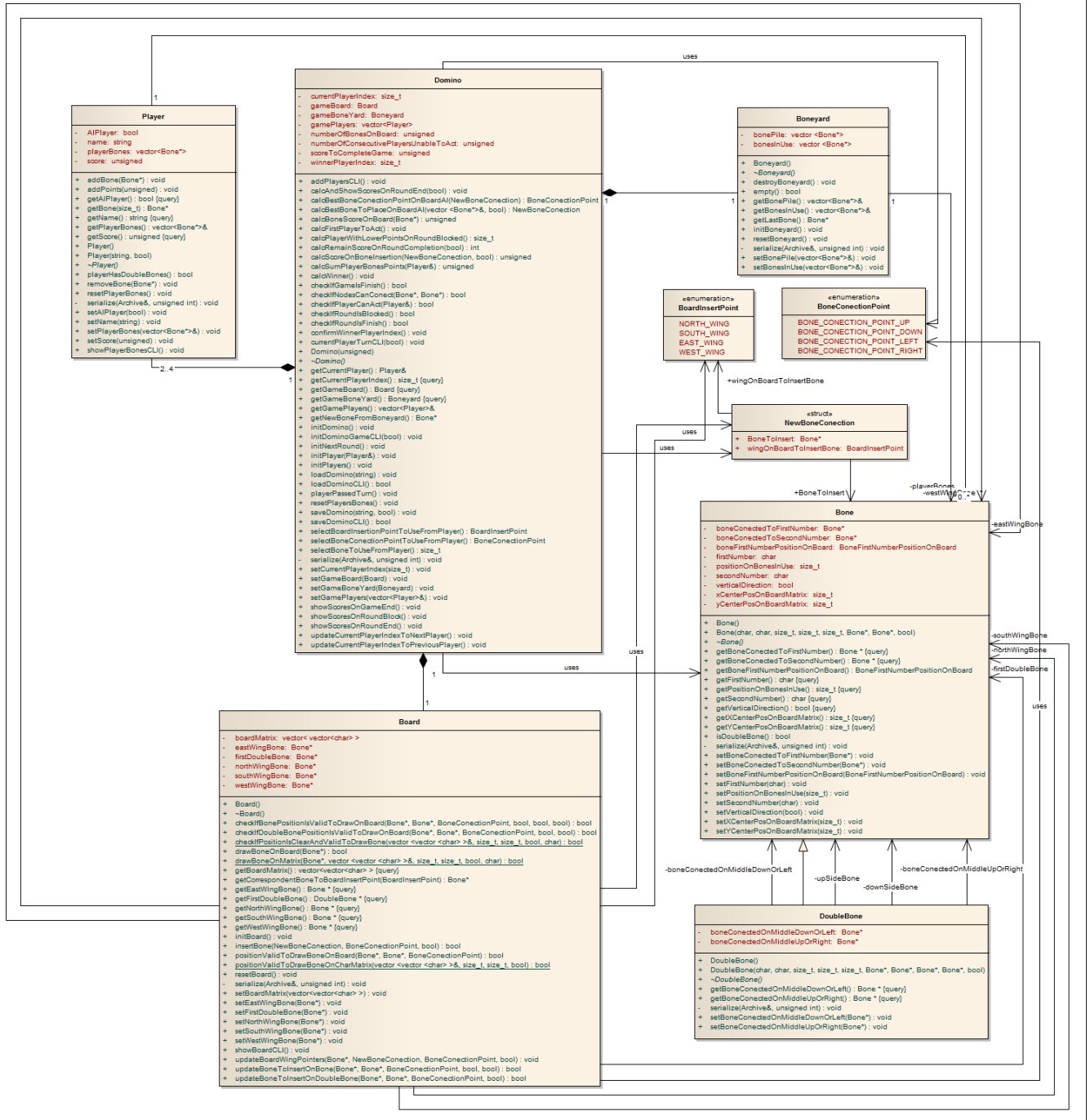




Dominó All fives

class Projecto 2 de Programação - 2010-2011 - MIEIC

Name: Projecto 2 de Programação - 2010-2011 - MIEIC
Package: Dominó
Version: 1.0
Author: Carlos Costa



Grupo: M_ei09097

Carlos Miguel Correia da Costa

Entregue no dia 21 / 05 / 2011

i. Introdução

A presente aplicação tem como objectivo simular um jogo de Domino All Fives, entre vários jogadores, podendo esses jogadores serem ou não controlados por uma AI, (de acordo com o pedido de quem está a usar a aplicação, no início do jogo).

Tal como foi especificado, o jogo pode ter entre 2 a 4 jogadores, sendo que no início de cada round são distribuídas 7 peças a cada um.

Começa a jogar quem tiver o maior DoubleBone do jogo ou quem tenha a peça de maior pontuação.

O primeiro DoubleBone é o único que em que se pode colocar Bones nos seus 4 lados, sendo que os restantes estão sempre colocados de forma perpendicular aos restantes Bones no Board (e só podem ter 2 ligações).

O objectivo do jogo é arrecadar o maior número de pontos possível, sendo que os pontos são calculados somando as extremidades das possíveis conexões de Bones no Board (no máximo 4, após a inserção do primeiro DoubleBone).

Os pontos apenas são válidos se forem múltiplos de 5.

O jogo termina quando 1 dos jogadores, após terminar uma round tiver atingido a pontuação mínima de conclusão do jogo, que é definida no início.

Como tal, poderão haver várias rounds até se atingir essa pontuação.

No final de cada round, o vencedor adiciona à sua pontuação a soma dos pontos dos adversários arredondada para o múltiplo de 5 mais próximo.

Caso a round fique bloqueada (nenhum dos jogadores tem Bones válidos e não há mais Bones disponíveis no Boneyard), o utilizador que tiver menos pontos na sua hand é que ganha os pontos extra, e a esses pontos extra tira os pontos que tem na sua hand.

Todas as funcionalidades pedidas foram implementadas, (incluindo a parte de valorização – AI), e para facilitar o jogo quando o número de pontos para terminar for elevado, foi acrescentada a possibilidade de guardar o jogo, para mais tarde poder continuar.

Esta funcionalidade foi implementada de 2 maneiras.

Na mais básica, apenas guarda as informações dos jogadores, (incluindo pontuação), em como as definições do jogo.

Numa mais avançada, foi usado a biblioteca do boost para proceder ao backup de todo o jogo, podendo continuar mais tarde com todas as definições que tinha antes, (incluindo, claro está o Board e Bones de cada player).

Nesta funcionalidade extra, é verificada a existência do ficheiro onde se vai guardar e informa-se o utilizador que quer fazer overWrite ou não.

ii. Estrutura e implementação de classes

A implementação do simulador de Dominó foi feita tendo em mente a interdependência entre cada uma das classes de forma a otimizar a performance, mas também foi tido em conta a modularidade e independência de cada um dos componentes de forma a tornar o código mais robusto e facilitar o seu upgrade em termos de funcionalidades.

Toda a implementação foi feita com um estilo de codificação que facilita a leitura e debug do código desenvolvido, em que para além da nomenclatura usada no nome das variáveis é fornecida a documentação de cada uma das funcionalidades e estruturado o código em secções, de forma a permitir uma pesquisa mais eficiente.

Para facilitar a consulta e pesquisa de alguma das funcionalidades implementadas, foi gerado a documentação Doxygen, que para além de ser uma forma flexível de pesquisar pelo código, proporciona muitos diagramas elucidativos de onde e como cada um dos subcomponentes está a ser usado e qual o seu relacionamento com a restante implementação.

O diagrama de classe principal está na capa do relatório, mas na documentação estão incluídos diagramas muito mais completos e muito mais detalhados.

Relativamente às classes implementadas, passo a uma breve descrição da funcionalidade de cada uma delas:

Classe Domino

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

É onde a verificação de grande parte está implementada, e onde foi incluída a AI do jogo.

Nesta classe é armazenada toda a informação necessária para o gerenciamento de um jogo de dominó.

Ou seja, é onde estão armazenados os jogadores, o Board, o Boneyard, os Bones e todas as definições relativas ao jogo.

Também é nesta classe que estão implementados os loads e saves (com ou sem o uso do Boost).

Classe Player

Classe que contém a informação relativa a um dado utilizador do jogo do dominó.

Caso este seja um jogador “virtual”, a flag AIPlayer deve estar a “true”, para que a lógica do jogo faça com que seja seleccionado o melhor Bone a inserir de acordo com as heurísticas sugeridas.

Esta flag, na CLI, é definida aquando da criação do Player, no início do jogo.

Classe Boneyard

Classe que trata da gestão dos Bones do dominó.

É aqui onde está implementado a sua inicialização e manutenção em memória, através de alocação dinâmica com “new”.

Disponibiliza também a API necessária para a “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 associados aos Players, apesar de já não estarem disponíveis para “compra”.

Desta forma aquando da destruição do Boneyard é garantido que não existem Bones por libertar de memória.

Classe Board

Classe que trata da gestão do tabuleiro de 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.

Grande parte das regras relativas à forma como os Bones podem ser inseridos no Board estão implementadas aqui.

São mantidos apontadores para os Bones onde é possível inserir novas peças de forma a aumentar a performance e facilitar a verificação da validade das inserções bem como a pontuação que essas inserções de Bones podem obter.

O cálculo das pontuações é feito antes da inserção de forma a ser utilizado pela AI para determinar qual a peça que tem a melhor pontuação. Assim evita-se redundância de código com funcionalidade semelhante.

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 numa API diferente, 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, bem como uma enumeração que indica a posição do primeiro número da peça.

Desta forma, é possível reconstituir um jogo, numa interface completamente diferente sem ser a CLI (por exemplo numa GUI), e para além disso torna o código mais extensível e facilita também toda a lógica de verificação da validade de inserções de Bones, porque cada Bone possui toda a informação necessária para saber onde está e o contexto á sua volta.

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 em que as peças DoubleBone permitem inserção de Bones.

Classe Exceptions

Classe de onde todas as excepções do programa derivam.

As excepções foram feitas para tratar de inputs inválidos por parte do utilizador.

Um desses casos está relacionado com os loads e saves (por exemplo quando o utilizador dá um nome a dar ao ficheiro de output que já está associado a outro ficheiro e que iria provocar overWrite, ou quando fornece o nome de um ficheiro para carregar que não existe...).

SubClasses:

- | | |
|----------------------------------|----------------------------|
| 1) FileDoesntExistException | 5) EmptyContainerException |
| 2) FileAlreadyExistException | 6) CorruptedFileException |
| 3) ElementAlreadyExistsException | 7) ExOutOfRange |
| 4) ElementDoesntExistsException | 8) ExNoInterval |

Notas acerca da implementação

Para o programa correr da forma como foi desenhado, a consola deve ter um buffer e uma janela de pelo menos 170 caracteres de largura e de 65 ou mais de altura (para permitir scroll, 1000 é o aconselhado para o buffer).

Isso pode ser feito nas propriedades da consola (ou nas predefinições, para se manter entre execuções do programa), clicando com o lado direito do rato na parte de cima da janela da consola.

No ficheiro utils.h e .cpp com o namespace utils, estão implementadas um conjunto de funções genéricas para a interacção com o utilizador usando a interface CLI.

No ficheiro defs.h estão as definições principais do programa que podem ser parametrizadas aqui.

Aqui se inclui o tamanho do Board, que se adapta de acordo com os defines lá feitos.

Para usar ou não o Boost para os loads e saves é só comentar (desactiva) ou descomentar (activa) o define:

#define USE_BOOST 1 (usa o Boost)

//#define USE_BOOST 1 (não usa o Boost)

Para além disso, é preciso compilar o Boost primeiro, porque a biblioteca de serialização não pode ser usada directamente a partir do headers <...>.hpp.

Para compila-lo com o Visual Studio ou Eclipse é só seguir o tutorial em:

http://www.boost.org/doc/libs/1_46_1/more/getting_started/windows.html

http://www.boost.org/doc/libs/1_46_1/more/getting_started/unix-variants.html

Depois é preciso adicionar a directoria

C:\Program Files\boost\boost_1_46_1, aos includes do compilador e

C:\Program Files\boost\boost_1_46_1\stage\lib, ao linker.

No Eclipse, ao contrário do Visual Studio (que tem auto_linker), é necessário adicionar que bibliotecas se vai usar. Neste caso usei:

boost_serialization-mgw45-mt-s-1_46_1

<http://theseekersquill.wordpress.com/2010/08/24/howto-boost-mingw/>

iii. Conclusão

A aplicação faz o que foi pedido na especificação e fá-lo duma maneira intuitiva para o utilizador final, de forma que apesar de usar uma interface CLI, é fácil de usar e bastante informativa relativamente ao que está a fazer na sua lógica interna.

Foram implementadas todas as especificações pedidas no enunciado incluindo todos os parâmetros sugeridos para a valorização (AI).

Ou seja, a AI primeiro tenta encontrar a peça que permite arrecadar o maior número de pontos possível.

Caso nenhuma delas forneça pontos ao inserir, tenta inserir o seu maior DoubleBone.

Se não tiver doubleBones, tenta inserir a peça de maior pontuação, visto que assim caso a round seja ganha por um adversário, ao jogar as peças de maior pontuação primeiro, estará a dar menos pontos extra a quem ganhar essa round.

iv. Apêndices

Visto que os screenShoots do programa em execução são relativamente grandes e à muitos casos particulares a testar, vou apenas apresentar uma “sessão típica do programa”.

No entanto, todos os parâmetros que foram especificados no enunciado foram testados e estão a funcionar.

Para além disso a aplicação é robusta contra inputs inválidos do utilizador, mostrando o que o utilizador tem que inserir se o input não for o esperado.

Na interface CLI pedida, esta aplicação tem 2 releases possíveis.

Uma usando o Boost que guarda / carrega o estado do jogo completo e outra que guarda apenas parte.

Para facilitar os testes são fornecidos os binários de ambas as versões no modo release.

Como em termos de interface são iguais, apenas varia o load/save (com as implicações anteriormente referidas que isso tem após o load – ver descrição da classe Domino), vou usar a que não usa o Boost, porque não tem dependências externas. Mas ambas estão a funcionar, e têm interface igual.

No início é apresentado o menu inicial da aplicação, que permite iniciar um novo jogo ou carregar um antigo (bem como sair da aplicação).

```
##### PROGRAMACAO - PROJECTO 2 #####
>>>          DOMINO - ALL FIVES          <<<
#####
1 - Iniciar novo jogo
2 - Carregar jogo
0 - Sair

>>> Opcao:
```

Se for escolhido para carregar o jogo é apresentada a seguinte mensagem, que pede o nome do ficheiro a carregar.

Se o nome do ficheiro existir carrega de lá o jogo. Senão pergunta ao utilizador se quer introduzir novamente o nome do ficheiro de onde quer carregar o jogo. Se o utilizador responder S pergunta novamente pelo nome do ficheiro senão, se responder N, volta ao menu inicial.

```
>>> Introduza o nome do ficheiro onde esta guardado o jogo que quer carregar:
>>> Introduza o nome do ficheiro onde esta guardado o jogo que quer carregar: nao existe
O nome do ficheiro que introduziu nao existe!
Prima ENTER para continuar...
Deseja tentar novamente com outro nome (S/N)? S
```

Se for escolhido iniciar um novo jogo, é perguntado quantos utilizadores o jogo terá e de seguida pergunta-se o nome e se é para usar AI ou não (em cada um).

```
>>> Quantos jogadores pretende que o jogo tenha: 2

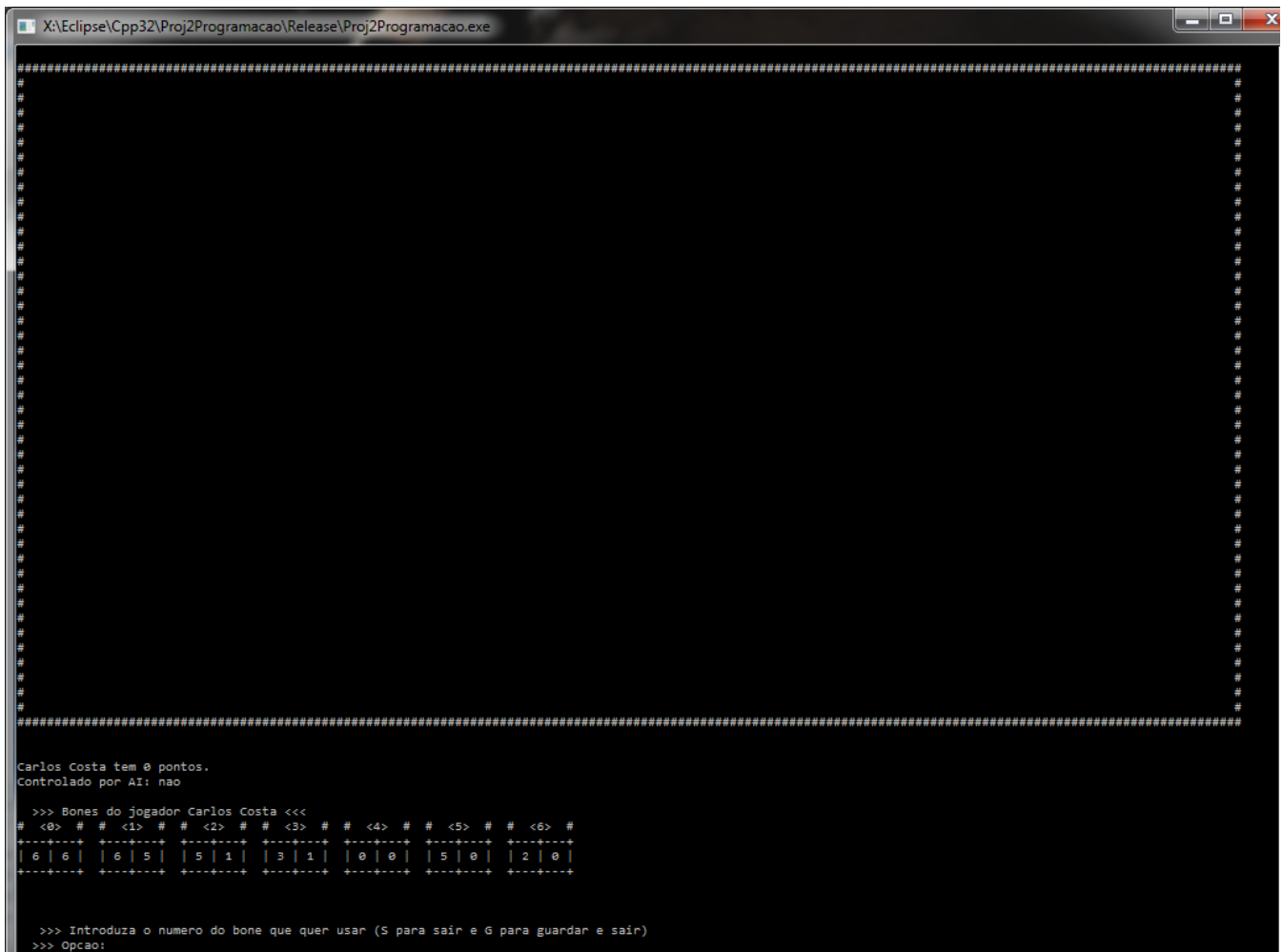
>>> Introduza o nome do jogador 0: Carlos Costa
>>> Pretende que o player seja controlado por AI (S/N): n

>>> Introduza o nome do jogador 1: Jose Costa
>>> Pretende que o player seja controlado por AI (S/N): s
```

Depois é perguntado qual o número de pontos mínimo que os jogadores devem fazer para ganhar o jogo.

```
Numero de pontos mínimo para terminar o jogo:
```

De seguida, o jogador que tiver o maior doubleBone ou o Bone com mais pontos começa a jogar:



```
>>> Bones do jogador Carlos Costa <<<
# <0> # # <1> # # <2> # # <3> # # <4> # # <5> # # <6> #
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 6 | 6 | | 6 | 5 | | 5 | 1 | | 3 | 1 | | 0 | 0 | | 5 | 0 | | 2 | 0 |
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+

>>> Introduza o numero do bone que quer usar (5 para sair e 6 para guardar e sair)
>>> Opcao:
```

A interface de inserção de Bones no Board é bastante intuitiva.

Primeiro selecciona-se o Bone que se quer inserir dando o índice que aparece sobre o desenho dele.

Depois selecciona-se em que “posição de inserção” se quer introduzir o Bone (as posições apresentadas na tabela são relativas ao utilizador, por isso se o utilizador quiser que east_wing seja para a esquerda em vez de ser para a direita... Isso é da responsabilidade do utilizador, tem é que ser consistente com a escolha que faz para os primeiros Bones que são inseridos nessas alas de inserção – e que inicializam os pointers respectivos no Board).

Depois de seleccionar um dos 2 ou 4 sítios possíveis de inserção (dependendo se já há um doubleBone ou não), pede-se que se insira a direcção em que o Bone seleccionado irá ter no Bone que já está no Board.

Na AI, esta ultima parte foi deixada para o utilizador visto que assim dá mais liberdade ao jogador para organizar os Bones no ecrã.

Ou seja, a AI selecciona o Bone e a posição de inserção e o utilizador define a direcção de inserção do Bone.

Depois de estas 3 informações serem recolhidas, a lógica da aplicação verifica se é possível inserir o Bone especificado, na posição e direcção fornecida, e caso seja inválido mostra uma mensagem de erro e pede que introduza novamente o Bone. Se for válido, mostra qual é a soma dos pontos na board e se os pontos forem múltiplos de 5 mostra quantos novos pontos é que ganhou.

Ter em atenção que a verificação se a round ou o jogo já terminou ou ficou bloqueado é feita automaticamente, bem como a verificação se um Player tem ou não Bones possíveis para inserção.

Ou seja, se um player não tiver Bones válidos para inserir, são automaticamente comprados Bones ao Boneyard (até estar vazio ou encontrar uma peça válida).

Portanto, quando se é dito ao utilizador para seleccionar um Bone, já se sabe que pelo menos um dos Bones na sua hand dá para inserir.

Para além disso, se o player não tiver Bones válidos e o Boneyard estiver vazio, é passada a vez automaticamente. E se todos os utilizadores passarem a vez a round é dada como bloqueada e termina, determinando-se o player que tem menos pontos na hand para saber quem é que ganha a round e fica com os pontos dos adversários.

Eis alguns exemplos do que foi referido:

Nota: ter em atenção que o resultado de uma inserção apenas é mostrado na vez do player a seguir, por isso o resultado das acções seguintes poderão estar nas imagens a seguir aquela em que se executa o exemplo.

Caso do primeiro Bone seleccionado por um utilizador:

```

X:\Eclipse\Cpp32\Proj2Programacao\Release\Proj2Programacao.exe
#####

Carlos Costa tem 0 pontos.
Controlado por AI: nao

>>> Bones do jogador Carlos Costa <<<
# <0> # # <1> # # <2> # # <3> # # <4> # # <5> # # <6> #
+---+ +---+ +---+ +---+ +---+ +---+ +---+
| 2 | 0 | | 4 | 1 | | 5 | 4 | | 5 | 0 | | 6 | 6 | | 1 | 1 | | 6 | 0 |
+---+ +---+ +---+ +---+ +---+ +---+ +---+

>>> Introduza o numero do bone que quer usar (5 para sair e 6 para guardar e sair)
>>> Opcao: 1

Numero de pontos feitos nesta jogada: 5

Foram acrescentados 5 pontos a sua pontuacao!
Carlos Costa tem agora 5 pontos!

Prima ENTER para continuar...

```

Caso do primeiro Bone seleccionado pela AI:

```
X:\Eclipse\Cpp32\Proj2Programacao\Release\Proj2Programacao.exe
#####

Jose Costa tem 0 pontos.
Controlado por AI: sim

>>> Bones do jogador Jose Costa <<<
# <0> # # <1> # # <2> # # <3> # # <4> # # <5> # # <6> #
+---+---+ +---+---+ +---+---+ +---+---+ +---+---+ +---+---+
| 3 | 2 | | 0 | 0 | | 6 | 4 | | 5 | 5 | | 6 | 6 | | 3 | 3 | | 5 | 1 |
+---+---+ +---+---+ +---+---+ +---+---+ +---+---+ +---+---+

A AI seleccionou o bone |4|6| para inserir na EAST_WING

Numero de pontos feitos nesta jogada: 10
Foram acrescentados 10 pontos a sua pontuacao!
Jose Costa tem agora 10 pontos!
Prima ENTER para continuar...
```


Inserções seguintes da AI:

```

X:\Eclipse\Cpp32\Proj2Programacao\Release\Proj2Programacao.exe
#####

      +---+
      | 4 | 6 | 6 | 0 |
      +---+

#####

Jose Costa tem 10 pontos.
Controlado por AI: sim

>>> Banes do jogador Jose Costa <<<
# <0> # # <1> # # <2> # # <3> # # <4> # # <5> #
+---+ +---+ +---+ +---+ +---+ +---+
| 3 | 2 | | 0 | 0 | | 5 | 5 | | 6 | 6 | | 3 | 3 | | 5 | 1 |
+---+ +---+ +---+ +---+ +---+ +---+

A AI seleccionou o bone |0|0| para inserir na EAST_WING
Selecione em que posicao quer inserir esse bone...

+---+
| # | Pontos de conexao no bone seleccionado |
+---+
| 0 |          BONE_CONNECTION_POINT_UP      |
| 1 |          BONE_CONNECTION_POINT_DOWN    |
| 2 |          BONE_CONNECTION_POINT_LEFT    |
| 3 |          BONE_CONNECTION_POINT_RIGHT    |
+---+

>>> Indice do ponto de conexao: 1

Numero de pontos feitos nesta jogada: 4
Prima ENTER para continuar...
    
```

Resultados das acções anteriores:

[illegible]

Exemplo do fim de uma round:

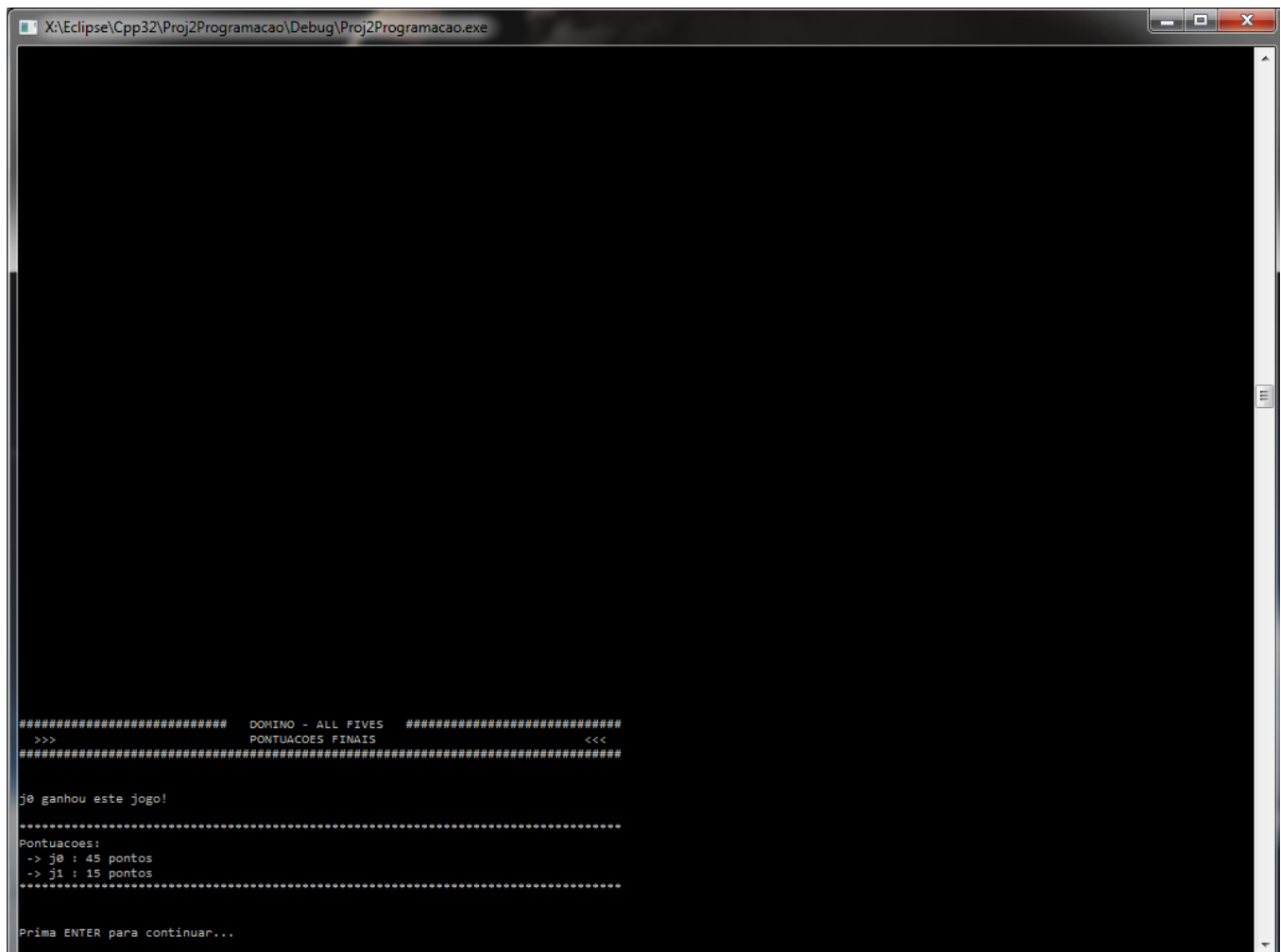
```
X:\Eclipse\Cpp32\Proj2Programacao\Debug\Proj2Programacao.exe

##### DOMINO - ALL FIVES #####
>>>          PONTUACOES NO FIM DESTA ROUND          <<<
#####

-----
* j0 tinha 0 pontos no fim da round!
* j1 tinha 15 pontos no fim da round!
Total de pontos: 15
Total de pontos arredondado: 15
-----
j0 ganhou esta round e obteve 15 pontos de bonus dos seus adversarios!
-----
Pontuacoes:
-> j0 : 45 pontos
-> j1 : 15 pontos
-----

Prima ENTER para continuar...
```

Exemplo do fim de um jogo:



```
X:\Eclipse\Cpp32\Proj2Programacao\Debug\Proj2Programacao.exe

##### DOMINO - ALL FIVES #####
>>>          PONTUACOES FINAIS          <<<
#####

j0 ganhou este jogo!
.....
Pontuacoes:
-> j0 : 45 pontos
-> j1 : 15 pontos
.....

Prima ENTER para continuar...
```


Exemplo de passagem de vez:

```
Jose Costa nao tem Bones possiveis para jogar e nao existem mais bones disponiveis para compra!  
Passando a vez...  
Prima ENTER para continuar...
```

Exemplo de uma round bloqueada por nenhum dos players ter peças válidas.

O bónus é a dado ao player que tenha o menor número de pontos na sua hand (neste caso Jose Costa), e é calculando fazendo a soma dos pontos dos adversários menos o numero de pontos de quem ganhou (neste caso $20 - 10 = 10$).

```
##### DOMINO - ALL FIVES #####  
>>> PONTUACOES NO FIM DESTA ROUND BLOCKED <<<  
#####  
  
-----  
* Carlos Costa tinha 20 pontos no fim da round!  
* Jose Costa tinha 10 pontos no fim da round!  
  
Total de pontos: 10  
Total de pontos arredondado: 10  
-----  
  
Jose Costa ganhou esta round e obteve 10 pontos de bonus dos seus adversarios!  
-----  
Pontuacoes:  
-> Carlos Costa : 0 pontos  
-> Jose Costa : 25 pontos  
-----  
  
Prima ENTER para continuar...
```

O load pode ser feito a partir do menu principal, mas os saves podem ser feitos em qualquer altura do programa.

Basta que quando seja a vez do utilizador seleccionar um dos seus Bones, insira G para guardar.

Se inserir S sai da aplicação.

Se inserir G guarda o estado do jogo e sai.

Exemplo:

[illegible]

```
>>> Introduza o nome do ficheiro onde quer guardar o jogo: domino.txt
```

Se o nome do ficheiro já existir avisa e pergunta se quer fazer overWrite.

Se não quiser fazer overWrite, pergunta se quer introduzir outro nome de ficheiro.

```
>>> Introduza o nome do ficheiro onde quer guardar o jogo: domino.txt  
O nome do ficheiro que introduziu existe!  
Prima ENTER para continuar...  
  
Deseja substituir o ficheiro que ja existe (S/N)? n  
Deseja tentar novamente com outro nome (S/N)? s
```