**INSTUTO FEDERAL DO ESPÍRITO SANTO**
Douglas Campos Sutil
Michelle Borges Botelho

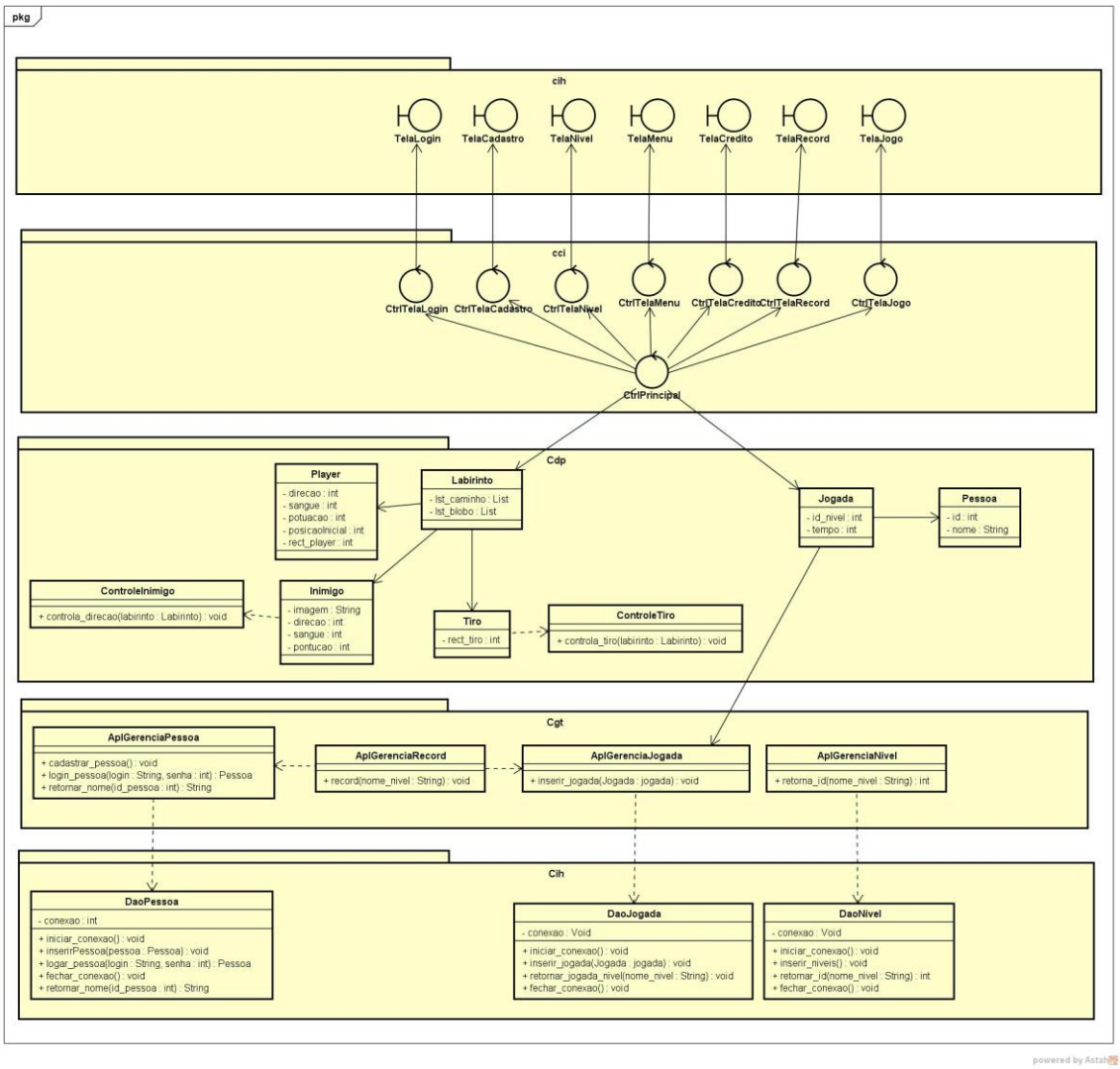
# Trabalho 4
# Padrões de Projeto Comportamentais

SERRA

**2016**

# Conteúdo

## 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

**Memento**

Memento é um padrão de projeto que permite armazenar o estado interno de um objeto em um determinado momento, para que seja possível retorná-lo a este estado, caso necessário. No código, esse padrão foi utilizado para guardar a coordenada do player antes da movimentação ser executada. Pois quando a posição do player é atualizado, caso haja uma colisão o estado anterior é recuperado.

```
if labirinto.player.rect_player.top>=0 :
    dy=labirinto.player.rect_player[1]
    labirinto.player.rect_player[1]-=self.speed
    for bloco in labirinto.lst_blocos[:]:
        if labirinto.player.rect_player.colliderect(bloco):
            labirinto.player.rect_player[1]=dy
    for inimig in labirinto.lst_inimigos:
        if labirinto.player.rect_player.colliderect(inimig.rect_inimigo):
            self.opcao=10
```

Guarda a Posição

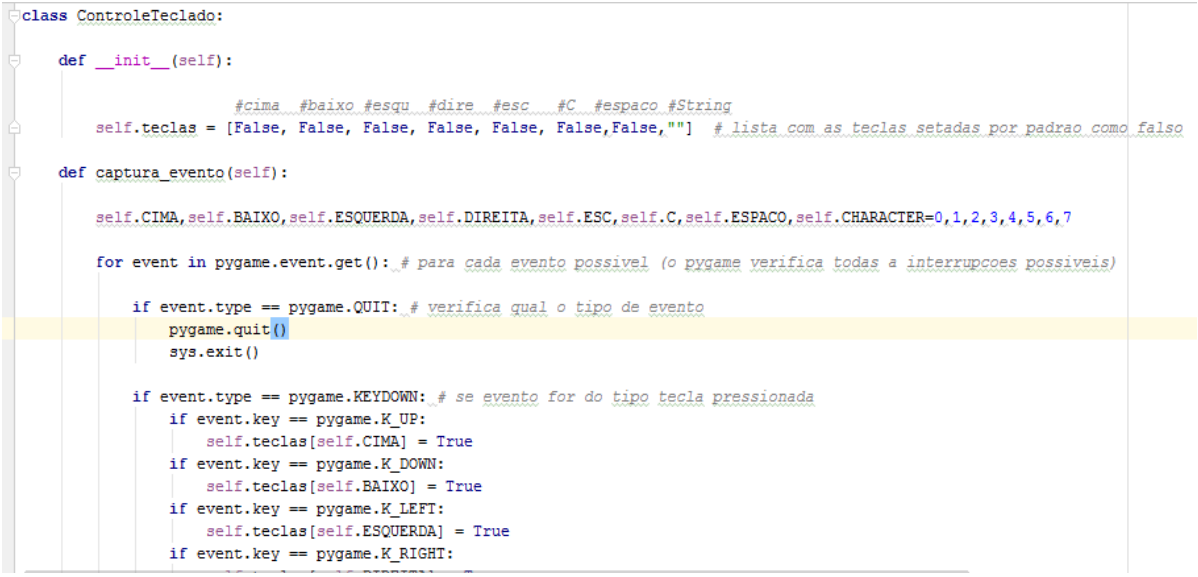
**State**

Esse padrão permite um objeto alterar seu comportamento quando o seu estado interno muda. Esta mudança é significativa e visível para o usuário, assemelhando-se a troca de classe. No jogo utilizamos esse padrão nas mudanças de tela, onde cada tela é um estado e dependendo do que o usuário apertar muda o comportamento.

**Observador**

O padrão define uma dependência um para muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente.

Foi utilizado na classe "ControleTeclado" onde observa as teclas pressionadas pelo usuário. A captura do que foi apertado do teclado é implementado pelo pygame. No exemplo abaixo quando um tecla é apertada, verifica qual é a tecla e modifica a posição na lista referente a essa tecla para True.



```
class ControleTeclado:

    def __init__(self):

                    #cima  #baixo #esqu  #dire  #esc   #C  #espaco #String
        self.teclas = [False, False, False, False, False, False,False,""]  # lista com as teclas setadas por padrao como falso

    def captura_evento(self):

        self.CIMA,self.BAIXO,self.ESQUERDA,self.DIREITA,self.ESC,self.C,self.ESPACO,self.CHARACTER=0,1,2,3,4,5,6,7

        for event in pygame.event.get(): # para cada evento possivel (o pygame verifica todas a interrupcoes possiveis)

            if event.type == pygame.QUIT: # verifica qual o tipo de evento
                pygame.quit()
                sys.exit()

            if event.type == pygame.KEYDOWN: # se evento for do tipo tecla pressionada
                if event.key == pygame.K_UP:
                    self.teclas[self.CIMA] = True
                if event.key == pygame.K_DOWN:
                    self.teclas[self.BAIXO] = True
                if event.key == pygame.K_LEFT:
                    self.teclas[self.ESQUERDA] = True
                if event.key == pygame.K_RIGHT:
```

Também foi aplicado na lógica de colisão do player com as paredes do labirinto e com os inimigos.

**Mediador**

O padrão de projetos Mediator é um padrão utilizado para gerenciar a comunicação entre dois ou mais objetos. Promove um baixo acoplamento evitando que os objetos façam referência uns aos outros de forma explícita.

No código o mediador é a classe "ControlePrincipal", é responsável por gerenciar a comunicação com as outra classes do jogo como por exemplo os controles das telas do jogo.

**Cadeia de responsabilidade**

Evitar o acoplamento do remetente de uma solicitação ao seu receptor, ao dar a mais de um objeto a oportunidade de tratar a solicitação. Encadear os objetos receptores, passando a solicitação ao longo da cadeia até que um objeto a trate. Como o funcionamento do jogo é um loop, a aplicação desse projeto fica complicado necessitando fazer "gambiarras" para implementa-lo.

**Iterator**

Prover um meio de acesso sequencial aos elementos de uma estrutura, sem informar a representação da estrutura. Não houve necessidade de aplicar, pois no código só tem um tipo de estrutura que é a lista.

**Command**

O Padrão Command encapsula uma solicitação como um objeto, o que permite parametrizar outros objetos com diferentes solicitações, enfileirar ou registrar solicitações e implementar recursos de cancelamento de operações. No nosso jogo não houve necessidade de aplicar essa padrão

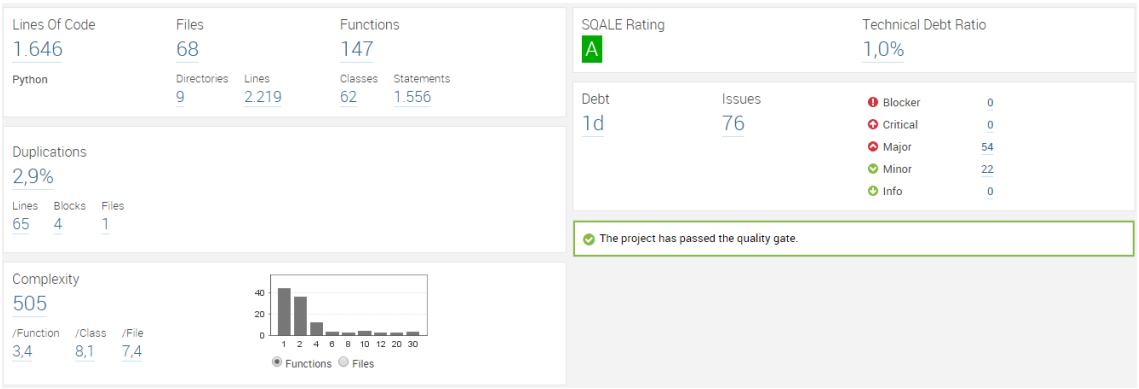
**Strategy**
Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam. Permite configurar uma classe com um de vários comportamentos, utilizando composição.No jogo poderia aplicar no inimigos se ele tivessem comportamentos diferentes.

**Visitador**
Permite definir uma nova operação sem mudar as classes dos elementos sobre os quais opera. Durante o desenvolvimento não houve necessidade de aplicar esse padrão de projeto.

## Mudanças Observadas com Implementação de Padrões

Nessa etapa do trabalho não houve a necessidade de mudar muita coisa código, pois sem perceber aplicamos a maioria dos padrões comportamentais anteriormente. A aplicação dos padrões de projeto nos ajudou a olha o código de outra maneira, não mas escrevê-lo de qualquer forma pra rodar , mas sim se preocupar como cada parte esta sendo construída. A aplicação de MVC ajudou muito na organização do código facilitando a busca de partes do código. E os testes? Ahh os testes!! O que dizer sobre eles?!?! é difícil admitir mas foram importantes para o andamento do código, diminuindo o retrabalho.

## Avaliação do Sonar




Em relação ao código anterior não correram muitas mudanças, a complexidade continua a mesma. Devido a grande quantidade de verificações com "if" a complexidade continua alta.