O interpretador de comandos `GandaGaloShell` para o jogo Gandagalo possui os seguintes métodos (todos os métodos que começam por `eng.` dizem respeito ao módulo `GandaGaloEngine`):

`do_mostrar()` - comando **mostrar** que leva como parâmetro o nome de um ficheiro onde está guardado um tabuleiro a mostrar graficamente. Verifica se já há algum tabuleiro através do método `eng.gettabuleiro()`, e se houver, guarda-o em memória através do método `eng.backup()`. Lê o ficheiro através do método `eng.ler_tabuleiro_ficheiro()`, imprime o puzzle através do método `eng.printpuzzle()` e mostra graficamente o tabuleiro. Devolve mensagem de erro se o número de parâmetros inserido for inválido ou se houver erro ao mostrar o puzzle;

`do_abrir()` - comando **abrir** que leva como parâmetro o nome de um ficheiro onde está guardado um tabuleiro a abrir. Se já existir algum tabuleiro aberto, devolve uma mensagem para o jogador fechar esse tabuleiro. Caso contrário, lê o ficheiro, imprime o puzzle se o tabuleiro não estiver vazio e motra graficamente o tabuleiro. Devolve mensagem de erro se o número de parâmetros inserido for inválido ou se houver erro ao mostrar o puzzle;

`do_gravar()` - comando **gravar** que leva como parâmetro o nome de um ficheiro para guardar um tabuleiro aberto. Se houver um tabuleiro aberto, guarda-o no ficheiro com o nome especificado através do método `eng.guardar_tabuleiro()`, caso contrário devolve uma mensagem a informar que não há dados para gravar. Devolve mensagem de erro se o número de parâmetros inserido for inválido ou se houver erro ao guardar o puzzle;

`do_jogar()` - comando **jogar** que leva como parâmetros o caractere referente à peça a ser jogada ('X' ou 'O) e dois inteiros que indicam o número da linha e o número da coluna, respetivamente, onde jogar. Verifica se o tabuleiro existente é válido através do método `eng.tabuleiro_valido()` e se for, insere a jogada através do método `eng.jogada()` e mostra graficamente o tabuleiro, caso contrário devolve uma mensagem a informar que o tabuleiro não é válido. Devolve mensagem de erro se o caractere a jogar não for X ou O, se o número da linha ou coluna a jogar não forem números inteiros ou se o número de parâmetros inserido for inválido.

`do_validar()` - comando **validar** que verifica se o tabuleiro é válido. Verifica se o tabuleiro existente é válido através do método `eng.tabuleiro_valido()` e se for, devolve mensagem a informar que o tabuleiro é válido, caso contrário, devolve mensagem a informar que o tabuleiro não é válido. Devolve mensagem de erro se forem inseridos quaisquer parâmetros ou se houver erro a validar o tabuleiro;

`do_ajuda()` - comando **ajuda** que indica a próxima casa lógica a ser jogada, sem indicar a peça a ser colocada. Verifica se o tabuleiro existente é válido e se for, devolve uma próxima jogada possível através do método `eng.ajuda_jogada()` e mostra graficamente o tabuleiro, caso contrário devolve uma mensagem a informar que o tabuleiro não é válido. Devolve mensagem de erro se forem inseridos quaisquer parâmetros ou se houver erro na execução do comando;

`do_undo()` - comando **undo** para anular movimentos, ou seja, permite retroceder no jogo. Se existir um tabuleiro aberto, volta a jogada anterior através do método `eng.return_undo()`, imprime o puzzle e mostra graficamente o tabuleiro, aso contrário devolve mensagem a informar que não há nenhum tabuleiro aberto. Devolve mensagem de erro se forem inseridos quaisquer parâmetros ou se houver erro na execução do undo.

`do_resolver()` - comando **resolver** que devolve o puzzle resolvido. Resolve o puzzle através do método `eng.def_resolver_auto()` e mostra graficamente o tabuleiro resolvido. Devolve mensagem de erro se forem inseridos quaisquer parâmetros ou se houver erro na resolução do tabuleiro;

`do_ancora()` - comando **ancora** que guarda o ponto actual do jogo para permitir mais tarde voltar a este ponto, através do método `eng.nova_ancora()`. Devolve mensagem de erro se forem inseridos quaisquer parâmetros ou se houver erro a definir a âncora;

`do_undoancora()` - comando **undoancora** que permite voltar à última ancora registada, através do método `eng.volta_unancora()`, imprime o puzzle e mostra graficamente o tabuleiro. Devolve mensagem de erro se o número de parâmetros inserido for inválido ou se houver erro a voltar à âncora;

`do_gerar()` - comando **gerar** que gera puzzles com solução única e leva três números inteiros como parâmetros: o nível de dificuldade (1 para 'fácil' e 2 para 'difícil'), o número de linhas e o número de colunas do puzzle. Gera um tabuleiro de acordo com os parâmetros inseridos, através do método `eng.gerar_tabuleiro()`. Devolve mensagem de erro se forem inseridos quaisquer parâmetros ou se houver erro a gerar o puzzle;

`do_ver()` - comando **ver** para visualizar o puzzle actual em ambiente gráfico. Se já houver uma janela aberta, fecha-a, e se houver algum puzzle aberto, abre uma janela em ambiente gráfico com o número de linhas e colunas do puzzle (através dos métodos `eng.getlinhas()` e `eng.getcolunas()`) e mostra o tabuleiro. Devolve mensagem de erro se se ainda não existir nenhum tabuleiro aberto, se forem inseridos quaisquer parâmetros ou se houver erro na execução do comando;

`do_sair()` - comando **sair** para sair do jogo. Imprime uma mensagem de saída e se houver uma janela aberta, fecha-a.

A classe `GandaGaloEngine` é inicializada definindo as seguintes variáveis:

`self``.linhas` - número de linhas do tabuleiro, inicializado a zero;

`self``.colunas` - número de colunas do tabuleiro, inicializado a zero;

`self``.tabuleiro` - matriz que representa o puzzle;

`self``.ancora` - stack que guarda os números das jogadas para as quais se definiu âncoras;

`self``.historico` - stack que guarda o histórico das jogadas ao longo do jogo;

`self``.jogada_n` - número da jogada, inicializada a zero (irá diminuir caso se use o undoancora ou undo, por se voltar atrás no jogo no jogo);

`self``.tab_backup` - backup do puzzle, inicializado a zero. Usado sempre que se mostra um tabuleiro enquanto outro já está aberto;

`self``.lin_backup` - backup do número de linhas do tabuleiro, inicializado a zero. Usado sempre que se mostra um tabuleiro enquanto outro já esta aberto;

`self``.col_backup` - backup do número de colunas do tabuleiro, inicializada a zero. Usado sempre que se mostra um tabuleiro enquanto outro já esta aberto;

`self``.resolve_auto_bifurc` - stack usada no comando **resolver** que guarda o número das jogadas nas quais, numa dada casa, é possível jogar tanto X como O (bifurcações). É útil sempre que o tabuleiro deixe de ser válido, podendo voltar-se atrás no jogo.

`self``.conta_bifurc_gerar` - variável que conta o número de jogadas em que se é possível jogar tanto X como O. Se este exceder um dado fator, o tabuleiro passa a ser considerado difícil.

O GandaGaloEngine é composto pelos seguintes métodos:

`ler_tabuleiro_ficheiro()` - tem como parâmetro o nome do ficheiro do qual lê o tabuleiro. Se o parâmetro *`abrir`* for igual a 0, o método é usado no comando **mostrar**, caso contrário é usado no **abrir**. Se o parâmetro *`aberto`* for igual a 0, significa que não há outro tabuleiro aberto, caso contrário, guardar esse mesmo tabuleiro em backup. Abre o ficheiro, lê todas as linhas e verifica se contém dois valores na primeira linha correspondentes ao número de linhas e colunas do puzzle, caso contrário o ficheiro é inválido, verificando de seguida se se trata de um puzzle no mínimo de 3x3. Conta o número de linhas e colunas do puzzle e verifica se correspondem aos valores da primeira linha do ficheiro, caso contrário trata-se de um ficheiro inválido. Se o ficheiro for válido, guarda o puzzle na variável *`self`*`.tabuleiro`.

`backup()` - usado quando se mostra um tabuleiro com outro aberto. Os valores do tabuleiro passam a ser os do tabuleiro em backup, limpando de seguida os valores do backup.

`tabuleiro_valido()` - compara todas as posições dos caracteres X ou O do puzzle com as posições anteriores ou seguintes em cada linha, coluna ou diagonal do puzzle, de modo a verificar se existem três caracteres X ou O iguais em linha, e se for o caso, determina que o puzzle não é válido.

`jogada()` - tem como parâmetros o caratere, a linha e a coluna a jogar. Verifica se o caratere a jogar é válido, se a posição a jogar não está bloqueada, se a posição a jogar se enconta dentro dos limites do puzzle e completa a jogada guardando a posição do caracter jogado no tabuleiro, guarda a jogada no histórico e incrementa o número da jogada. Se não houver espaços para mais jogadas no tabuleiro e se este estiver válido, o jogo está completo.

`ajuda_jogada()` - tem como parâmetro *`resolver_auto`* que se for igual a 0, determina que o método é usado para o comando **ajuda**, caso contrário, é usado para o comando **resolver**. Para todas as posições livres no tabuleiro, começa por determinar quais delas se encontram antes, depois ou no meio de duas peças iguais seguidas no puzzle, para todas as direcções, pois estas são jogadas importantes e prioritárias no jogo. Caso haja alguma situação dessas, guarda para cada uma delas o caratere oposto (para não bloquear o jogo) e a respectiva posição na lista `sugestoes_2`, e escolhe uma aleatoriamente para retornar como sugestão, retornando apenas a posição se o método for usado no comando **ajuda**. Se não se verificar nenhuma situação dessas, guarda como sugestão a posição da primeira casa disponível no puzzle e ambos os caracteres X e O.

`def_resolver_auto()` - método usado no comando **resolver**. Enquanto o tabuleiro estiver válido e não resolvido, verifica a sugestão de jogada do método `ajuda_jogada()` e implementa a sugestão como jogada. Se for possível jogar tanto o X como o O, guarda o número da jogada, seleciona uma peça a jogar (por defeito o X) e implementa a sugestão como jogada. Conta o número de bifurcações encontradas até encontrar a solução por incrementação da variável *`self`*`.conta_bifurc_gerar`. Conta o número de casa livres no tabuleiro e se não houver nenhuma, verifica se o tabuleiro é válido e imprime cada jogada até à resolução do puzzle. Se tabuleiro não for válido, e houver bifurcações possíveis, guarda a posição da casa onde jogar O antes da bifurcação. Joga de seguida O no local da jogada da bifurcação, onde por defeito estava X, e imprime cada jogada da resolução se o parâmetro *`gerar`* for igual de 0. Caso o tabuleiro esteja resolvido e válido, o jogo acabou, caso contrário o puzzle não tem solução. A stack de bifurcações é eliminada e é criada uma nova, exceto quando é usada no método `gerar_tabuleiro()`, para verificar a existência de pelo menos uma bifurcação no puzzle; não é apagada para posteriormente se verificar a existência de outras soluções para o mesmo tabuleiro.

`return_undo()` - permite voltar à jogada anterior. Se o parâmetro *`anc`* for diferente de 0, é usado no método `volta_unancora()`, se o parâmetro *`volta_atras_res_aut`* for diferente de 0, é usando no método `def_resolver_auto()`, quando este último não consegue acabar um tabuleiro mas há pelo

menos uma jogada em que se pode escolher entre X e O. Caso já tenham sido feitas jogadas anteriormente, retira a última jogada feita do histórico, retira a última peça jogada do puzzle substiuindo-a por uma casa vazia na posição da última jogada, e decrementa o número de jogadas feitas. Se o parâmetro *volta_atras_res_aut* for diferente de 0, guarda a última jogada feita e retorna-a.

nova_ancora() - guarda o número da jogada na stack *self*.ancora. Enquanto não se volta à bifurcação

volta_unancora() - volta à âncora anterior do jogo. Se a jogada atual precede a última ancora, isto é, se através do **undo** o jogador passou para uma jogada anterior à âncora, esta é eliminada. Caso contrário, a última âncora é eliminada e volta uma jogada atrás até chegar à última ancora.

gerar_tabuleiro() - tem como parâmetros a dificuldade e as dimensões do puzzle a gerar. Se o nível de dificuldade for 1 (fácil) ou 2 (díficil), e se a dimensão do tabuleiro for pelo menos 3x3, faz backup dos dados do tabuleiro e cria um tabuleiro vazio. São geradas 3% a 10% de casas bloqueadas do número total de casas do tabuleiro e são colocadas em casas aleatórias. O tabuleiro é resolvido e se for válido, retiram-se casas com X ou O aleatoriamente, até ficar 10 a 25% do tabuleiro preenchido, e é definida uma âncora com o estado do tabuleiro (ainda sem jogadas). Verifica depois se o tabuleiro não tem mais do que uma solução e calcula o número de casas jogadas onde seria possível jogar tanto X ou O sobre o número de casas vazias, o que vai determinar a dificuldade do jogo (fácil se menor do que 0.15 ou difícil se maior do que 0.15), retornando à âncora com o tabuleiro inicial. Se o puzzle tiver mais do que uma solução, o tabuleiro é limpo e continua o ciclo até ser gerado um tabuleiro váldo e com as características descritas anteriormente. O tabuleiro é depois gravado num ficheiro com o nome contendo as características do puzzle (dificuldade e tamanho), e as variáveis retornam aos seus valores originais.

(Observação: este método devolve frequentemente erro ao tentar verificar se existe mais do que uma solução (através do método ver_mais_1_sol()) para o puzzle. O motivo deve-se à stack resolve_auto_bifurc que se não estiver vazia faz com que dê erro ao entrar de novo no método def_resolver_auto(), mas para este método, tal stack não pode estar vazia, caso contrário não se verificariam as bifurcações encontradas.)

ver_mais_1_sol() - verifica se o tabuleiro do gerar tem mais que uma solução. Se houver bifurcações possíveis, guarda a posição da casa onde jogar O antes da bifurcação. Joga de seguida O no local da jogada da bifurcação, onde por defeito estava X, e resolve o tabuleiro. Se o tabuleiro tiver uma segunda solução devolve True, caso contrário devolve False.

gerar_novas_variaveis() - limpa o histórico e as âncoras para o método gerar_tabuleiro().

getlinhas() - retorna o nº de linhas do tabuleiro.

getcolunas() - retorna o nº de colunas do tabuleiro.

gettabuleiro() - retorna o tabuleiro.

settabuleiro() - tem como parâmetro um tabuleiro e define-o como o tabuleiro atual.

printpuzzle() - imprime o puzzle.

guardar_tabuleiro() - tem como parâmetro o nome de um ficheiro para guardar o tabuleiro atual. Abre o ficheiro, escreve linha a linha os carateres do puzzle separados por espaços.