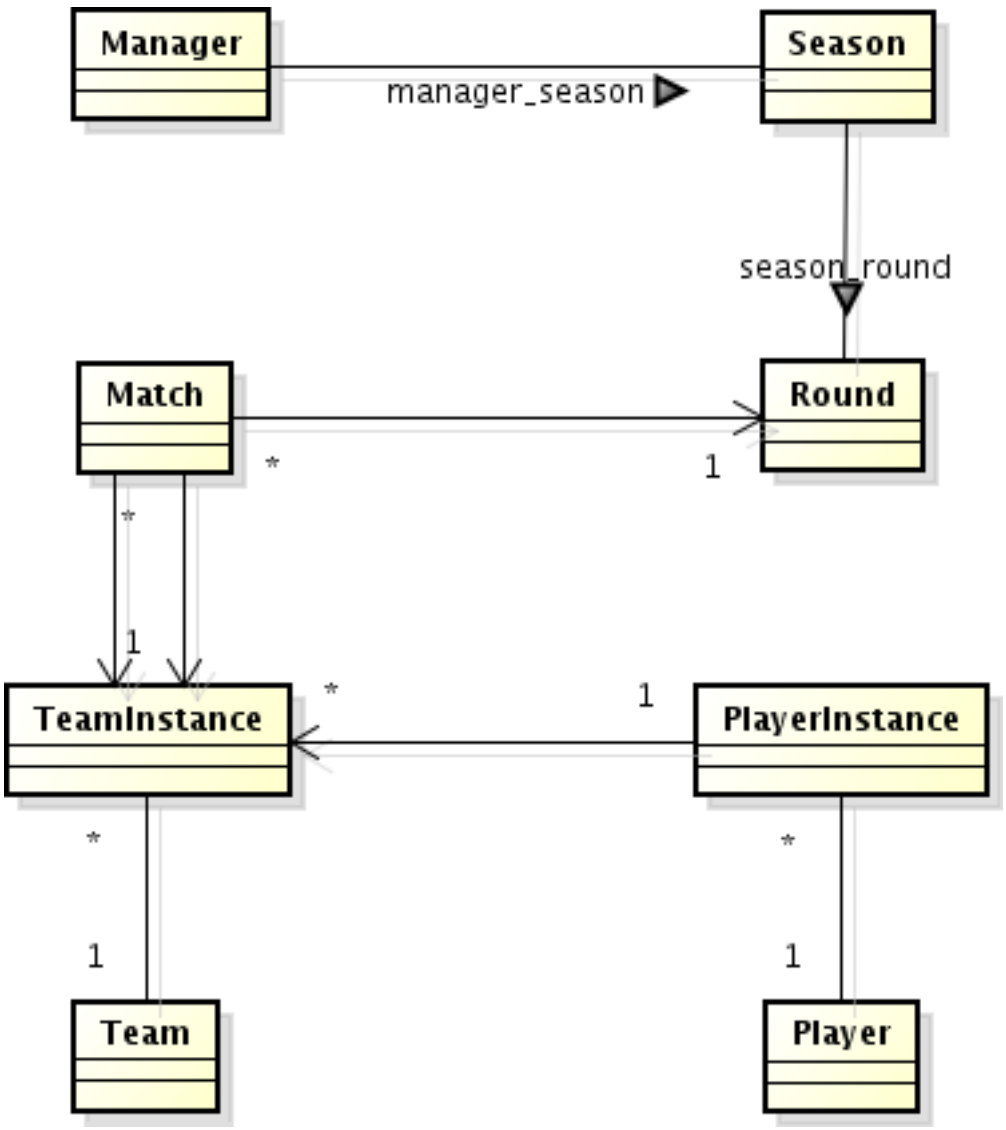

Figura 1: Diagrama de Classes

Manager

Esta classe representa o usuário do jogo, que exerce papel de treinador de uma equipe. Mantivemos o isolamento dos dados do usuário, necessários para o jogo, daqueles necessários para controle de sessão e autenticação. Concentra todos os atributos necessários para um jogador (gerente do time).

Season

Esta classe representa uma temporada, ou seja, um campeonato das 4 divisões do início ao fim. Um manager está vinculado a uma temporada. O time que o manager gerencia é mapeado através desta representação. Uma temporada possui a menor quantidade de rounds possível, respeitando o fato de que todos times devem jogar contra todos em turno e returno, e que todos os times devem jogar em todas rodadas.

Round

Esta classe representa uma rodada dos 4 campeonatos. Uma rodada possui partidas entre todas equipes de todas divisões.

Match

Esta classe representa uma partida entre dois times de uma mesma divisão. Possui dos times associados e as estatísticas do jogo.

Team, TeamInstance, Player, PlayerInstance

Estas classes representam os times e os jogadores. Um time possui diversos jogadores, enquanto que um jogador pode estar associado a apenas um time.

Inicialmente, havíamos optado por criar uma tabela de relacionamento entre matches-players-managers, porém tal configuração atrapalharia a implementação da funcionalidade multiusuário. Para solucionar esta questão, decidimos modelar times e jogadores em duas classes cada: uma base, representando os dados originais e uma instância (representadas pelas classes 'Instance'), representando os dados alterados para uma sessão específica de um usuário.

Os dados iniciais populados foram importados do jogo Elifoot original, com algumas personalizações. Tal procedimento de importação também está presente no código, no pacote gameapp.database.

Encapsulamentos

O framework escolhido (Django) opera com 3 camadas, utilizando o padrão de projeto MVC (Model-View-Controller). A camada de visão (View) é representada por *templates*, que são arquivos HTML contendo marcações especiais utilizados para exibição dos dados). O *controle* (Control) é representado por métodos responsáveis por coletar os dados necessários às visões e passá-los aos templates da aplicação. Os *modelos* (Model), apresentados anteriormente na forma de classes, são a representação dos dados da aplicação.

Uma das vantagens da utilização do framework Django é a ocultação da camada de persistência com a base de dados, ou seja, o banco de dados é oculto do desenvolvedor, que utiliza métodos de alto nível para salvar e carregar dados. Os principais disponíveis são:

- *all:* carrega todos os dados pertencentes a determinado modelo.
- *get:* permite buscar um elemento relacionado a determinado modelo através da aplicação de filtros.
- *filter:* permite buscar vários elementos de um determinado modelo através de filtros.
- *save:* permite persistir uma instância de um determinado modelo no banco de dados.

Outro ponto importante com relação a encapsulamentos diz respeito a forma como os dados são copiados das classes bases (*Team* e *Player*) para as instâncias (*TeamInstance* e *PlayerIntance*). O conhecimento desses métodos é exclusivo das classes e estes são utilizados pelos controladores durante a inicialização das temporadas.

Composição e Herança

Na versão inicial de nossa aplicação utilizamos uma classe *Person* (vide revisão 21 no nosso repositório de versões de código - http://code.google.com/p/elyphoot/source/browse/trunk/src/game/models.py?r=21), herdada por *Player* e *Manager*, que representava dados de uma determinada pessoa. No intuito de otimizar e isolar partes da aplicação, levando em conta a modularidade do framework escolhido, optamos por remover esta herança. De qualquer forma a herança é utilizada para que os modelos de nossa aplicação sejam corretamente interpretados pelo Django, ou seja, todas as classes da aplicação estendem a classe Model, do framework, indicando que tal classe deve ser persistida na base de dados.

Por se tratar de uma aplicação melhor modelada através do paradigma relacional, utilizamos extensivamente composições durante o desenvolvimento do jogo. As principais são:

Player e Team: um jogador pode fazer parte de 0 ou 1 times. Um time pode possuir diversos jogadores.

TeamInstance e Team: uma instância de um time, representada pela classe *TeamInstance*, referencia um único time base (*Team*).

PlayerInstance e Player: uma instância de jogador, representada pela classe *PlayerInstance*, referencia um único jogador base (*Player*).

PlayerInstance e *TeamInstance*: é uma relação equivalente aquela existente entre Team e Player, com a diferença de que ela é dinâmica, ou seja, varia de acordo com a temporada atual, enquanto aquela é fixa e utilizada como ponto de partida para criação das instâncias.

Match: uma partida se relaciona com dois times e uma rodada.

Season: uma temporada se relaciona com 32 instâncias de times (*TeamInstance*) e 14 rodadas (*Round*), além de possuir relacionamentos de um para um com times (campeão e time atualmente controlado pelo manager) e com rodadas (rodada atual).

Manager: um gerenciador de times possui diversas temporadas e uma temporada atual, além de um relacionamento com o usuário (*User*) utilizado pelo framework para controle da sessão do usuário.

Polimorfismo por Inclusão e Paramétrico

Como todas as classes de Python são herdeiras de um *Object*, utilizamos a redefinição de métodos (polimorfismo por inclusão) para facilitar a visualização dos dados. Isso foi obtido sobrescrevendo o método **__unicode__** em cada um dos modelos (classes) utilizados. O método **__unicode__** é invocado quando o Python precisa de uma representação amigável (Unicode) dos objetos.

Utilizamos o polimorfismo paramétrico de forma indireta, pois o framework cria automaticamente métodos para filtragem de dados que aceitam cada um dos atributos da classe como filtros. Assim podemos utilizar os exemplos a seguir:

- **teamInstance.players.filter(squad_member=True, base_player__position=2):** recupera os jogadores do time selecionado que estão atualmente escalados (squad_member=True) e jogam na zaga (base_player__position=2). Note-se que o framework permite o acesso aos relacionamentos da classe, em um filtro, através do "operador" **__**.
- **teamInstance.players.filter(goals_gt=2):** o mesmo método filter utilizado acima, agora buscando os jogadores que fizeram mais de dois gols (*gt* é um "operador" de comparação para filtragem do Django, também criado em tempo de carga dos modelos da aplicação).

Funções Como Elementos de Primeira Ordem

Utilizamos funções como elementos de primeira ordem em vários pontos da aplicação, tanto na camada de lógica de negócio, desenvolvida em Python, quanto na camada de apresentação, desenvolvida em HTML e JavaScript. Os principais pontos são:

- **Mapeamento de URL:** o framework permite que, utilizando expressões regulares, façamos o mapeamento de um tipo de endereço a uma função no controlador. Esse mapeamento pode ser feito diretamente passando-se a função (o binding é executado quando

a aplicação é carregada no WebServer) ou passando-se o nome da função (o binding é feito quando a URL relativa a função é encontrada). Nas primeiras versões utilizamos o mapeamento diretamente à funções, mas optamos, em seguida, por utilizar apenas o nome da função, pois encontramos fontes indicando que esse modo aumenta o desempenho da aplicação (URLs que nunca são chamadas não têm suas funções carregadas na memória). Exemplo: *url(r'^create_account/', 'views.create_account')* - mapeia urls terminadas em *create_account* para a função *views.create_account*.

- **Mapeamentos de listas:** utilizamos a função map do Python para mapear funções a listas de elementos. A função map recebe, como parâmetros, uma função e uma lista de elementos a serem mapeados para tal função. Exemplo: map(gameapp.match.controller.run_match, game_round.matches.all()) - mapeia a função *gameapp.match.controller.run_match* para todos os elementos da lista *game_round.matches.all()*.
- **Na camada de apresentação para atrasar a chamada de uma função.** O JavaScript permite, através do método *setTimeout*, que atrasemos a execução de uma determinada função. Este método recebe, como parâmetros, uma função e um tempo, em milissegundos, após o qual a função deve ser invocada. Exemplo: setTimeout(runRoundStep, 1000); - faz com que a função runRoundStep seja invocada após 1000 milissegundos.

Funções de Ordem Maior

Utilizamos funções de mapeamento (map) tanto na camada de negócio quanto na camada de apresentação. Um exemplo da camada de negócio foi apresentado anteriormente (**Mapeamento de listas**, acima). Outros exemplos incluem:

- **Tratamento e exibição de dados recebidos do servidor.** Utilizamos o método each, fornecido pela biblioteca jQuery, que mapeia uma função a uma lista de objetos, fornecendo, além do objeto, um índice de tal objeto para a função mapeada, aumentando o poder da função. Exemplo: *jQuery.each(matches, function(i, value) { ... })* - que mapeia a função não nomeada que recebe dois parâmetros à lista de matches.

Apesar de não utilizada, poderíamos ter optado pelo uso do método *reduce* do Python para computar o poder de um time - computado atualmente através do método *game_app.match.controller.team_power()* -, por exemplo. O Python não possui, nativamente, outras funções de ordem maior.

Manipulação de Listas Através de Funções Puras e Recursão

Um exemplo de aplicação de função pura para a manipulação de listas dentro de nossa aplicação é aquela que utilizamos, na camada de aplicação, para operar os dados recebidos do servidor, a cada passo de uma rodada, ou

seja, a cada "minuto" da rodada. Ela opera sobre todos os resultados (matches) recebidos e, utilizando apenas variáveis locais que não são salvas, atualiza a exibição (mostrando os gols feitos no último minuto e o resultado corrente das partidas.

Essa função é uma função não nomeada e encontra-se mapeada à lista de partidas através da biblioteca jQuery (vide acima). *jQuery.each(matches, function(i, value) { ... })*.

Não utilizamos recursão para manipular listas na aplicação, pois não sentimos necessidade das mesmas uma vez que tínhamos métodos como map e blocos for orientados a elementos (for elemento in lista) que supriram as necessidades do nosso jogo.

Currying

Utilizamos currying em dois pontos da camada de aplicação, uma vez que o JavaScript permite que façamos a definição de métodos inline (função não nomeada) que facilita a interpretação do código pelos desenvolvedores:

- **jQuery.each(matches, function(i, value)) { ... }):** utilizada para atualizar a exibição dos resultados, recebe como parâmetros o índice atual do elemento sendo processado (i) e o elemento em si (value).
- **$.getJSON("/game/play_round_step/", function(data) { ... }):** trata o resultado de uma chamada (GET) no servidor, representando um minuto da rodada.

Casamento de Padrões

O framework Django permite que façamos o reconhecimento de endereços através de expressões regulares e façamos o mapeamento das mesmas aos métodos de controle:

- **url(r*'^logout/$'*, *'gameapp.views.logout'*):** endereços contendo logout são direcionadas para o método gameapp.views.logout.
- **url(r*'^$'*, *'gameapp.views.index'*, name=*"index"*):** todos os endereços que não possuem qualquer sequência (apenas o domínio) são redirecionados para o método gameapp.views.index.
- **url(r*'^game/'*, include(*'gameapp.urls'*)):** todos os endereços contendo game no final de sua sequência devem ser tratados pelos padrões específicos da aplicação gameapp (arquivo gameapp/urls.py).

Pacotes e Estrutura

Quanto a sua estrutura física, nosso software foi organizado da seguinte maneira, de acordo com as características do framework escolhido:

- Projeto ‘elyphoot’.
    - Aplicação ‘gameapp’.
        - Pacote database.
        - Pacote manager.
        - Pacote match.
        - Pacote round.
        - Pacote season.
        - Modelos.
        - Visões.
        - Mapeamentos (urls).
- Database.
- Documentação.

O framework Django, apesar de permitir que os modelos sejam separados em pacotes, não permite que isso seja feita de forma simples e elegante, por isso optamos por manter todos modelos na raiz da aplicação, em um único arquivo models.py, e utilizamos os pacotes para armazenar o código de controle referente a cada modelo. O código relacionado as visões também foi mantido em um único arquivo, uma vez que ele foi simplificado (concentrado nos controladores) e o framework trabalha melhor desta forma.

Cabe ressaltar que as visões, no framework, são divididas em duas partes: uma responsável por tratar as requisições do usuário e outra por tratar a exibição (representada por modelos escritos em HTML, contendo marcações e estruturas de exibição de dados). Isso permite grande flexibilidade no desenvolvimento da aplicação.

Análise das Características e Critérios da Linguagem (Python)

Característica	Nota	Justificativa
Legibilidade	9	Utiliza indentação como delimitador de bloco. Polimorfismo universal para diversas operações básicas. Não permite sobrecarga dos operadores básicos.
Redigibilidade	9	Sintaxe intuitiva. Polimorfismo universal permite maior abstração para construção dos TADs. Alto grau de expressividade em comandos de iteração.
Confiabilidade	7	Fortemente tipado. Possui STE. As palavras chave da linguagem são reservadas (o interpretador gera um alerta de sobrescrita caso você tente utilizá-las). Tipagem implícita e dinâmica pode levar o programador ao erro mais facilmente.
Simplicidade	10	Utiliza indentação como delimitador de bloco.
Ortogonalidade	7	Polimorfismo universal para diversas operações básicas. Não permite sobrecarga dos operadores básicos.
Portabilidade	9	Por ser linguagem interpretada, o porte é dependente das bibliotecas utilizadas.
Expressividade	9	Alto grau de expressividade, especialmente em comandos de iteração.
Reusabilidade	9	Polimorfismo universal; composição. Como Python permite múltipla herança, o efeito de interface em Java pode ser obtido via classes abstratas.
Estruturas de Controle	9	Possui pacote de maioria dos tipos de dado modelados para serem manipulados por iteradores. Fornece ao programador ferramentas para iterar sobre seus TADs.
Tipos de Dados	9	Básicos, listas, dicionários, coleções, tuplas de dados.
Estruturas de Dados	9	Classes.
Tratamento de Exceções	9	Possui. É possível estender a classe de exceções, permitindo ao programador criar suas próprias exceções.
Restrições de Aliasing	7	Atribuição de variável para variável é por referência. O interpretador Python é escrito em C e compilado usando *gcc* com a opção *-fno-strict-aliasing*, que evita conflitos na utilização de alias com otimizações realizadas pelo compilador.
Checagem de Tipos	10	Tipagem forte e dinâmica.

Tabela 2: Avaliação da Linguagem

Conclusões

Foi uma ótima experiência alinhar aplicação de técnicas de programação com a modelagem de um jogo que tanto nos encantou, há mais de uma década, do ponto de vista do usuário. Conseguimos colocar em prática conceitos apresentados durante o semestre em uma linguagem de programação que não era de nosso domínio, mas que se mostrou poderosa e de uma simplicidade singular. Aproveitamos o desafio e optamos por trabalhar com um framework web que exigiu esforço inicial para conhecê-lo, recompensado pelas facilidades que ele trouxe posteriormente, especialmente quanto a sua API para mapeamento objeto-relacional.

Um marco em nosso projeto foi quando nos deparamos com o desafio de implementarmos o requisito multiusuário. Descobrimos que nosso modelo original dificultaria bastante tal implementação. Fizemos algumas reuniões de alinhamento e optamos por manter o requisito, sendo necessário algumas alterações no modelo do projeto para tal.

Também pensamos, no meio do projeto, em mudar a linguagem e bibliotecas utilizadas, pois a curva de aprendizado do framework escolhido estava se mostrando bastante íngreme. Após realivação decidimos continuar com as nossas primeiras decisões e implementar o jogo em Python utilizando o framework Django.

Após a apresentação ao professor fizemos diversas modificações no relatório pois não tínhamos compreendido a obrigatoriedade de utilização de dos itens apresentados na implementação do trabalho. De qualquer forma utilizamos, de forma consciente, a maioria deles e, de forma inconsciente (por considerarmos boas práticas de programação) alguns outros. Acreditamos que um ponto que poderíamos melhorar seria o preparo antes da apresentação uma vez que não conseguimos defender a completude de nosso jogo ante os questionamentos do orientador.

Bibliografia

ELIAS, A. **Elifoot - Site Oficial**, 2012. Disponível em: <http://www.elifoot.net/>. Acesso em: 17 jun. 2012.

GOOGLE INC. Google's Python Class. **Google Code University**. Disponível em: <http://code.google.com/intl/pt-BR/edu/languages/google-python-class/>. Acesso em: 22 maio 2012.

HOLOVATY, A.; KAPLAN-MOSS, J. The Django Book: Version 2.0. **The Django Book**, 16 mar. 2009. Disponível em: <http://www.djangobook.com/en/2.0/>. Acesso em: 01 jun. 2012.

LARMAN, C. **Applying UML and Patterns:** An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd Edition. ed. New Jersey: Pearson Education, Inc., 2004.

PYTHON SOFTWARE FOUNDATION. Python v2.6.8 documentation. **Python Docs**, 12 abr. 2012. Disponível em: <http://docs.python.org/release/2.6.8/>. Acesso em: 29 maio 2012.

SEBESTA, R. **Conceitos de Linguagens de Programação**. Tradução de Eduardo Kessler Piveta. Nona Edição. ed. Porto Alegre: Bookman Companhia Editora LTDA, 2010.

WIKIPEDIA. Elifoot. **Wikipedia**, 20 abr. 2012. Disponível em: <http://pt.wikipedia.org/wiki/Elifoot>. Acesso em: 15 jun. 2012.