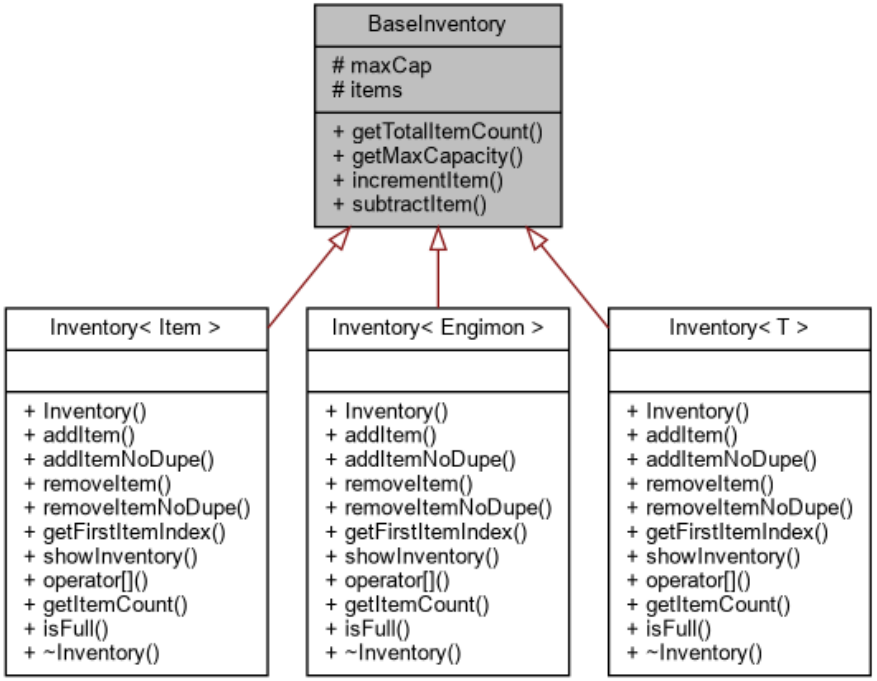
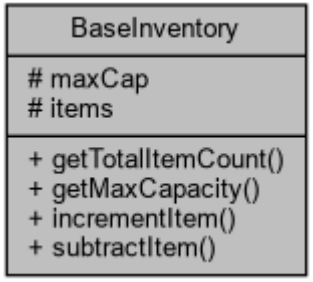
Kelas : 03
Nama Kelompok : y e e wangy wangy
1. 13519116 / Jeane Mikha Erwansyah
2. 13519118 / Cynthia Rusadi
3. 13519124 / Fransiskus Febryan Suryawan
4. 13519131 / Hera Shafira
5. 13519163 / Alvin Wilta
6. 13519164 / Josep Marcello
Asisten Pembimbing : Muhammad Fariz Luthfan Wakan

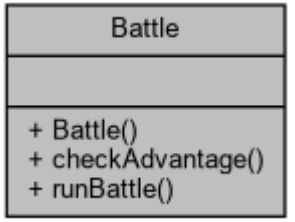
## 1. Diagram Kelas



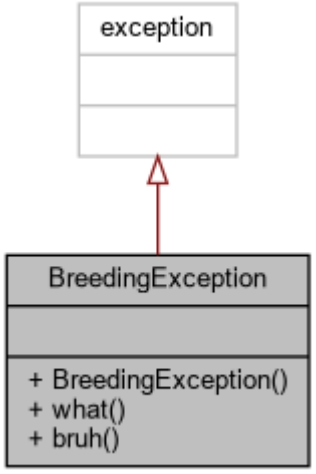

*Gambar 1.1 Diagram Inheritance BaseInventory*



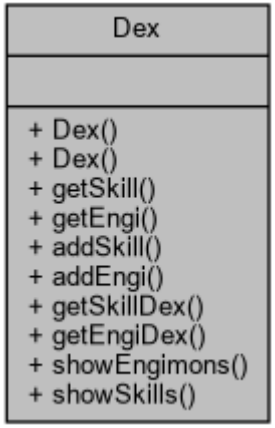

*Gambar 1.2 Diagram Collaboration BaseInventory*



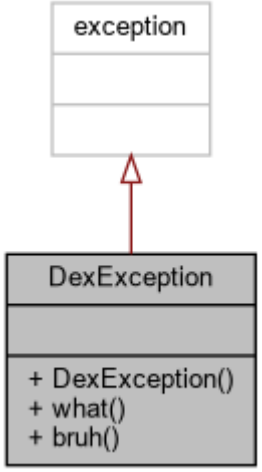

*Gambar 1.3 Diagram Collaboration Battle*



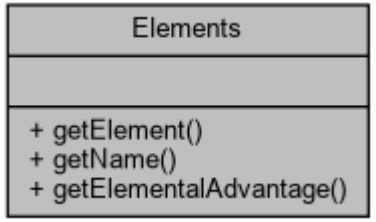

*Gambar 1.4 Diagram Inheritance dan Collaboration BreedingException*



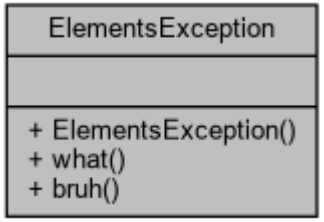

*Gambar 1.5 Diagram Collaboration Dex*



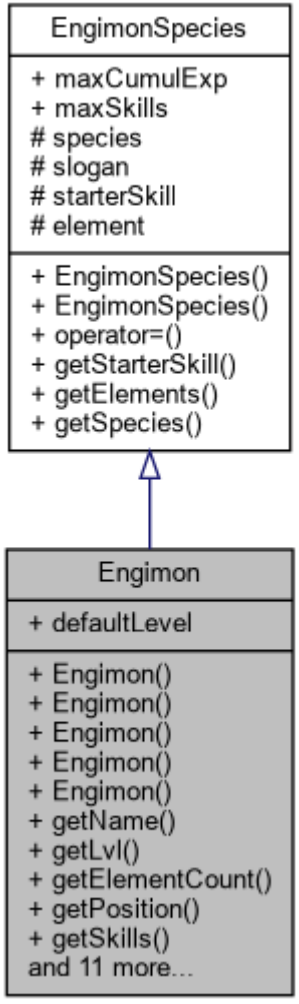

*Gambar 1.6 Diagram Inheritance dan Collaboration DexException*



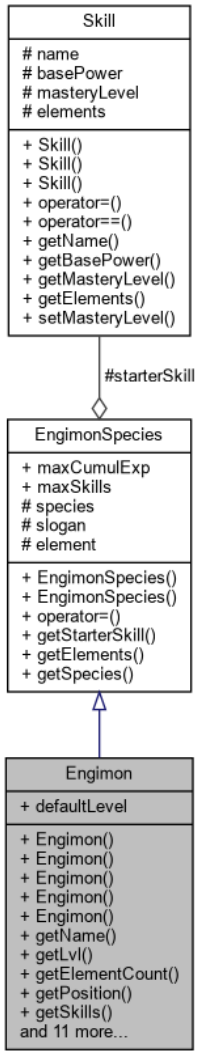

*Gambar 1.7 Diagram Collaboration Elements*



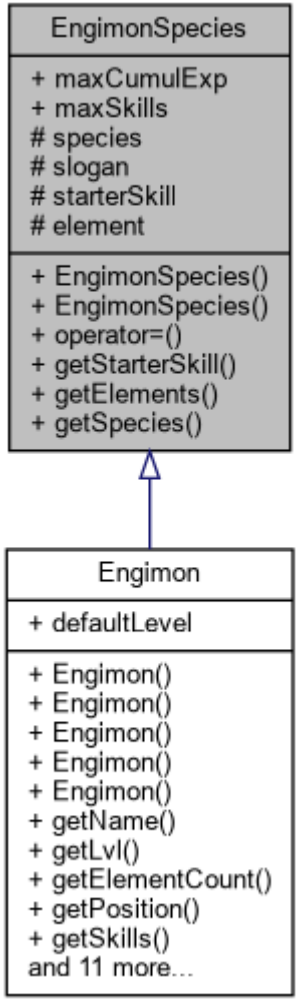

*Gambar 1.8 Diagram Collaboration ElementsException*



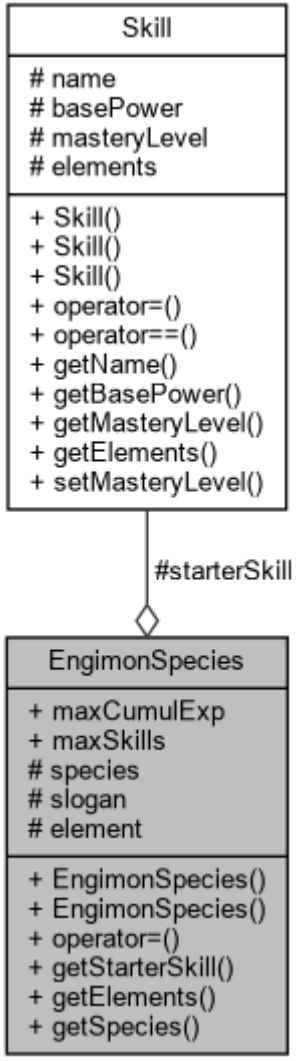

*Gambar 1.9 Diagram Inheritance Diagram*



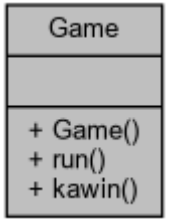

*Gambar 1.10 Diagram Collaboration Engimon*



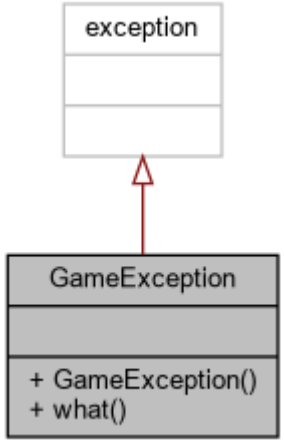

*Gambar 1.11 Diagram Inheritance EngimonSpecies*



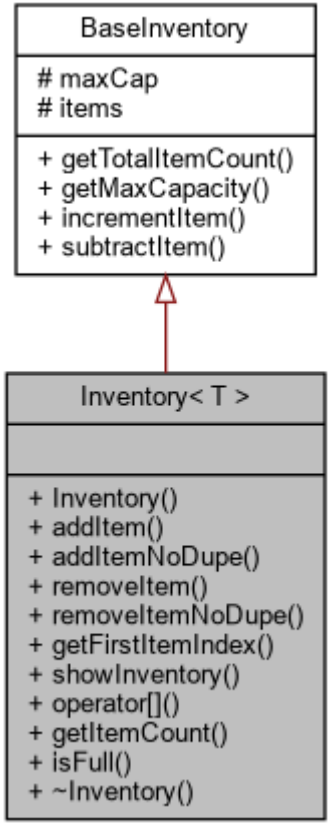

*Gambar 1.12 Diagram Colaboration EngimonSpecies*



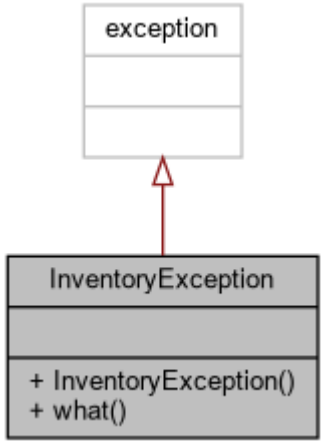

*Gambar 1.12 Diagram Collaboration Game*



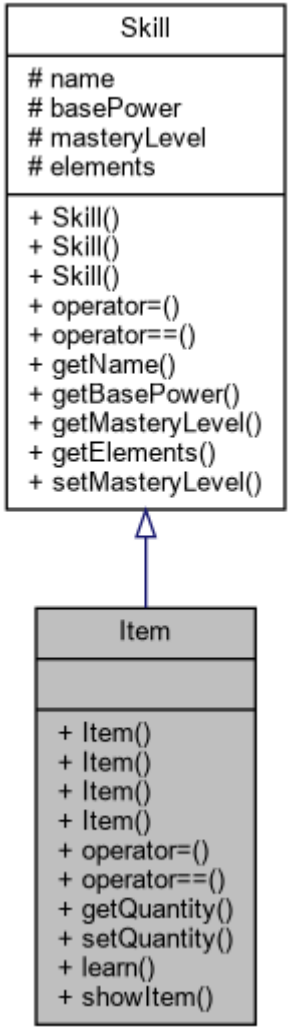

*Gambar 1.12 Diagram Inheritance dan Collaboration GameException*



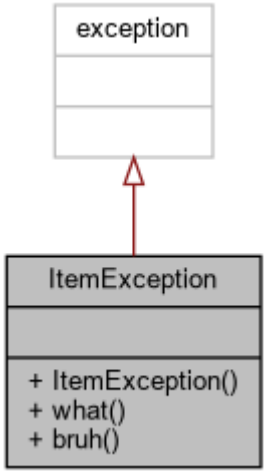

*Gambar 1.13 Diagram Inheritance dan Collaboration Inventory*



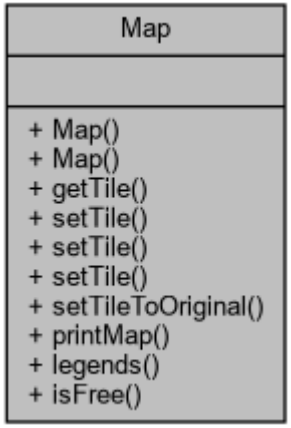

*Gambar 1.14 Diagram Inheritance dan Collaboration InventoryException*



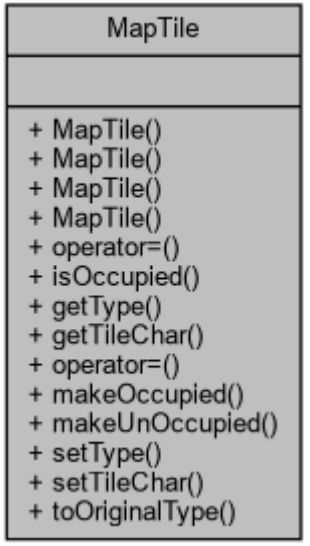

*Gambar 1.15 Diagram Inheritance dan Collaboration Item*



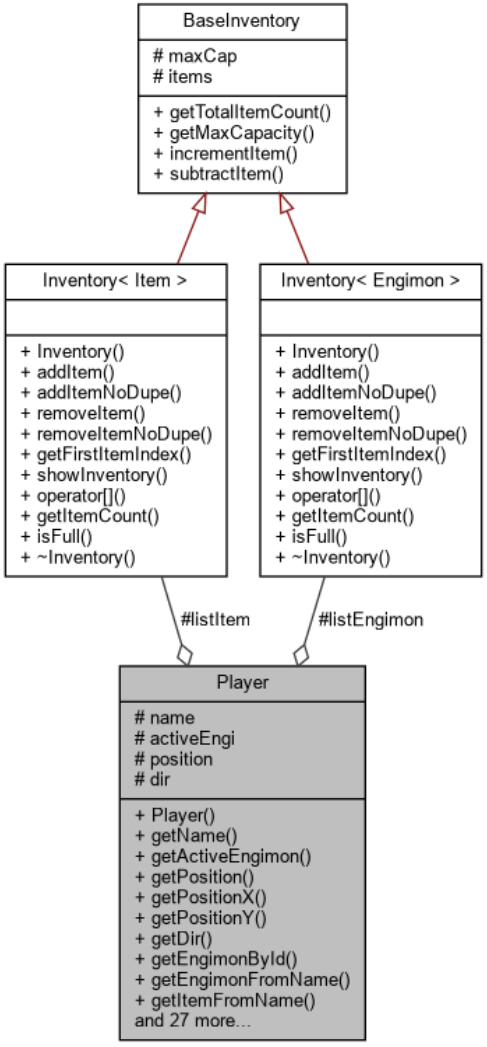

*Gambar 1.16 Diagram Collaboration ItemException*



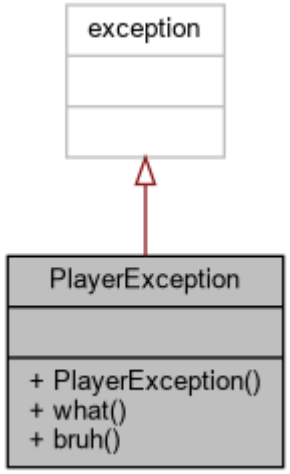

*Gambar 1.17 Diagram Collaboration Map*



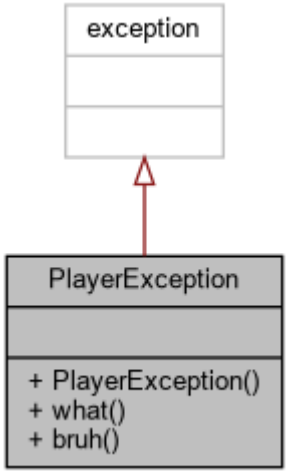

*Gambar 1.18 Diagram Inheritance MapTile*



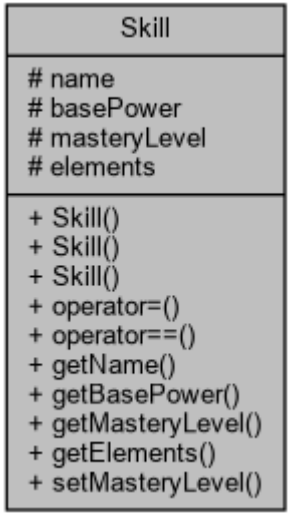

*Gambar 1.19 Diagram Collaboration Player*



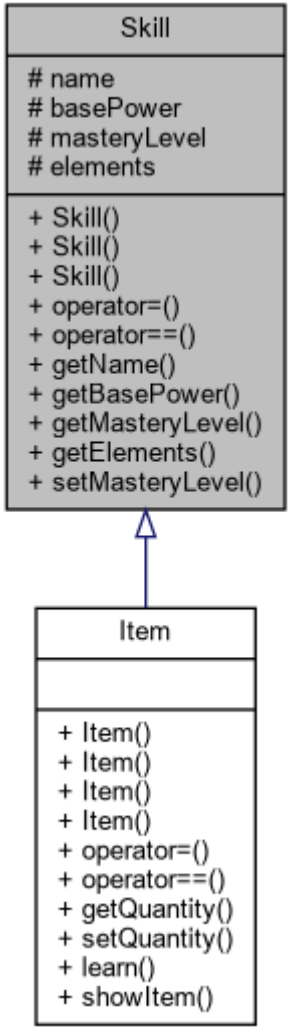

*Gambar 1.20 Diagram Collaboration PlayerExecption*



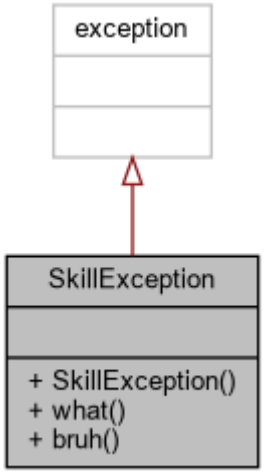

*Gambar 1.21 Diagram Inheritance PlayerExecption*



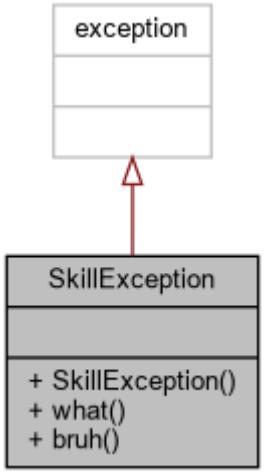

*Gambar 1.22 Diagram Collaboration Skill*




*Gambar 1.23 Diagram Inheritance Skill*




*Gambar 1.24 Diagram Inheritance SkillException*




*Gambar 1.25 Diagram Collaboration SkillException*

# 2. Penerapan Konsep OOP

## 2.1. Inheritance & Polymorphism

### 2.1.1. Kelas EngimonSpecies - Kelas Engimon

Kelas Engimon diturunkan dari kelas EngimonSpecies karena satu spesies Engimon dapat muncul berkali-kali, yang diwakilkan oleh Engimon. Sedangkan kelas EngimonSpecies adalah kelas dasar yang memberikan sifat-sifat yang dimiliki oleh spesies engimon tertentu.

```
class EngimonSpecies {
protected:
    // Nama spesies
    string species;
    string slogan;
```

```
        Skill starterSkill;
        // Elemen yang dimiliki
        vector<Elements::el> element;
    ...
    };
```

```
    class Engimon : public EngimonSpecies {
    private:
        string name;
        tuple<string, string> parents[2];
        vector<Skill> skills;
        int lvl;
        unsigned exp;
        unsigned cexp;
        tuple<int, int> location;

    ...
    };
```

### 2.1.2. Kelas Skill - Kelas Item

Kelas Item diturunkan dari kelas Skill karena Item merepresentasikan Skill Item, yang tak lain adalah turunan dari Skill. Kelas Item harus memiliki semua yang dimiliki oleh kelas Skill, ditambah beberapa sifat yang hanya dimiliki oleh Item, sehingga kelas Item diturunkan dari kelas Skill.

```
class Skill {
protected:
    string name;
    unsigned basePower;
    unsigned masteryLevel;
    vector<Elements::el> elements;  // elemen yang dapat mempelajari skill ini
...
```

```
};
```

```
class Item : public Skill {
private:
    unsigned quantity;
...
};
```

### 2.1.3. Kelas BaseInventory - Kelas Inventory

Kelas Inventory merupakan kelas generic, sehingga atribut static akan berbeda nilainya untuk setiap tipe yang berbeda. Untuk mengatasi hal tersebut, maka kelas Inventory diturunkan dari kelas BaseInventory, yang akan menyimpan jumlah total semua item dan kapasitasnya, sedangkan kelas Inventory yang akan menyimpan barang-barang dalam inventory.

```
class BaseInventory {
protected:
    static const int maxCap;  // kata josep 500
    static int items;
...
};
```

```
class Inventory : BaseInventory {
private:
    /* data */
    vector<T> cont;
    int items;
...
};
```

## 2.2. Method/Operator Overloading

Terdapat *method overloading* pada beberapa kelas. Salah satu contoh untuk *method overloading* ada pada kelas Player, yaitu showEngimon, addEngimon, removeEngimon, showItem, addItem, dan removeItem. showEngimon menerima parameter integer dan kosong, showItem menerima parameter string, Engimon/Item, dan kosong, sedangkan addEngimon, removeEngimon, addItem, dan removeItem hanya menerima 2 parameter, yaitu string dan Engimon/Item. Hal ini dilakukan agar yang menangani perbedaan *input* kelas hanyalah kelas Player karena Player yang memiliki atribut Inventory bertipe Engimon dan Inventory bertipe Item. *Method overloading* dan *operator overloading* juga diimplementasikan pada kelas Engimon untuk konstruktor-konstruktornya dengan alasan adanya banyak tipe Engimon yang dapat bermunculan pada *game*, seperti Engimon starter, wild Engimon, Engimon dengan *custom name* dari pengguna, Engimon yang merupakan hasil breeding. *Operator overloading* yang diimplementasikan pada kelas ini adalah operator '==', operator '=', dan operator '<<', digunakan agar dapat mengecek apakah kedua Engimon sama atau tidak, memindahkan isi dari kelas Engimon ke suatu variabel dan juga digunakan untuk keperluan *print* pada Inventory yang bersifat generic.

```
    void checkActiveEngimon();
    void switchEngimon(int);
    void showEngimon(int);
    void showEngimon();
    void addEngimon(Engimon);
    void addEngimon(string);
    void removeEngimon(Engimon&);
    void removeEngimon(string);
    void showItem(string);
    void showItem(int);
    void showItem() const;
    void useItem(int, int, const Dex&);
    void addItem(Item);
    void addItem(string);
    void removeItem(Item);
    void removeItem(string);
    void interact();
    void engimonIsEmpty();
    void itemIsEmpty();
    void inventoryIsFull();
```

```
    void addExp(int exp);
    unsigned getBattlePower(int elmtAdv);
    void showEngimon();
    void interact() const;
    bool operator==(const Engimon &Eng) const;
    Engimon &operator=(const Engimon &Eng);
    friend ostream &operator<<(ostream &os, const Engimon &src);
```

## 2.3. Template & Generic Classes

Template dan Generic Class hanya digunakan pada kelas Inventory, karena Inventory harus menampung dua tipe berbeda. Karena perilaku inventory tetap sama, maka cocok menggunakan template untuk inventory.

## 2.4. Exception

### 2.4.1. Kelas Parser

Penggunaan exception dalam kelas Parser untuk mengatasi kesalahan saat membuka file. Exception cocok digunakan dalam kasus tersebut karena kelas Parser tidak terkait dengan program tertentu, sehingga harus melempar exception untuk memberitahu pemanggil bahwa pembacaan file gagal. Apabila tidak menggunakan exception, maka pemanggil tidak akan tahu bahwa pembacaan tidak berhasil.

```
class Parser {
private:
    char delim;
    std::string filePath;
    std::vector<std::string> parseLine(std::string) const;
...
};

class ParserException : std::exception {
```

```
private:
    const int msgID;
    static std::string msg[];
...
};
```

```
ParserException::ParserException(int id) : msgID(id) {}
const char* ParserException::what() { return msg[msgID].c_str(); }
std::string ParserException::msg[] = {"Terjadi kesalahan saat membaca file.",
                                      "Gagal membuka file."};
```

### 2.4.2. Kelas Dex

Penggunaan exception pada kelas Dex untuk mengatasi beragam kesalahan yang ada. Karena Dex harus membaca dan menerjemahkan isi file menjadi data Engimon dan Skill. Ada kemungkinan bahwa file yang dibaca tidak sesuai format, yang menyebabkan kegagalan pembacaan file, sehingga Dex harus melempar exception untuk memberi pesan pada pemanggilnya. Penggunaan exception membuat kelas Dex tidak perlu mengetahui status program saat ini.

```
class DexException : exception {
private:
    const int msgID;
    static string msg[];

public:
    DexException(int);
    const char* what();
    void bruh();
};
```

```
DexException::DexException(int id) : msgID(id) {}
```

```
const char* DexException::what() { return msg[msgID].c_str(); }
void DexException::bruh() { cout << what() << endl; }
string DexException::msg[] = {"Format file invalid.",
                              "Element invalid.",
                              "First skill Engimon invalid.",
                              "Skill tidak ada.",
                              "Engimon tidak ada.",
                              "Skill sudah ada.",
                              "Engimon sudah ada."};
```

### 2.4.3. Kelas Elements

Penggunaan exception pada kelas Elements adalah untuk mengatasi kasus ketika Element yang hendak diakses tidak valid. Ketika Element tidak valid, maka salah satu cara untuk memberitahu pemanggil bahwa Element tidak valid adalah dengan melempar exception. Penggunaan exception memberikan pembeda keluaran hasil pemanggilan yang valid dan keluaran pemanggilan yang tidak valid.

```
class ElementsException {
private:
    const int msgID;
    static std::string msg[];

public:
    ElementsException(int);
    const char* what();
    void bruh();
};
```

```
ElementsException::ElementsException(int id) : msgID(id) {}
const char* ElementsException::what() { return msg[msgID].c_str(); }
void ElementsException::bruh() { cout << what() << endl; }
string ElementsException::msg[] = {"Element tidak ada."};
```

### 2.4.4. Kelas Inventory

Penggunaan exception pada kelas Inventory adalah untuk mengatasi kasus pemanggilan metode yang tidak valid. Metode yang dimaksud adalah metode addItem ketika inventory sudah penuh, metode removeItem ketika inventory kosong, operator indexing ketika index invalid, metode getFirstItemIndex ketika item yang diminta tidak ada atau inventory kosong. Keuntungan penggunaan exception adalah untuk membedakan keluaran item yang valid dan error.

```
class InventoryException : exception {
private:
    const int exceptionID;
    static const string msg[];

public:
    InventoryException(int);
    const char* what();
};
```

```
const string InventoryException::msg[] = {
    "Inventory penuh", "Inventory kosong",
    "Item tersebut tidak ada di inventory",
    "Tidak ada item di indeks tersebut"};

InventoryException::InventoryException(int id) : exceptionID(id) {}
const char* InventoryException::what() {
    return msg[this->exceptionID].c_str();
}
```

### 2.4.5. Kelas Item

Kelas Item menggunakan exception pada penanganan kasus item tidak dapat digunakan. Karena kelas Item melibatkan kelas Engimon dalam metode learn, maka kelas Item harus memberitahu kepada pemanggil secara langsung apabila

terjadi kesalahan. Keuntungannya adalah metode learn tidak perlu memberi keluaran, dan dapat menggunakan exception untuk berkomunikasi dengan pemanggil.

```
class ItemException : exception {
private:
    const int msgID;
    static string msg[];

public:
    ItemException(int);
    const char* what();
    void bruh();
};
```

```
ItemException::ItemException(int x) : msgID(x) {}
const char* ItemException::what() { return msg[msgID].c_str(); }
void ItemException::bruh() { cout << what() << endl; }
string ItemException::msg[] = {
    "Mastery level item bukan 1", "Skill item tidak cocok dengan Engimon",
    "Skill item sudah pernah dipelajari",
    "Input pilihan untuk mengganti skill di luar batas"};
```

### 2.4.6. Kelas Skill

Kelas Skill menggunakan exception dalam menangani kasus Element yang diberikan dalam konstruktor tidak valid. Karena konstruktor tidak dapat memberi nilai keluaran, maka cara berkomunikasi dengan pemanggil adalah dengan menggunakan exception. Exception dilemparkan oleh konstruktor Skill agar pemanggil tahu bahwa Element yang diberikan tidak valid.

```
class SkillException : exception {
private:
    const int msgID;
```

```
SkillException::SkillException(int x) : msgID(x) {}
const char* SkillException::what() { return msg[msgID].c_str(); }
void SkillException::bruh() { cout << what() << endl; }
```

```
    static string msg[];

public:
    SkillException(int);
    const char* what();
    void bruh();
};
```

```
string SkillException::msg[] = {"Element tidak valid"};
```

## 2.5. C++ Standard Template Library

### 2.5.1. Vector

Beberapa kelas mengimplementasikan STL vector, seperti pada kelas Inventory, Engimon, EngimonSpecies, dan Skill. STL ini digunakan karena merupakan array dinamis, dengan *container*-nya sudah menangani penyimpanannya secara otomatis. Vector diimplementasikan pada kelas Inventory karena Inventory-nya diimplementasikan secara *generic* dan Vector dapat menampung elemen yang bersifat *generic*. Selain itu, Engimon dan Skill Item dapat dimasukkan dan dihapuskan dari Vector tersebut secara mudah dengan penggunaan metode-metode yang sudah tersedia untuk *container* ini. Begitu juga pada kelas Engimon, yang menyimpan atribut skills menggunakan Vector, dengan parameternya adalah Skill, yang juga menandakan bahwa Vector bersifat fleksibel dengan parameternya. Atribut skills menggunakan Vector karena Engimon diperbolehkan untuk memiliki maksimal 4 skill dan atribut ini tidak menggunakan array statis untuk mencegah adanya pengambilan elemen yang tidak ada di array statis. Skill menggunakan Vector untuk keperluan menyimpan elemen-elemen yang diperlukan untuk skill tersebut dengan alasan Vector yang bersifat dinamis.

```
class Inventory : BaseInventory {
private:
    /* data */
    vector<T> cont;
    int items;
```

```
class EngimonSpecies {
protected:
    // Nama spesies
    string species;
    string slogan;
    Skill starterSkill;
    // Elemen yang dimiliki
```

| | |
|---|---|
| | `vector<Elements::el> element;` |
| `class Engimon : public EngimonSpecies {`<br>`private:`<br>`  string name;`<br>`  tuple<string, string> parents[2];`<br>`  vector<Skill> skills;`<br>`  int lvl;`<br>`  unsigned exp;`<br>`  unsigned cexp;`<br>`  tuple<int, int> location;` | `class Skill {`<br>`protected:`<br>`  string name;`<br>`  unsigned basePower;`<br>`  unsigned masteryLevel;`<br>`  // elemen yang dapat mempelajari skill ini`<br>`  vector<Elements::el> elements;` |

### 2.5.2. Tuple

Posisi Player dan Engimon disimpan di dalam STL Tuple karena penggunaannya dan pemanggilannya yang mudah dan operasi-operasi yang banyak digunakan adalah get() dan make_tuple(). Penentuan posisi sudah sesuai dengan penggunaan Tuple karena yang implementasi yang dilakukan pada posisi adalah untuk membuat sebuah tuple, sebagai konstruktor, dan mencari tahu posisi player atau engimon pada saat itu. Selain posisi, Tuple juga diimplementasikan pada Engimon untuk menentukan nama dari setiap orang tuanya.

```
class Engimon : public EngimonSpecies {
private:
    string name;
    tuple<string, string> parents[2];
    vector<Skill> skills;
    int lvl;
    unsigned exp;
    unsigned cexp;
    tuple<int, int> location;
```

```
class Player {
protected:
    string name;
    Inventory<Engimon> listEngimon;
    Inventory<Item> listItem;
    int activeEngi;
    tuple<int, int> position;
    char dir;
```

### 2.5.3. Unordered_map

*Collection* pada STL C++ ini digunakan untuk menyimpan data menggunakan *hash table*. unordered_map dipilih karena elemen yang disimpan pada *hash table* tidak terurut sehingga penambahan elemen memiliki kompleksitas memasukkan elemen O(1). *Collection* ini dipilih untuk *class* Dex. Di *class* Dex, *collection* ini digunakan untuk menyimpan *list* Engimon dan Skill yang dibaca dari *file*. Dipilih menggunakan (*unordered_*)*map* agar pengambilan Engimon ataupun Skill cukup menggunakan spesies Engimon ataupun nama Skill.

```
class Dex {
private:
    unordered_map<string, Skill> skillDex;
    unordered_map<string, EngimonSpecies> engiDex;
...
};
```

### 2.5.4. Pair

STL Pair diimplementasikan dalam kelas Game untuk keperluan looping. Pair yang digunakan merupakan elemen dari STL unordered_map, digunakan untuk melakukan pencocokan dengan Engimon, penambahan skill, dan pencocokan

dengan elemen, dan digunakan karena Pair dapat menampung 2 elemen dan dapat langsung mendapatkan 2 nilai dari iterator tersebut untuk digunakan.

```
for (pair<string, EngimonSpecies> a : dex.getEngiDex()) {
        if (idx == 0) engieSpecies = a.second;
        idx--;
}
```

```
for (pair<string, Skill> a : dex.getSkillDex()) {
        // cek elemen
        for (Elements::el engieEl : engie.getElements()) {
            for (Elements::el skillEl : a.second.getElements()) {
                compat = engieEl == skillEl;
                if (compat) {
                    filtered.emplace(a.first, a.second);
                    break;
                }
            }
            if (compat) break;
        }
    }
```

```
for (pair<string, Skill> a : filtered) {
            if (idx == 0) {
                engie.setSkills(i, a.second);
                break;
            }
            idx--;
        }
```

```
for (pair<string, EngimonSpecies> esMap : dex.getEngiDex()) {
                vector<Elements::el> el = esMap.second.getElements();
                if (std::find(el.begin(), el.end(), elementBapak) != el.end()) {
```

```
                if (std::find(el.begin(), el.end(), elementEmak) !=
                    el.end()) {
                    spesies = esMap.second;
                    break;
                }
            }
        }
```

### 2.5.5. Find

STL Find adalah fungsi untuk mencari indeks elemen dalam suatu koleksi. Dalam kelas Inventory, STL Find digunakan untuk menguji apakah barang tertentu ada dalam item yang disimpan dalam inventory, untuk mencegah terjadinya pemasukan item yang berulang.

```
        for (pair<string, EngimonSpecies> esMap : dex.getEngiDex()) {
            vector<Elements::el> el = esMap.second.getElements();
            if (std::find(el.begin(), el.end(), elementBapak) != el.end()) {
                if (std::find(el.begin(), el.end(), elementEmak) !=
                    el.end()) {
                    spesies = esMap.second;
                    break;
                }
            }
        }
```

```
    if(std::find(cont.begin(), cont.end(), item) == cont.end()) {
        addItem(item);
    }
```

### 2.5.6. Unique

STL ini digunakan di Dex ketika *loading* Engimon dan Skill dari *file*. Digunakan *unique* untuk menghapus elemen duplikat pada Skill ataupun Engimon. Digunakan bersamaan dengan STL *sort*.

```
 sort(elements.begin(), elements.end());
     vector<Elements::el>::iterator it =
         unique(elements.begin(), elements.end());
```

### 2.5.7. Sort

STL ini akan mengurutkan sebuah *collections* (bisa *vector*, *array*, dll.). STL ini digunakan untuk mengurutkan *vector* Elements::el agar dapat dihapus dengan STL *unique*.

```
sort(elements.begin(), elements.end());
     vector<Elements::el>::iterator it =
         unique(elements.begin(), elements.end());
```

# 3. Bonus Yang dikerjakan

## 3.1. Bonus yang diusulkan oleh spek

### 3.1.1. Dual Element Breeding

Implementasi dual element breeding dilakukan dengan mengambil salah satu element random dari kedua orang tua, kemudian menerapkan aturan breeding menggunakan satu element random. Pengambilan element random menggunakan fungsi rand dalam pustaka stdlib.

```
Elements::el elementBapak, elementEmak;
    if (bapak.getElements().size() == 2)
        elementBapak = bapak.getElements().at(rand() % 2);
    else
        elementBapak = bapak.getElements().at(0);
    if (emak.getElements().size() == 2)
        elementEmak = emak.getElements().at(rand() % 2);
    else
        elementEmak = emak.getElements().at(0);
    if (elementBapak == elementEmak) {
        // spesies sama dengan bapak ato emak
        if (rand() % 2 == 0)
            spesies = dex.getEngi(bapak.getSpecies());
        else
            spesies = dex.getEngi(emak.getSpecies());
    } else {
        // Hitung Element Advantage

        double advBapak =
            Elements::getElementalAdvantage(elementBapak, elementEmak);
```

```
        double advEmak =
            Elements::getElementalAdvantage(elementEmak, elementBapak);
        if (advBapak > advEmak) {
            spesies = dex.getEngi(bapak.getSpecies());
        } else if (advEmak > advBapak) {
            spesies = dex.getEngi(emak.getSpecies());
        } else {
            for (pair<string, EngimonSpecies> esMap : dex.getEngiDex()) {
                vector<Elements::el> el = esMap.second.getElements();
                if (std::find(el.begin(), el.end(), elementBapak) != el.end()) {
                    if (std::find(el.begin(), el.end(), elementEmak) !=
                        el.end()) {
                        spesies = esMap.second;
                        break;
                    }
                }
            }
        }
    }
```

### 3.1.2. Purely Random Wild Engimon Generation

Purely random wild engimon generation diimplementasikan dengan mengambil semua tile kosong yang terdapat dalam peta, kemudian mengambil posisi acak untuk diisi Engimon acak sejumlah yang diinginkan, sesuai dengan parameter count pada metode.

```
Engimon Game::makeRandomEngimon() const {
    int idx = rand() % dex.getEngiDex().size();
    bool compat = false;
```

```
    EngimonSpecies engieSpecies;

    // dapetin spesies engimonnya
    for (pair<string, EngimonSpecies> a : dex.getEngiDex()) {
        if (idx == 0) engieSpecies = a.second;
        idx--;
    }

    Engimon engie(engieSpecies);
    engie.addExp((rand() % wildEngimonLevelBound) *
                 100);  // karena tiap level butuh 100 exp dan sementara dibikin
                        // random 1 sampe 100

    unordered_map<string, Skill> filtered;
    for (pair<string, Skill> a : dex.getSkillDex()) {
        // cek elemen
        for (Elements::el engieEl : engie.getElements()) {
            for (Elements::el skillEl : a.second.getElements()) {
                compat = engieEl == skillEl;
                if (compat) {
                    filtered.emplace(a.first, a.second);
                    break;
                }
            }
            if (compat) break;
        }
    }

    // dapetin n move acak
    for (size_t i = 0;
```

```
         i < min(filtered.size(), (size_t)wildEngimonMoveSetBound); i++) {
        idx = rand() % filtered.size();
        for (pair<string, Skill> a : filtered) {
            if (idx == 0) {
                // belom cek tipe, do it later
                engie.setSkills(i, a.second);
                break;
            }
            idx--;
        }
    }

    return engie;
}

void Game::spawnWildEngimon(unsigned count) {
    wildEngimonLevelBound = (unsigned)player.getActiveEngimon().getLvl() + 5;
    wildEngimonCaptilizeTileCharLevelBound =
        (unsigned)player.getActiveEngimon().getLvl();
    wildEngimonMoveSetBound = (unsigned)player.getActiveEngimon().getLvl() / 25;

    unordered_map<string, EngimonSpecies> engies = dex.getEngiDex();
    // dapetin tile kosong
    vector<tuple<int, int>> freeSpaces = getEmptyMapTile();

    count =
        (count < getEmptyMapTile().size()) ? count : getEmptyMapTile().size();

    // bikin `count` jumlah engimon lalu taruh di map
    for (size_t i = 0; i < count; ++i) {
```

```
        // bikin engimon random
        Engimon engie = makeRandomEngimon();

        // taruh engimon di map
        bool isPlaced = false;
        for (unsigned j = 0; !isPlaced && j < (mapX * mapY); ++j) {
            unsigned randIdx = rand() % freeSpaces.size();
            tuple<int, int> pos = freeSpaces[randIdx];
            vector<Elements::el> engieEl = engie.getElements();
            char engieChar = 0;
            bool isCapitilized = (unsigned)engie.getLvl() >=
                                 wildEngimonCaptilizeTileCharLevelBound;

            // cari tipenya
            if (engieEl.size() == 2) {
                if (((engieEl[0] == Elements::ICE &&
                      engieEl[1] == Elements::WATER) ||
                     (engieEl[0] == Elements::WATER &&
                      engieEl[1] == Elements::ICE))) {
                    engieChar = 'S' * isCapitilized + 's' * !isCapitilized;
                } else if (((engieEl[0] == Elements::WATER &&
                             engieEl[1] == Elements::GROUND) ||
                            (engieEl[0] == Elements::GROUND &&
                             engieEl[1] == Elements::WATER))) {
                    engieChar = 'N' * isCapitilized + 'n' * !isCapitilized;
                } else {
                    engieChar = 'L' * isCapitilized + 'l' * !isCapitilized;
                }
            } else {  // ukurannya 1
                if (engieEl[0] == Elements::FIRE) {
```

```
            engieChar = 'F' * isCapitilized + 'f' * !isCapitilized;
        } else if (engieEl[0] == Elements::ICE) {
            engieChar = 'I' * isCapitilized + 'i' * !isCapitilized;
        } else if (engieEl[0] == Elements::WATER) {
            engieChar = 'W' * isCapitilized + 'w' * !isCapitilized;
        } else if (engieEl[0] == Elements::GROUND) {
            engieChar = 'G' * isCapitilized + 'g' * !isCapitilized;
        } else {
            engieChar = 'E' * isCapitilized + 'e' * !isCapitilized;
        }
    }

    // cek tipe tyle lalu ganti tipenya kalau cocok
    if (map.getTile(get<0>(pos), get<1>(pos)).getType() ==
        MapTile::WATER) {
        if (!(engieChar == 'F' || engieChar == 'G' ||
              engieChar == 'E' || engieChar == 'L')) {
            map.setTile(get<0>(pos), get<1>(pos), engieChar);
            engie.setPos(get<0>(pos), get<1>(pos));
            freeSpaces.erase(freeSpaces.begin() + randIdx);
            isPlaced = true;
        }
    } else {  // char-nya: '-'
        if (engieChar == 'F' || engieChar == 'G' || engieChar == 'E' ||
            engieChar == 'L') {
            map.setTile(get<0>(pos), get<1>(pos), engieChar);
            engie.setPos(get<0>(pos), get<1>(pos));
            freeSpaces.erase(freeSpaces.begin() + randIdx);
            isPlaced = true;
        }
```

```
            }
        }

        if (!isPlaced) {
            // error
            --i;  // ga diitung langkah ini
        } else {
            wildEngimons.push_back(engie);
        }
    }
}
```

### 3.1.3. Unit Testing Implementation

Unit Testing Implementation menggunakan kerangka gtest buatan Google untuk menguji kode. Contoh *screenshot* di bawah hanya sebagian, untuk lebih lengkapnya berada pada pengumpulan tugas besar.

```
#include <gtest/gtest.h>

#include "../src/headers/BaseInventory.hpp"
#include "../src/headers/InventoryException.hpp"
#include "../src/impl/Inventory.cpp"

// uji Inventory dengan int
// uji konstruktor
TEST(Inventory, IntInventoryConstructor) {
    Inventory<int> inv = Inventory<int>();
    EXPECT_EQ(inv.getItemCount(), 0);
    EXPECT_EQ(BaseInventory::getTotalItemCount(), 0);
}
// uji tambah item
```

```
TEST(Inventory, IntInventoryAddItem) {
    Inventory<int> inv = Inventory<int>();
    int item = 0;
    EXPECT_NO_THROW(inv.addItem(item));
    EXPECT_EQ(inv.getItemCount(), 1);
    EXPECT_EQ(BaseInventory::getTotalItemCount(), 1);
}
// uji inventory penuh
TEST(Inventory, IntInventoryFullAddItem) {
    Inventory<int> inv = Inventory<int>();
    for (int i = 0; i < BaseInventory::getMaxCapacity(); i++) {
        BaseInventory::incrementItem();
    }
    int item = 0;
    EXPECT_THROW(inv.addItem(item), InventoryException);
    for (int i = 0; i < BaseInventory::getMaxCapacity(); i++) {
        BaseInventory::subtractItem();
    }
}
// uji hapus item
TEST(Inventory, IntInventoryRemoveItem) {
    Inventory<int> inv = Inventory<int>();
    int item = 0;
    inv.addItem(item);
    EXPECT_NO_THROW(inv.removeItem(item));
    EXPECT_EQ(inv.getItemCount(), 0);
    EXPECT_EQ(BaseInventory::getTotalItemCount(), 0);
}
// uji inventory kosong
TEST(Inventory, IntInventoryEmptyRemoveItem) {
    Inventory<int> inv = Inventory<int>();
    int item = 0;
    EXPECT_THROW(inv.removeItem(item), InventoryException);
```

```
}
// uji hapus item yang tidak ada
TEST(Inventory, IntInventoryNonexistentRemoveItem) {
    Inventory<int> inv = Inventory<int>();
    int item1 = 0, item2 = 1;
    int invaliditem = 2;
    inv.addItem(item1);
    inv.addItem(item2);
    EXPECT_THROW(inv.removeItem(invaliditem), InventoryException);
}
// uji pengambilan elemen ke-i
TEST(Inventory, IntInventoryBracketOperator) {
    Inventory<int> inv = Inventory<int>();
    int items[5] = {0, 2, 4, 6, 8};
    for (int i = 0; i < 5; i++) {
        inv.addItem(items[i]);
    }
    for (int i = 4; i >= 0; i--) {
        EXPECT_EQ(inv[i], i * 2);
    }
}
// uji pengambilan indeks tidak valid
TEST(Inventory, IntInventoryBracketOperatorInvalid) {
    Inventory<int> inv = Inventory<int>();
    EXPECT_THROW(inv[0], InventoryException);
}

// uji dua inventory berbeda
TEST(Inventory, IntInventoryTwoInstance) {
    Inventory<int> inv1 = Inventory<int>();
    Inventory<int> inv2 = Inventory<int>();
    int item = 0;
    inv1.addItem(item);
```

```
    inv2.addItem(item);
    EXPECT_EQ(inv1.getItemCount(), 1);
    EXPECT_EQ(inv2.getItemCount(), 1);
    EXPECT_EQ(BaseInventory::getTotalItemCount(), 2);
}
// uji penghapusan dari satu dari dua instansi
TEST(Inventory, IntInventoryTwoInstanceRemove) {
    Inventory<int> inv1 = Inventory<int>();
    Inventory<int> inv2 = Inventory<int>();
    int item = 0;
    inv1.addItem(item);
    inv2.addItem(item);
    inv2.removeItem(item);
    EXPECT_EQ(inv1.getItemCount(), 1);
    EXPECT_EQ(inv2.getItemCount(), 0);
    EXPECT_EQ(BaseInventory::getTotalItemCount(), 1);
}
// uji dua tipe berbeda
TEST(Inventory, InventoryMultipleType) {
    Inventory<int> inv1 = Inventory<int>();
    Inventory<float> inv2 = Inventory<float>();
    int item1 = 0;
    float item2 = 0;
    inv1.addItem(item1);
    inv2.addItem(item2);
    EXPECT_EQ(inv1.getItemCount(), 1);
    EXPECT_EQ(inv2.getItemCount(), 1);
    EXPECT_EQ(BaseInventory::getTotalItemCount(), 2);
}
// uji penghapusan dari satu dari dua instansi
TEST(Inventory, InventoryMultipleTypeRemove) {
    Inventory<int> inv1 = Inventory<int>();
    Inventory<float> inv2 = Inventory<float>();
```

```
    int item1 = 0;
    float item2 = 0;
    inv1.addItem(item1);
    inv2.addItem(item2);
    inv2.removeItem(item2);
    EXPECT_EQ(inv1.getItemCount(), 1);
    EXPECT_EQ(inv2.getItemCount(), 0);
    EXPECT_EQ(BaseInventory::getTotalItemCount(), 1);
}
```

```
#include <gtest/gtest.h>

#include "../src/headers/BaseInventory.hpp"
#include "../src/headers/InventoryException.hpp"
#include "../src/headers/Player.hpp"

// uji konstruktor
TEST(Player, ConstructorPlayer) {
    Player p = Player();
    EXPECT_EQ("yee", p.getName());
    EXPECT_EQ(1, p.getPositionX());
    EXPECT_EQ(1, p.getPositionY());
    EXPECT_EQ('a', p.getDir());
}

// uji setters
TEST(Player, SetterPlayer) {
    Player p = Player();
    p.setName("imba");
    EXPECT_EQ("imba", p.getName());
    p.setPosition(make_tuple(5, 6));
    EXPECT_EQ(5, p.getPositionX());
    EXPECT_EQ(6, p.getPositionY());
```

```
    p.setDir('d');
    EXPECT_EQ('d', p.getDir());
}
```

```
#include <gtest/gtest.h>

#include <vector>

#include "../src/headers/Item.hpp"

TEST(Item, ConstructorVector) {
    vector<Elements::el> el;
    EXPECT_THROW(Item("test 1", 0, 100, el, 10), SkillException);

    EXPECT_NO_THROW(Item("test 1", 0, 100, Elements::ELECTRIC, 10));

    el.push_back(Elements::ELECTRIC);
    el.push_back(Elements::FIRE);
    ASSERT_NO_THROW(Item("test 1", 0, 100, el, 10));

    Item t1("test 1", 0, 100, el, 10);

    EXPECT_EQ(t1.getName(), "test 1");
    EXPECT_EQ(t1.getQuantity(), 10);
    EXPECT_EQ(t1.getElements()[0], Elements::ELECTRIC);
}

TEST(Item, ConstructorElement) {
    Item t1("test 1", 0, 100, Elements::ELECTRIC, 10);

    EXPECT_EQ(t1.getName(), "test 1");
    EXPECT_EQ(t1.getQuantity(), 10);
    EXPECT_EQ(t1.getElements()[0], Elements::ELECTRIC);
```

```
}

TEST(Item, CopyConstructor) {
    Item t1("test 1", 100, 1,
            vector<Elements::el>{Elements::ELECTRIC, Elements::FIRE,
                                 Elements::GROUND},
            10);
    ASSERT_NO_THROW(Item t2(t1));
    Item t2(t1);

    EXPECT_EQ(t1.getName(), t2.getName());
    EXPECT_EQ(t1.getQuantity(), t2.getQuantity());
    EXPECT_EQ(t1.getElements(), t2.getElements());
}

TEST(Item, OperatorEq) {
    Item t1("test 1", 100, 1,
            vector<Elements::el>{Elements::ELECTRIC, Elements::FIRE,
                                 Elements::GROUND},
            10);
    Item t2(t1);
    Item t3(t1);
    t3.setQuantity(5);

    EXPECT_EQ(t2 == t1, true);
    EXPECT_EQ(t3 == t1, false);
}

TEST(Item, Learn) {
    Dex d("data/Test_Engimons.csv", "data/Test_Skills.csv");
    Engimon e(d.getEngi("Picakhu"));
    Item t(d.getSkill("Tackle"), 10);
```

```
    ASSERT_NO_THROW(t.learn(e, d));
    EXPECT_EQ(t.getQuantity(), 9);
    EXPECT_EQ(e.getSkillsCount(), 2);
    EXPECT_EQ(e.getSkillByIndex(1).getName(), d.getSkill("Tackle").getName());
    EXPECT_EQ(e.getSkillByIndex(1).getBasePower(),
              d.getSkill("Tackle").getBasePower());
    EXPECT_EQ(e.getSkillByIndex(1).getElements(),
              d.getSkill("Tackle").getElements());
    EXPECT_EQ(e.getSkillByIndex(1).getMasteryLevel(),
              d.getSkill("Tackle").getMasteryLevel());
}
```

```
#include <gtest/gtest.h>

#include <vector>

#include "../src/headers/Elements.hpp"
#include "../src/headers/Skill.hpp"

// nguji konstruktor Skill pake vektor
TEST(Skill, ConstructorVector) {
    std::vector<Elements::el> vec;
    EXPECT_THROW(Skill s1("test 1", 0, 100, vec), SkillException);

    vec.push_back(Elements::FIRE);
    vec.push_back(Elements::ELECTRIC);
    EXPECT_NO_THROW(Skill s1("test 1", 0, 100, vec));
}

// nguji getters dari Skill
TEST(Skill, Getter) {
    std::vector<Elements::el> vec;
    vec.push_back(Elements::FIRE);
```

```
    vec.push_back(Elements::ELECTRIC);
    vec.push_back(Elements::GROUND);

    Skill s1("test 1", 100, 0, vec);

    EXPECT_EQ(s1.getName(), "test 1");
    EXPECT_EQ(s1.getBasePower(), 100);
    EXPECT_EQ(s1.getMasteryLevel(), 0);
    EXPECT_EQ(s1.getElements()[0], Elements::FIRE);
    EXPECT_EQ(s1.getElements()[1], Elements::ELECTRIC);
    EXPECT_EQ(s1.getElements()[2], Elements::GROUND);
}

// nguji konstruktor Skill pake satu elemen enum Elements
TEST(Skill, ConstructorSingleElement) {
    EXPECT_NO_THROW(Skill s1("test 1", 100, 0, Elements::ELECTRIC));

    Skill s1("test 1", 100, 0, Elements::ELECTRIC);
    EXPECT_EQ(s1.getElements()[0], Elements::ELECTRIC);
}

// nguji copy constructor Skill
TEST(Skill, CopyConstructor) {
    std::vector<Elements::el> vec;
    vec.push_back(Elements::FIRE);
    vec.push_back(Elements::ELECTRIC);

    Skill s1("test 1", 100, 0, vec);
    Skill s2(s1);

    EXPECT_EQ(s1.getName(), s2.getName());
    EXPECT_EQ(s1.getBasePower(), s2.getBasePower());
    EXPECT_EQ(s1.getMasteryLevel(), s2.getMasteryLevel());
```

```
    EXPECT_EQ(s1.getElements()[0], s2.getElements()[0]);
    EXPECT_EQ(s1.getElements()[1], s2.getElements()[1]);
}

TEST(Skill, OperatorAssign) {
    std::vector<Elements::el> vec;
    vec.push_back(Elements::FIRE);
    vec.push_back(Elements::ELECTRIC);

    Skill s1("test 1", 100, 0, vec);
    Skill s2("test 2", 0, 0, Elements::ELECTRIC);
    s2 = s1;
    EXPECT_EQ(s1.getName(), s2.getName());
    EXPECT_EQ(s1.getBasePower(), s2.getBasePower());
    EXPECT_EQ(s1.getMasteryLevel(), s2.getMasteryLevel());
    EXPECT_EQ(s1.getElements()[0], s2.getElements()[0]);
    EXPECT_EQ(s1.getElements()[1], s2.getElements()[1]);
}

TEST(Skill, OperatorEqual) {
    std::vector<Elements::el> vec;
    vec.push_back(Elements::FIRE);
    vec.push_back(Elements::ELECTRIC);

    Skill s1("test 1", 100, 0, vec);
    Skill s2(s1);
    EXPECT_EQ(s1 == s2, true);
}

// nguji setter Skill
TEST(Skill, Setter) {
    Skill s1("test 1", 100, 0, Elements::ELECTRIC);
    s1.setMasteryLevel(10);
```

```
    EXPECT_EQ(s1.getMasteryLevel(), 10);
}
```

## 3.2. Bonus Kreasi Mandiri

### 3.2.1. EngiDex

EngiDex merupakan hasil dari kelas Dex, yang menyimpan nama, spesies, dan skill dari setiap Engimon yang tersedia di dalam game. EngiDex ini akan digunakan pada program utama untuk melihat semua nama Engimon yang ada berada di game. Selain itu, EngiDex juga berguna untuk membantu *generate wild* Engimon.

### 3.2.2. Random Player Spawn

*Random player spawn* diimplementasikan dengan mengambil semua tile kosong yang terdapat dalam peta, kemudian mengambil posisi acak untuk diisi player dengan Engimon aktifnya di belakangnya.

### 3.2.3. ASCII Art

ASCII Art ditampilkan saat permainan dimulai dan diakhiri. Saat di awal-awal permainan, akan ditampilkan logo EngiBall. Saat pemain melakukan Battle juga akan ditampilkan ASCII Art untuk memperlihatkan hasil battle. Jika pemain menang akan ditampilkan kembang api dan jika pemain kalah akan ditampilkan batu nisan.

## 4. Pembagian Tugas

| Modul (dalam poin spek) | Designer | Implementer |
|---|---|---|
| 1. Engimon | 13519163 | 13519163 |
| 2. Skill | 13519164 | 13519164 |
| 3.a. Command Player | 13519118 | 13519116, 13519118, 13519124, 13519131, 13519163, 13519164 |
| 3.b. Inventory | 13519124 | 13519124 |
| 3.b.i.2 Skill Item | 13519116 | 13519116, 13519164 |
| 3.c. Active Engimon | 13519118, 13519164 | 13519118, 13519164 |
| 4. Battle | 13519124, 13519131, 13519164 | 13519116, 13519118, 13519124, 13519131, 13519163, 13519164 |
| 5. Breeding | 13519124 | 13519124 |
| 6. Peta | 13519131, 13519164 | 13519164 |
| Bonus 1: Dual Element Breeding | 13519124 | 13519124 |
| Bonus 2: Purely Random Wild Engimon Generation | 13519164 | 13519124, 13519164 |
| Bonus 3: Unit Testing | 13519116, 13519118, 13519124, 13519131, 13519163, 13519164 | 13519116, 13519118, 13519124, 13519131, 13519163, 13519164 |
| Extra Bonus 1: EngiDex | 13519163, 13519164 | 13519164 |
| Extra Bonus 2: Random Player Spawn | 13519164 | 13519164 |
| Extra Bonus 3: ASCII Art | 13519131 | 13519131 |

| | | |
|---|---|---|