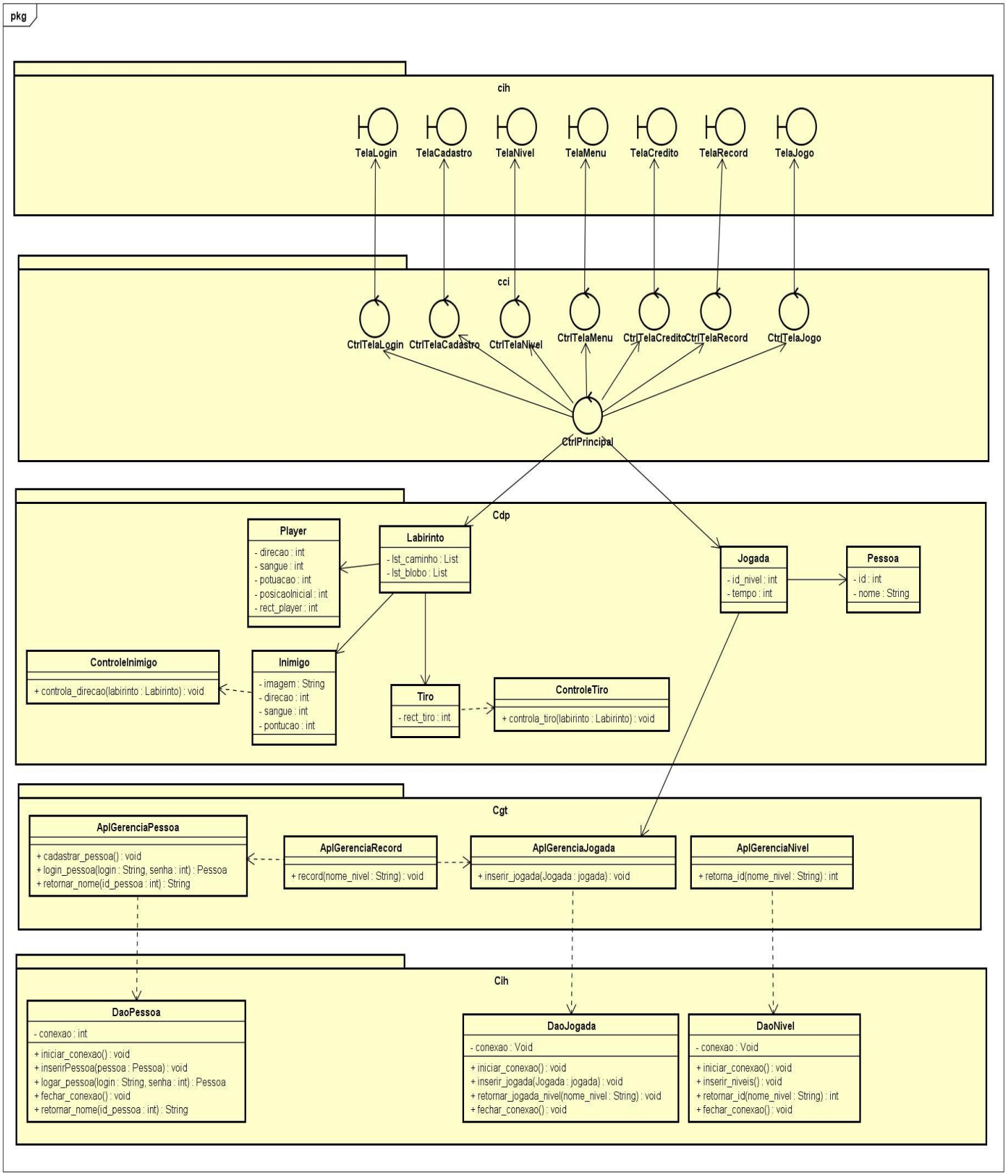
Mini Mundo

O software consiste em um jogo do tipo labirinto em 2D, construído em python utilizando a biblioteca pygame. De acordo com o nível de dificuldade (fácil, médio ou difícil) é gerado aleatoriamente um labirinto e seu tamanho varia de acordo com o nível de dificuldade(quanto maior o nível de dificuldade, maior o labirinto e a quantidade de inimigos). O objetivo é chegar ao final do labirinto, sem que os inimigos o matem, no menor tempo possível . Para chegar à saída, você possui uma arma e sua velocidade. O jogo guardará os melhores recordes de cada nível. Haverá (ainda não implementado) a possibilidade de jogo multiplayer.

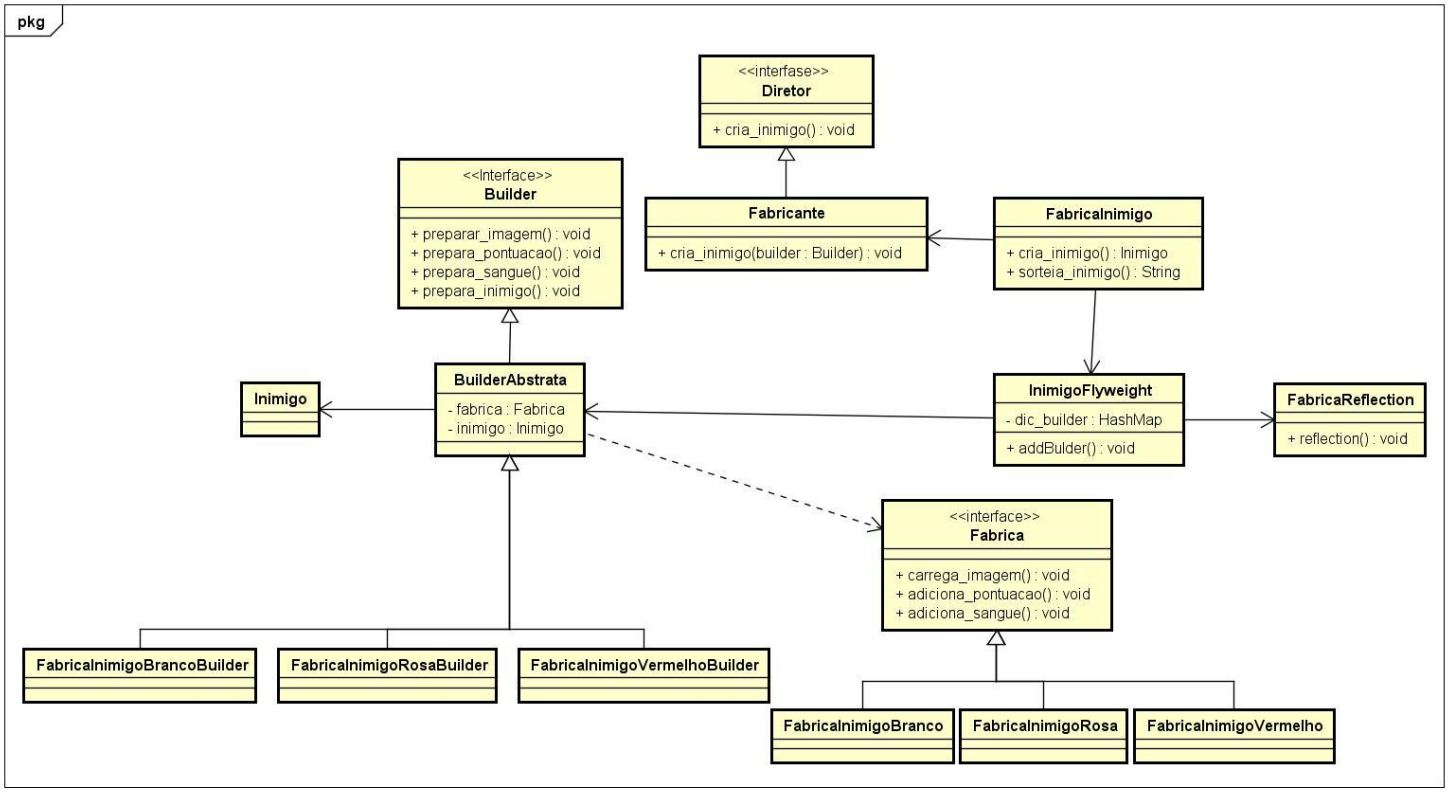
Para iniciar o jogo são necessários um login e senha, mas caso você não possua um, é possível fazer um cadastro.

Descrição da utilização do padrão MVC



No código aplicamos o padrão MVC . A pasta cih contém as telas do jogo, cada tela tem seu controlador que fica na pasta cci, que por sua vez, tem o controlador principal que é responsável pelo loop do jogo e por fazer a comunicação da camada de visão com a camada modelo.Na cdp temos o labirinto que é composta por lista de caminho, blocos, tiros e o player que são utilizadas para controlar as movimentações do labirinto. A classe jogada

tem o objetivo de salvar informações das jogadas de uma pessoa para salvar no banco de dados. Na pasta cgt ficam as apls para gerenciar as persistências do banco e na pasta cih ficam as classe responsáveis por acessar o banco de dados



Padrões de Projeto

Para criar as melhores soluções, é preciso seguir um processo detalhado para obter uma análise dos requisitos, funcionais ou não funcionais, e desenvolver um projeto que os satisfaça e que possibilite submetê-los a teste, para constatar eventuais falhas, além do mais se deseja que o projeto tenha uma arquitetura flexível para acomodar futuros problemas e requisitos sem a necessidade da realização do re-projeto.

Os padrões de projetos tornam mais fáceis reutilizar soluções e arquiteturas bem sucedidas para construir softwares orientados a objetos de forma flexível e fácil de manter. O uso de padrões de projeto pode reduzir a complexidade do processo de projetar software. Além disso, o software orientado a objetos bem projetado possibilita aos projetistas reutilizar e empregar componentes preexistentes em sistemas futuros. [Conheça os Padrões de Projeto http://www.devmedia.com.br/conheca-os-padroes-de-projeto/957#ixzz3sMLPK1al]

Padrão Composite

Composite é um padrão de projeto de software utilizado para representar um objeto que é constituído pela composição de objetos similares a ele. Neste padrão, o objeto composto possui um conjunto de outros objetos que estão na mesma hierarquia de classes a que ele pertence. O padrão composite é normalmente utilizado para representar listas recorrentes - ou recursivas - de elementos. Além disso, esta forma de representar elementos compostos em uma hierarquia de classes permite que os elementos contidos em um objeto composto sejam tratados como se fossem um único objeto. Desta forma, todos os métodos comuns às classes que representam objetos atômicos da hierarquia poderão ser aplicáveis também ao conjunto de objetos agrupados no objeto composto.

No trabalho foi utilizado esse padrão no objeto Labirinto, que é composto por: Player,Inimigos,Parede,Saída,Tiros.

Padrão Fábrica

O padrão fábrica foi utilizado para fabricar inimigos. O padrão Fábrica fornece uma interface para a criação de famílias de objetos correlatos ou dependentes sem a necessidade de especificar a classe concreta destes objetos. Ele é útil quando você precisa criar objetos dinamicamente sem conhecer a classe de implementação, somente sua interface (o padrão Fábrica estabelece uma forma de desenvolver objetos que são responsáveis pela criação de outros objetos).

Na classe FabricaInimigo instanciamos a classe InimigoFlyweight que possui um dicionário de builders, nesse dicionário as builders são adicionadas dinamicamente de acordo com a criação de cada tipo de inimigo. Ainda na classe InimigoFlyweight é instanciada a classe FabricaReflection que é responsável por identificar as builders por meio do tipo do personagem. A builder tem o objetivo de separar a construção de objetos complexos (inimigo) de sua representação de modo que o mesmo o mesmo processo de construção possa criar diferentes representações.

Padrão Flyweight

Um Flyweight é um objeto que minimiza o uso de memória através da partilha de dados, tanto quanto possível com outros objetos semelhantes. É uma maneira de usar objetos em grandes números quando uma simples representação repetida usaria uma quantidade inaceitável de memória.

Para aplicar esse padrão utilizamos a classe InimigoFlyweight que possui um dicionário onde a chave é o nome do inimigo e o conteúdo é a builder. O conteúdo do dicionário é adicionado só quando utilizado pela primeira vez e fica guardado para ser utilizado uma próxima pelo mesmo tipo de inimigo.Com isso economizamos recurso.

Padrão Proxy

O Pattern Proxy é um padrão Estrutural definido pelo GoF (Gang of Four).

O seu objetivo principal é encapsular um objeto através de um outro objeto que possui a mesma interface, de forma que o segundo objeto, conhecido como “Proxy”, controla o acesso ao primeiro, que é o objeto real. [http://www.devmedia.com.br/conheca-o-pattern-proxy-gof-gang-of-four/4066#ixzz3sMaSUJGH]

Utilizamos o Proxy ao iniciar o jogo, em uma janela com a nossa “logo”, até que o jogo seja carregado e a tela de Login seja apresentada, como um intermediário entre o usuário e o banco de dados. Por ele, verificamos quem pode ter acesso as informações do jogo. Para ter acesso ao jogo são necessários um Login e Senha que pode ser criado na janela de Cadastro.

Padrão Facade

Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. Facade define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.

No trabalho esse padrão foi utilizado pela AplGerenciaJogada que gerencia DAOJogada no método inserir_jogada,que por sua vez, objetiva inserir as informações de uma jogada no banco de dados. Esse método encapsula todo o processo de inserção no banco.

```
class AplGerenciaJogada:

    def Inserir_jogada(self,jogada):
        dao_jogada = DaoJogada()
        dao_jogada.inciar_conexao()
        dao_jogada.inserir_jogada(jogada)
        dao_jogada.fechar_conexao()


    def  retorna_jogadas_nivel(self,nome_nivel):
        dao_Jogada = DaoJogada()
        dao_Jogada.inciar_conexao()
        lista_jogada= dao_Jogada.retorna_jogadas_nivel(nome_nivel)
        dao_Jogada.fechar_conexao()
        return lista_jogada
```

Reflection

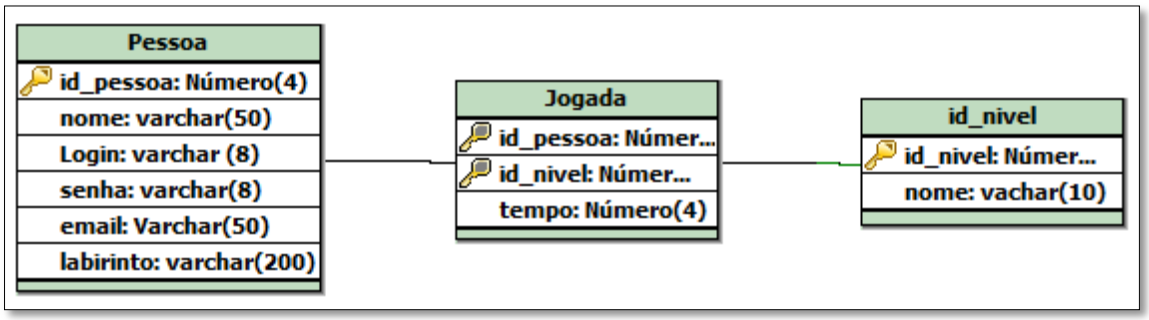
Reflection é uma maneira de se descobrir dados de uma classe/objeto/interface em tempo de execução. Foi utilizado na Fábrica de Inimigos com o objetivo de apontar para uma Fábrica específica de um tipo de inimigo, evitando a verificação por intermédio de IF's.

```
class FabricaReflection:

    def reflection(self,nome_inimigo):
        contrutor="FabricaInimigo"+nome_inimigo +"Builder"
        nome=[contrutor]
        path = "util." + contrutor
        _temp = __import__(path, fromlist=nome)

        builder= getattr(_temp, contrutor)
        builder = builder()
        return builder
```

Banco de Dados

É um conjunto de arquivos relacionados entre si com registros sobre pessoas, lugares ou coisas. São coleções organizadas de dados que se relacionam de forma a criar algum sentido (Informação) e dar mais eficiência durante uma pesquisa ou estudo.

Em nosso trabalho utilizamos o padrão DAO com banco de dados SQlite3, para salvar o cadastro do jogador.



Mudanças Observadas com Implementação de Padrões

Podemos notar que ao implementar padrões o código tornou-se mais legível. Começando pela utilização do MVC, que dividiu o código em "regiões", quando precisava de alguma coisa, sabia mais ou menos aonde procurar.

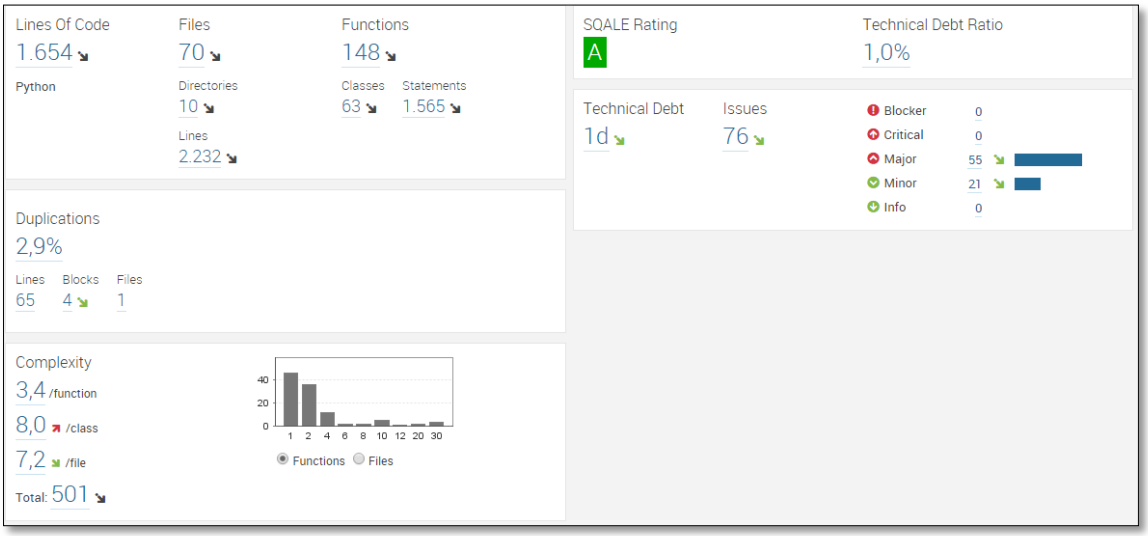
Com os padrões criativos obtivemos um maior encapsulamento e abstração para uso do código.

Subindo para os padrões estruturais, conseguimos aumentar o "reuso/compartilhamento" de código e obtivemos uma maior economia no uso de memória e de códigos.

Decorator

Os Decorators fornecem uma alternativa flexível ao uso de subclasses para extensão de funcionalidades. O decorator tem como objetivo adicionar responsabilidades a um objeto dinamicamente. Ainda não utilizamos tal padrão, mas como pretendemos adicionar novos poderes ao player, tal recurso será muito útil.

Avaliação do Sonar



Em relação ao trabalho anterior a complexidade diminuiu de 3,6 para 3,4 depois da implementação de padrões de projeto e refatoração do código para o MVC. Mas a complexidade do código continua elevada. Acreditamos que o motivo possa ser a elevada quantidade de verificações com "if" como podemos verificar na figura abaixo.