Foi escolhida a linguagem C++ para a implementação do jogo de Pokemon, devido à minha familiaridade com a linguagem e seus processos de compilação, além de ser uma escolha fácil para ambientes Linux.

Utilizou-se a IDE Visual Studio Code para o desenvolvimento e depuração. As dependências de pacotes de software, para se executar numa distribuição Linux Ubuntu, versão 18, estão no arquivo **Ubuntu-18-Dependencies-CLANG.sh** (para o compilador C++ Clang) e **Ubuntu-18-Dependencies-CLANG.sh** (para o compilador C++ GCC). A compilação, execução da aplicação e execução dos testes unitários podem ser feitas em linha de comando, sem a necessidade de uma IDE:

*mkdir build/*

*cd build/*

*cmake -DCMAKE\_BUILD\_TYPE=Debug ../.*

*cmake --build .*

**Para executar:**

*cd src/Pokemon*

*./Pokemon*

**Para executar os testes unitários:**

*cd -*

*ctest -V*

A aplicação é componentizada, da seguinte forma, com os comentários abaixo como justificativas:

**Pokemon**

É o binário (executável) que representa a aplicação.

**PokemonBLL**

É a biblioteca (executável de vínculo dinâmico) que contém toda a lógica da aplicação (Business Layer Logic), de forma a desacoplar a implementação da interface do programa. É também uma estratégia vital para se implementar os testes unitários.

Em termos de criação de classes, cada uma foi criada com seu par declaração (ou cabeçalho) .hpp e implementação .cpp. As classes descritas abaixo referem-se somente ao módulo **PokemonBLL**, pois o módulo Pokemon não tem classes, apenas coleta as escolhas das cartas das rodadas e mostra o resultado.

**Jogo** – encapsula as rodadas de uma partida, que compõem o jogo.

**Pokemon** – encapsula as propriedades das cartas Pokemon.

**Pokemongenerator** – encapsula o mecanismo que ‘embaralha’ as cartas e cria as rodadas para alimentarem o jogo.

**Pokemonrodada** – encapsula as escolhas de cartas dos dois jogadores, e seu resultado.

Dadas as condições do projeto, não se viu necessidade de usar sobrecarga de método ou derivadas de classes a partir de uma superclasse. Os 16 tipos de cartas, com suas propriedade, viraram 16 estruturas (C++ strut) que alimentam a classe Pokemon. Dessa forma, um ‘baralho’ foi criado na forma de variáveis constantes estáticas, facilitando a geração de cartas aleatórias na hora de se criar um jogo.

C++, ao contrário de Java, tem na própria definição da linguagem as bibliotecas de algoritmos (STL – Standard Template Library), com facilidades como algoritmos e coleções nativas na linguagem. Tudo que é necessário fazer é apenas declarar seus cabeçalhos. Os mais relevantes são:

*std::vector* (coleção)

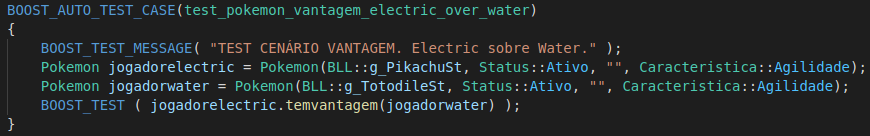
*std::algorithm* (vários algoritmos de busca, ordenação e outros)

*std::string* (manipulação de caracteres)

**Entendimento e facilidade de leitura da aplicação utilizando Testes Unitários**

**TDD** (*Test Driven Development*) é a metodologia *de facto* hoje para desenvolvimento no mercado de trabalho. Dentre as diversas vantagens da metodologia, como facilidade de prototipação e antecipação do desenvolvimento a partir das regras de negócio, se destaca também a facilidade de introduzir novos desenvolvedores às regras e conceitos da aplicação, facilitando a leitura intuitiva do código a partir da forma como ele deve funcionar.

Exemplo de clareza na descrição de um teste. Abaixo, testamos de forma individual o ciclo de efetividade, como descrito no enunciado do projeto:



Exemplo de saída dos testes unitários:

