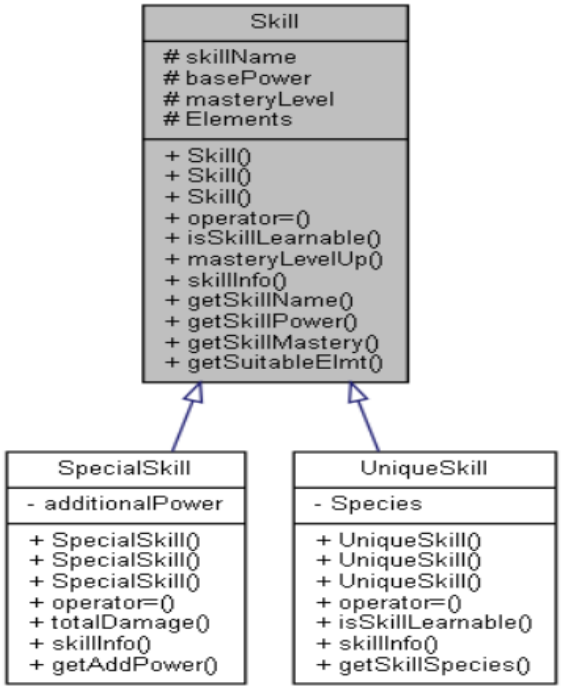
Kelas : 04
Nama Kelompok : POO

1. 13519169 / David Owen Adiwiguna
2. 13519180 / Karel Renaldi
3. 13519185 / Richard Rivaldo
4. 13519191 / Kevin Ryan
5. 13519215 / Leonard Matheus
6. 13519217 / Hughie Alghaniyyu Emiliano

Asisten Pembimbing : Willy Santoso

1. Diagram Kelas



lic Member Functions

	Skill ()
	Skill (string name, int power, vector< string > Elmts, int mastery=1)
	Skill (const **Skill** &skill)
Skill &	**operator=** (const **Skill** &skill)
virtual const bool	**isSkillLearnable** (string engimonElmt)
void	**masteryLevelUp** (int bpIncrease)
virtual const void	**skillInfo** ()
string	**getSkillName** ()
const int	**getSkillPower** ()
const int	**getSkillMastery** ()
t vector< string >	**getSuitableElmt** ()

tected Attributes

string	**skillName**
int	**basePower**
int	**masteryLevel**
or< string >	**Elements**

Public Member Functions

	SpecialSkill ()
	SpecialSkill (string name, int **basePower**, vector< string > elmts, int addPower, int mastery=1)
	SpecialSkill (const **SpecialSkill** &special)
SpecialSkill &	**operator=** (const **SpecialSkill** &special)
const int	**totalDamage** ()
const void	**skillInfo** ()
const int	**getAddPower** ()

▸ Public Member Functions inherited from **Skill**

Private Attributes

int	**additionalPower**

Public Member Functions

	UniqueSkill ()
	UniqueSkill (string name, int **basePower**, vector< string > elmts, string species, int mastery=1)
	UniqueSkill (const **UniqueSkill** &unique)
UniqueSkill &	**operator=** (const **UniqueSkill** &unique)
const bool	**isSkillLearnable** (string engimonElmt, string species)
const void	**skillInfo** ()
const string	**getSkillSpecies** ()

▸ Public Member Functions inherited from **Skill**

Private Attributes

string	**Species**

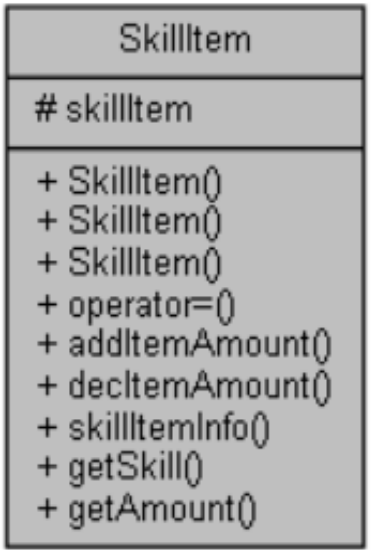
Kelas *skill* didesain sebagai sebuah kelas dasar yang secara kontekstual merupakan *skill* yang bersifat reguler atau biasa. Di dalam kelas *skill* didefinisikan atribut-atribut yang dimiliki kelas ini, seperti nama, *base power*, *mastery level*, dan daftar elemen yang bisa mempelajari *skill* tersebut. Untuk menambah variasi *skill* berdasarkan jenisnya, maka kemudian dari kelas ini akan diturunkan dua buah kelas anak, yaitu *Special Skill* dan *Unique Skill*. *Special Skill* akan memiliki atribut tambahan berupa *additional power*, sedangkan *Unique Skill* memiliki spesies tertentu yang bisa mempelajarinya.

Public Member Functions

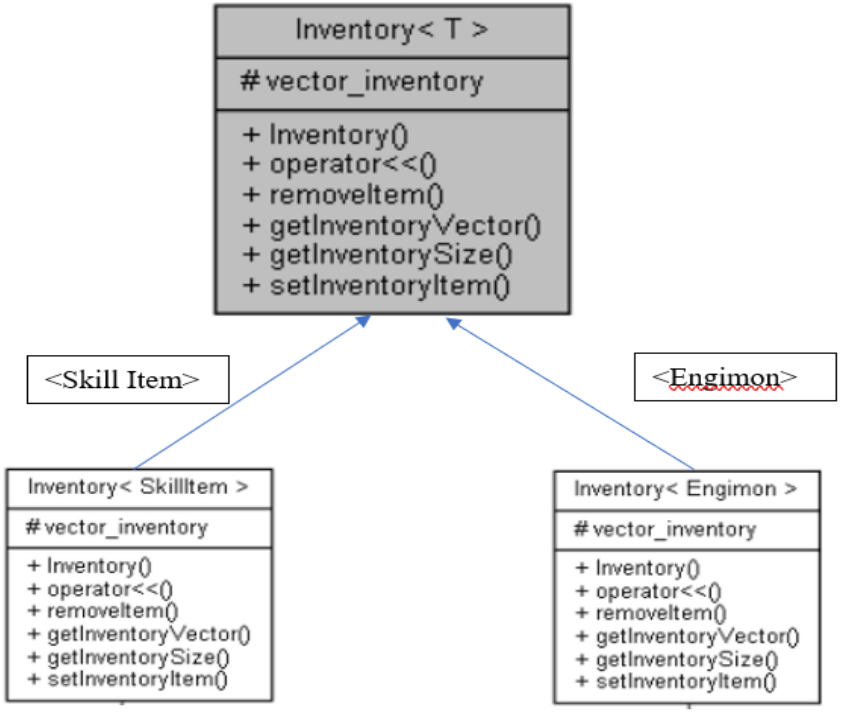
	SkillItem ()
	SkillItem (**Skill** skill, int amount=1)
	SkillItem (const **SkillItem** &item)
SkillItem &	**operator=** (const **SkillItem** &item)
void	**addItemAmount** (int amount=1)
void	**decItemAmount** (int amount=1)
const void	**skillItemInfo** ()
Skill	**getSkill** ()
const int	**getAmount** ()

Protected Attributes

pair< **Skill**, int * >	**skillItem**



Kelas *Skill Item* didesain sebagai sebuah kelas yang memiliki atribut berupa *pair*. *Pair* akan menyimpan *Skill* yang disimpan di dalam *Skill Item* tersebut lengkap dengan atribut jumlah dari setiap *Skill*. Dengan desain demikian, maka nantinya *Inventory* hanya perlu memasukkan *Skill Item* ke dalam *Container* tanpa perlu mencatat secara eksplisit jumlah dari setiap *Skill Item*. Kesulitan dalam perancangan kelas *Skill* dan *Skill Item* adalah menentukan beberapa *method* yang sebenarnya tidak terlalu tepat jika dimasukkan ke dalam *method skill* ataupun *skill item*, yaitu misalnya *learn* dan *replace*. Oleh karena itu, *learn* dan *replace* ini dimasukkan ke dalam kelas lain, yaitu *Player* karena dari kelas ini, dapat dilakukan pengecekan syarat-syarat terhadap atribut-atribut *player* sesuai dengan prinsip kerja *learn* dan *replace*. Penggunaan *pointer to integer* untuk mencatat jumlah *amount* tiap item dilakukan supaya pemrogram bisa melakukan pengurangan atau penambahan jumlah dengan mengubah objek secara langsung, bukan hanya melakukan *shallow copy* sehingga perubahan yang terjadi bersifat aktual.



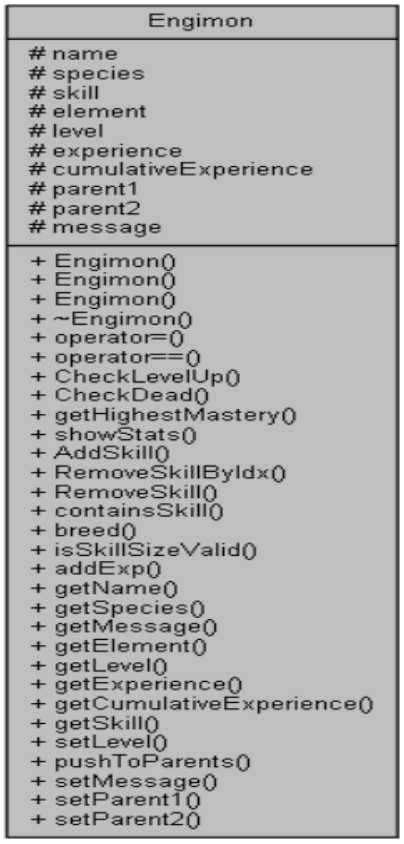
Kelas *Inventory* merupakan kelas yang dibuat dengan menggunakan konsep generik. Dengan adanya kelas generik sebagai *template* ini, maka nantinya pemrogram bisa melakukan penyimpanan jenis *item* yang berbeda di dalam *inventory* pemain. Dalam hal ini, jenis *item* tersebut merupakan *Skill Item* dan *Engimon*. Di dalam *Inventory* ini akan didefinisikan *method* umum yang bisa digunakan untuk mengolah sebuah *item* ketika dimasukkan dan diambil dari *inventory.* Hal ini akan memudahkan pemrogram dalam memrogram kelas *Inventory* untuk digunakan pada dua buah jenis *item* yang berbeda, namun memiliki perlakuan yang sama terhadap keduanya.

Public Member Functions

	Engimon ()
	Engimon (string nama, string species, vector< string > element)
	Engimon (const Engimon &engimon)
	~Engimon ()
Engimon &	operator= (const Engimon &)
bool	operator== (const Engimon &engimon) const
void	CheckLevelUp (Engimon engimon)
bool	CheckDead (Engimon engimon)
const Skill	getHighestMastery ()
void	showStats ()
void	AddSkill (Skill skill)
void	RemoveSkillByIdx (int skillidx)
void	RemoveSkill (Skill skill)
bool	containsSkill (list< Skill > listSkill, string skillName)
Engimon	breed (Engimon anotherEngimon)
bool	isSkillSizeValid (Engimon engimon)
void	addExp (int exp)
string	getName ()
string	getSpecies ()
string	getMessage ()
vector< string >	getElement ()
int	getLevel ()
int	getExperience ()
int	getCumulativeExperience ()
list< Skill >	getSkill ()
void	setLevel (int level)
void	pushToParents (Engimon parent)
void	setMessage (string newMessage)
void	setParent1 (string engimon1)
void	setParent2 (string engimon2)

Protected Attributes

string	name
string	species
list< Skill >	skill
vector< string >	element
int *	level
int *	experience
int *	cumulativeExperience
string *	parent1
string *	parent2
string	message



Public Member Functions beserta Protected Attributes dari Engimon. Dalam class Engimon, juga dilengkapi method breeding yang dapat digunakan untuk mengawinkan 2 engimon dan menghasilkan anak dengan elemen yang sama dengan parentsnya.

Map Class Reference

Public Member Functions

void	**printMap** ()
int	**getmaxEngimonLiar** ()
Position	**getplayerPosition** ()
vector< pair< Position, Engimon > >	**getengimonLiar** ()
int	**getplayerPositionX** ()
int	**getplayerPositionY** ()
int	**getxmax** ()
int	**getymax** ()
void	**setplayerPosition** (int _x, int _y)
Position	**getactiveEngimonPosition** ()
int	**getactiveEngimonPositionX** ()
int	**getactiveEngimonPositionY** ()
void	**setactiveEngimonPosition** (int _x, int _y)
void	**addEngimonLiar** ()
void	**removeEngimonLiar** (Engimon engimon)
void	**moveAllEngimonLiar** ()
void	**setAlphabet** (Engimon engimonbaru, int coorY, int coorX)
void	**bacaMap** (string namafile)

Position Class Reference

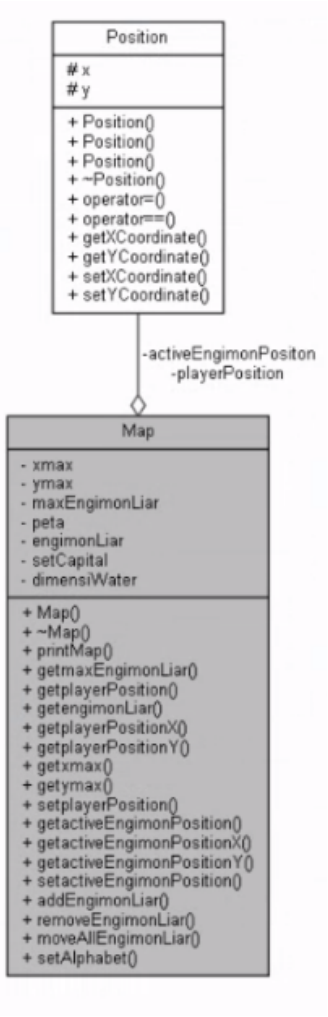
`#include <Position.hpp>`

Public Member Functions

	Position ()
	Position (int coordinate_x, int coordinate_y)
	Position (const Position &)
	~Position ()
Position &	**operator=** (const Position &pos)
bool	**operator==** (const Position &pos) const
int	**getXCoordinate** ()
int	**getYCoordinate** ()
void	**setXCoordinate** (int _x)
void	**setYCoordinate** (int _y)

Protected Attributes

int	**x**
int	**y**



Kelas Map menggunakan Kelas Position untuk menyimpan setiap posisi yang dihasilkan oleh player, aktif Engimon, dan *wild* engimon. Kelas ini akan diakses untuk memberikan tampilan pergerakan engimonnya.

Public Member Functions

	Battle (Player player, Engimon engimonPlayer, Engimon engimonWild)
	~Battle ()
int	getlevelEngimonPlayer ()
int	getlevelEngimonWild ()
int	getMultiplierEngimonPlayer ()
int	getMultiplierEngimonWild ()
int	getskillPowerPlayer ()
int	getskillPowerWild ()
float	getTotalPowerPlayer ()
float	getTotalPowerWild ()
string	getWinner ()
void	setMultiplierEngimon ()
void	setSkillPower ()
void	setTotalPower ()
void	setElement ()
float	checkMultiplier (string elemen1, string elemen2)
void	showTotalPower ()
void	doBattle ()
SkillItem	getRandomSkill (vector< SkillItem > fireItems, vector< SkillItem > waterItems, vector< SkillItem > electricItems, vector< SkillItem > groundItems, vector< SkillItem > iceItems)

Private Attributes

Player	player
Engimon	engimonPlayer
Engimon	engimonWild
int	levelEngimonPlayer
int	levelEngimonWild
float	multiplierEngimonPlayer
float	multiplierEngimonWild
int	skillPowerPlayer
int	skillPowerWild
float	totalPowerPlayer
float	totalPowerWild
string	winner
string	loser

Berikut *link* gambar diagram kelas secara utuh (karena gambar terlalu besar sehingga tidak bisa dicantumkan dengan jelas dalam laporan ini). *Drive class diagram*: https://drive.google.com/file/d/1HCFIk8qBCOJuhnpMqCDIVy7eBNXSFido/view?usp=sharing.

Kelas battle didesain menjadi sebuah kelas yang menyimpan atribut-atribut yang berasal dari player, engimon player, dan engimon liar dan fungsi-fungsi yang digunakan untuk melakukan battle. Kelas ini digunakan untuk melakukan battle antara engimon player dan engimon liar, seperti menghitung besarnya power dari engimon player dan engimon liar, menentukan siapa yang menang dalam pertarungan, dan memberikan player random skill item.

Public Member Functions

	Player ()
	Player (**Engimon** activeEngimon)
Map	**getMap** ()
Inventory< **Engimon** >	**getInventoryEngimon** ()
Inventory< **SkillItem** >	**getInventorySkillItem** ()
Engimon	**getActiveEngimon** ()
void	**addSkillItem** (**SkillItem** newSkillItem)
void	**addEngimon** (**Engimon** newEngimon)
void	**moveUp** ()
void	**moveDown** ()
void	**moveLeft** ()
void	**moveRight** ()
void	**showOwnedEngimon** ()
void	**showStatsEngimon** (string engimonName)
void	**showActiveEngimon** ()
void	**swapActiveEngimon** (string newEngimonName)
void	**interactWithEngimon** ()
void	**showOwnedItems** ()
void	**doBreed** (string firstEngimonName, string secondEngimonName)
void	**doBattle** ()
void	**learnSkill** (string SkillName)
void	**replaceSkillItem** (int replacedSkillIdx, **Skill** newSkill)

Private Attributes

Engimon	**ActiveEngimon**
Inventory< **Engimon** >	**inventoryEngimon**
Inventory< **SkillItem** >	**inventorySkillItem**
Map	**map**

Static Private Attributes

static const int	**MAX_CAPACITY** = 20

Kelas *Player* merupakan kelas yang berisikan *Command* yang bisa dimainkan atau digunakan oleh pengguna. Secara umum, kelas ini akan menggunakan kelas-kelas lainnya dalam menjalankan *Command* tersebut. Dengan menggunakan kelas ini, maka nantinya pembuatan program utama bisa menjadi lebih mudah, karena semua *Command* yang ada di dalam program tinggal dipanggil dari objek *Player* yang telah dikonstruksi sebelumnya. Di dalam kelas ini juga terdapat sebuah atribut statik sebagai penanda batas maksimum untuk *Inventory* pemain.

Selain kelas-kelas utama di atas, juga ditambahkan sebuah *header* berisi implementasi fungsi-fungsi antara yang digunakan untuk memudahkan pembuatan *database Skill* dan *Skill Items* yang digunakan di dalam permainan. Hal ini juga memungkinkan penggunaan *Skill* ini untuk digunakan di kelas yang lain, seperti misalnya *Battle* yang juga akan membutuhkan *Skill* dalam kasus menambahkan *Skill Items* saat memenangkan *battle*.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

```
class UniqueSkill: public Skill{
    private:
        // Species of the Unique Ski
        string Species;
    public:
        // Default Constructor
        UniqueSkill();
```

```
class SpecialSkill: public Skill{
    private:
        // Additional Power of The Special Skill
        int additionalPower;
    public:
        // Default Constructor
        SpecialSkill();
        // User-Defined Constructor
```

Variasi *Skill* yang ada di dalam program sebenarnya bisa disatukan ke dalam sebuah kelas. Namun, untuk memudahkan pemrogram dalam mengelompokkan *skill* ini, maka *Skill* dibuat sebagai sebuah *parent class* yang menurunkan *skills* lain berdasarkan klasifikasinya. Dengan menggunakan konsep pewarisan pada *Skill*, maka turunan dari *Skill* bisa diberikan perilaku yang berbeda-beda. Selain itu, penggunaan metode *Inheritance* ini memungkinkan pengembangan *skill* lebih lanjut. Dalam implementasinya, kedua kelas turunan ini juga akan memanggil *method* miliki kelas *Skill* sebagai *parent class*.

```
class SkillItem
{
protected:
    // Map to contain skill item and number of the items
    pair<Skill, int> skillItem;

public:
    // Default Constructor, fill with default value
    SkillItem();
    // User-Defined Constructor
    SkillItem(Skill skill, int amount = 1);
```

Konstruksi *Skill Item* dilakukan dengan menerima sebuah argumen bertipe kelas *Skill* dan sebuah integer yang menyatakan jumlah yang diinginkan dari *Skill Item* tersebut. Artinya, karena *skill* terdiri dari beberapa jenis, maka konstruksi *Skill Item* akan menggunakan konsep Polymorphism, menyesuaikan dengan kelas atau jenis *Skill* yang dibuat *item*-nya.

2.2. Method/Operator Overloading

```
SpecialSkill& SpecialSkill::operator=(const SpecialSkill& special){
    Skill::operator=(special);
    this->additionalPower = special.additionalPower;

    return *this;
}
```

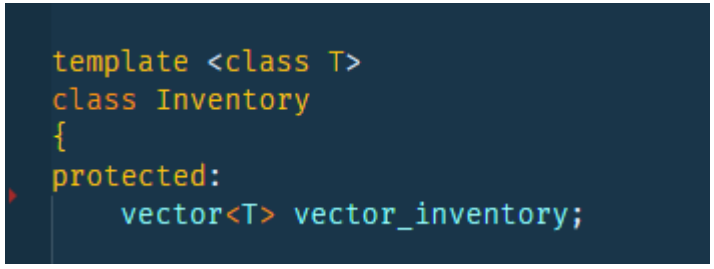
```
Inventory<T> &operator<<(const T &inventory_item)
{
    this->vector_inventory.push_back(inventory_item);

    return *this;
}
```

Untuk *operator* yang didefinisikan kepada *child class* dari *Skill*, akan dilakukan *operator overloading* untuk operator (=), menyesuaikan dengan keperluan dari *child class* ini. Selain itu pada kelas inventory digunakan *operator overloading* untuk operator (<<), operator ini diimplementasikan untuk method memasukan sebuah elemen kedalam objek hasil instansiasi kelas inventory. Konsep operator overloading dipilih karena dalam kasus method seperti menambahkan / mengurangi suatu atribut penyimpanan lebih cocok menggunakan konsep ini karena konsep ini juga mendukung chaining dan juga konsep ini lebih *readable* dibandingkan dengan method biasa.

2.3. Template & Generic Classes

Kelas *inventory* menerapkan konsep dari *template & generic classes*. Konsep *template & generic classes* pada kelas inventory ini digunakan pada atribut *vector_inventory* untuk menyimpan elemen" dengan tipe generic. Konsep *template & generic classes* ini sangat cocok untuk kasus kelas ini karena implementasinya nanti lebih baik dan lebih efisien karena didalam spek ini minimal terdapat 2 inventory dengan tipe engimon

dan skill dimana perilaku inventory dari masing" tipe sama dan hanya berbeda pada tipe dari elemen yang disimpannya oleh karena itu konsep ini sangat tepat digunakan. Keuntungan menggunakan konsep ini juga kita dapat memperkecil kompleksitas dimana kita dapat menghindari konsep inheritance yang mana membuat kelas baru dimana hal itu akan meningkatkan kompleksitas dari suatu program. Jadi *konsep template & generic class* ini juga bisa mereduksi kompleksitas terutama kompleksitas memori sehingga program lebih efisien lagi. Berikut cuplikan kode penggunaan generic class



```
template <class T>
class Inventory
{
protected:
    vector<T> vector_inventory;
```

2.4. Exception

Kelas *inventory* menerapkan konsep exception yang mana konsep ini digunakan pada method operator overloading. Konsep ini digunakan pada method remove item berdasarkan indeks juga digunakan konsep exception ini sehingga jika indeks yang diberikan salah / *out of range* maka method akan melakukan throw error. Konsep ini digunakan karena konsep ini lebih *best practice* dibandingkan hanya menggunakan if saja dan konsep ini juga lebih mudah dilakukan *debugging* dimana kita tinggal menggunakan keyword *try catch* saja pada main driver yang kita buat dan error dari throw tersebut akan di catch jika memang method melakukan throw error. Berikut cuplikan kode untuk penggunaan exception pada method yang ada di kelas player.

```
void removeItem(int index)
{
    if (index > this->vector_inventory.size() - 1)
        throw "index out of bound";
    else
        this->vector_inventory.erase(this->vector_inventory.begin() + index);
}
```

Kelas Player juga menerapkan konsep exception. Konsep ini digunakan karena kemungkinan saat memasukan sebuah elemen kedalam *vector* jumlah elemen dalam *vector* sudah maksimum sehingga tidak bisa lagi menampung elemen maka dari itu exception throw digunakan pada method ini. Berikut cuplikan kode untuk penggunaan exception pada method yang ada di kelas player. Secara umum, *exception* yang digunakan akan mengirimkan sebuah pesan kesalahan yang sesuai dengan pelanggaran yang dilakukan.

```
// Inventory
void Player::addSkillItem(SkillItem newSkillItem)
{
    if (this->inventoryEngimon.getInventorySize() + this->inventorySkillItem.getInventorySize() + 1 > Player::MAX_CAPACITY)
    {
        throw "Maximum capacity";
        return;
    }

    int i = 0;
    while (i < this->inventorySkillItem.getInventorySize())
    {
        if (this->inventorySkillItem.getInventoryVector()[i].getSkill().getSkillName() == newSkillItem.getSkill().getSkillName())
        {
            cout << "Menambahkan Skill Item untuk " << newSkillItem.getSkill().getSkillName() << endl;
            this->inventorySkillItem.getInventoryVector()[i].addItemAmount();
            return;
        }
        i++;
    }

    this->inventorySkillItem << newSkillItem;
    cout << "Berhasil menambahkan skill item baru!" << endl;
}
```

2.5. C++ Standard Template Library

```
class Skill
{
protected:
    // Skill name
    string skillName;
    // Power
    int basePower;
    // Mastery Level
    int masteryLevel;
    // Elements
    vector<string> Elements;
```

```
class SkillItem
{
protected:
    // Map to contain skill item and number of the items
    pair<Skill, int> skillItem;
```

Pada implementasi *Skill* dan *Skill Item,* digunakan dua buah jenis C++ STL *Containers*, yaitu *Vector* dan *Pair* (meskipun lebih tepat apabila dikatakan sebagai sebuah kelas layaknya *Tuple* atau bahkan *struct*). *Vector* pada kelas *Skill* digunakan untuk menampung nilai-nilai elemen berupa *string* yang bisa mempelajari *skill* tersebut. Hal ini memungkinkan sebuah *skill* bisa memiliki lebih dari satu buah elemen yang bisa mempelajari *skill* tersebut.

Dalam hal ini, *Pair* digunakan pada kelas *Skill Item* untuk mencatat *Skill* dari *Skill Item* tersebut serta jumlah dari *Skill Item* untuk *Skill* tersebut di dalam *inventory* nantinya. Struktur ini dibuat demikian untuk memudahkan pengimplementasian *inventory*, sehingga *inventory* hanya perlu memasukkan instansi dari kelas *Skill Item* tanpa perlu mencatat jumlah dari setiap *Skill Item* lagi secara eksplisit.

```
class Inventory
{
protected:
    vector<T> vector_inventory;
```

Selain itu pada implementasi kelas *Inventory*, digunakan satu buah jenis C++ STL *Containers*, yaitu *Vector*. *Vector* pada kelas ini digunakan untuk menampung elemen-elemen dari objek inventory yang memiliki tipe generic. *Container Vector* dipilih karena fungsi / method bawaan dari *vector* ini sangat bervariasi dan juga *vector* ini memungkinkan menyimpan elemen dengan tipe yang serupa. Juga terdapat penggunaan kombinasi *container* dalam *container* seperti yang terlihat pada pembuatan peta permainan untuk menampung nilai pasangan koordinat X dan Y.

2.6. Konsep OOP Lain

```
class SkillItem
{
protected:
    // Map to contain skill item and number of the items
    pair<Skill, int> skillItem;
```

Salah satu konsep OOP lain yang digunakan di dalam pengerjaan Tugas Besar kali ini adalah konsep *Composition*. Konsep ini digunakan dalam pembuatan kelas *Skill Item*. Dalam hal ini, *Skill Item* dianggap kurang cocok dalam menggunakan konsep *Inheritance*, meskipun dengan menggunakan pewarisan kode yang dibuat bisa menjadi lebih praktis. Namun, karena *Skill Item* lebih tepat menggambarkan hubungan *is-a* dengan *Skill*, maka konsep *Composition* yang kemudian dipakai, bukan *Inheritance*. Selain itu, juga digunakan *constructor* dengan parameter yang nilainya diisi secara *default*, terutama untuk *mastery level* dan *amount* dari *Skill Item*, untuk memudahkan pembuatan instansi dari kelas. Hal lain yang perlu diketahui mengenai pembuatan instansi *skill* adalah bahwa *Skill* yang ada di dalam program utama dibuat sebagai objek biasa yang tidak dimasukkan ke dalam *Container* sama sekali.

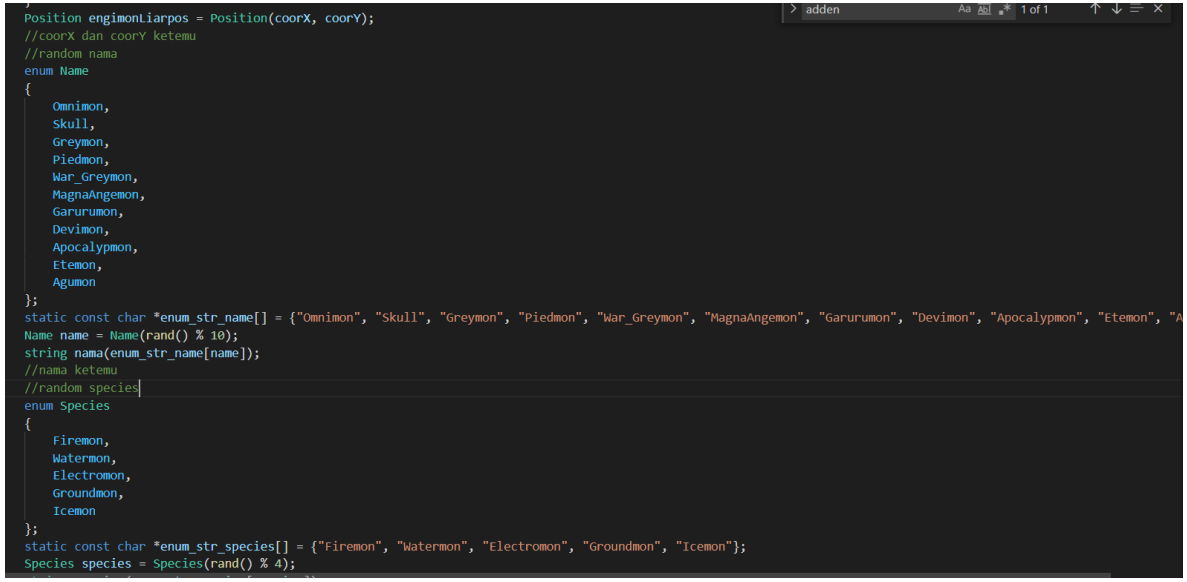
Karena *Skill* terdiri dari berbagai macam jenis yang merupakan turunan dari *parent class Skill*, maka nantinya untuk menampilkan informasi tambahan mengenai atribut yang ada dari jenis-jenis *skill* yang berbeda, akan digunakan *method overriding* untuk perilaku-perilaku yang perlu diganti dan disesuaikan dengan ekstensi dari *parent class* nya, misalnya penampilan informasi mengenai atribut tambahan yang hanya ada di kelas turunan tersebut.

Selain itu, di dalam pembuatan program ini juga digunakan beberapa konsep Pemrograman Berorientasi Objek lainnya, seperti misalnya pembuatan objek yang memiliki komposisi berupa objek lain dengan menggunakan *Constructor Initialization List*, penggunaan *method static* untuk atribut yang dimiliki suatu kelas secara keseluruhan, konsep *Shallow Copy* dan *Deep Copy* untuk mengubah nilai aktual suatu objek secara langsung, dan penggunaan *keyword const* dan *virtual* yang sebisa mungkin disesuaikan dengan konsep dan kebutuhan program dalam mengolah atribut objek tersebut.

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Purely Random Wild Engimon Generation



```
Position engimonLiarpos = Position(coorX, coorY);
//coorX dan coorY ketemu
//random nama
enum Name
{
    Omnimon,
    Skull,
    Greymon,
    Piedmon,
    War_Greymon,
    MagnaAngemon,
    Garurumon,
    Devimon,
    Apocalypmon,
    Etemon,
    Agumon
};
static const char *enum_str_name[] = {"Omnimon", "Skull", "Greymon", "Piedmon", "War_Greymon", "MagnaAngemon", "Garurumon", "Devimon", "Apocalypmon", "Etemon", "A
Name name = Name(rand() % 10);
string nama(enum_str_name[name]);
//nama ketemu
//random species
enum Species
{
    Firemon,
    Watermon,
    Electromon,
    Groundmon,
    Icemon
};
static const char *enum_str_species[] = {"Firemon", "Watermon", "Electromon", "Groundmon", "Icemon"};
Species species = Species(rand() % 4);
```

Wild Engimon akan dimasukkan dengan turn random dengan peluang 50% menggunakan random generator. Selain itu, untuk penentuan species, nama, dan elemen, program akan mengenerate posisi secara random juga. Apabila posisi yang dihasilkan berada pada area grassland, maka hanya element yang eligible saja yang berada di grassland begitu pula yang terjadi pada water. Untuk pemilihan spesies akan dilakukan secara random juga.

3.1.2. Unit Testing Implementation

```
void test_constructor_skill(){
    // Init ELements
    vector<string> elmt = {"Fire", "Water", "Wind", "Earth"};

    // Construct the object
    Skill s1;
    Skill s2("Autokill", 100000, elmt);

    // Print Skill Info
    s1.skillInfo();
    cout << "--------------------" << endl;
    s2.skillInfo();
    cout << "--------------------" << endl;

    assert(s2.getSkillName() == "Autokill");
    assert(s2.getSkillMastery() == 1);
    assert(s1.getSkillPower() == 0);
    assert(s2.getSuitableElmt() == elmt);
}
```

Telah diimplementasikan *Unit Testing* untuk kelas Engimon, *Skill* dan *Skill Item*. *Unit Testing* ini diimplementasikan secara bersamaan dengan pengecekan kode program dan *testing* dilakukan dengan pengecekan pada *method* kelas yang sering digunakan, seperti konstruktor dan metode-metode lain kelas tersebut yang akan banyak digunakan dalam program utama. Implementasi *Unit Tests* dibuat dengan menggunakan *Assertion* dan mengecek atribut-atribut dari objek sesuai dengan perlakuan *method* yang diberikan.

3.2. Bonus Kreasi Mandiri

```
vector<Skill> createElectricSkills(){
    UniqueSkill Thunderslash("Thunderslash", 20, vector<string> {ELECTRIC}, "Electromon");
    Skill PlasmaFist("PlasmaFist", 10, vector<string> {ELECTRIC});
    Skill PikaPapow("Pika Papow", 12, vector<string> {ELECTRIC});
    Skill Flamethrower("Flame Thrower", 12, vector<string> {FIRE, ELECTRIC});
    Skill StokedSparksurfer("Stoked Sparksurfer", 14, vector<string> {ELECTRIC, GROUND});
    SpecialSkill BOLT("10,000,000 Volt Thunderbolt", 75, vector<string>{ELECTRIC}, 10);
    SpecialSkill ForkedLightning("Forked Lightning", 60, vector<string>{ELECTRIC}, 10);
```

Di dalam spesifikasi Tugas Besar pada poin 2.a *Skill* dikatakan bahwa "*Skill memiliki berbagai macam (banyak jenis skill).*" Hal ini cukup rancu dan kurang terlalu jelas dalam menggambarkan spesifikasi tersebut. Oleh karena itu, pemrogram kemudian memutuskan untuk mendesain sistem klasifikasi kelas *Skill* dengan spesifikasi sebagai berikut. *Skill* dapat dikelompokkan berdasarkan elemen dan spesies yang bisa mempelajari dan memiliki *skill* tersebut serta klasifikasi terhadap tipe atau tingkatan *skill*. Ketiga hal ini saling berkaitan satu sama lain dan dirancang sebagai bagian dari kelas *Skill*.

Klasifikasi berdasarkan spesies dirancang sebagai *Unique Skill* yang nantinya hanya bisa dimiliki oleh spesies tertentu saja. Dalam hal ini, satu *unique skill* dirancang hanya bisa dimiliki khusus satu spesies saja. Selain itu, *skill* diklasifikasikan berdasarkan elemen yang bisa menggunakan *skill* tersebut, namun untuk *skill* tertentu bisa saja dipelajari oleh lebih dari satu elemen. Berdasarkan tipenya, *skill* bisa dikelompokkan menjadi *skill* biasa, *Unique Skill*, dan *Special Skill*.

Secara keseluruhan, *Unique Skill* dapat dikatakan sebagai sebuah *skill* yang hanya bisa dimiliki oleh spesies tertentu saja. Tentunya karena satu spesies harus memiliki elemen yang konsisten, maka hal ini menyebabkan implikasi bahwa *skill* ini juga hanya bisa dimiliki oleh satu elemen saja. Karena jenis *skill* ini sudah dimiliki sejak awal permainan, maka *base power* untuk *skill* ini sengaja didesain rendah dan sama untuk semua *Unique Skill*. Berbeda dengan *Unique Skill*, *skill* biasa bisa dimiliki oleh spesies manapun dan jenis *skill* ini bisa dimiliki oleh lebih dari satu buah elemen. Selain itu, jenis *skill* ini memiliki *base power* yang beragam, dan kombinasi elemen yang bisa mempelajari *skill* ini juga dibuat secara *random*. Jenis terakhir, yaitu *Special Skill*, merupakan *skill* yang memiliki *base power* paling tinggi. Setiap *skill* ini memiliki

tambahan atribut berupa *additional power* yang nantinya juga akan menambah kekuatan total. Jenis ini juga memiliki *base power* yang beragam dan sebisa mungkin diseimbangkan untuk tiap elemen.

Di dalam program sendiri terdapat lima buah *Unique Skill* yang dibuat untuk setiap spesies (dan elemen) tertentu. Untuk *skill* biasa telah terdapat tiga buah *skill* untuk masing-masing elemen. Terakhir, setiap elemen juga akan memiliki dua buah *Special Skill* yang didesain memiliki dua *base power* yang berbeda. Pemilihan nama dan kombinasi elemen untuk setiap *skill* diambil atau berasal dari referensi Pokemon berikut: *https://pokemondb.net/move/all*.

4. Pembagian Tugas

Modul (dalam poin spek)	**Designer**	**Implementer**
1. Engimon	13519191, 13519185, 13519180	13519191, 13519185, 13519180
2. *Skill*	13519185	13519185
3. Player	13519185, 13519180, 13519191, 13519169	13519185, 13519180, 13519191
3.b.2 *Skill Item*	13519185, 13519180	13519185, 13519180
3.b Inventory	13519180, 13519185	13519180, 13519185
4. Battle	13519217	13519217
5. Breeding	13519191, 13519185, 13519180	13519191. 13519185, 13519180
6. Peta	13519215	13519215
Bonus 2: Purely Random Wild Engimon Generation	13519215	13519215
Bonus 3: Unit Testing Implementation	13519185, 13519191	13519185, 13519191
Lain-Lain: SkillandElementsInit.hpp	13519185	13519185

Lain-Lain: Position.cpp	13519215	13519215
Main Program	13519185, 13519180, 13519191, 13519215, 13519217	13519185, 13519180, 13519191

Kelas : 04
Nama Kelompok : POO
1. <NIM> / <Nama Anggota>
2. 13519180 / Karel Renaldi
3. 13519185 / Richard Rivaldo
4. 13519191 / Kevin Ryan
5. 13519215 / Leonard Matheus
6. 13519217 / Hughie Alghaniyyu Emiliano

Asisten Pembimbing : Willy Santoso

1. Konten Diskusi

Konten diskusi berisi setiap pertanyaan dan jawaban yang dibahas saat asistensi. Dapat ditulis dalam format poin-per-poin atau paragraf.

- Update progress pengerjaan
- Kendala dalam pengerjaan tugas besar
- Max capacity bebas aja?
  Iya bebas tapi jangan terlalu besar supaya demonya ga terlalu susah nanti.
- Method learn sama replace skill pas pake skill item mungkin kurang cocok dibuat di Skill Item, jadi boleh dibuat di Player aja supaya lebih pas?
  Boleh, tapi dicantumkan aja nanti di laporan soalnya asisten nanti refernya di laporan. Sama nanti di laporan tulisin aja bonus-bonus yang dibuat kalo misalnya ada buat pertimbangan bonus.
- Maksud dari setiap spesies memiliki satu skill bawaan yang unik sedangkan engimon dapat memiliki lebih dari 1 elemen?
  Hanya skillnya saja yang unik, untuk elemen dari engimonnya tetap dapat lebih dari 1 elemen.
- Pada poin 3a iii. Apakah Engimon yang data lengkapnya ditampilkan terbatas pada Engimon yang kita miliki atau bisa semua Engimon yang ada?
- Untuk spesies Engimon dan Elemen, apakah setiap spesies hanya bisa memiliki 1 elemen?
  Tidak, 1 spesies diasosiasikan dengan 1 kombinasi elemen
- Change Active Engimon di inventory atau di player?
  Bebas, tapi boleh di player.

- Di main boleh inisialisasi objek-objek skill sebagai list skill yang ada di gamenya tapi ga di container gapapa?
  Boleh, tinggal dicantumkan aja di laporannya.
- Untuk elemen yang cuma 1 tipe, apakah tulisannya Elemen/Elemen atau Elemen/Null?
  Bebas, yang penting nanti bisa dibedain
- Untuk random spawn, bisa ditentukan mau spawn di titik mana, asalkan kasih batasan yang bisa di grassland dan di sea
- Untuk load file, boleh load game [BONUS] atau load peta hurufnya aja.
- Tiap turn, semua engimon liar harus bergerak
- Tiap turn, tidak boleh direset, pokemon liar yang udah dispawn harus tetap ada di peta
- Kalau mentok di ujung, cukup throw exception aja

2. Screenshot Bukti