Tic Tac Toe

11052316 - Bryan Bialokur da Cruz

11040016 - Filipi de Carvalho Brabo

11036516 - Jonatas Duarte Souza

11072216 - Melissa Gabriela P. da Soledade Perrone

31 de Julho de 2018

Turma: A2 - Matutino

1 - Introdução

O projeto abordado foi o desenvolvimento de um Jogo da Velha (também conhecido como Tic-Tac-Toe) e de uma inteligência artificial para se jogar contra, utilizando conceitos de programação orientada à objetos.

2 - Descrição das classes

Classe "Interface":

É uma classe abstrata responsável pela interação com o usuário. Ela imprime informações e "cenas" do jogo no console (linha de comando) e coleta as entradas do usuário.

Algumas cenas:

- Tabuleiro: Imprime o estado atual do tabuleiro do jogo
- Tela inicial: Mostra as opções de jogo (jogador contra jogador, jogador contra computador ou computador contra computador);
- Tela de jogada: Informa quem é o atual jogador e a célula escolhida por ele. Faz também o tratamento de erros caso seja inserido algo não permitido para o valor de uma célula;

Classe "UserInput":

Faz a leitura e tratamento de erros das opções que o usuário precisa inserir dentro do jogo (através da classe "Interface")

Classe "Engine":

Classe abstrata responsável pelas funções do jogo. Tem métodos como:

- Limpar o tabuleiro, i.e., remover quaisquer valores de cada célula, deixando-as com valor nulo (símbolo: '-');

- Definir uma jogada, ou seja, alocar o valor 1 ou -1 (i.e., ‘X’ ou ‘O’) em uma célula do tabuleiro;
- Checar se uma célula está vazia;
- Checar se o jogo acabou.

Classe “Board”:

Classe que representa o tabuleiro. Seus métodos são:

- Modificar os valores da matriz que representa o tabuleiro.
- Acessar as células do tabuleiro;
- Modificar células do tabuleiro (não o valor da mesma, e sim a célula em si);
- Acessar o tamanho da matriz (tabuleiro).

Classe “Cell”:

Representa cada célula do tabuleiro. Controla os valores e os símbolos das células do tabuleiro

Interface “IPlayer”:

Interface utilizada para todos players do jogo. Possui métodos que devem obrigatoriamente implementados. Todo jogador deve ter:

- Um método que faça a escolha de uma jogada;
- Um método para definir e outro para retornar seu número;
- Um método que retorna o nome do jogador.

Classe “Player”:

Classe utilizada como “mãe” das outras classes “*Players*”, ela implementa os métodos que definem e retornam o número do jogador.

Classe “Human”:

Classe que herda a classe “Player”. Seus métodos são:

- Acessar e modificar o nome do(s) jogador(es);
- Fazer o jogador escolher um par de valores pelo console (linha de comando) e retornar essas coordenadas.

Classe “IA”:

Classe que herda a classe “Player”. Seus métodos são:

- Retornar a posição para a jogada da IA. Utiliza dois modos de estratégia, o primeiro retorna aleatoriamente a posição a ser inserida a célula. O segundo utiliza uma inteligência artificial que utiliza o método “minimax” para decidir a melhor jogada. A IA prevê uma certa quantidade de movimentos, quanto maior a dificuldade mais movimentos ela consegue prever, e assim, tomar uma melhor decisão.

- Acessar e modificar o nível de dificuldade da IA.
- Retornar o nome da IA (será sempre “Computador #”, onde ‘#’ é o número de jogador da IA)

Classe “Game”:

Monta o jogo com as configurações desejadas. Ela é quem cria todos os elementos necessários e faz a interação entre eles.

Classe “Main”:

Entrypoint do programa, lê as configurações inseridas pelo usuário e cria jogos com base nelas. Essas configurações são o modo de jogo (*singleplayer, multiplayer* e *IA vs IA)* e dificuldade (fácil, médio, difícil e impossível). Além disso, permite que o jogador escolha entre começar outro jogo ou sair do programa quando um jogo tiver acabado;

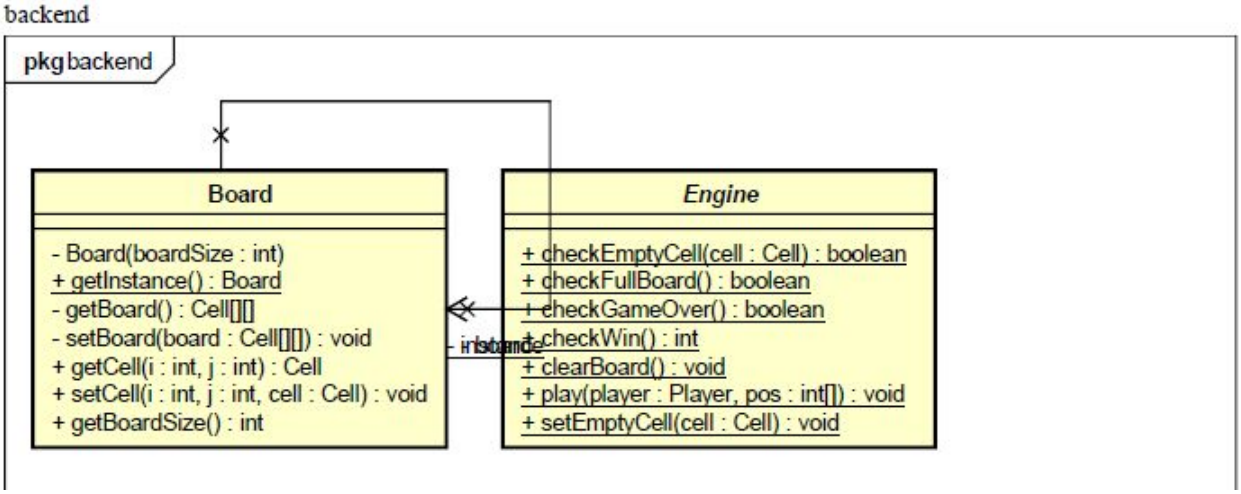

Figura 1: Classes do package backend

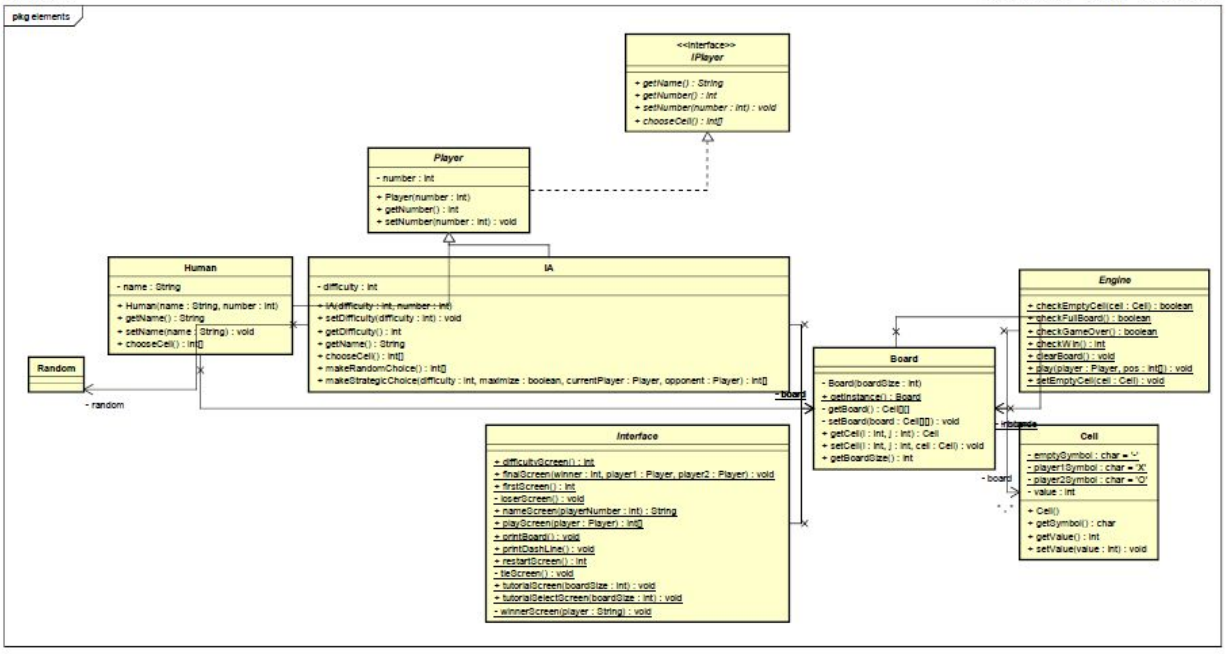

Figura 2: classes do package elements

exceptions

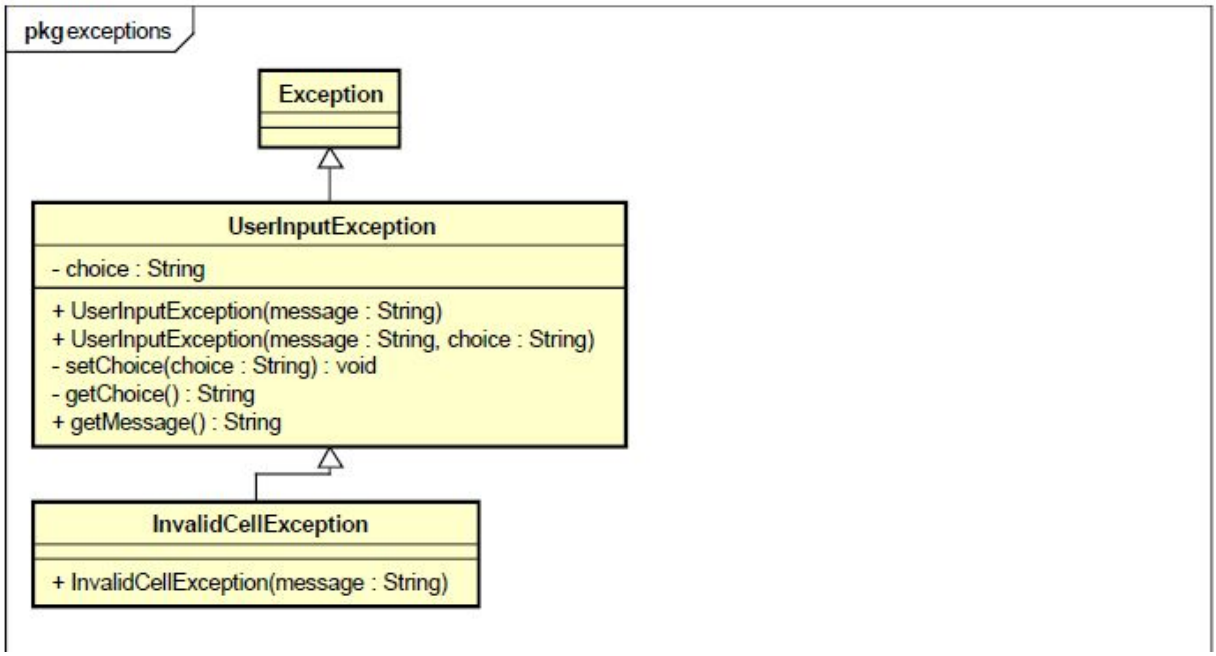

Figura 3: classes do package exceptions

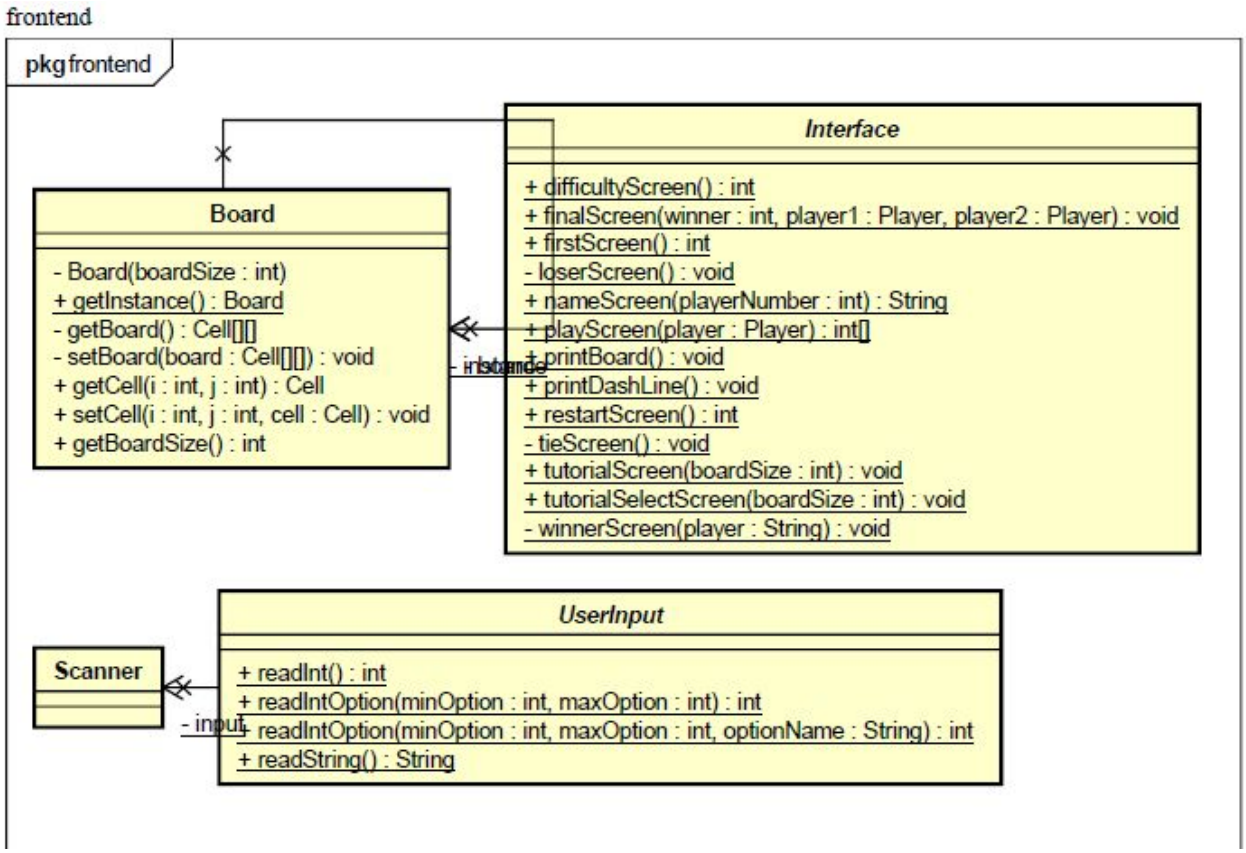

Figura 4: classes do package frontend

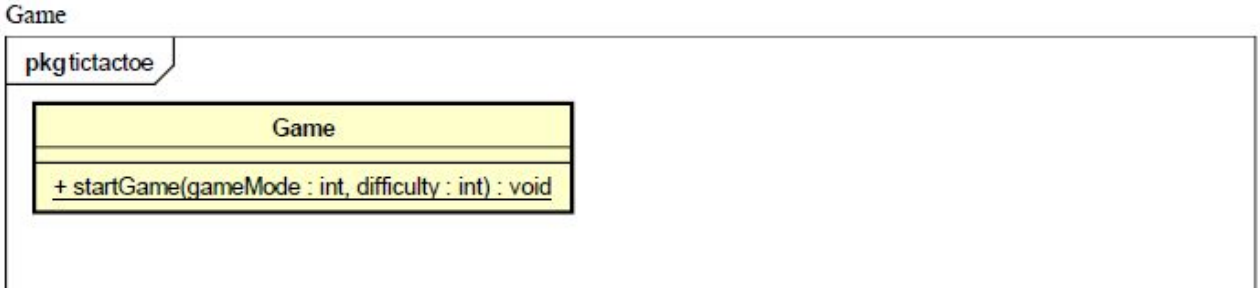

Figura 5: classe Game em package tictactoe

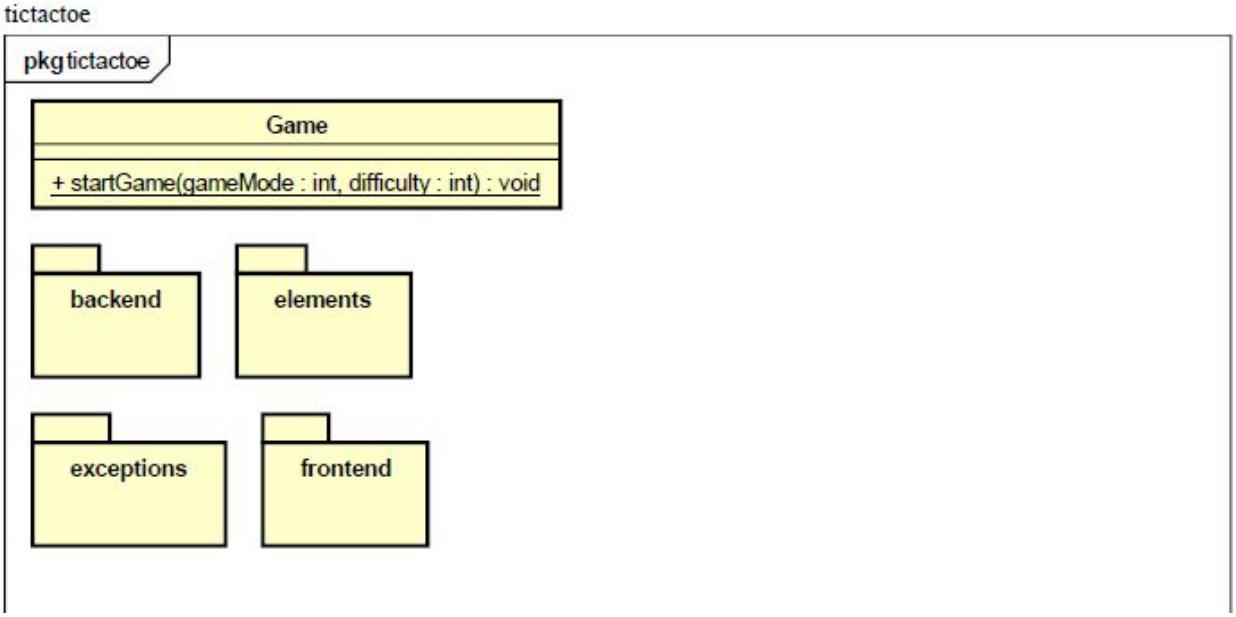

Figura 6: package tictactoe

3 - Conceitos de orientação a objetos aplicados

- **Encapsulamento:** o encapsulamento controla o acesso de atributos e métodos de uma classe. Isso protege a manipulação de dados de tal classe. Utilizou-se métodos e atributos private para que os mesmos não sofressem alterações inadequadas. Para modificar/acessar um atributo, utilizou-se métodos get/set.
- **Métodos get/set**: são métodos acessores e modificadores que reforçam o encapsulamento, pois impedem que os atributos de uma classe pública sejam diretamente acessados. Foram utilizados nas classes Board, Cell, Human, IA e Player.
- **Alta coesão:** a alta coesão ocorre quando um método executa uma única tarefa. A alta coesão foi aplicada em todos os métodos. Alguns exemplos: método “loserScreen” na classe Interface - imprime a tela informando que o jogador perdeu. Método “checkEmptyCell” em Engine - verifica se aquela célula já está ocupada (se já ocorreu uma jogada naquela posição).
- **Construtor:** construtores são métodos chamados quando a classe é instanciada. Foram utilizados nas classes Board, Cell, Human, IA e Player.

- **Sobrecarga de métodos:** é um tipo de polimorfismo. Permite a existência de vários métodos com o mesmo nome (identificador), mas com parâmetros diferentes. Permite que métodos com tarefas similares tenham o mesmo nome. Um exemplo: métodos “readIntOption” na classe UserInput. O primeiro possui os parâmetros minOption, maxOption e optionName. O segundo possui minOption e maxOption apenas, criando indiretamente um valor default para optionName.
- **Sobrescrita de métodos:** A sobrescrita ocorre quando copiamos um método de uma classe, normalmente indicamos com um “@Override”, para uma que a herde. Utilizamos nas classes *Human*, *IA* e *UserInputException*.
- **Interface e Herança:** Uma interface define como um objeto interage com outro. Ela define um tipo de dados que não pode ser instanciado. As interfaces não possuem construtores e podemos realizar implementações múltiplas. Já a herança é o princípio que permite a criação de classes através de uma classe-mãe (superclasse). A superclasse é a classe previamente criada. A partir dela, as subclasses (classes-filhas) herdam todas as suas características. Utilizamos a interface IPlayer, que é implementada pela classe Player, que por sua vez é da qual as classes Human e IA herdam.
- **Tratamento de Exceções:** é um evento que quebra o fluxo normal do programa. Criamos as seguintes exceções: 1. UserInputException: Erro na leitura de algum dado na interface; 2 - InvalidCellException: Erro na leitura das coordenadas de uma célula.
- **Design Pattern:** Utilizamos o padrão Strategy para o jogador, no qual não importa qual tipo de jogador seja instanciado para o jogo poder fluir. A mudança principal está no método “chooseCell” de cada tipo de jogador, o usuário insere a escolha dele no console e a IA faz a escolha baseada em funções pré-definidas.

4 - Participação de cada integrante do grupo

Todos integrantes ajudaram na realização do projeto e a divisão não foi tão explícita, pois utilizamos ferramentas para trabalharmos juntos, como git e chat por voz, e com isso acabamos tendo certa flexibilidade na hora de realizar o projeto, podendo cada integrante modificar (com certa justificação) uma parte do código ou relatório. Apesar disso, cada um de nós acabou por focar em algumas partes do projeto.

O Bryan focou mais na parte de estrutura/modelagem do projeto, o Jonatas focou na abstração e no desenvolvimento da IA, o Filipi também ficou com o desenvolvimento da IA e ajudou no relatório, a Melissa acabou por focar mais na parte da realização do relatório e todos ajudaram na parte de implementação e debug do código.