

Kode Kelompok : LAH

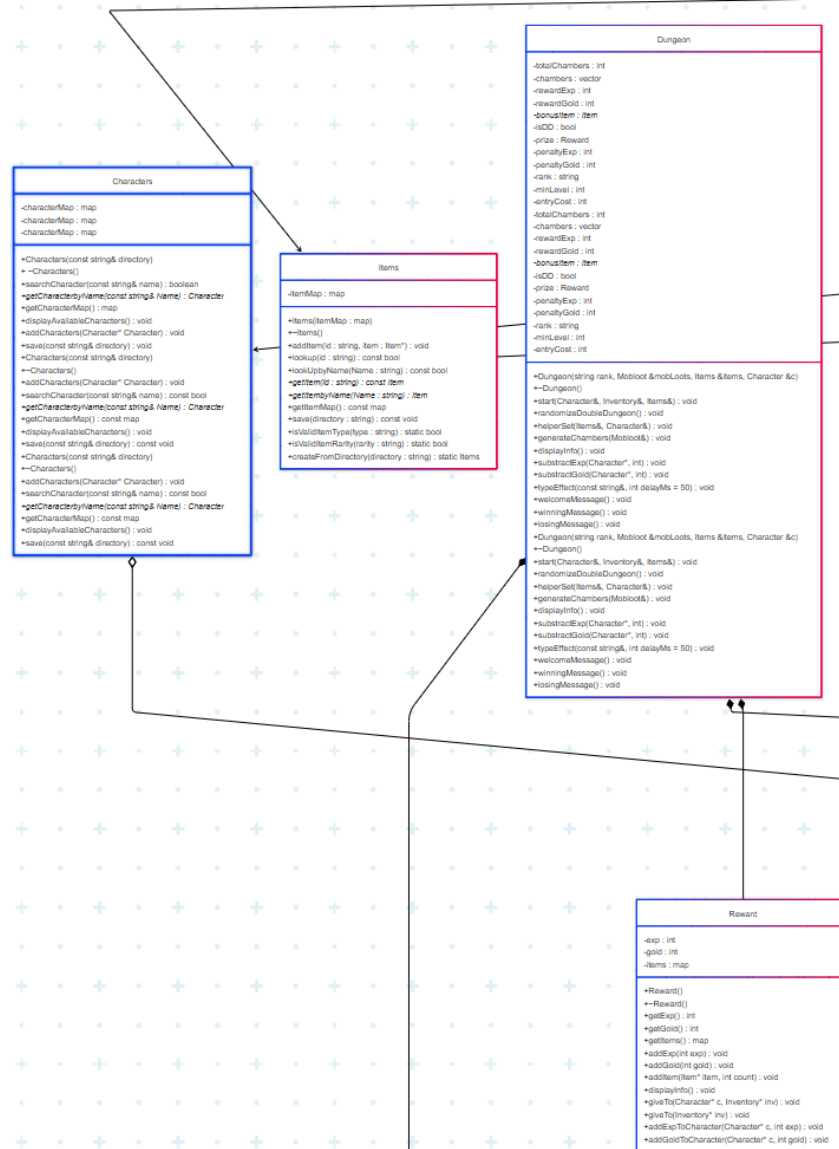
Nama Kelompok : oopsie

1. 13523023 / Muhammad Aufa Farabi
2. 13523025 / Joel Hotlan Haris Siahaan
3. 13523030 / Julius Arthur
4. 13523051 / Ferdinand Gabe Tua Sinaga
5. 13523111 / Muhammad Iqbal Haidar

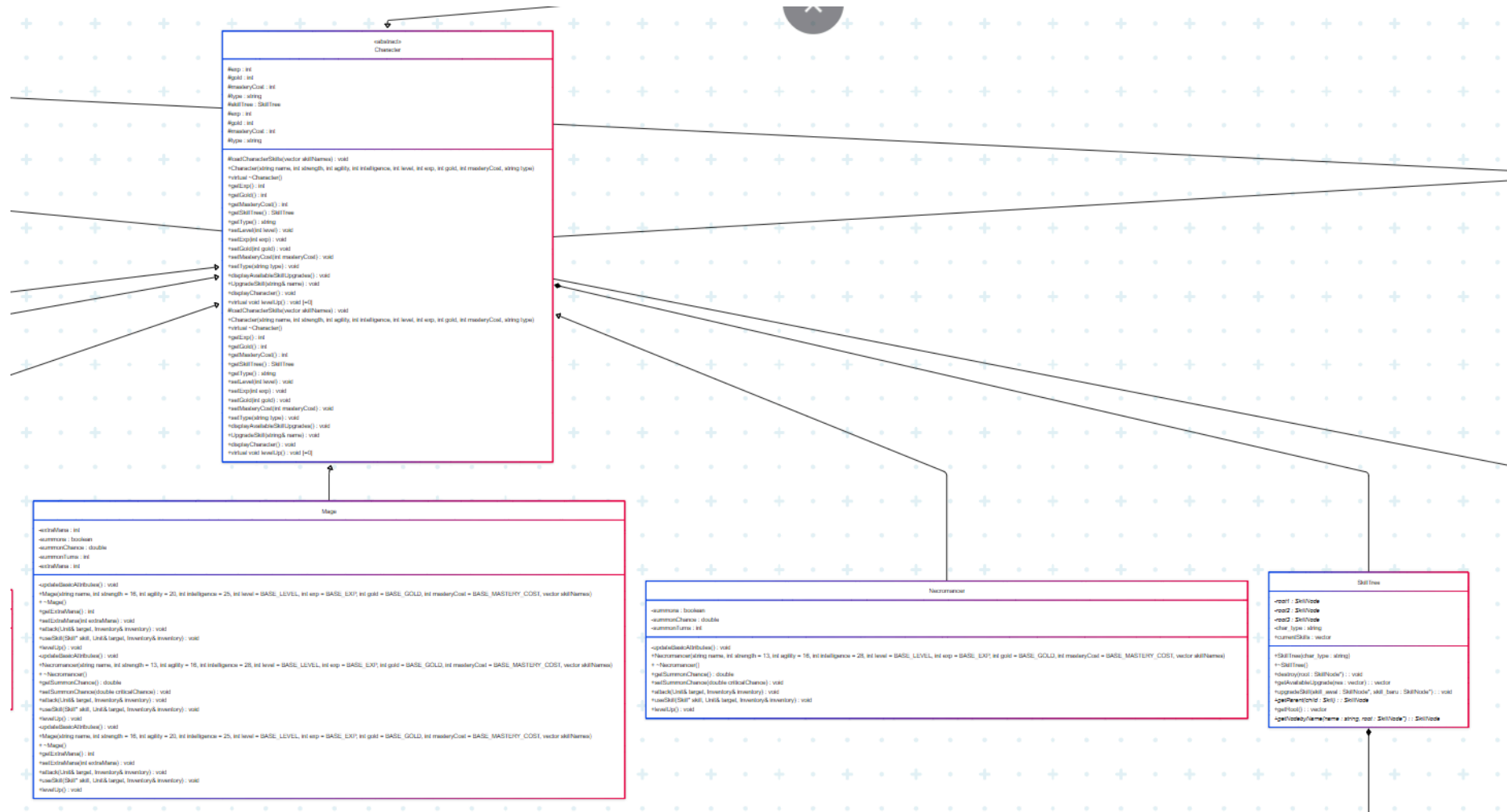
Asisten Pembimbing : Jeffrey Chow

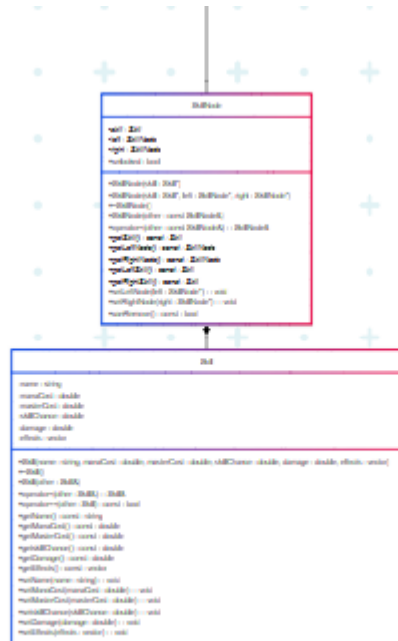
1. Diagram Kelas

<https://www.mermaidchart.com/raw/200737ac-7d1c-40eb-ad17-f021457c9261?theme=light&version=v0.1&format=svg>

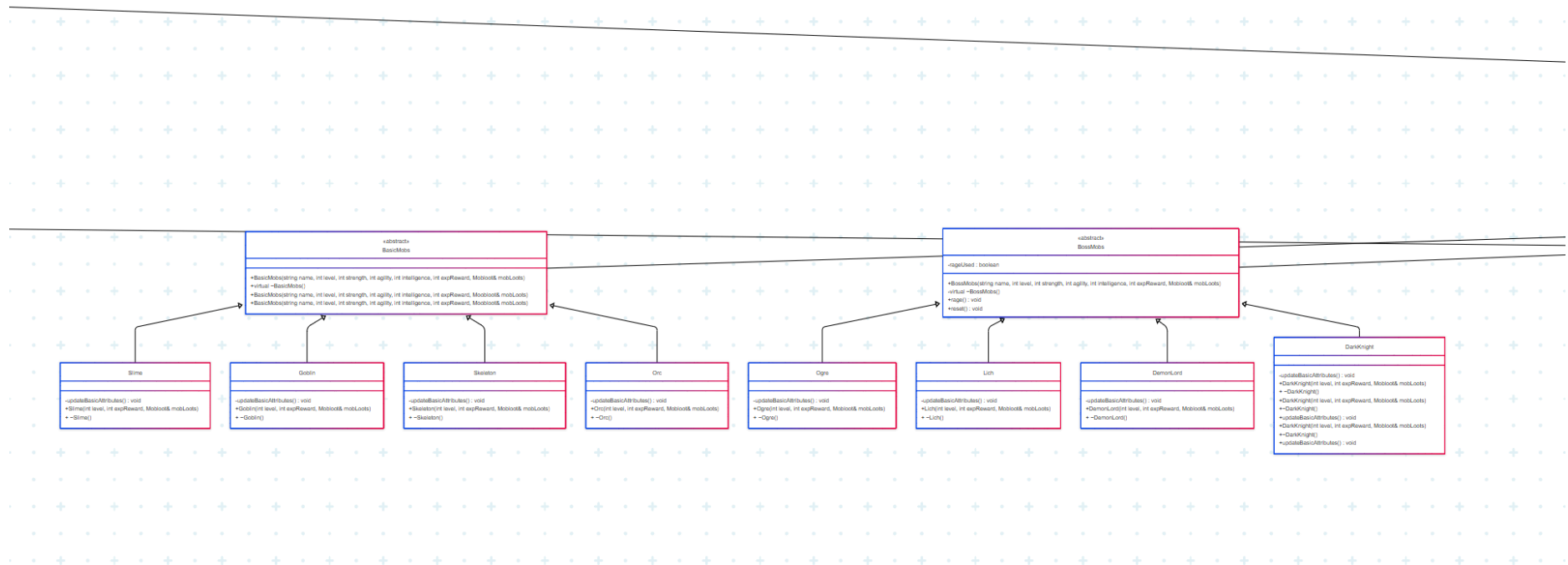


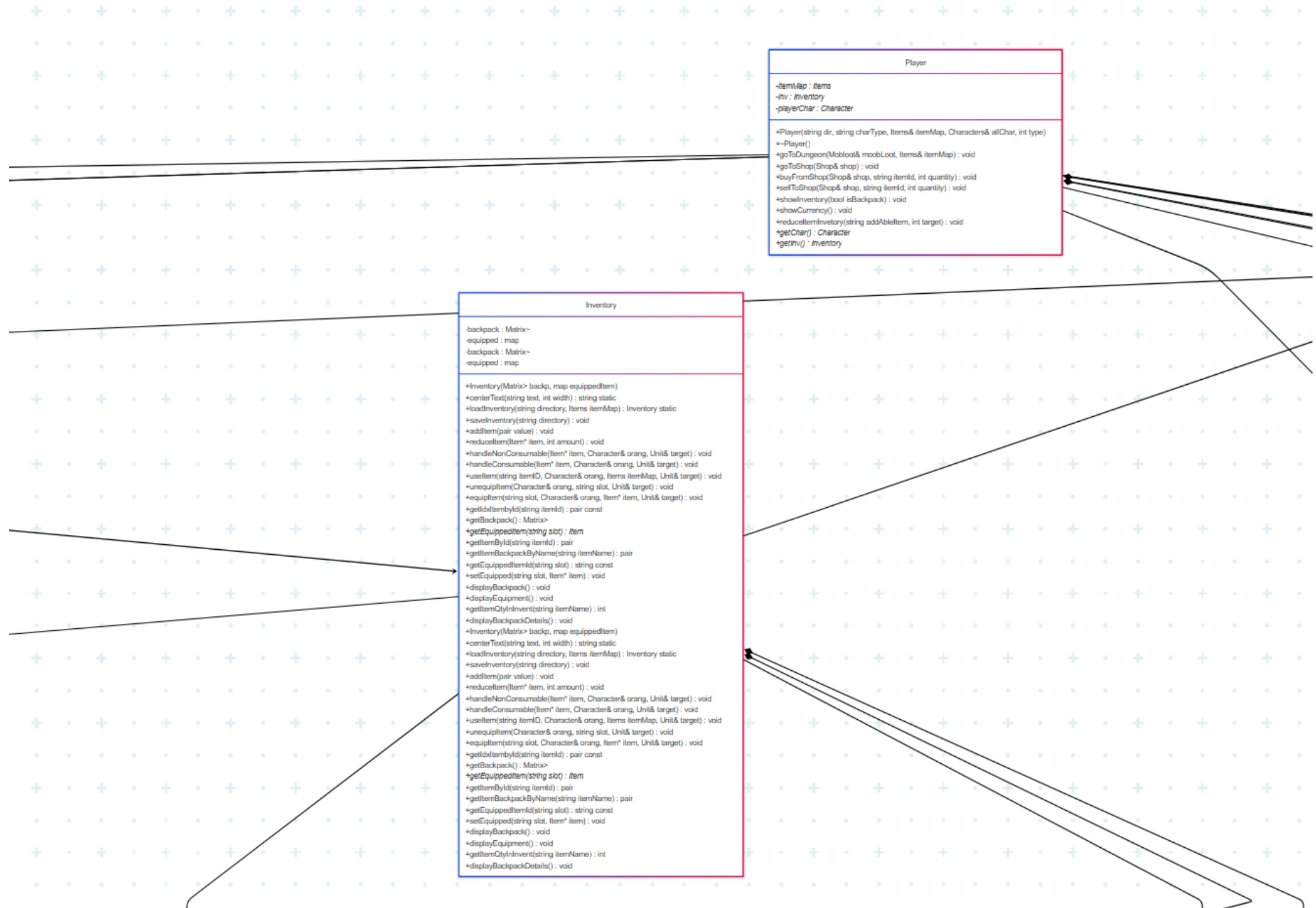


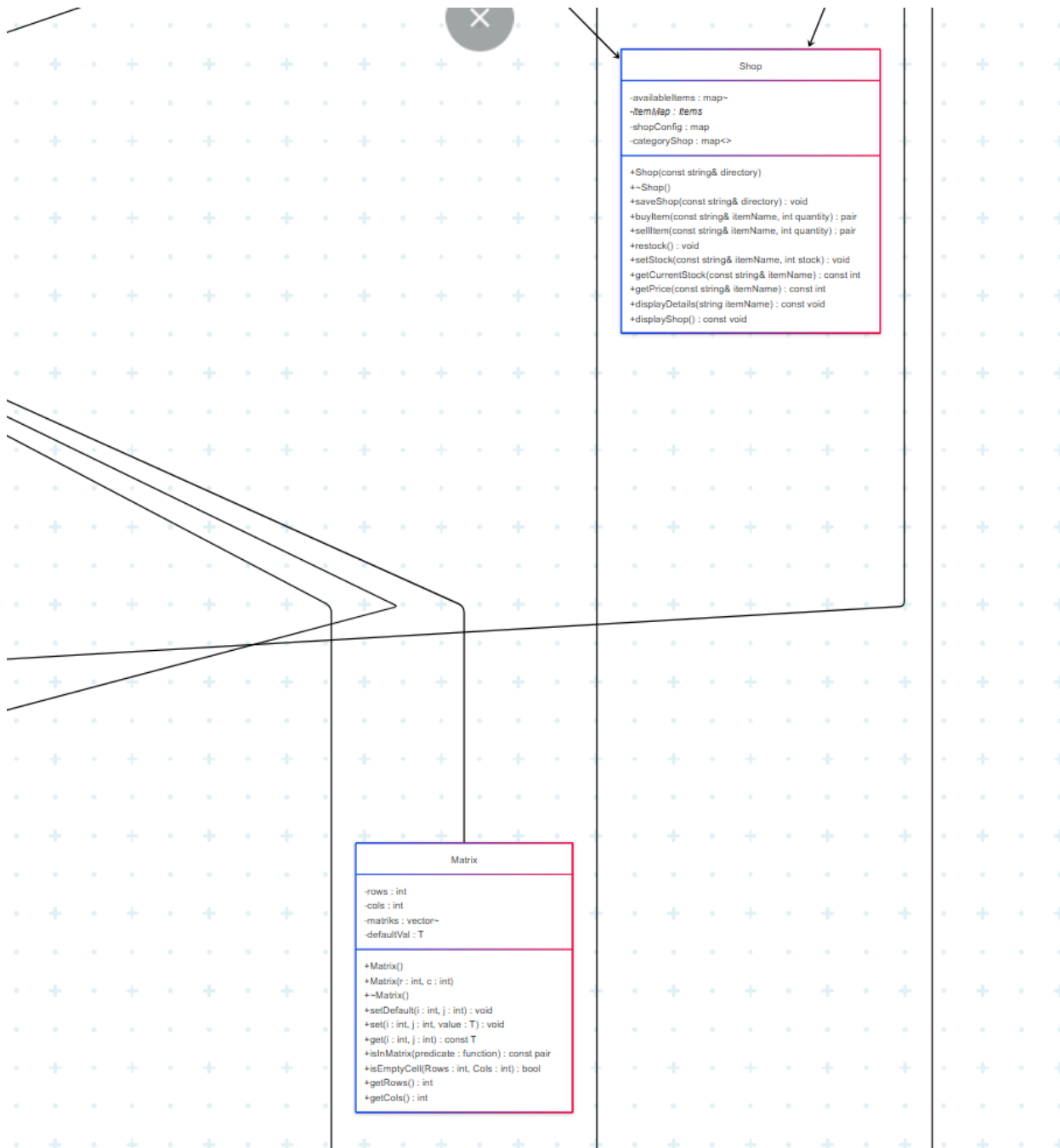


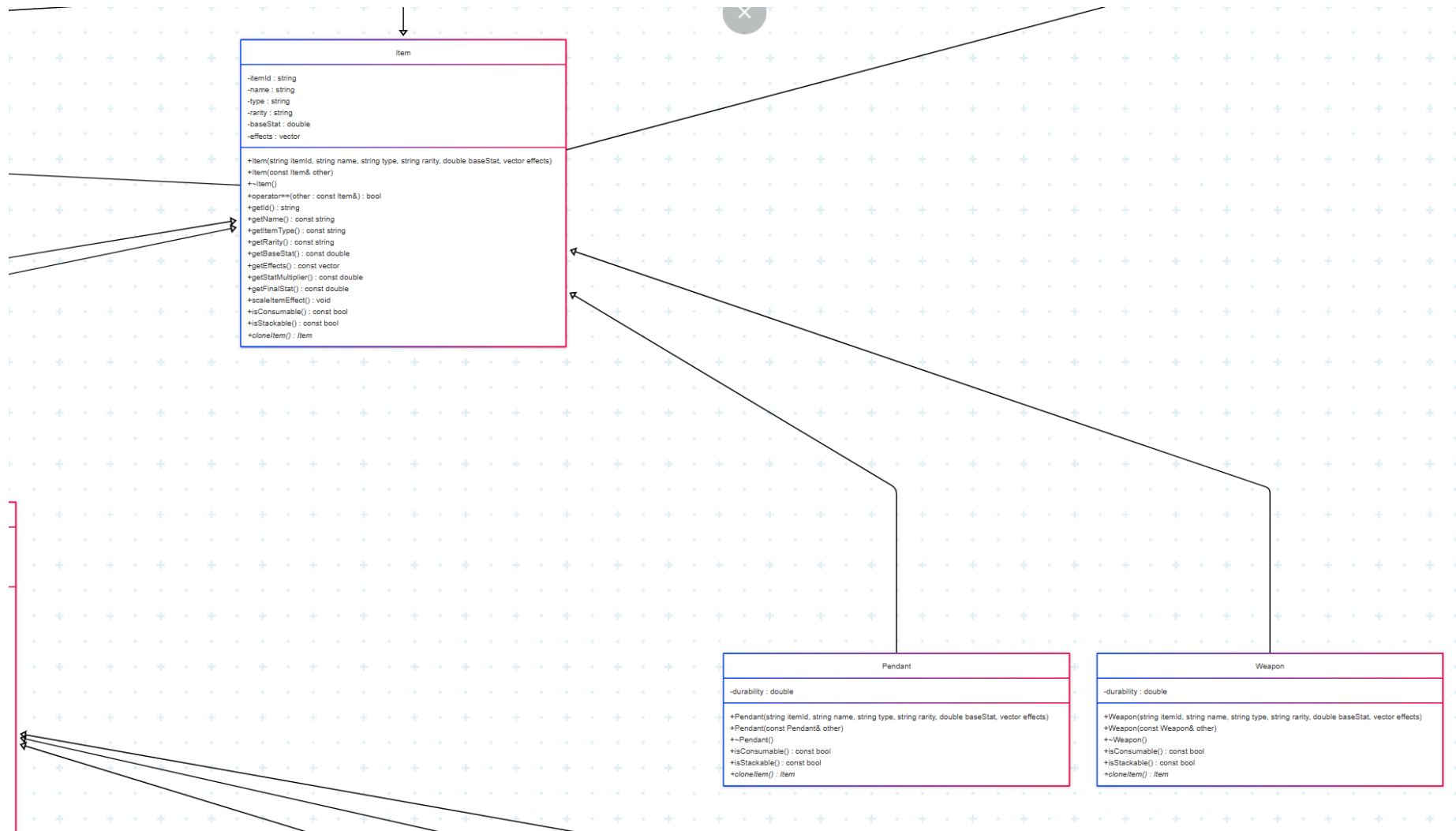


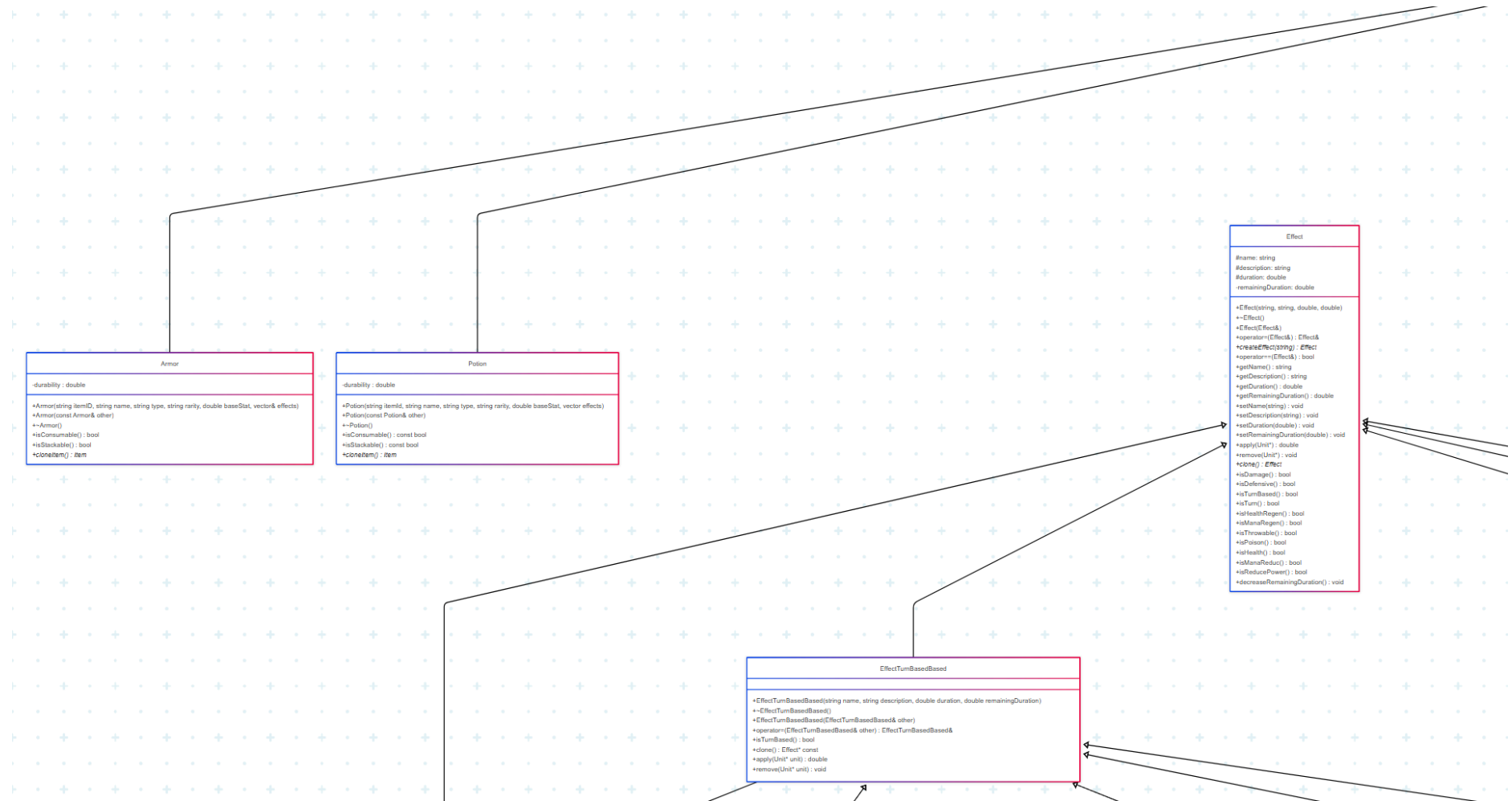


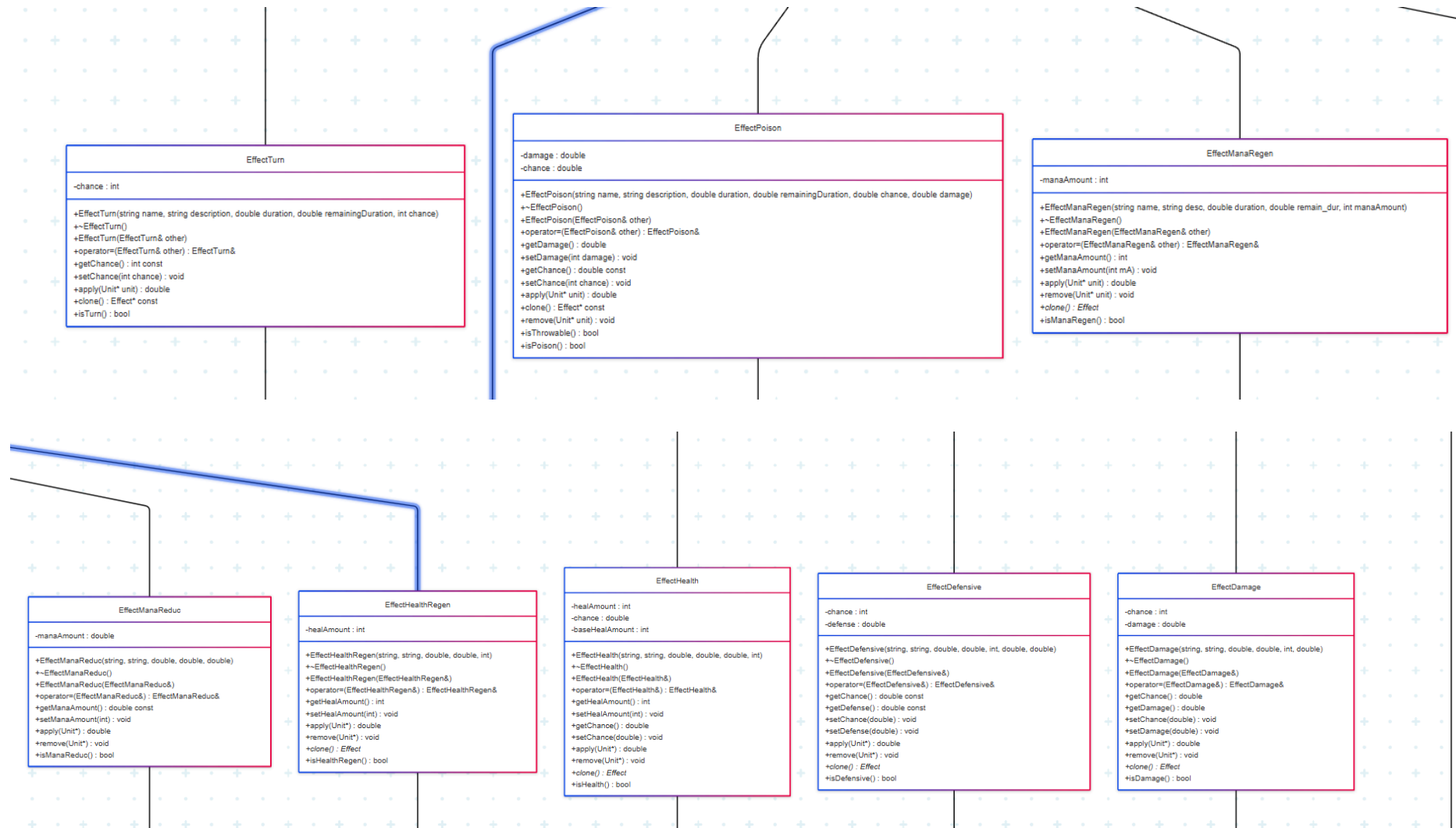


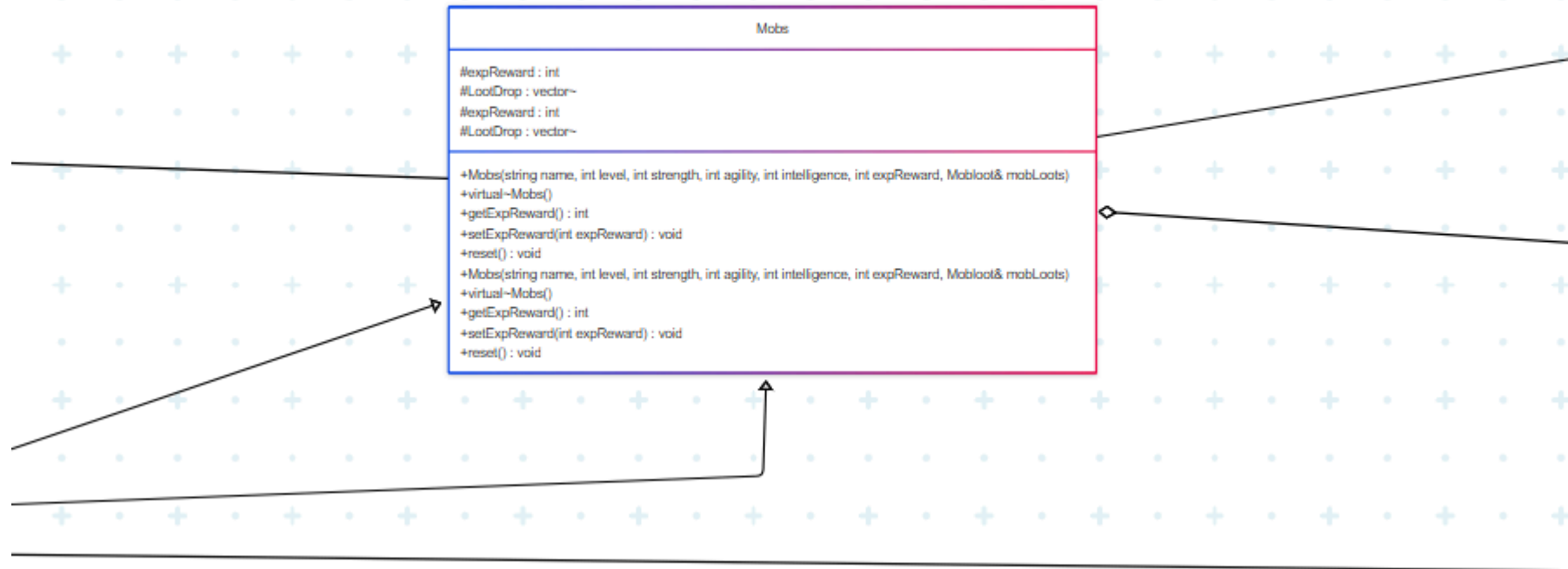


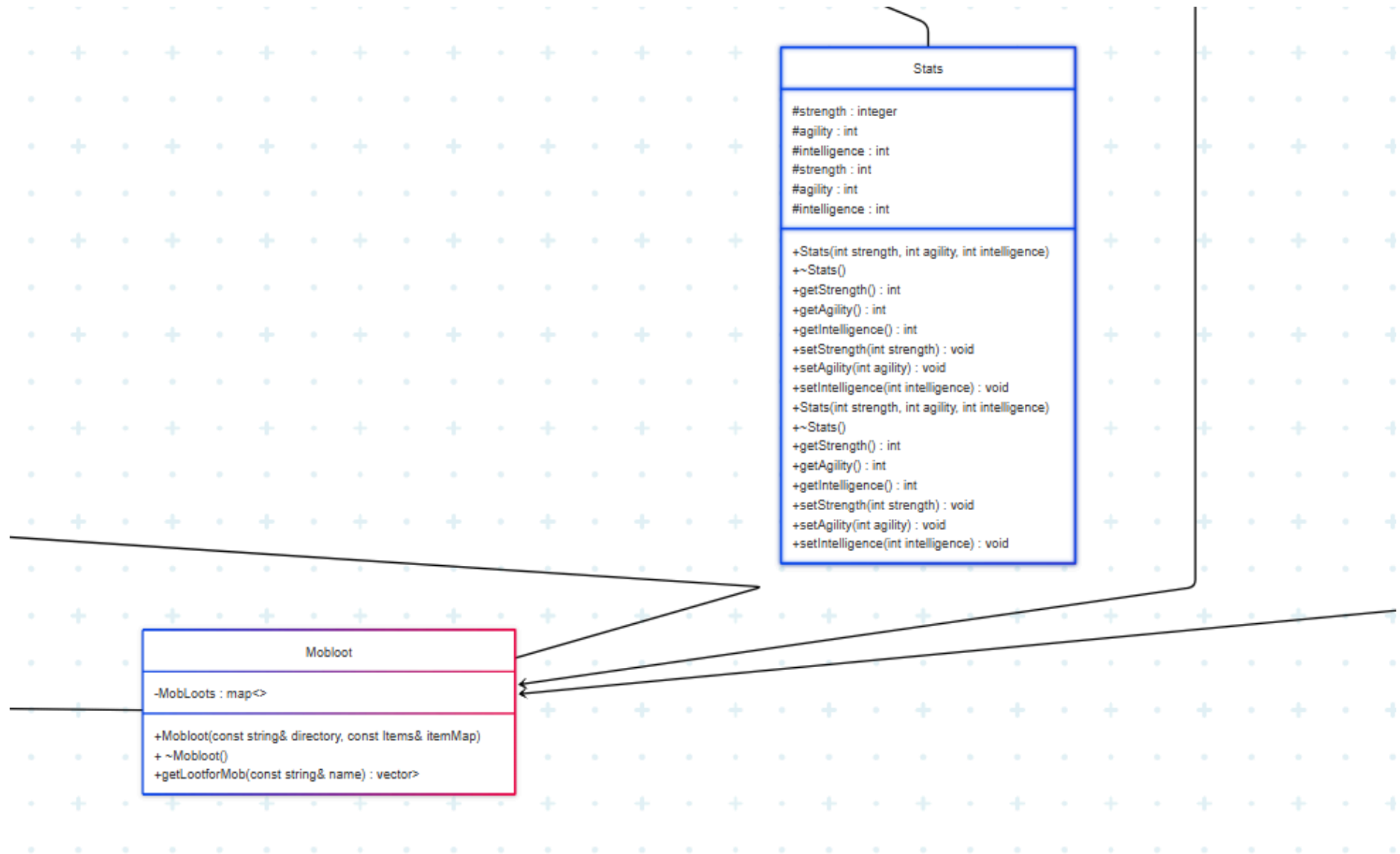












Sejalan dengan penjelasan pada saat perkuliahan dengan dosen IF2010 Pemrograman Berorientasi Objek yakni Pak Catur, kami memanfaatkan beberapa konsep OOP yang telah diajarkan seperti Inheritance pada kelas effect beserta turunannya yakni beberapa jenis efek. Kami juga mengimplementasikan konsep virtual, ABC pada kelas unit, mobs, basic mobs, boss mobs, dan character. Kami juga membuat kelas bertipe generik yakni matriks sehingga membuatnya lebih fleksibel terhadap banyak tipe data yang akan disimpannya. Dengan memanfaatkan konsep-konsep tersebut, desain kelas pada program ini menjadi lebih mudah untuk dikembangkan, dipelihara, dan diubah sesuai kebutuhan. Secara keseluruhan, desain program kami berupaya menjaga modularitas dan extensibility, sehingga apabila di kemudian hari dibutuhkan penambahan jenis efek, tipe unit, atau format penyimpanan data lain, struktur program telah siap untuk mengakomodasi perubahan tersebut dengan minimum modifikasi.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Inheritance

```
class EffectDamage : public Effect {  
    private:  
        int chance;  
        double damage;
```

EffectDamage merupakan turunan dari class Effect. Hal ini dilakukan karena EffectDamage, beserta berbagai turunan class Effect lainnya memiliki implementasi berbeda beda, terutama pada metode apply. Selain itu, setiap class turunan Effect juga memiliki atribut berbeda - beda, seperti ManaHeal, HealthHeal, Damage, Defend, dan lain - lain. Dengan ini, setiap jenis effect yang berbeda dapat memiliki implementasinya yang berbeda - beda dan memungkinkan Polymorphism.


```
class BossMobs : public Mobs {
protected:
    bool rageUsed;
public:
    // ctor dtor
    BossMobs(string name, int level, int strength,
    virtual ~BossMobs();
    bool isRageUsed() const;
    void rage();
    void reset();
```

```
class Mobs : public Unit {
protected:
    int expReward;
    vector<pair<Item*, double>> LootDrop;

public:
    // ctor dtor
    Mobs(string name, int level, int strength, int agility, int intelligence,
    virtual ~Mobs();

    int getExpReward() const;
    void setExpReward(int expReward);
    vector<Item*> dropLoot();
    void reset();
```

BossMob merupakan turunan dari class Mobs. hal ini dilakukan karena BossMob perlu memiliki implementasi method dan atributnya sendiri seperti bool rageUsed dan method seperti rage(). Namun, BossMob merupakan Mob sehingga memiliki atribut expReward dan method-method pada Mobs.

```
class Armor : public Item
{
private:
    double durability;
public:
    Armor(std::string itemID, std::string name, std::string type, std::string rarity,
        double baseStat,
        const std::vector<Effect*>& effects);
    Armor(const Armor& other);
    ~Armor();
```

Item-item seperti armor, weapon, pendant dan potion di implementasi menggunakan konsep inheritance karena seluruh item-item tersebut memiliki atribut yang sangat mirip satu dengan yang lainnya hanya saja ia memiliki implementasi method yang sedikit berbeda di tiap itemnya.

```
class Character : public Unit
protected:
    int exp;
    int gold;
    int masteryCost;
    string type;
    SkillTree skillTree;
    void loadCharacterSkills(vector<string> skillNames);
public:
    // ctor dtor
    Character(string name, int strength, int agility, int intelligence, int level, int exp, int gold, int masteryCost, vector<string> skillNames, string type);
    virtual ~Character();

    // setter getter
    int getExp() const;
    int getGold() const;
    int getMasteryCost() const;
    SkillTree getSkillTree() const;
    string getType() const;
    void setLevel(int level);
    void setExp(int exp);
    void setGold(int gold);
    void setMasteryCost(int masteryCost);
    void setType(string type);
    void displayAvailableSkillUpgrades();
    void upgradeSkill(string& name);
    void displayCharacter();

    // Fungsi
    virtual void levelUp() = 0;

    // Fungsi override Unit
    void reset();
```

Character merupakan turunan dari class Unit. Character seluruh atribut dari class Unit beserta beberapa atribut tambahan seperti exp, gold, masteryCost, type, dan SkillTree. Class ini juga memiliki beberapa metode yang berguna untuk sebuah Character. Inheritance memungkinkan perluasan tanggung jawab yang dapat dimiliki oleh Character dibandingkan class unit.

Polymorphism

```

1  #ifndef UNIT_HPP
27  class Unit {
76
77      // Fungsi
78      virtual void attack(Unit& target, Inventory& inventory);
79      virtual void takeDamage(int damage, Inventory& inventory);
80      virtual void heal(int amount);
81      virtual void restoreMana(int amount);
82      virtual void useSkill(Skill* skill, Unit& target, Inventory& inventory);
83      virtual void addSkill(Skill* skill);
84      virtual void removeSkill(Skill* skill);
85      void addActiveEffect(Effect* effect);
86      void removeActiveEffect(Effect* effect);
87      void applyActiveEffect();
88      virtual void reset() = 0;
89
90

```

Implementasi dari polimorfisme berada pada metode `reset()` pada class `Unit` yang merupakan metode pure virtual. Metode ini dioverride pada subclass-subclassnya, contohnya pada class `BossMobs` dan class `Character`, yang akan memiliki implementasi yang berbeda pada metodenya. Implementasi metode `reset()` pada class `Character` akan mereset nilai atribut-atribut unit yang diwarisi pada `character`, sedangkan implementasi metode `reset()` pada class `BossMobs` selain mereset nilai atribut-atribut unit yang diwarisinya, juga mereset status atribut `isRage` yang dimiliki.

```
#ifndef ITEM_HPP
class Item {
    Item(const Item& other);
    ~Item();

    bool operator==(const Item& other);

    std::string getId(){return itemId;};
    std::string getName() const;
    std::string getItemType() const;
    std::string getRarity() const;
    double getBaseStat() const;
    std::vector<Effect*> getEffects() const;
    double getStatMultiplier() const;
    double getFinalStat() const;
    void scaleItemEffect() ;
    virtual bool isConsumable() const {return false;};
    virtual bool isStackable() const {return true;};
    virtual Item* cloneItem();

};
#endif
```

Method-method seperti isConsumable, isStackable dan clone item memiliki implementasi yang berbeda beda di tiap anaknya dimana virtual fungsi yang mengembalikan bool digunakan untuk memetakan perilaku item tersebut dalam inventory ataupun battle.

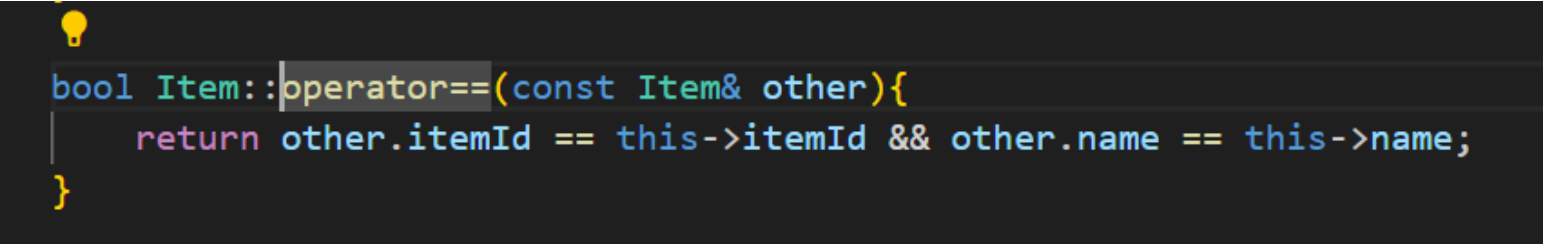
```
class Fighter : public Character {
private:
    float blockChance;
    void updateBasicAttributes() override;
public:
    // ctor dtor
    Fighter(string name, int strength = 27, int agility = 17, int intelligence = 15, int level = BASE_LEVEL, int exp = BASE_EXP, int gold = BASE_GOLD, int masteryCost = BASE_MASTERY_COST, vector<string>
    ~Fighter();

    // setter getter
    float getBlockChance() const;
    void setBlockChance(float blockChance);

    // Fungsi
    void takeDamage(int damage, Inventory& inventory);
    void useSkill(Skill* skill, Unit& target, Inventory& inventory);
    void levelUp();
};
```

Fighter, Mage, Assassin, Necromancer, Berseker merupakan turunan dari class Character. Hal ini diperlukan karena setiap turunan class Character memiliki atribut dan metode yang unik untuk setiap class nya seperti blockChance, extraMana, summonTurns, dan lain - lain. Dengan *polymorphism*, setiap turunan class Character dapat mengimplementasikan dan updateAttributes() masing masing.

2.2. Method/Operator Overloading



```
bool Item::operator==(const Item& other){  
    return other.itemId == this->itemId && other.name == this->name;  
}
```

Salah satu contoh operator overloading ada di implementasi item. Salah satu keuntungannya adalah memudahkan dalam validasi perbandingan apakah item yang kita sedang tinjau sama dengan item yang kita inginkan

2.3. Template & Generic Classes

```

template<class T>
class Matrix {
private:
    int rows, cols;
    std::vector<std::vector<T>> matriks;
    T defaultVal = T();

public:
    Matrix() : rows(8), cols(8) {}
    Matrix(int r, int c) : rows(r), cols(c), matriks(r, std::vector<T>(c, T())) {}
    ~Matrix() {}

    void setDefault(int i, int j){
        matriks[i][j] = defaultVal;
    }

    void set(int i, int j, T value) {
        if (i >= 0 && i < rows && j >= 0 && j < cols) {
            matriks[i][j] = value;
        }
    }

    T get(int i, int j) const {
        return matriks[i][j];
    }
}

```

Template class Matrix dibuat untuk implementasi inventory yang elemen-elemen matrix berupa objek-objek item.

2.4. Exception

```

Items Items :: createFromDirectory(const std::string& directory) {
    std::map<std::string, Item*> itemMap;
    std::string filename = directory + "item.txt";
    if (!fs::exists(directory) || !fs::is_directory(directory)) {
        throw InputOutputException("Directory tidak ditemukan");
    }

    std::ifstream file(filename);
    if (file.fail()) {
        throw InputOutputException("File item.txt tidak ditemukan");
    }
}

```

Method createDirectory akan mengembalikan objek Items yang telah di-load dari path tertentu, tetapi ada kasus ketika directory atau file konfigurasi item.txt tidak ditemukan. Untuk mengatasi kasus-kasus tersebut dilempar InputOutputException sehingga dapat keluar dari method dengan aman tanpa perlu mengembalikan objek Items.

```

// Jika masih ada sisa yang tidak bisa dimasukkan, lempar exception
if (quantity > 0) {
    throw InventoryFull("Backpack penuh, tidak bisa menambahkan sisa item", quantity);
}
}

```

Pada method AddItem yang menambah item ke inventory terdapat kasus kuantitas item tidak bisa ditampung dengan kapasitas inventory. Dalam kasus ini, akan dilempar exception InventoryFull dengan pesan keterangan dan membatalkan penambahan item ke inventory.

2.5. C++ Standard Template Library

```
include > mobloot.hpp > ...
1  ∨ #ifndef Mobloot_HPP
2    #define Mobloot_HPP
3
4  ∨ #include "items.hpp"
5    #include <map>
6    #include <string>
7    #include <vector>
8    #include <utility>
9    using namespace std;
10
11  ∨ class Mobloot {
12    private:
13        map<string, vector<pair<Item*, double>>> MobLoots;
14
15    public:
16        Mobloot(const string& directory, const Items& itemMap);
17        ~Mobloot();
18        vector<pair<Item*, double>> getLootforMob(const string& name);
19
20
21
22    };
23
24    #endif
```

Pemakaian collections berupa map, vector, dan pair dari STL untuk menyimpan daftar mapping nama mobs dengan objek item untuk hasil loot nya dan probabilitas item dropnya. STL Library digunakan agar tidak perlu membuat class yang harus secara khusus untuk merepresentasikan collections, seperti array dinamik (vector) dan map, serta STL Library telah menyediakan method-method yang memudahkan dalam operasi yang berhubungan dengan pemakaian collections.


```

class Characters {
private:
    map<string, Character*> characterMap;

public:
    Characters(const string& directory);
    ~Characters();

    void addCharacters(Character* Character);
    bool searchCharacter(const string& name) const;
    Character* getCharacterByName(const string& Name);
    map<string, Character*> getCharacterMap() const { return characterMap; }
    void displayAvailableCharacters();
}

```

Pemakaian map untuk menyimpan daftar mapping nama character dengan objek character-nya. STL Library digunakan agar tidak perlu membuat class yang harus secara khusus untuk merepresentasikan collection map, serta STL Library telah menyediakan method-method yang memudahkan dalam operasi yang berhubungan dengan pemakaian collections.

2.6. Konsep OOP lain

Sama seperti poin-poin sebelumnya, tuliskanlah konsep OOP lainnya yang ada terapkan selain dari 5 yang diwajibkan. Poin ini akan dipertimbangkan untuk menjadi **nilai bonus**. Contoh: menerapkan Abstract Base Class, Aggregation, atau Composition.

Konsep Komposisi digunakan dalam implementasi fitur Dungeon. Sebagaimana dikatakan di spesifikasi bahwa sebuah dungeon dapat memiliki beberapa chamber berisi monster yang harus dikalahkan oleh pemain. Disini terlihat ada hubungan Komposisi antara objek dungeon dan chamber dimana sebuah dungeon harus memiliki chamber dan chamber tidak dapat exist tanpa adanya objek dungeon. Pada implementasinya, dungeon memiliki vector of chamber sebagai salah satu member attribute. Begitu juga dengan hubungan antara objek chamber dan objek mobs yang dimana chamber dapat memiliki beberapa mobs tetapi mobs tidak bisa exist secara independen.

Konsep Agregasi digunakan dalam implementasi fitur Skill. Setiap skill dapat memiliki beberapa objek pointer ke effect, yang disimpan dalam vektor berisinya, atau tidak sama sekali. Agregasi sendiri berarti suatu class memiliki hubungan “memiliki” atau “mengandung” dengan class lain, tetapi class lain tersebut dapat berdiri sendiri. Karena class Skill dapat memiliki atribut kumpulan objek effect*, tetapi class effect dapat berdiri sendiri, maka Skill dan Effect memiliki hubungan agregasi.

Konsep Abstract Base Class digunakan dalam implementasi fitur Unit. Class Unit dijadikan sebagai abstract base class karena Class unit berperan sebagai blueprint untuk subclass nya, yakni subclass-subclass dari subclass mobs dan subclass character, artinya juga class mobs dan class character di sini juga merupakan abstract base class. Implementasi abstract base class pada Unit dilakukan dengan menjadikan metode reset() pada unit sebagai method pure virtual. Akibatnya, instansiasi objek hanya bisa dilakukan pada subclass dari class unit yang tidak bersifat abstract base class, bukan pada class unit.

Konsep OOP lain yang digunakan adalah forward declaration untuk class. Hal ini dilakukan pada class Effect, yakni terdapat Forward Declaration Class Unit di dalamnya. Pemakaian forward declaration untuk class Unit pada class Effect dilakukan untuk menghindari circular dependency karena class Unit memiliki Effect sebagai atributnya, sedangkan class Effect membutuhkan class Unit untuk mekanisme pemberian efeknya.

3. Bonus Yang dikerjakan

Hapuslah poin di bawah jika tidak dikerjakan. Jika dikerjakan, jelaskanlah pendekatan yang anda gunakan untuk menyelesaikan masalah tersebut berikut dengan screenshot cuplikan kode yang relevan untuk menjelaskan pendekatan anda!

3.1. Bonus yang diusulkan oleh spek

3.1.1. Cheat

Pada bagian Dungeon telah diimplementasikan fitur cheat sebanyak 3 buah yakni AutoWin, EnemyAlwaysStunned, dan MassiveAttackDamage. Pendekatan yang digunakan untuk mengimplementasikan

fitur ini adalah dengan menambahkan variabel boolean sebagai atribut kelas Chamber yang merepresentasikan state apakah cheat tersebut digunakan atau tidak. Lalu pada alur battle akan dilakukan pengecekan apakah variabel tersebut bernilai true atau false. Dan ada berbagai cheat lainnya didalam battle, umumnya implementasinya sama yakni menggunakan variabel boolean.

```

1 class Chamber {
2     private:
3         int rewardExp;
4         int rewardGold;
5         int enemyCount;
6         std::vector<Mobs *> enemies;
7         bool isLastChamber;
8         int minMobLevel;
9         int maxMobLevel;
10        MobLoot *mobLoots;
11        bool autoMenang;
12        bool cheatEnemyStun;
13        bool cheatDamage;

```

Variabel State Cheat

```

1 if (opt == 1) {
2     autoMenang = true;
3 } else if (opt == 2) {
4     cheatEnemyStun = true;
5 } else if (opt == 3) {
6     c.setAttackDamage(c.getAttackDamage() + BIG_DAMAGE);
7     cheatDamage = true;
8 }

```

Pengecekan cheat EnemyAlwaysStunned

```

1 bool Chamber::battle(Character& c, Inventory& inv, Reward& prize, Items& items) {
2     for (int i = 0; i < enemyCount; i++) {
3         bool isCharTurn = true;
4         int turnCtr = 0;
5         while (enemies[i]→getCurrentHealth() > 0 && c.getCurrentHealth() > 0 && !autoMenang) {

```

Pengecekan cheat AutoWin

```

1 if (enemies[i]→getTurnEffectStatus("Stun") || cheatEnemyStun) {

```

Pengecekan cheat EnemyAlwaysStunned

3.1.2. Antarmuka Pengguna

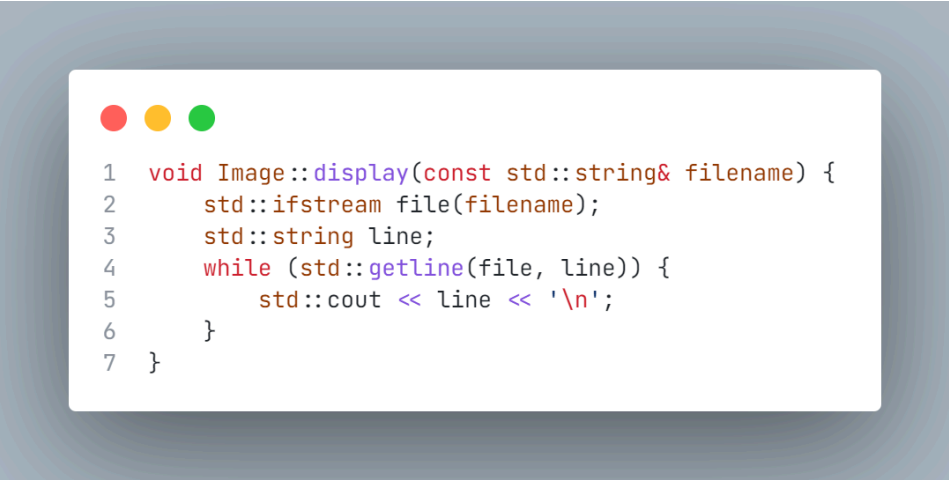
Kami mengimplementasikan antarmuka pengguna sederhana dengan menampilkan status dari setiap pemain dan monster saat dilakukan dungeon. Kami memanfaatkan terminal dengan warna ANSI untuk mendisplay health bar serta mana bar dari character maupun mobs. Berikut ini adalah cuplikan implementasi kode untuk mendisplay bar. Selain itu kami juga membuat metode untuk mendisplay ascii art dari dosen IF2010 Pak Catur sebagai bentuk penghormatan.

```

1 void Chamber::printBar(int value, int maxValue, const string& bgColor) {
2     int width = 30;
3     int filled = (value * width) / maxValue;
4
5     // Tampilkan bagian bar yang terisi (warna latar)
6     std::cout << "[";
7     cout << bgColor;
8     for (int i = 0; i < filled; ++i)
9         cout << " ";
10
11     // Reset warna dan tampilkan bagian yang belum terisi (background gelap)
12     cout << "\033[0m";
13     for (int i = filled; i < width; ++i)
14         cout << " ";
15
16     cout << "\033[0m" << " ] (" << value << "/" << maxValue << " | +";
17 }

```

Method printBar()




```
1 void Image::display(const std::string& filename) {
2     std::ifstream file(filename);
3     std::string line;
4     while (std::getline(file, line)) {
5         std::cout << line << '\n';
6     }
7 }
```

Method display()

3.1.3. Cerita Latar / Lore

Kami mengimplementasikan fitur berupa cerita latar yang dapat dibaca oleh pengguna sembari bermain. Pendekatan yang kami gunakan untuk mengimplementasikan fitur ini adalah dengan menyimpan cerita dalam suatu variabel string lalu dibuatkan sebuah method yang seolah-olah dapat memberikan efek ketikan saat menampilkan cerita latar. Method tersebut mengimpor library chrono dan thread karena memanfaatkan method `sleep_for()`.




```

1 void typeEffect(const std::string &text, int delayMs) {
2     for (char c : text) {
3         std::cout << c << std::flush;
4         std::this_thread::sleep_for(std::chrono::milliseconds(delayMs));
5     }
6 }

```

Method typeEffect()



```

1 std::string kalimat =
2     "Kau berdiri di hadapan gerbang kuno yang mengarah ke " + dungeonNames[rank] +
3     ", tempat di mana cahaya dunia \nluar sirna, dan hanya keberanianmu yang jadi penerang.\n\n"
4     "Di dalamnya tersebar " + to_string(totalChambers) + " chamber, masing-masing menyimpan "
5     "monster yang akan menguji \nkekuatan, akal, dan nyalimu.\n\n";

```

Variabel menyimpan kalimat

3.2. Bonus Kreasi Mandiri

3.2.1. Save and Load Characters ke Config

Selain items, shop, dan inventory dapat disimpan ke dalam config. Kami juga mengimplementasikan fitur save dan load Character untuk config. Jadi, pengguna dapat memilih character yang sudah ada dari

characters config pada awal game atau membuat character baru yang dapat disave ke dalam config characters

3.2.2. Fitur Player

Player merupakan suatu class yang menjadi manager untuk pengguna yang menghubungkan character player yang dipilih, inventory, dan items yang ada.

Daftarkan setiap kreasi tambahan yang anda buat dibagian ini serta deskripsi singkat mengenai implementasinya. Setiap kreasi tambahan yang anda buat akan dihargai jadi daftarkanlah selengkap-lengkapny. Poin ini akan dipertimbangkan sebagai nilai tambahan ke nilai akhir kelompok dan individu.

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Unit	13523023	13523023
Character	13523023	13523023
Skill dan Skillpath	13523030	13523030, 13523051, 13523023, 13523025, 13523111
Effect	13523030	13523030, 13523051, 13523023, 13523025, 13523111
Dungeon	13523111	13523030, 13523051, 13523023, 13523025, 13523111
Mobs	13523023, 13523025	13523023

Shop	13523025	13523030, 13523051, 13523023, 13523025, 13523111
Items	13523051	13523030, 13523051, 13523023, 13523025, 13523111
Damage	13523023	13523023
Inventory	13523051	13523030, 13523051, 13523023, 13523025, 13523111
Save/Load	13523030, 13523051, 13523023, 13523025, 13523111	13523030, 13523051, 13523023, 13523025, 13523111