Kelas : 02
Nama Kelompok : OOPokemon
1. 13518014 / Ignatius David Partogi
2. 13519066 / Almeiza Arvin Muzaki
3. 13519075 / Juan Louis Rombetasik
4. 13519102 / Leonardus Brandon Luwianto
5. 13519107 / Daffa Ananda Pratama Resyaly
6. 13519109 / Christian Tobing Alexandro
Asisten Pembimbing : Ikraduya Edian (13517106)

1. Diagram Kelas

Desain dari kelas kami adalah pertama dimulai dari cell, di dalam cell terdapat objek posisi yaitu x dan y untuk menandai lokasi cell tersebut. Cell memiliki atribut tipe cell, cell juga memiliki atribut Occupier jika cell ditempati, namun pada keadaan normal atribut tersebut akan bernilai nullptr.

Kemudian ada Map, kami mendesain Map dengan membuat array of object cell sebanyak MAX_X * MAX_Y untuk mengakses cell dengan koordinat tertentu kami gunakan hashing dengan nilai koordinat X ditambah koordinat Y * MAX_X, dan untuk mengakses koordinat tertentu dengan indeks array, Indeks array dilakukan modulo dengan MAX_X untuk mendapat koordinat X dan dilakukan pembagian integer (div) dengan MAX_X untuk mendapat koordinat Y.

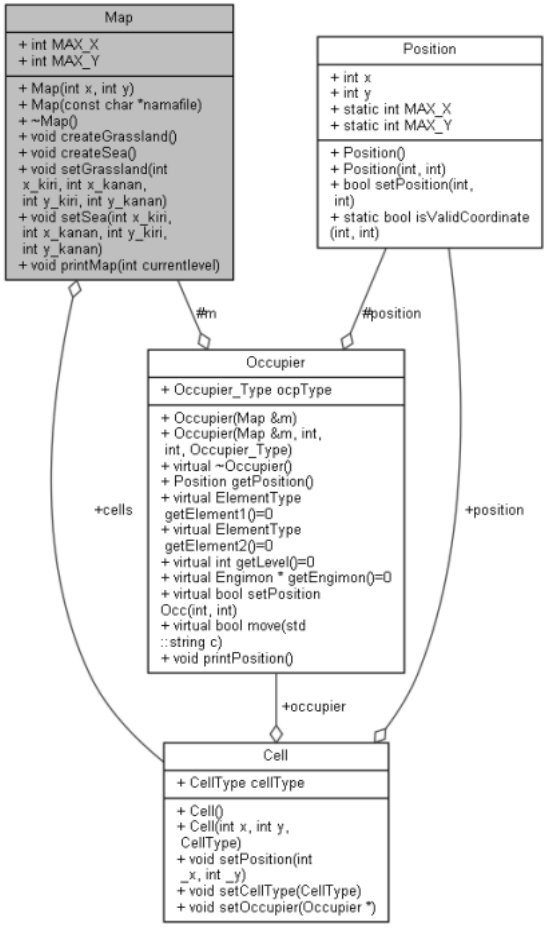
Player (Player), Wild Engimon (Enemy) dan Active Engimon (ActiveEngimon) yang masing masing merupakan kelas turunan dari Abstract class Occupier karena mereka menempati map.

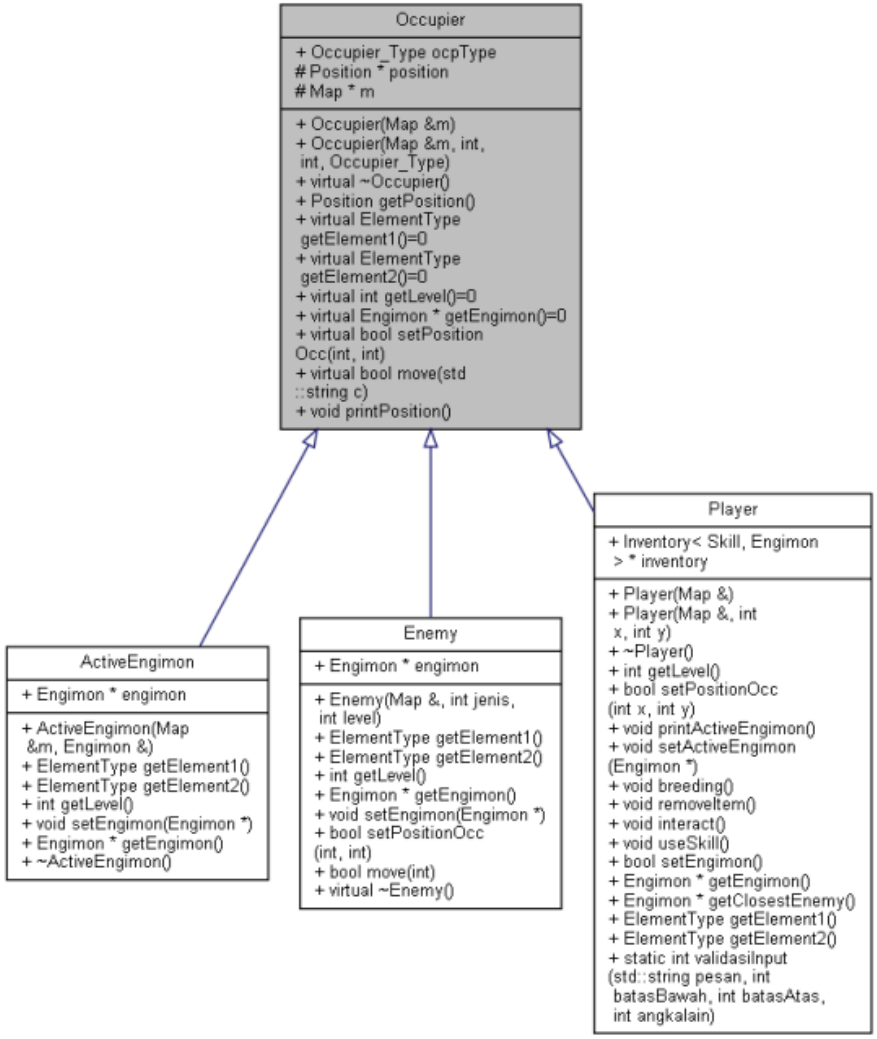
Kelas Element memiliki satu atribut yaitu dari enumerator ElementType, alasan kenapa tidak langsung memakai enumerator dan perlu ditampung dalam kelas sendiri karena terdapat method-method yang diperlukan berbagai kelas.

Engimon pada umum adalah kelas biasa, awalnya kami ingin menjadikan Engimon sebagai kelas abstrak namun tidak jadi karena akan membuat rumit pada saat proses breeding.

Skill juga adalah kelas pada umumnya

Class Map & Occupier:

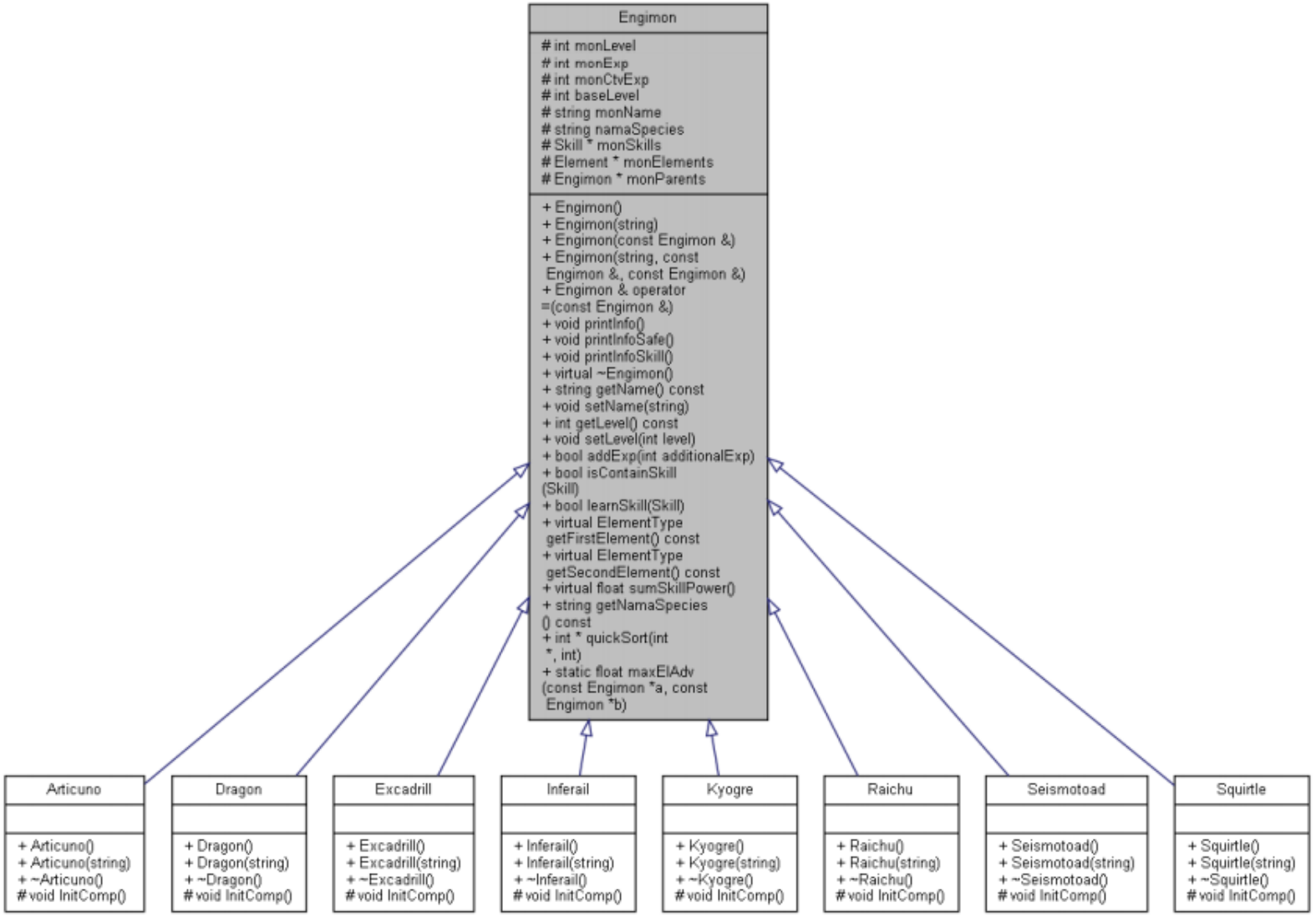


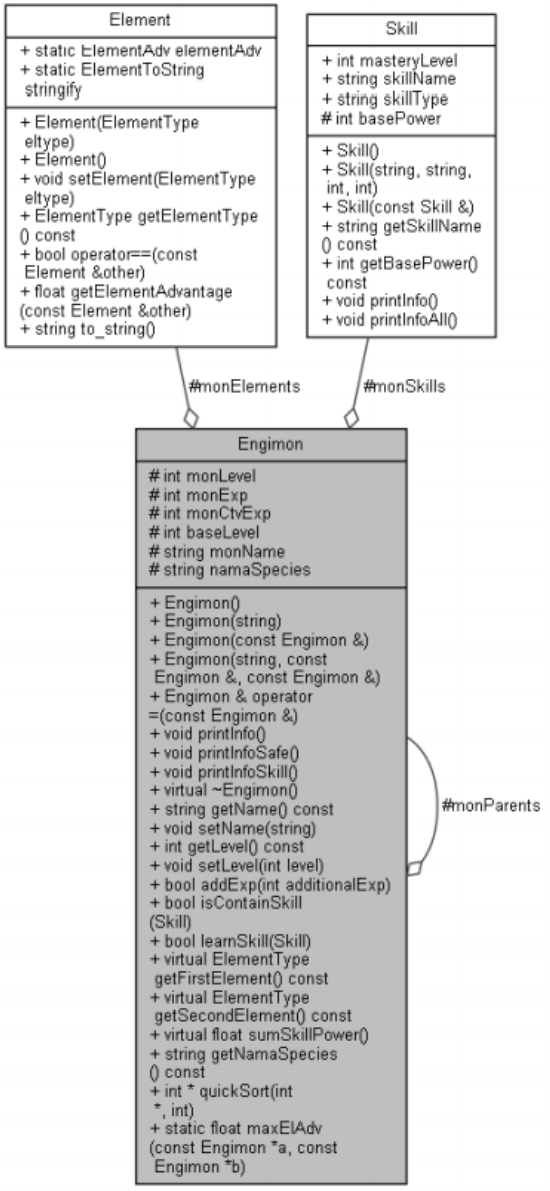


Enum OccupierType : { Player_Type, Enemy_Type, Pet_Type }

Enum CellType: { Sea_Cell, Grassland_Cell, Rancuh }

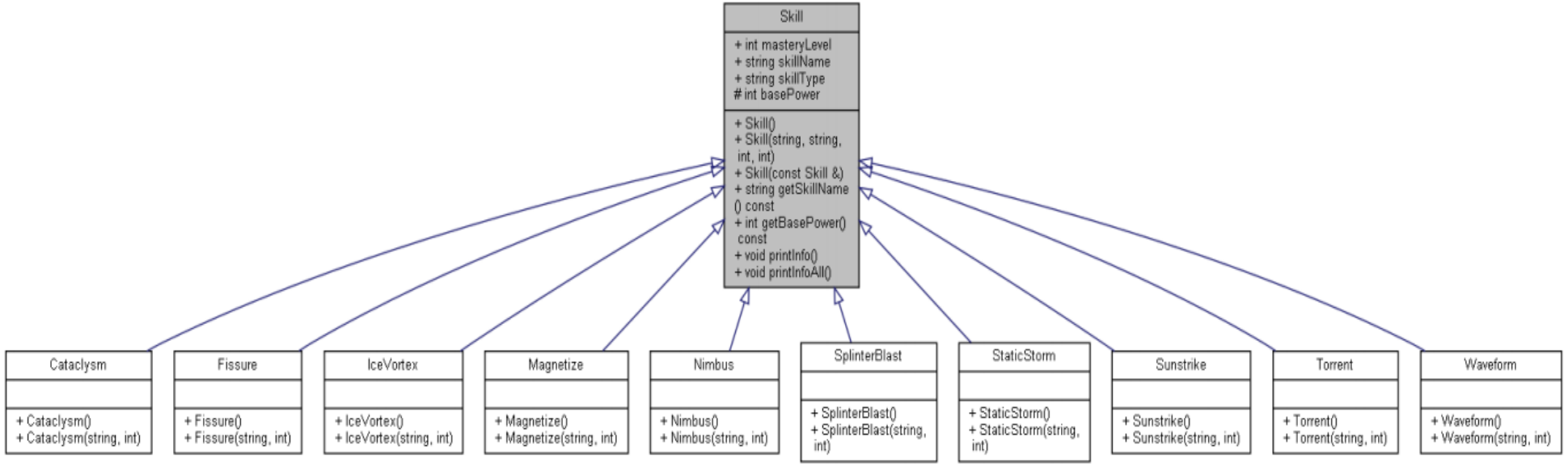
Class Engimon:




Element
+ static ElementAdv elementAdv
+ static ElementToString stringify
+ Element(ElementType eltype)
+ Element()
+ void setElement(ElementType eltype)
+ ElementType getElementType() const
+ bool operator==(const Element &other)
+ float getElementAdvantage(const Element &other)
+ string to_string()
Skill
+ int masteryLevel
+ string skillName
+ string skillType
# int basePower
+ Skill()
+ Skill(string, string, int, int)
+ Skill(const Skill &)
+ string getSkillName() const
+ int getBasePower() const
+ void printInfo()
+ void printInfoAll()
#monElements
#monSkills
Engimon
int monLevel
int monExp
int monCtvExp
int baseLevel
string monName
string namaSpecies
+ Engimon()
+ Engimon(string)
+ Engimon(const Engimon &)
+ Engimon(string, const Engimon &, const Engimon &)
+ Engimon & operator=(const Engimon &)
+ void printInfo()
+ void printInfoSafe()
+ void printInfoSkill()
+ virtual ~Engimon()
+ string getName() const
+ void setName(string)
+ int getLevel() const
+ void setLevel(int level)
+ bool addExp(int additionalExp)
+ bool isContainSkill(Skill)
+ bool learnSkill(Skill)
+ virtual ElementType getFirstElement() const
+ virtual ElementType getSecondElement() const
+ virtual float sumSkillPower()
+ string getNamaSpecies() const
+ int * quickSort(int *, int)
+ static float maxElAdv(const Engimon *a, const Engimon *b)
#monParents

Enum ElementType : { Fire, Water, Electric, Ground, Ice, None }

Class Skill:



2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Pada tugas besar ini, konsep *inheritance* atau pewarisan dan *polymorphism* digunakan dalam tiga kelas dasar (*base class*) yaitu kelas Occupier, kelas Engimon, dan kelas Skill.

2.1.1 Kelas Occupier

Terdapat tiga kelas turunan (*derived class*) yang mewarisi kelas dasar Occupier yaitu kelas Player, ActiveEngimon, dan Enemy. Berikut cuplikan kode dari kelas turunan Player yang mewarisi kelas dasar Occupier:

```
class Player : public Occupier
{
      private:
            ActiveEngimon* activeEngimon;

      public:
            Inventory<Skill, Engimon> *inventory;

            Player(Map&);
            Player(Map&, int x, int y);
            ~Player();
            int getLevel();
            bool setPositionOcc(int x, int y);
            void printActiveEngimon();
            void setActiveEngimon(Engimon*);
            void breeding();
```

```
            void removeItem();
            void interact();
            bool setEngimon();
            Engimon* getEngimon();
            Engimon* getClosestEnemy();
            ElementType getElement();
            static int validasiInput(std::string pesan, int batasBawah, int batasAtas, int angkalain);

};
#endif
```

Alasan kami memakai abstract class occupier karena mereka yang menempati cell bisa berupa player, wild engimon maupun engimon sendiri, ini juga merupakan kasus polymorphism

2.1.2 Kelas Engimon

Terdapat delapan kelas turunan (*derived class*) yang mewarisi kelas dasar Engimon yaitu kelas Articuno, Dragon, Excadrill, Raichu, Squirtle, Inferail, Kyogre, dan Seismotoad. Berikut cuplikan kode dari kelas turunan Dragon yang mewarisi kelas dasar Engimon:

```
class Dragon : public Engimon {
private:
        void InitComp();
```

```
public:
        Dragon();
        Dragon(string);
        ~Dragon();
};

#endif
```

Kelas-kelas Species banyak memiliki kesamaan informasi seperti nama, nama parent, skill, level, experience sehingga akan lebih baik jika kesamaan tersebut disimpan pada superclassnya, Engimon, dan diakses melalui inheritance. Adapun perbedaan informasi seperti tipe elemen akan diimplementasikan di kelas Species masing-masing.

2.1.3 Kelas Skill

Terdapat sepuluh kelas turunan (*derived class*) yang mewarisi kelas dasar Skill yaitu kelas Cataclysm, Fissure, IceVortex, Magnetize, Nimbus, SplinterBlast, StaticStorm, Sunstrike, Torrent, dan Waveform. Berikut cuplikan kode dari kelas turunan Cataclysm yang mewarisi kelas dasar Skill:

```
class Cataclysm : public Skill {
private:
        string species;

public:
        Cataclysm();
        Cataclysm(string, int);
```

```
};
```

Hampir serupa seperti poin Engimon sebelumnya, kelas-kelas Skill memiliki atribut yang hampir serupa, hanya terdapat perbedaan atribut mengenai jenis spesies yang eligible. Oleh karena itu, lebih dirasa efisien ketika setiap Skill yang spesifik dibuat turunan dari superclass Skill.

2.2. Method/Operator Overloading

Terdapat beberapa kelas yang menggunakan operator overloading seperti contoh dibawah ini yaitu kelas element dan terutama kelas Skill, kelas skill sangat memerlukan operator overloading karena dilakukan untuk membandingkan skill satu dengan yang lain, dan Engimon satu dengan yang lain

```
using namespace std;
enum ElementType {
      None,
      Fire,
      Water,
      Electric,
      Ground,
      Ice
};

class Element
{
      private:
            ElementType element;
      public:
```

```
            typedef std::map<std::pair<ElementType,ElementType>, float> ElementAdv;
            typedef std::map<ElementType, std::string> ElementToString;
            Element(ElementType eltype);
            Element();
            void setElement(ElementType eltype);
            ElementType getElementType() const;
            bool operator==(const Element& other);
            float getElementAdvantage(const Element& other);
            string to_string();
            static ElementAdv elementAdv;
            static ElementToString stringify;
};
#endif
```

```
class Skill {
    protected:
        int basePower;

        friend ostream& operator<<(ostream& os, const Skill& s);
    public:
        int masteryLevel;
        string skillName;
        string skillType;
        Skill();
        Skill(string, string, int, int);
        Skill(const Skill&);
        string getSkillName() const;
```

```
        int getBasePower() const;
        void printInfo();
        void printInfoAll();
        friend bool operator==(const Skill& c1, const Skill& c2);
        friend bool operator!=(const Skill& c1, const Skill& c2);
        friend bool operator>(const Skill &c1, const Skill &c2);
        friend bool operator<(const Skill &c1, const Skill &c2);
        friend bool operator>=(const Skill &c1, const Skill &c2);
        friend bool operator!=(const Skill& c1, const Skill& c2);
};
```

Operator overloading diperlukan pada element karena untuk perbandingan antara element. Operator overloading juga diperlukan pada kelas skill karena akan digunakan pada saat mensorting Skill berdasarkan element mastery terbesar.

2.3. Template & Generic Classes

Penggunaan fungsi template dan kelas generik dapat ditemukan pada kelas Bag dan kelas Inventory. Berikut cuplikan kode dari header kelas Bag,

```
template<class T>
class Bag
{
      public:
            T** listItem;
            int neff;
            bool Add(T& other);
            void printAllInfo();
            Bag();
```

```
            ~Bag();

};
```

Terdapat dua jenis Bag pada Inventory, yaitu BagSkill dan BagEngimon. Karena Skill dan Engimon merupakan dua objek yang berbeda, Bag sebagai penampung harus bisa menampung keduanya sehingga pemilihan class template untuk Bag dirasa paling tepat.

2.4. Exception

Penanganan kasus menggunakan *exception handling* diimplementasikan pada method static `validasiInput` milik kelas Player. Berikut cuplikan kode *try block* dan *catch block* dari kelas Player:

```
int Player::validasiInput(std::string pesan, int batasBawah, int batasAtas, int angkalain)
{
    int n2;
    while (true)
    {

        std::cout << pesan;
        std::cin >> n2;
        try
        {
            // JIKA gagal, reset input buffer
            if (std::cin.fail())
            {
            std::cin.clear();
```

```
            std::cin.ignore(INT_MAX,'\n');
            throw "Masukkan angka";
            }
            else if (n2 < batasBawah || n2 > batasAtas) throw "Masukkan angka yang valid";
            else if (n2 == angkalain) throw "Masukkan angka bukan sebelumnya";
            else return n2;
      }
      catch (char const* error)
      {
            std::cerr << error << std::endl;
            continue;
      }
    }
      return n2;
}
```

Validasi input memanfaatkan exception guna menghindari error yang dapat terjadi apabila terdapat kesalahan input dari pengguna.

2.5. C++ Standard Template Library

Salah satu *C++ Standard Template Library* yang digunakan dalam tugas ini adalah STD vector. Penggunaan STD vector dapat dilihat dari cuplikan kode pada header kelas Inventory berikut:

```
template <class T>
class Bag;

template <class T1,class T2>
```

```
class Inventory {

private:
      vector<T1> bagSkills;
      vector<T2> bagEngimon;
      // typedef unordered_map<Skill, int, SkillHashFunction> umapSkill;
      static unordered_map<Skill, int, SkillHashFunction> skillDict;

public:
      /* tidak ditampilkan di sini */
};
#endif
```

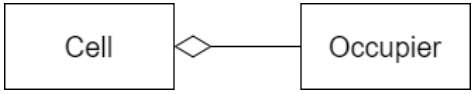
Alasan penggunaan STL std::vector pada template class Inventory ini adalah karena std::vector memiliki fungsi-fungsi default yang dapat memudahkan eksplorasi item-item di dalamnya, seperti iterator, capacity dan modifier. Mengingat item-item dalam Bag cukup sering dieksploitasi, penggunaan fungsi-fungsi default std::vector akan sangat mempermudah proses eksploitasi item tersebut.

Selain pada Inventory, STL juga digunakan untuk me-mapping element advantage antara tiap-tiap element dengan std::map

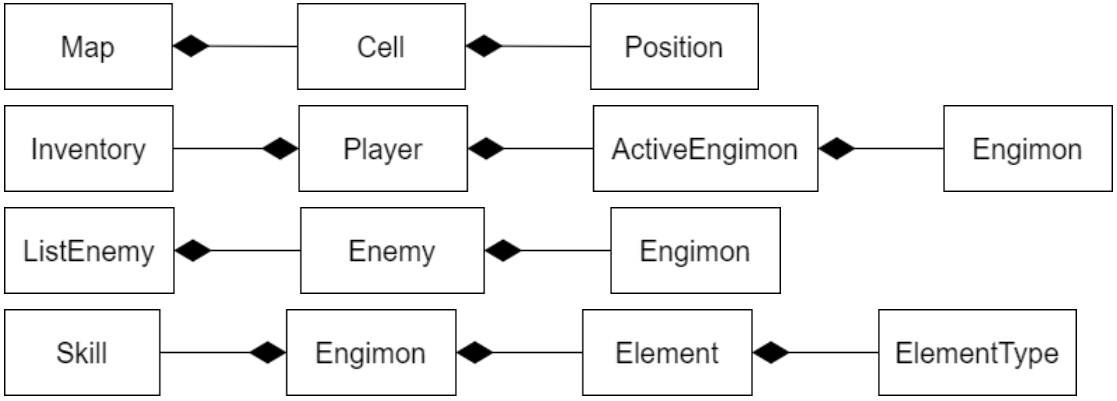
2.6. Konsep OOP lain

Abstract Base Class : Kelas Occupier (mereka yang menempati suatu cell/tile pada map)

Aggregation :



Composition :



Dan masih banyak lagi...

Modularitas :

- Penambahan jumlah musuh bisa dilakukan dengan memberi nilai pada pembuatan object list musuh
- Penambahan jenis element hanya perlu menambahkan enumerasi baru pada ElemenType dan kamus element advantage
- Penambahan jenis Cell Type hanya perlu menambahkan enumerasi pada CellType
- Jika ada “mungkin” fitur yang tidak sesuai, tidak akan mengganggu jalannya program

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Dual Element Breeding

Kasus jika breeding antara engimon dua element dengan satu element, engimon anak anak mewarisi element orang-tua yang memiliki dua element, jika antara dua element, untuk mempermudah persoalan, engimon hanya mewarisi engimon ayah (engimon pilihan pertama)

```
// Kasus double element
  else if (elbokap2 == None)
  {
      this->monElements[0].setElement(elnyokap1);
      this->monElements[1].setElement(elnyokap2);
  }
  else
  {
      this->monElements[0].setElement(elbokap1);
      this->monElements[1].setElement(elbokap2);
  }
```

3.1.2. Purely Random Wild Engimon Generation

Untuk persoalan purely random wild engimon generation, kami membuat kelas baru (List Enemy) yang berisi array of pointer Enemy untuk menspawn engimon liar dengan merandom angka dari 0-7 untuk tipe dan dari (player-level (active engimon) level) + 5) - (player-level -5) untuk levelnya.

Setiap kali player mengalahkan engimon lawan, list enemy akan men-destruct Enemy yang dikalahkan dan membuat new Enemy dengan cara yang sama.

```
ListEnemy::ListEnemy(Map& m, Player* player, int size)
{
    this->map = &m;
    this->currentplayer = player;
    this->listEnemy = new Enemy*[size];
    this->jmlhMusuh = size;
    srand(time(0));
    for (int i = 0; i < size; i++)
    {
        // Ngerandom dari 0-7 untuk tipe;
        // Ngerandom dari  (player level-5) - (player level + 5)
        this->listEnemy[i] = new Enemy(m, rand() % 8, abs(rand()
        % (player->getLevel() + 5) + (player->getLevel() - 5)));
    }
}
```

3.2. Bonus Kreasi Mandiri

Print Map with Color

Dengan Library Windows.h,

```
HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
SetConsoleTextAttribute(hConsole, colorMultiplier);
```

colorMultiplier bernilai 0-255, Foreground (Text) (0-15) + Background (0-15)*16

0 = Hitam , 1 = Biru, 2 = Hijau, 3 = Biru Muda, 4 = Merah … 15 = Putih

Contoh pada persoalan diatas, Mendapatkan warna text ‘F’ merah pada background biru muda (Element Fire pada Sea_Cell), fungsi print map akan mengecek cell tersebut apakah ditempati atau tidak lalu jika ya akan mengecek jenis Elemen pada cell tersebut, jika element Fire, color multiplier akan di set menjadi 4 dan lalu menambahkannya dengan 3x16 untuk backgroundnya.

Map Creation

```
class Map
{
	public:
		int MAX_X, MAX_Y;

		Cell* cells;

		Map(int x, int y);
		Map(const char* namafile);
		~Map();
		// Membuat Grassland sesuai luas MAX_Y * MAX_X
		void createGrassland();

		// Membuat Sea sesuai luas MAX_Y * MAX_X
		void createSea();

		// Membuat Grassland sesuai area
		void setGrassland(int x_kiri, int x_kanan, int y_kiri, int y_kanan);

		// Membuat Grassland sesuai area
		void setSea(int x_kiri, int x_kanan, int y_kiri, int y_kanan);
```

```
            void printMap(int currentlevel);

};
```

Adanya method setGrassland dan setSea adanya untuk membuat cell Grassland dan Sea sesuai area, tidak ada throw exception karena ada pengecekan koordinat di dalam prosedur tersebut.

Adanya fungsi di atas memungkinkan kami untuk membuat method random map creation sehingga ketika bermain, map akan di generate secara random. Dikarenakan waktu yang singkat, kami tidak sempat membuat method random map creation

4. Pembagian Tugas

Modul (dalam poin spek)	Designer	Implementer
1.a-f	13518014 13519075 13519107 13519109	13518014 13519075 13519107 13519109
2.a-c	13518014 13519102 13519107	13518014 13519066 13519102 13519107
3.a	13519066 13519102 13519107	13519066 13519075 13519102
3.b	13519109	13519075

		13519109
3.c	13519075 13519109	13519075 13519109
4.a-f	13519075	13519075 13519107 13519109
5.a-f	13519066 13519075 13519102 13519107	13519066 13519075 13519102 13519107
6.a-j	13519075 13519109	13518014 13519066 13519075 13519102 13519107 13519109