Métodos utilizando no projeto do jogo de 20 perguntas:
Métodos usados na StringNode.hpp:
  StringNode()
    Construtor da classe, sem parâmetros, cria um nodo com string vazia, ponteiro esquerdo e direito vazios.
  StringNode(string initial_text)
    Construtor da classe, initial text é o valor inciial da string do nodo, ponteiro esquerdo e direito vazios.
  ~StringNode()
    Destrutor da classe.

  shared_ptr<StringNode> getLeftNode(void)
    Retorna o ponteiro para o nodo à esquerda. Não tem parâmetros.
  shared_ptr<StringNode> getRightNode(void)
    Retorna o ponteiro para o nodo à direita. Não tem parâmetros.
  string getText(void)
    Retorna o conteúdo do nodo. Não tem parâmetros.

  int setText(string newText)
    Retorna um inteiro 1 para sucesso e 0 para erro.
    O parâmetro é a string que será o novo texto do nodo.
  int insertNode(string initial_new_node_text)
    Retorna um inteiro 1 para sucesso e 0 para erro.
    O parâmetro é a string do novo nodo criado à esquerda do atual.
    Se já houver um nodo à esquerda, cria à direita.
    Se já houver à direita, retorna Error(int 0);
  int insertLeftNode(void)
    Retorna um inteiro para sucesso e 0 para erro.
    Não tem parâmetro.
    Um novo nodo vazio é criado à esquerda do atual.
  int insertLeftNode(string initial_new_node_text)
    Retorna um inteiro para sucesso e 0 para erro.
    O parâmetro é a string do novo nodo criado à esquerda do atual.
  int insertRightNode(void)
    Retorna um inteiro para sucesso e 0 para erro.
    Não tem parâmetro.
    Um novo nodo vazio é criado à direita do atual.
  int insertRightNode(string initial_new_node_text)
    Retorna um inteiro para sucesso e 0 para erro.
    O parâmetro é a string do novo nodo criado à direita do atual.
  int cutNode(void)
    Retorna um inteiro 1 para sucesso e 0 para erro.
    Não tem parâmetros. Limpa recursivamente os nodos em seus galhos até limpar a si mesmo.
  int clearLeft(void)
    Retorna um inteiro 1 para sucesso e 0 para erro.
    Não tem parâmetros. Apaga o nodo no galho à esquerda.
  int clearRight(void)
    Retorna um inteiro 1 para sucesso e 0 para erro.
    Não tem parâmetros. Apaga o nodo no galho à direita.

Métodos usados na BTree.hpp:
  BTree()
    Construtor. Sem parâmetros. Cria uma árvore com um nodo inicial vazio.
  BTree(string initial_text);
    Construtor. Uma string de parâmetro que será o valor inicial do nodo raiz da árvore criada.
  PStringNode getRoot(void)
    Retorna um ponteiro para a raiz da árvore. Sem parâmetros.

Tanto a classe StringNode quanto a classe Btree foram testadas no arquivo tests_btree.cpp
Os testes foram:

```
TEST_CASE( "Binary Tree Create and Read", "[binary_tree]" ) {
  SECTION( "Creating an empty tree root" )
  SECTION( "Creating a non empty tree root")
  SECTION( "Inserting left branch on root")
  SECTION( "Inserting right branch on root")
  SECTION( "Inserting left branch on left branch of root")
  SECTION( "Inserting right and left branch on left branch of root" )
  SECTION( "Inserting third branch on same node results in error" )
  SECTION( "Inserting three levels deep of root" )
} //TestCase Binary Tree Create and Read

TEST_CASE( "Binary Tree Update", "[binary_tree]" ) {
  SECTION( "Changing root node text value" )
  SECTION( "Changing branch node text value" )
}//TestCase Binary Tree Update

TEST_CASE( "Binary Tree Delete", "[binary_tree]" ) {
  SECTION( "root can be deleted" )
  SECTION( "a branch can be deleted and those above it are not affected" )
  SECTION( "a branch can be deleted and those below it are deleted too")
  SECTION("A node can have it's left branch deleted")
  SECTION("A node can have it's right branch deleted")
}//TEST CASE DELETE BINARY TREE


Métodos usados na biblioteca game_engine.hpp:
 GameEngine();
    Construtor. Sem parâmetros. Cria uma game engine com árvore
    interna vazia.
 GameEngine( string initial_text )
    Construtor. O parâmetro será o conteúdo da raiz da árvore
    interna.

 PStringNode getStart(void)
    Retorna um ponteiro para raiz da árvore interna. Sem parâmetros.
 PStringNode getActualNode(void)
    Retorna um ponteiro para o nodo atual da árvore interna.
    Sem parâmetros.
 PStringNode popLastNode(void)
    Retorna um ponteiro para o último nodo visitado. Sem parâmetros.
 int setActualNode( PStringNode p_next_node )
    Retorna um inteiro 1 para Sucesso ou 0 para falha. Parâmetro é
    um ponteiro para o nodo que passará a ser o nodo atual.
 int pushLastNode( void )
    Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
    Adiciona à pilha de últimos nós visitados o nó atual.
 int pushLastNode( PStringNode p_next_node );
    Retorna um inteiro 1 para Sucesso ou 0 para falhas. O parâmaetro é
    um ponteiro para o nodo que será armazenado na pilha de últimos nós
    visitados.

 int moveToYes( void )
    Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
    Muda o nó atual para o nó do galho à esquerda. Caso não haja nó,
    nada é feito e se retorna 0(Falha).
 int moveToNo( void );
    Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
    Muda o nó atual para o nó do galho à direita. Caso não haja nó,
    nada é feito e se retorna 0(Falha).
 int moveBack( void );
    Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
    Muda o nó atual para o nó anteriormente visitado. Caso não haja nó,
    nada é feito e se retorna 0(Falha).
```

```
int restart( void )
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
   Apaga-se a árvore interna, os ponteiros para o nó atual e se esvazia
   a pilha de nós anteriores.
   Então se cria uma nova árvore iniciada com o valor na raiz de "New game".
   O nó atual passa a ser a nova raiz.

string readActualNode( void )
   Retorna o conteúdo do nó atual. Não há parâmetros.
int writeInActualNode( string new_text )
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. O parâmetro string
   passado é sobrescrito no nodo atual. Apaga-se o texto anteior.

int removeActualNode( void );
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
   Apaga-se o nó atual. O nó atual passa a ser o anteriormente visitado.

int newYesAnswer( void );
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
   Cria-se um novo nó vazio no galho esquerdo ao atual.
int newYesAnswer(string initial_text)
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Uma string como
   parâmetro. Cria-se um novo nó à esquerda do atual com valor inicial
   recebido como parâmetro.
int newNoAnswer( void );
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
   Cria-se um novo nó vazio no galho direito ao atual.
int newNoAnswer(string initial_text);
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Uma string como
   parâmetro. Cria-se um novo nó à direita do atual com valor inicial
   recebido como parâmetro.
int newYesQuestion(string initial_question);
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Uma string como
   parâmetro. Cria-se um novo nó à esquerda do atual com valor inicial
   recebido como parâmetro. Também se criam dois nodos vazios em seus
   respectivos galhos.
PStringNode getYes( void );
   Retorna um ponteiro para o galho à esquerda do nó atual. Sem parâmetros.
PStringNode getNo( void );
   Retorna um ponteiro para o galho à direita do nó atual. Sem parâmetros.

int checkGuess( void );
   Retorna 1 caso o nó atual tenha ambos os galhos nulos, 0 caso contrário.

int saveGame( void );
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
   Chama o método saveGame com parâmetro "autosave".

int saveGame( string file_name )
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Uma string como
   parâmetro. Salva o conteúdo da árvore interna em um arquivo txt com nome
   igual ao parâmetro passado, dentro da pasta "saved_games".
   Galhos com ponteiros nulos são representados com o símbolo "#".
int loadGame( void );
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Sem parâmetros.
  Chama o método loadGame com parâmetro "autosave".
int loadGame( string file_name );
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Uma string como
   parâmetro. Recupera o conteúdo da árvore interna através de um arquivo
    txt com nome igual ao parâmetro passado, dentro da pasta "saved_games".
int readFile( fstream &p_file_to_read );
   Retorna um inteiro 1 para Sucesso ou 0 para falhas. Um ponteiro para
   ponteiro de arquivo como parâmetro. Lê o arquivo apontado pelo
   ponteiro resursivamente e altera árvore interna de acordo com o que lê.
```

```
 int writeInFile( fstream &p_file_to_write );
    Retorna um inteiro 1 para Sucesso ou 0 para falhas. Um ponteiro para
    ponteiro de arquivo como parâmetro. Escreve no arquivo apontado pelo
    ponteiro resursivamentede acordo com o que lê na árvore interna.

A game engine foi testada nos arquivos tests_game_engine.cpp e
tests_game_statements.cpp
Os testes foram:

TEST_CASE( "Move to Yes or No", "[20_QUESTION_GAME_ENGINE]" )
    SECTION("Can decide if there's no statement on Yes")
    SECTION("Can decide if there's no statement on NO")
    SECTION( "move to No if there's an statement on No" )
} //TEST CASE MOVE TO YES OR NO

TEST_CASE( "GUESSING", "[20_QUESTION_GAME_ENGINE]" )
    SECTION( "Game engine can check if it's a guess" )
} //TEST CASE GUESSING

TEST_CASE( "Restart", "[20_QUESTION_GAME_ENGINE]" )
    SECTION( "a game can be restarted" )
}
TEST_CASE( "Game can be saved on a file", "[20_QUESTION_GAME_ENGINE]")
{
    SECTION("Game engine can write on files already opened")
    SECTION( "Game engine can open files to write" )
}//TEST_CASE GAME SAVE

TEST_CASE( "Game load", "[20_QUESTION_GAME_ENGINE]")
    SECTION("Game engine can read files already opened")
    SECTION( "Game engine can load games" )


TEST_CASE( "CREATE STATEMENTS", "[20_QUESTION_GAME_STATEMENT]" )
  SECTION( "creating game engine" )
  SECTION( "creating an empty statement" )
  SECTION( "creating a non empty statement" )
  SECTION( "creating an empty answer" )
  SECTION( "creating a non-empty answer" )
  SECTION( "creating a question" )
} //TEST CASE CREATING  STATEMENTS

TEST_CASE( "READ STATEMENTS", "[20_QUESTION_GAME_STATEMENT]" )
  SECTION( "An statement on root can be read" )
  SECTION( "An statement on branch can be read")
} //TEST CASE READ STATEMENTS


TEST_CASE( "UPDATE STATEMENTS", "[20_QUESTION_GAME_STATEMENT]" )
  SECTION( "An statement on root can be updated" )
  SECTION( "An statement on branch can be updated")
} //TEST CASE UPDATE STATEMENTS

TEST_CASE( "DELETE STATEMENTS", "[20_QUESTION_GAME_STATEMENT]" )
  SECTION( "A statament can be deleted" )
  SECTION( "The engine will backtrace to the  statement before the one erased" )
{
  SECTION( "The engine will backtrace to the  statement before the one erased
many times" )


Métodos usados na biblioteca game_interface.hpp:
  GameInterface( void );
```

Construtor. Sem parâmetros. Cria uma interface com engine de árvore interna vazia.

```
  PGameEngine getEngine( void );
    Retorna um ponteiro para a engine da internface. Sem parâmetros.
  int openMenu( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Executa várias impressões no terminal e recebe entradas do usuário.
    Dependendo dos valores de entrada pode começar um novo jogo, salvar
    jogo atual, carregar um anterior, editar um jogo atual ou sair
    dessa rotina.

  int startNewGame( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Começa um novo jogo começando com a resposta: "New game"

  int loadSavedGame( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Imprime no terminal e recebe entradas do usuário.
    Carrega um jogo anteriormente salvo pelo usuário dentro da pasta
    "saved_games".
  int saveActualGame( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Imprime no terminal e recebe entradas do usuário.
    Armazena o jogo atual dentro da pasta
    "saved_games". O nome do arquivo salvo é passado pelo usuário.
  int playingRoutine( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Inicia a rotina de perguntas e respostas do jogo.
  int editRoutine( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Iniciai uma rotina que permite editar ou deletar os perguntas e
    respostas do jogo.
  int exitGame(void);
    Retorna o código para encerrar o jogo. Sem parâmetros.

  int doRound( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Lê um nodo. Decide se é uma pergunta ou resposta, imprime para o
    usuário. Daí pode encerrar o jogo, ir para outro nodo, criar uma
    nova resposta ou uma nova pergunta.
  int doEditRound( void )
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Lê um nodo. Pode apagá-lo da árvore(junto de seus galhos), pode
    editá-lo, passar para seus galhos ou retornar a nó anterior.
  int gotAnswer( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Executa as ações necessárias quando um nó é uma folha, ou seja,
    uma resposta.
  int gotQuestion( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Executa as ações necessárias quando um nó não é uma folha, ou seja,
    uma pergunta.
  int validYesInput( string user_input );
    Retorna um inteiro 1, caso seja uma entrada afirmativa válida. 0
    caso contrário.
  int finishGame( void );
    Retorna um inteiro 1 para sucesso ou 0 para Erro. Sem parâmetros.
    Imprime na tela e armazena o estado atual da árvore no arquivo
"last_game.txt"

A game interface foi testada no arquivo tests_game_intercae.cpp
Os testes foram:
```

```
TEST_CASE( "Exists: ", "[20_QUESTION_GAME_INTERFACE]" )
    SECTION("a game interface can be created")
    SECTION(" a game interface have an engine")
}//TEST CASE EXISTS


TEST_CASE( "Round ", "[20_QUESTION_GAME_INTERFACE]" )
    SECTION("It can perform a final round with answer")
    SECTION("It can perform a final round with don't know")
    SECTION("It can perform multiple rounds")
}//TEST CASE EXISTS


TEST_CASE( "Menu", "[20_QUESTION_GAME_INTERFACE]" )
    SECTION("can start new game")
    SECTION("can load a saved game")
    SECTION("can save the actual game")
    SECTION("can exit the game")
    SECTION("has a playing routine")
    SECTION("has a edit routine")
    SECTION("Has a menu")
} //TEST CASE MENU
```