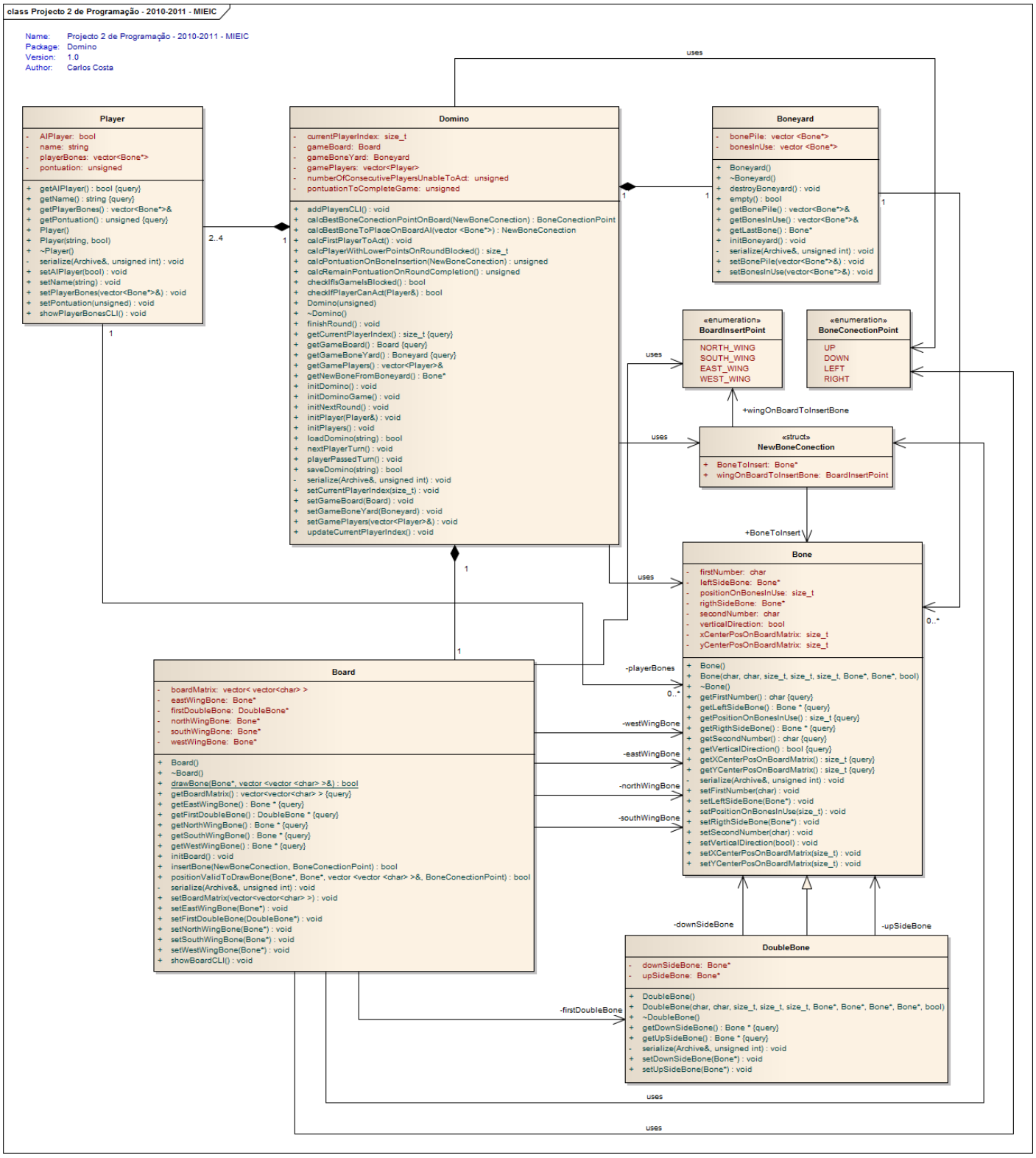
Dominó – Diagrama de classes



Grupo: M_ei09097

Carlos Miguel Correia da Costa

P.S.

Tal como é pedido no enunciado, apenas foram incluídas as descrições das classes e principais métodos.

Funções como os gets/sets, constructores e funções auxiliares de pequena importância não foram incluídas nesta "vista geral" do projecto 2.

Classe Domino

Classe que tratará de grande parte da lógica do jogo do dominó (incluindo AI), bem como de parte da interface CLI com o utilizador.

Funções principais:

- ✓ **bool loadDomino**(string filename) : Método que tratará do carregamento de um jogo de dominó guardado anteriormente num ficheiro de texto.
- ✓ **bool saveDomino**(string filename) : Método que tratará do backup dos dados de um jogo de dominó para um ficheiro de texto.
- ✓ **void initDominoGame**() : Método que inicializa um novo jogo de dominó.
- ✓ **void initDomino**() : Método auxiliar que inicializa o Board, Boneyard e Players.
- ✓ **void initPlayers**() : Método auxiliar que inicializa os Players do jogo.
- ✓ **void initPlayer**(Player& player) : Método auxiliar que inicializa um dado Player.
- ✓ **void initNextRound**() : Método que inicializa uma nova round do dominó (quando é feito dominó ou o jogo fica bloqueado e ainda não foi atingido a pontuação de terminação do jogo, começa-se uma nova round).
- ✓ **void finishRound**() : Método que tratará da terminação de uma round.
- ✓ **void addPlayersCLI**() : Método que adiciona os players que vão jogar através da interface CLI.
- ✓ Bone* **getNewBoneFromBoneyard**() : Método que vai "comprar" um Bone ao Boneyard.
- ✓ **void playerPassedTurn**() : Método que trata da passagem de vez de um utilizador quando este não tem Bones possíveis para jogar.
- ✓ **void calcFirstPlayerToAct**() : Método que determina que Player deve começar a round de acordo com as peças que recebeu do Boneyard.
- ✓ **void updateCurrentPlayerIndex**() : Método que trata da actualização de variáveis quando um player acaba a sua jogada.
- ✓ **void nextPlayerTurn**() : Método que trata da lógica associada a uma jogada do próximo utilizador.
- ✓ **bool checkIfPlayerCanAct**(Player& player) : Método que verifica se um utilizador pode jogar ou se é obrigado a passar (caso não tenha Bones adequados e não haja mais Bones no Boneyard).
- ✓ **bool checkIfIsGameIsBlocked**() : Método que verifica se o jogo está bloqueado (mais nenhum jogador tem Bones adequados para jogar).
- ✓ size_t **calcPlayerWithLowerPointsOnRoundBlocked**() : Método que determina quem é que ganhou a round quando este fica bloqueado (ganha quem tiver o menor número de pontos na sua hand). P.S. retorna índice que o jogador vencedor ocupa no vector<Player> do Domino.
- ✓ **unsigned calcRemainPontuationOnRoundCompletion**() : Método que trata da atribuição dos pontos dos adversários ao jogador que ganhou a round fazendo "dominó".
- ✓ **unsigned calcPontuationOnBoneInsertion**(NewBoneConection newBoneConection) : Método que calcula a pontuação resultante da inserção de um dado Bone no Board.
- ✓ NewBoneConection **calcBestBoneToPlaceOnBoardAI**(vector <Bone*> PlayerBonesAI) : Método que calcula qual o melhor Bone a jogar tendo em conta o Board e as regras que foram sugeridas no enunciado para a AI a ser implementada para o jogo de dominó contra o computador.
- ✓ BoneConectionPoint **calcBestBoneConectionPointOnBoard**(NewBoneConection newBoneConection) : Método que determina a melhor posição para inserir um Bone em relação a outro no Board (a ser usado na AI).

Classe Player

Classe que conterá a informação relativa a um dado player do jogo do dominó.

Caso este seja um player "virtual", a flag AIPlayer deve estar a "true".

Funções principais:

- ✓ `void showPlayerBonesCLI()` : Método que tratará da visualização da "hand" de Bones que um player possui, na interface CLI.

Classe Boneyard

Classe que fará a gestão da "compra" dos Bones durante o jogo.

P.S. Os Bones são alocados dinamicamente com o "new", por isso para evitar memory leaks e seguir a boa prática de programação de "quem aloca, dealoca" (memória), é mantido um vector auxiliar que tem os Bones que estão em uso no Board, apesar de já não estarem disponíveis para "compra". Desta forma os loads e saves ficam mais simplificados e evita-se muitos bugs relacionados com pointers inválidos, etc. (parte-se do principio que não há nem eliminação nem movimento dos Bones neste vector auxiliar…)

Funções principais:

- ✓ **void initBoneyard()** : Método que trata da inicialização do Boneyard (criação dos Bones).
- ✓ **void destroyBoneyard()** : Método que trata da dealocação da memória dos Bones que foi feita no initBoneyard().
- ✓ **bool empty()** : Método que retorna true caso não haja mais Bones para "compra".
- ✓ Bone* **getLastBone()** : Método que retorna um apontador para o último Bone que está na bonePile. Caso não exista nenhum retorna NULL.

Classe Board

Classe que trata da gestão da board do jogo, fazendo parte da lógica de verificação das inserções de Bones e mantendo uma matrix de char com o desenho dos Bones que já foram jogados.

Funções principais:

- ✓ **void initBoard()** : Método que trata da inicialização do board (em "branco" com os rebordos laterais).
- ✓ **void showBoardCLI()** : Método que mostra o board na interface CLI.
- ✓ **bool insertBone**(NewBoneConection newBoneConection, BoneConectionPoint boneConectionPoint) : Método que trata da inserção de um novo bone no Board. Se a inserção for válida faz o drawBone na boardMatrix, senão retorna false indicando que a inserção é inválida.
- ✓ **bool positionValidToDrawBone**(Bone* boneInBoardToCheck, Bone* boneToInsert, vector <vector <**char**> >& charMatrix, BoneConectionPoint boneConectionPoint) : Método auxiliar que verifica se o "desenho" de um determinado Bone na boardMatrix é possível.
- ✓ **static bool drawBone**(Bone* bone, vector <vector <**char**> >& charMatrix) : Método que desenha um determinado Bone na boardMatrix. Retorna false caso o "desenho" não seja possível.

Classe Bone

Classe que contém as informações relativas a uma peça de dominó.

Para facilitar os saves e loads e a possível reconstituição do jogo, tem a posição que ocupa no vector de bonesInUse no Boneyard e a posição central que ocupa na boardMatrix (x, y).

Para além disso cada peça tem apontadores para as peças que estão a ela conectadas e tem uma flag a indicar se foi desenhada na horizontal ou vertical na boardMatrix.

Desta forma, é possível reconstituir um jogo, numa interface completamente diferente sem ser a CLI (por exemplo numa GUI).

Como esta classe não tem funções importantes (para além dos gets, sets e construtores), ficam as suas variáveis:

Variáveis principais:

- ✓ **char** firstNumber : Primeiro número da peça.
- ✓ **char** secondNumber : Segundo número da peça.
- ✓ size_t positionOnBonesInUse : Posição que a peça ocupa no vector de bonesInUse no Boneyard.
- ✓ size_t xCenterPosOnBoardMatrix : Posição no eixo dos xx que a peça ocupa na boardMatrix (matrix de chars que tem as peças jogadas).
- ✓ size_t yCenterPosOnBoardMatrix : Posição no eixo dos yy que a peça ocupa na boardMatrix (matrix de chars que tem as peças jogadas).
- ✓ Bone* leftSideBone : Bone que está conectado do lado esquerdo (vendo a peça na horizontal).
- ✓ Bone* rigthSideBone : Bone que está conectado do lado direito (vendo a peça na horizontal).
- ✓ **bool** verticalDirection : Flag que indica a direcção com que o Bone foi desenhado na boardMatrix.

Classe DoubleBone

Subclasse de Bone que corresponde a uma especialização do Bone para as peças duplas.

Desta forma evita-se ter nas classes Bone, os 2 apontadores para Bone* extra, que correspondem aos 2 sítios extra m que as peças DoubleBone permitem inserção de Bones.

Variáveis principais:

Para além das variáveis que herdou de Bone tem:

- ✓ Bone* upSideBone : Bone que está conectado da parte de cima da peça (vendo a peça na horizontal).
- ✓ Bone* downSideBone : Bone que está conectado da parte de baixo da peça (vendo a peça na horizontal).