1. Dadas as classes abaixo, implementar os métodos int incluirDoInicio(Peca peca) e int incluirDoFim(Peca peca), conforme as seguintes regras:

Método incluirDoInicio

- Cada peça incluída é uma NOVA casa no tabuleiro.
- Se o tabuleiro estiver vazio, a primeira peça será automaticamente incluída e o retorno é igual a zero.
- Se houver pelo menos uma casa no tabuleiro, a peça pode ser incluída se uma das cabeças da peça recebida for igual a uma das cabeças da peça existente na casa do tabuleiro. Neste caso, a nova casa deve ser adicionada como a próxima casa, e o retorno deve ser igual a 1.
- Se houver mais de uma casa no tabuleiro, o método deve procurar, a partir do início, uma casa que se encaixe com a nova peça. Para isso, existem três possibilidades:
    - A primeira casa se encaixar, ou seja a sua cabeça esquerda ser igual a uma das cabeças da nova peça. Neste caso, a nova casa deve ser anterior à primeira casa e o retorno deve ser igual a 2.
    - A última casa se encaixar, ou seja a sua cabeça direita ser igual a uma das cabeças da nova peça. Neste caso, a nova casa deve ser posterior à última casa e o retorno deve ser igual a 1.
    - Algumas das casas intermediárias se encaixarem, ou seja a cabeça direita de uma casa e a cabeça esquerda da casa subsequente devem ser iguais às cabeças da nova peça. Neste caso, a nova casa deve entrar entre as duas casas encontradas, e o retorno deve ser igual ao tamanho do tabuleiro menos a quantidade de casas andadas menos 1.

Método incluirDoFim

- Cada peça incluída é uma NOVA casa no tabuleiro.
- Se o tabuleiro estiver vazio, a primeira peça será automaticamente incluída e o retorno é igual a zero.
- Se houver pelo menos uma casa no tabuleiro, a peça pode ser incluída se uma das cabeças da peça recebida for igual a uma das cabeças da peça existente na casa do tabuleiro. Neste caso, a nova casa deve ser adicionada como a primeira casa, e a casa atual fica sendo a última casa, e o retorno deve ser igual a 1.
- Se houver mais de uma casa no tabuleiro, o método deve procurar, a partir do fim, uma casa que se encaixe com a nova peça. Para isso, existem três possibilidades:
    - A primeira casa se encaixar, ou seja a sua cabeça esquerda ser igual a uma das cabeças da nova peça. Neste caso, a nova casa deve ser anterior à primeira casa e o retorno deve ser igual a 2.
    - A última casa se encaixar, ou seja a sua cabeça direita ser igual a uma das cabeças da nova peça. Neste caso, a nova casa deve ser posterior à última casa e o retorno deve ser igual a 1.
    - Algumas das casas intermediárias se encaixarem, ou seja a cabeça direita de uma casa e a cabeça esquerda da casa subsequente devem ser iguais às cabeças da nova peça. Neste caso, a nova casa deve entrar entre as duas casas encontradas, e o retorno deve ser igual ao tamanho do tabuleiro menos a quantidade de casas andadas menos 1.

```
enum CabecaPeca {
      BRANCO,
      PIO,
      DUQUE,
      TERNO,
      QUADRA,
      QUINA,
      SENA;
}
class Peca {
      CabecaPeca esquerda;
      CabecaPeca direita;
}
class CasaTabuleiro {
    Peca peca;
    CasaTabuleiro proximo;
    CasaTabuleiro anterior;
}
class Tabuleiro {
    CasaTabuleiro inicio;
    CasaTabuleiro fim;
    int tamanho;
    int incluirDoInicio(Peca peca) {
    }
    int incluirDoFim(Peca peca) {
    }
}
```

2. Dada a classe Tabuleiro acima já implementada, implementar a classe BurrinhoInteligente. Esta classe deve permitir dois jogadores jogarem o jogo conforme a dinâmica descrita na questão 1, e mais a lógica descrita a seguir.
    - Eles vão tirando aleatoriamente peças de um conjunto de peças que combinam todas as cabeças possíveis (o dominó) e decidem, a cada jogada, sem verem a peça que tiraram, se querem tentar encaixar a peça a partir do início ou do fim.
    - Peças encaixadas não podem mais ser tiradas, e peças não encaixadas voltam para o conjunto de onde é possível tirar peças.
    - O jogo acaba quando o conjunto de peças fica vazio.
    - A classe em questão deve suportar o modo “simulação”, onde não é necessário a página HTML perguntar se cada jogador quer procurar o encaixe do início ou do fim. No modo simulação, isto deve ser feito aleatoriamente.

3. Implementar uma página que realize um jogo de burrinho inteligente em modo simulação e em modo jogador.