Adailson Almeida dos Santos1; André Vinicius Vieira Mildemberg1;

*1 Graduando do curso de Análise e desenvolvimento de sistemas do Centro Universitário Unifacear;*

**RESUMO**

*Realizamos o desenvolvimento de um jogo de damas em linguagem Java, para realizar o planejamento do desenvolvimento do sistema utilizamos diagramas UML, como o diagrama de casos de uso e diagrama de classes.*

*O sistema consiste em um jogo de damas com as regras tradicionais do jogo, que compreende na partida ser realizada entre dois jogadores, em tabuleiro 8X8 (64 casas) alternadamente brancas e pretas, com 12 peças brancas e 12 peças pretas.*

*Para gerar uma maior competitividade nos jogadores o sistema conterá um ranqueamento entre todos os jogadores com base em pontos que eles ganharão nas vitorias de cada jogo.*

*Além disso para uma melhor organização no desenvolvimento utilização o padrão de projeto MVC (Model-View-Controller) afim de facilitar manutenções futuras no código e a reutilização do mesmo em outros projetos.*

*Palavras chave: damas, jogador, Java, diagramas.*

***ABSTRACT***

*We developed the game of checkers in Java language, to carry out the planning of the development of the system uses UML diagrams, such as the use-case diagram and class diagram.*

*The system consists of a game of checkers with the classic rules of the game, such as: The game is played between two players, on an 8X8 (64) board alternately white and black, with 12 white pieces and 12 black pieces.*

*To generate a greater characteristic in the players the system will contain a ranking among all the players based on points that they will win in the victories of each game. In addition, for a better organization in the development, use or MVC design pattern in order to facilitate maintenance in future code and the reuse of it in other projects.*

*Keywords: checkers, player, Java, diagrams.*

1. **INTRODUÇÃO**

Para este projeto nós desenvolvemos um jogo de damas em linguagem Java com base nos princípios de orientação a objetos (POO). Ademais, também foi utilizado o padrão de projeto MVC afim de organizar melhor a estrutura do sistema e facilitar futuras manutenções. Para nos auxiliar no desenvolvimento nos utilizamos a IDE Eclipse.

Além disso, no processo de desenvolvimento nós utilizamos diagramas baseados na UML para definirmos de forma mais concreta o que o sistema deverá conter, para desta forma, sermos mais produtivos no desenvolvimento. Os diagramas utilizados foram o diagrama de classes e o diagrama de casos de uso.

1. **DESENVOLVIMENTO**

No tópico de desenvolvimento abordaremos a variante de regras do jogo de damas escolhido que será transcrito no tópico 2.1. No item 2.2 falaremos sobre as ferramentas que utilizamos durante o desenvolvimento do sistema e como nós fizemos uso delas. No tópico 2.3 serão apresentados a prototipagem das principais telas do sistema.

Nos itens 2.4 e 2.5 serão expostos e elucidados os diagramas de casos de uso e de classes respectivamente, os quais foram elaborados com o intuito de facilitar o planejamento do projeto. Na sequência no tópico 2.6 contextualizaremos sobre o padrão de projeto model-view-controller que foi utilizado no desenvolvimento do sistema.

Finalmente no item 2.7 apresentaremos os principais algoritmos desenvolvidos para o sistema e uma breve explicação sobre cada um deles.

2.1 REGRAS

De acordo com o site *BrainKing* existem diversas versões variantes no jogo de damas entre elas as damas internacionais, Canadianas, alquerques, checas, brasileiras e tradicionais que inevitavelmente é a versão mais popular do jogo. O nosso jogo será baseado na versão tradicional. As regras do jogo de damas tradicional são:

* Tabuleiro 8X8
* Partida realizada entre dois jogadores
* 24 peças ao todo
* 12 pretas e 12 brancas
* Peças brancas começam
* A peça só anda para frente
* Anda uma casa por vez
* Quando atinge a última linha a peça vira dama
* A dama anda para frente e para trás, porém com o limite de uma casa
* Se houver a possibilidade a captura é obrigatória
* A peça só pode ser capturada se houver um espaço livre na mesma diagonal que a peça que irá capturar está
* As peças podem realizar a captura indo para trás
* Se no mesmo lance se apresentar mais de um modo de capturar, é obrigatório executar o lance que capture o maior número de peças (Lei da Maioria).
* Após 20 lances sucessivos, sem captura ou deslocamento de pedra, a partida é declarada empatada.

2.2 FERRAMENTAS UTILIZADAS NO DESENVOLVIMENTO

Para o desenvolvimento do projeto nós fizemos uso de diversas ferramentas a seguir iremos apresenta-las e falar um pouco sobre como o uso delas nos auxiliou no desenvolvimento do projeto.

2.2.1 Eclipse

Para a programação do software nós utilizamos o Eclipse que é uma IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado) que é focada em Java, porém tem suporte para algumas outras linguagens como PHP e Python. Além disso o Eclipse é um software de iniciativa open source, ou seja, ele é gratuito e tem milhares de colaboradores.

2.2.2 Draw.io

Criado em setembro de 2016, o Draw.io é um software online grátis usado para criar diagramas. Nele nós elaboramos os diagramas de classe e de casos de uso do projeto.

2.3 PROTOTIPAGEM

Como foi feito uso da biblioteca *Swing* do java nós tivemos um grande foco no desenvolvimento das interfaces gráficas do projeto. A seguir nas figuras 1 e 2 a prototipagem da tela inicial e da partida.



FIGURA 1: Tela Principal.

FONTE: Os autores (2020)

Na tela principal do sistema temos um menu com a opções jogar, que inicia uma partida, cadastrar-se e ranking para visualizar o ranking de jogadores.

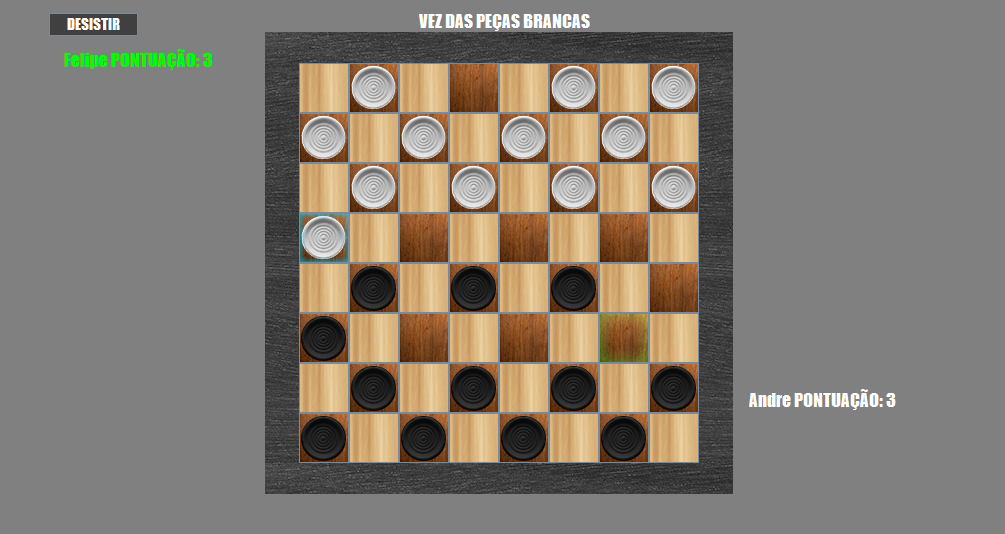


FIGURA 2: Tela do jogo.

FONTE: Os autores (2020)

Na tela da partida nos temos os nomes dos jogadores, quando for a vez de um jogador jogar o seu nome fica na cor verde, além disso temos um botão desistir que ao aperta-lo o jogador da vez desiste da partida e seu oponente ganha os pontos pela vitória.

2.3 DIAGRAMA DE CASOS DE USO

Um diagrama de casos de uso é um diagrama dinâmico ou de comportamento na UML (Linguagem de Modelagem Unificada). O cientista de computação JACOBSON descreve casos de uso da seguinte forma, “podemos dizer que um caso de uso é um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo.”.

Com base nesta descrição desenvolvemos o diagrama de casos de uso do projeto. A seguir na figura 3, o Diagrama de casos de uso do projeto:

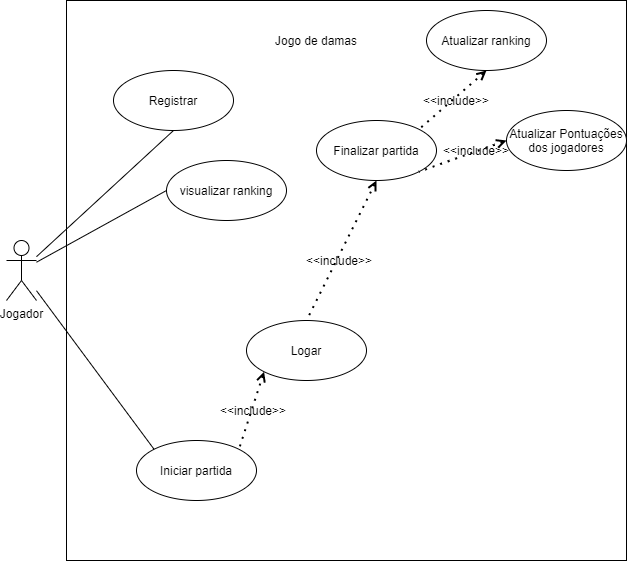


FIGURA 3: Diagrama de casos de uso do projeto.

FONTE: Os autores (2020)

2.4 DIAGRAMA DE CLASSES

Segundo BELL no livro Fundamentos básicos de UML: “O propósito do diagrama de classes é mostrar os tipos que estão sendo modelados no sistema e a relações entre eles”. Nessa perspectiva observa-se que é fulcral a elaboração de um diagrama de classes para um sistema afim de documentar corretamente o funcionamento do mesmo e facilitar futuras manutenções no código.

Na figura 4, temos representado a versão completa do diagrama de classes do projeto, porém pelo motivo dele ser relativamente extenso, para a melhor visualização neste artigo nos dividimos ele em cinco partes que estão representadas nas figuras 5, 6, 7, 8 e 9.

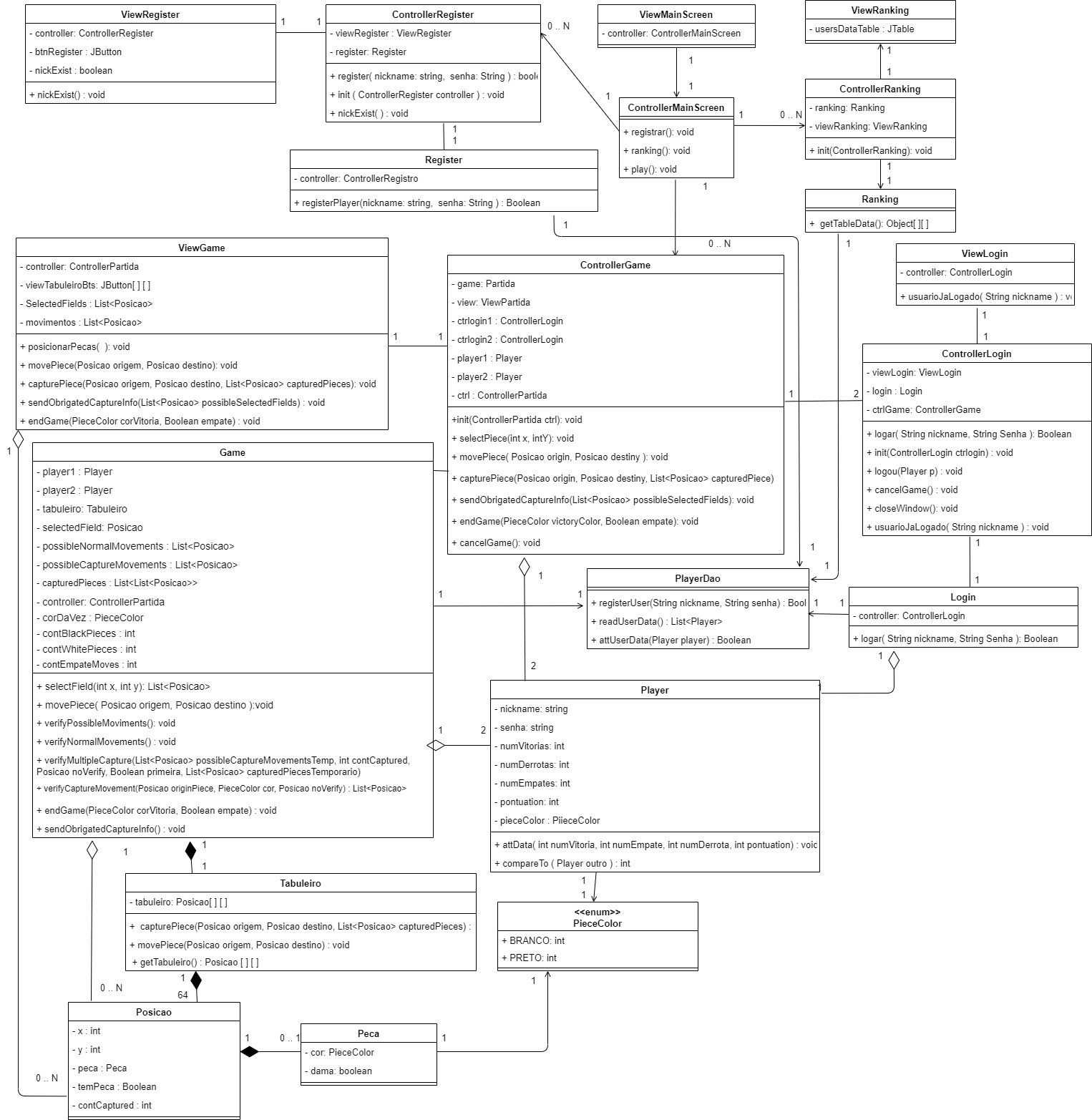


FIGURA 4: Diagrama de classes completo.

FONTE: Os autores (2020)

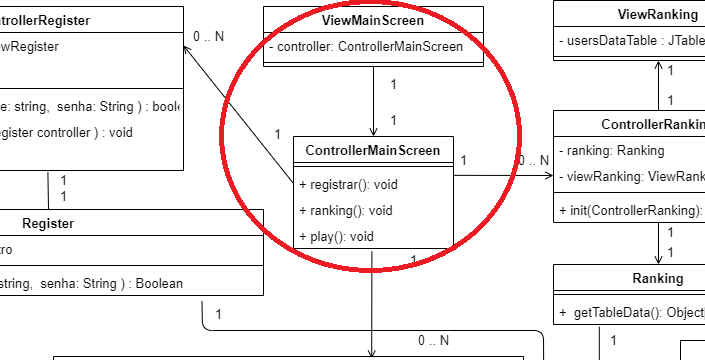


FIGURA 5: Parte central do diagrama.

FONTE: Os autores (2020)

Como nós desenvolvemos nosso sistema baseado nos padrões de projeto MVC, para funcionar como núcleo do sistema temos as classes ViewMainScreen e ControllerMainScreen representadas na figura 5. A partir delas podemos dividir o sistema em quatro ramificações que são as classes de registro, de ranking, a parte de login e a parte principal do sistema que são as classes da partida.

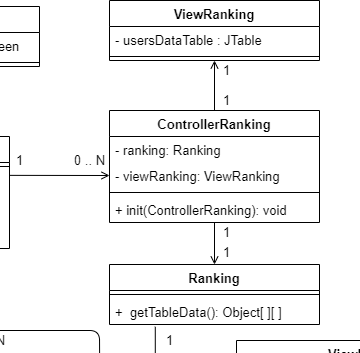


FIGURA 6: Classes de Ranking.

FONTE: Os autores (2020)

Na Ramificação representada na figura 6 do sistema estão as classes Ranking, ViewRanking e ControllerRanking. Por conta do corte do diagrama acima, não está aparecendo mas a classe Ranking se relaciona com a classe PlayerDao ( para um melhor entendimento olhar a figura 4 ) que é uma classe baseado no padrão de aplicação DAO ( Data Access Object ) ou objeto de acesso a dados ele é utilizado para separar as regras de negócio da aplicação das regras de acesso à base de dados, nesse caso o arquivo TXT, dessa forma deixando o código mais desacoplado e facilitando a reutilização uma vez que seja necessário usar um Banco de dados para aplicação caberia ao desenvolvedor simplesmente criar uma classe que herde PlayerDao e sobrescreva os métodos de acordo com a utilização desse banco.

Essas classes então serão responsáveis por extrair da base de dados do sistema as informações sobre as pontuações de cada jogador e exibir ao usuário ordenando do jogador com maior pontuação para o com menor.

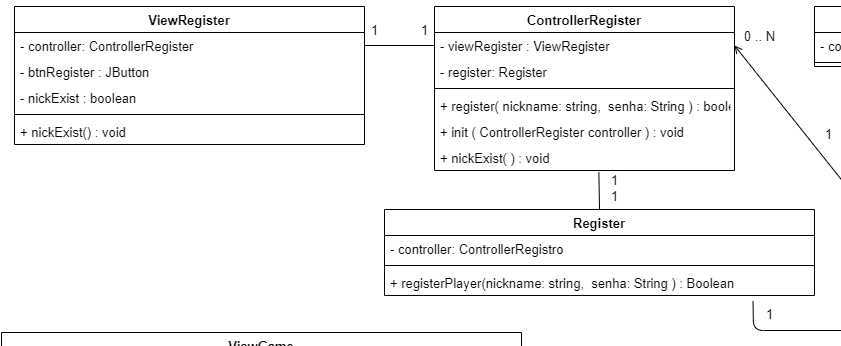


FIGURA 7: Classes de registro.

FONTE: Os autores (2020)

As classes de registro da figura 7 estão separadas nas classes Register, ViewRegister e ControllerRegister, além disso a classe Register se relaciona com a classe PlayerDao a qual já foi explicada no texto sobre a figura 6. Essas classes são responsáveis por validar e cadastrar os dados do jogador na base de dados, para ele poder realizar o login posteriormente, para se registrar o jogador precisa informar um nickname e senha além disso o nickname que ele informar não pode já estar registrado no sistema.

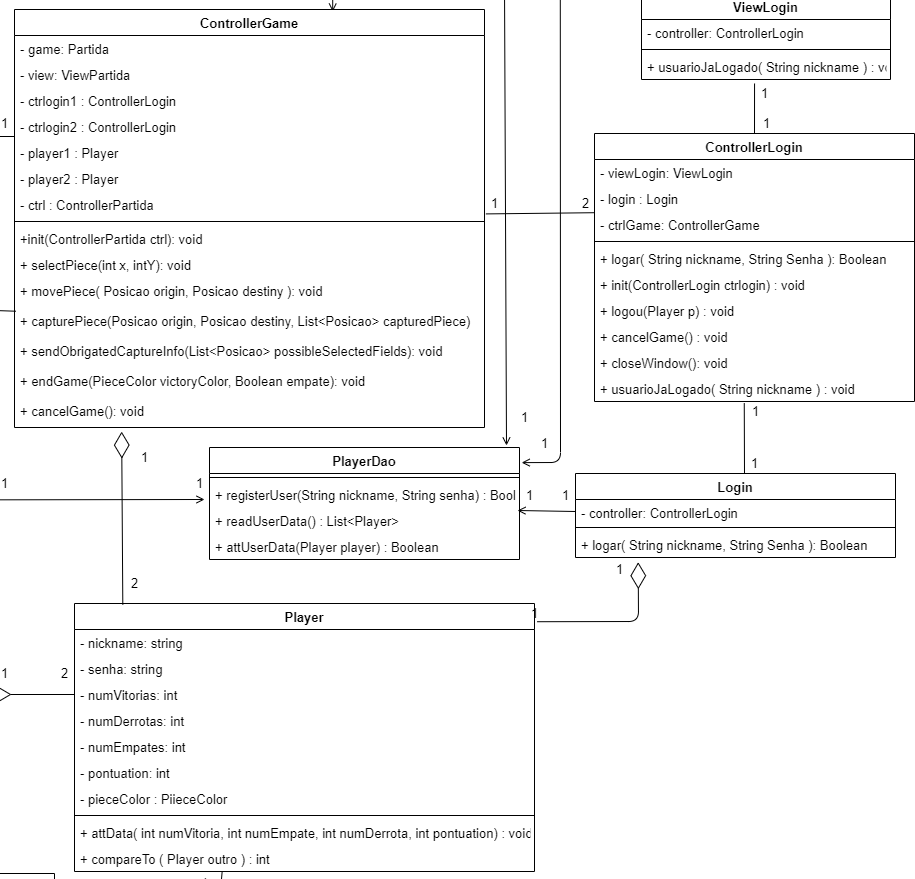


FIGURA 8: Classes de login.

FONTE: Os autores (2020)

Na figura 8 temos as classes utilizadas para realizar o Login dos jogadores antes de iniciar a partida. A partir do momento que a classe ControllerGame é chamada ela automaticamente instancia duas vezes a classe ControllerLogin que por sua vez chama as classes ViewLogin e Login na qual a classe Login também utiliza a classe PlayerDao para acessar os dados salvos no arquivo de texto, então quando o usuário informar dados validos a classe Login instancia a classe Player com os dados desse usuário e passa para a ControllerGame.

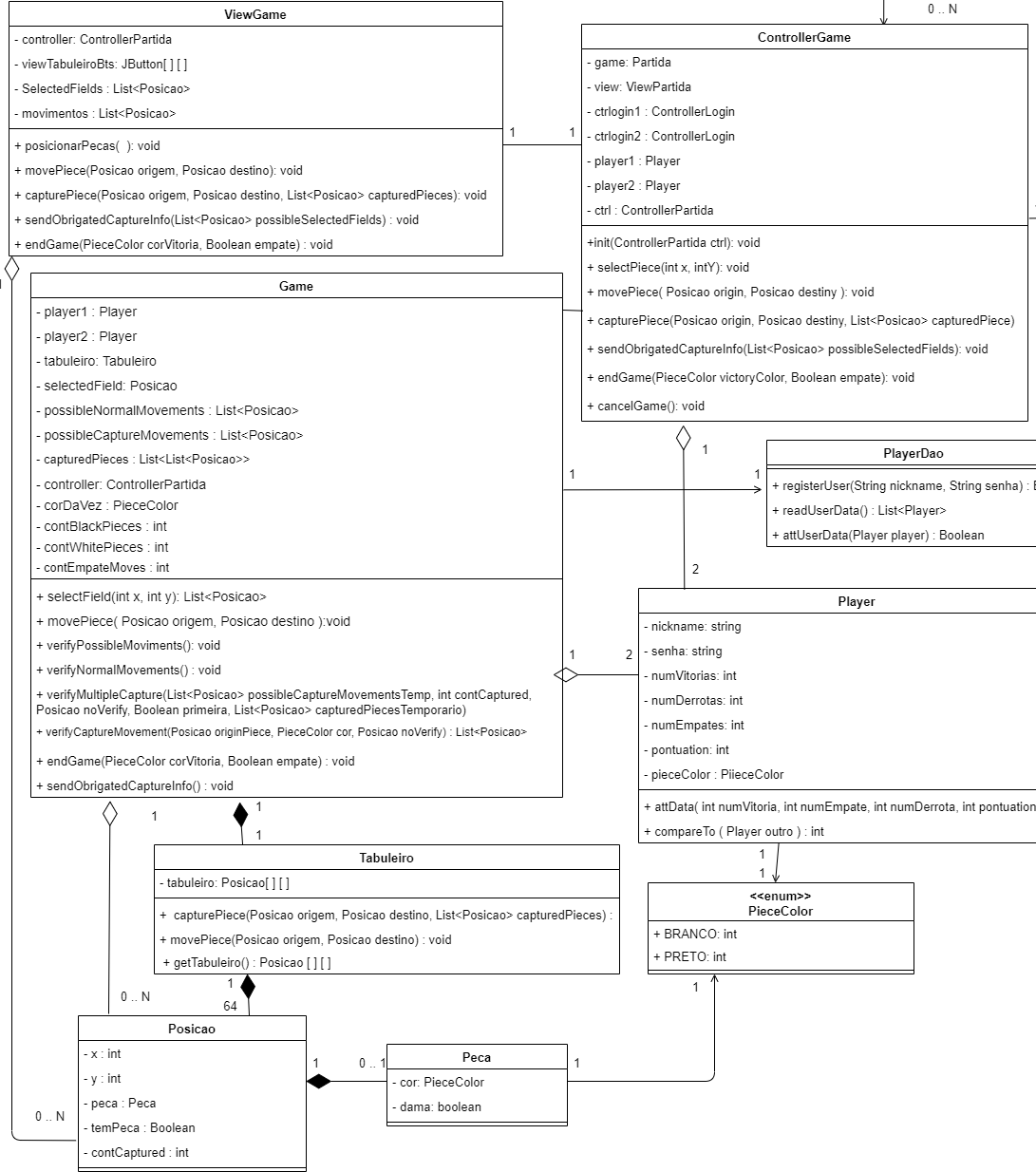


FIGURA 9: Classes da Partida.

FONTE: Os autores (2020)

Na figura 9 temos representadas as classes da Game, ViewGame e ControllerGame. Após os jogadores que jogarão a partida terem feito login a classe ControllerGame inicia o jogo chamando a classe *Game*.

A classe Game da partida contém todas as regras de negócio do jogo, é ela que realiza a validação dos movimentos da partida, organiza a vez de cada jogador, determina o fim da partida, atualiza as estatísticas de cada jogador e salva na base de dados através da classe PlayerDao.

A classe tabuleiro é instanciada pela classe Game, ela contém uma matriz de objetos da classe Posicao. Essa classe é utilizada para realizar as validações da classe Game.

A classe Posicao possui os atributos de coordenadas X e Y, um atributo peca que é um objeto da classe Peca e um atributo booleano *temPeca* para verificar se nessa posição tem uma peça.

A classe Peca tem dois atributos o primeiro contendo um valor da classe enumerada *PieceColor* para representar a cor dessa peça e um atributo booleano para verificar se a peça é uma dama.

2.5 PADRÃO DE PROJETO MVC

Para a organização do projeto optamos por utilizar o padrão *model-view-controller* ou simplesmente MVC que segundo GUEDES, “o princípio básico do MVC é a divisão da aplicação em três camadas: a camada de interação do usuário (***view***), a camada de manipulação dos dados *(****model****)* e a camada de controle (***controller***)”.

Na camada *view* como o próprio nome sugere temos a parte visual do sistema, ela não deve ter conhecimento das regras de negócio do software, sua única responsabilidade deve ser exibir as informações para o usuário capturar seus “*inputs*” e passar para a camada de controle.

Por sua vez, a camada *model* contém as regras de negócios da aplicação e o acesso à base de dados quando necessário, sua responsabilidade é receber os dados fazer os tratamentos específicos para eles e enviar para camada de controle não tendo assim nenhuma preocupação com a exibição dos dados. Para o nosso projeto além da camada model criamos uma sub camada DAO, que é responsável por todo o acesso a base de dados, nesse caso arquivo TXT, assim separando a lógica de acesso a base de dados das regras de negócio da aplicação.

Finalmente, a camada *controller* é um mediador entre a view e a model ela que transmite os dados de uma para a outra.

Dessa forma, utilizando o padrão MVC obtemos diversos benefícios como desacoplar a parte visual da parte das regras de negócio, assim facilitando futuras manutenções.

Porém usar o padrão MVC exige certa disciplina dos desenvolvedores uma vez que eles precisam tomar muito cuidado para não misturar códigos de uma camada com de outra, além disso usar o MVC em projetos pequenos pode deixá-los mais complexos do que necessitavam.

2.6 ALGORITMOS UTILIZADOS

Segundo Dasgupta, Papadimitriou e Vazirani, "algoritmos são procedimentos precisos, não ambíguos, mecânicos, eficientes e corretos". Em palavras mais coloquiais algoritmos são formulas exatas para resolver problemas.

A seguir iremos abordar alguns dos principais algoritmos utilizados no sistema e explicar um pouco seu funcionamento.

2.6.1 Lógica de exibição do tabuleiro

A seguir na figura 10 o código utilizado para exibição do tabuleiro:

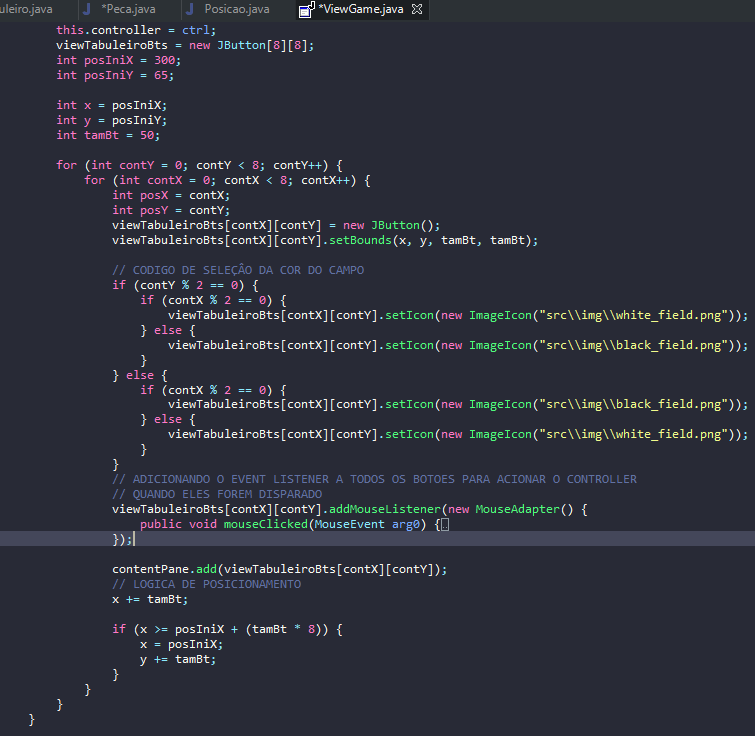


FIGURA 10: Algoritmo de exibição do tabuleiro.

FONTE: Os autores (2020)

Para a criação do tabuleiro nós criamos uma matriz 8X8 de componentes JButton da biblioteca Swing, as posições devem ser botões para poderem capturar o evento de clique do usuário, então após isso definimos as posições do eixo X e Y para iniciar o posicionamento das posições do tabuleiro e também a variável que guarda o tamanho em pixel de cada posição,

Após isso, com dois laços de repetição iteramos sobre a matriz de botões para posiciona-los através dos valores das variáveis x e y, e entramos em uma estrutura condicional para com base nos índices do botão definir se ele será uma posição de cor preta ou branca.

Finalmente, depois de definirmos o evento realizado no momento que o usuário clicar no botão, adicionamos o botão no objeto contentPane que no Swing é o componente que armazena todos os outros componentes da janela. Então acrescentamos ao valor da variável x o tamanho do botão, e verificamos com uma condicional se já foram posicionados 8 botões nessa linha, se já houverem o valor de x é redefinido para o valor inicial e é acrescentado ao valor de y o tamanho do botão.

2.6.2 Algoritmos de acesso aos dados no arquivo de texto

Como abordado anteriormente utilizamos o padrão de aplicação Dao para realizar o acesso aos dados no arquivo TXT, a seguir explicarei um pouco seu funcionamento.

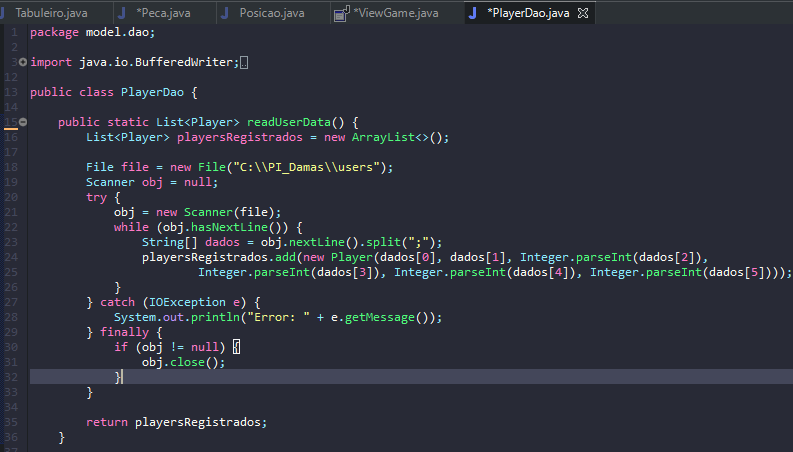


FIGURA 11: Método readUserData da classe PlayerDao

FONTE: Os autores (2020)

A classe PlayerDao possui 3 métodos baseados nas operações de um CRUD (*create, read, update e delete*) que são a readUserData para realizar a leitura de dados, a registerUser para realizar a criação de um registro de usuário e a attUserData para atualizar os dados do jogador.

Os métodos dessa classe são todos estáticos, ou seja, podem ser chamados sem a necessidade de instanciar a classe PlayerDao dessa forma deixando o código mais simples e legível.

Na figura 11 temos o método readUserData que cria uma lista de objetos da classe Player então ela abre o arquivo users, que foi criado pelo método registerUser, então ela lê cada linha desse arquivo separando os dados por ; , que foi o caractere escolhido para realizar separação dos dados, então ela adiciona para a lista playersRegistrados uma instancia da classe jogador com base nos dados lidos. Ao final ela retorna essa lista de jogadores para a classe que o chamou.

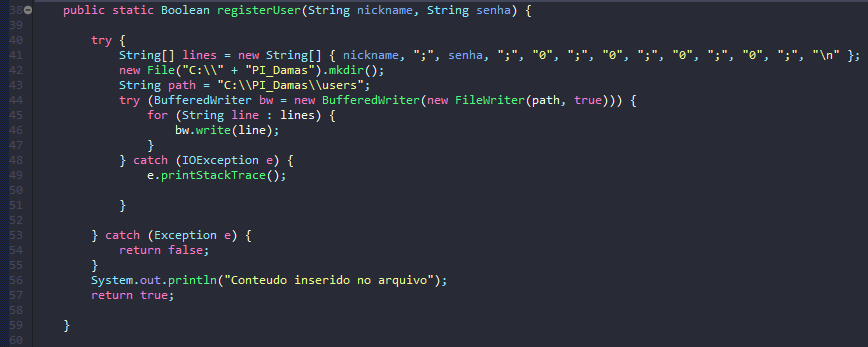


FIGURA 12: Método registerUser da classe PlayerDao

FONTE: Os autores (2020)

Na figura 12 temos o método registerUser que recebe como parâmetro o nickname e a senha do usuário, então dentro de uma estrutura try-catch ela salva os dados do usuário em um array de String separando-os com ; , vale ressaltar que no momento que o usuário insere seus dados de cadastro há uma tratativa que o impede de inserir o caractere ; , além do nickname e a senha que foram recebidos como parâmetro ele também salva como 0 os campos que representam a quantidade de vitórias, empates, derrotas e a pontuação do jogador respectivamente.

Após isso o método verifica se a pasta “PI\_Damas” e o arquivo de texto “user” dentro dela existem, se não existirem ele cria-os. Após isso ele salva os dados normalmente, caso haja algum erro o método retorna falso, se não houver retorna verdadeiro.

1. **CONCLUSÃO**

Dessa forma, através das metodologias e ferramentas apresentadas nesse artigo, eclipse para programação, biblioteca Swing do Java para criação das interfaces gráficas, a elaboração dos diagramas de classes e de casos de uso para ter uma visão melhor do escopo do projeto, o uso do padrão de projeto MVC para deixar o código mais desacoplado e facilitar a reutilização. Além de todos os conceitos de orientação a objetos vistos em aula possibilitaram o desenvolvimento e a finalização do projeto com todas as regras tradicionais do jogo de damas implementadas além de um sistema de login e registro de usuários com pontuações para cada jogador que podem ser acessadas por um ranking.

1. **REFERÊNCIAS**

### BELL, DONALD. Fundamentos básicos de UML: O diagrama de classes. 1 ed. IBM Corporation. 19, dezembro de 2016

DASGUPTA**,** SANJOY**.** PAPADIMITRIOU**,** CHRISTOS**,** VAZIRANI**,** UMESH.[***Algorithms***](http://highered.mheducation.com/sites/0073523402/index.html)*.* 1 ed. Editora:  McGraw-Hill Higher Education. 13, setembro de 2006.

GUEDES, MARYLENE. **O que é MVC?** 22, junho de 2020. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-mvc/>. Acesso em: 11, de novembro de 2020.

### JACOBSON, IVAN: [*Object Oriented Software Engineering: A Use Case Driven Approach*](https://www.amazon.com.br/Object-Oriented-Software-Engineering-Approach/dp/0201544350/ref=sr_1_3?qid=1585572037&refinements=p_27%3AIvar+Jacobson&s=books&sr=1-3). 1 ed. Editora: Addison Wesley. 1, julho de 1992.

### 

### Regras jogo de damas. 11, maio de 2008. Disponível em: https://brainking.com/GameRules >. Acesso em: 11, de setembro de 2020.