TPP2023-HW5-40822017L 吳謹言 40847029S 王麒翔 40847056S 何芷倩 40847009S 王芷鈴 **（請記得修改這裡！未填學號與姓名將會扣很大！若為多人作業，請列出所有成員。）**

```
/* ------------------------------------------------------------
// Game.hpp
// ------------------------------------------------------------ */
#ifndef Game_HPP
#define Game_HPP
#include <iostream>
#include "Plant.hpp"
#include "Map.hpp"
class Player;
class Zombie;
class Plant;
class Map;
class Land;
class Game
{
public:
	Game(CoinPlant& coinPlant,HornPlant& hornplant,BombPlant& bombplant,HealPlant& Healflower)
	{
		std::string file_name = "plants.txt" ;
		read_txt(file_name,coinPlant,hornplant,bombplant,Healflower);
		read_game_settings();
	};
	void start(CoinPlant &coinPlant,HornPlant &hornPlant,
	                 BombPlant &bombPlant,HealPlant &healPlant);
private:
	Map map_;
	Player player_;
	int numLands_ = 0;
	int numZombies_ = 0;
	int lastChoice_ = 4;
	bool gameEnd_= 0;
	bool winned_ = false;
	bool bombUsed_ = false;
	void read_txt(const std::string &fileName,CoinPlant &coinPlant,HornPlant &hornPlant,
	              BombPlant&bombPlant,HealPlant &healPlant);
	void read_game_settings();
	void move_player();
	void move_zombies();
	void display() const;
	void round();
};

#endif
/* -------------------------------------------------------------
// Game.cpp
// ------------------------------------------------------------ */
#include <algorithm>
#include <iostream>
#include <ctime>
#include <fstream>
#include <sstream>

#include "Plant.hpp"
#include "Player.hpp"
#include "Map.hpp"
#include "Game.hpp"
#include "Land.hpp"

using namespace std;
void Game::start(CoinPlant &coinPlant,HornPlant &hornPlant,BombPlant &bombPlant,HealPlant
&healPlant)
{
	while (!gameEnd_)
	{
		round();
	}//while
	if (winned_)
	{
		cout << "Congratulations! You have killed all zombies!" << endl;
	}//if
	else if(bombUsed_)
	{
		cout << "\n\nYou lose the game since you cannot use that many bomb plants!" << endl;
	}//else if
	else
	{
		cout<<"\n\nOh no... You have no plant on the map"<<endl;
	}//else
```

```
    system("pause");
    system("cls");
}//Game::start
void Game::read_txt(const std::string &fileName,CoinPlant &coinPlant,HornPlant &hornPlant,
                    BombPlant &bombPlant,HealPlant &healPlant)
{
    ifstream File(fileName);
    char plantType, dollarSign;
    std::string plantName;
    int cost, hp;
    int coinVisits, coinReward;
    int hornDamage;
    int healPoints;

    for (int i = 0; i < 4; ++i)
    {
        File >> plantType >> plantName >> dollarSign >> cost >> hp;
        if (plantType == 'C')
        {
            File >> coinVisits >> coinReward;
            coinPlant.set_properties(plantName, cost, hp, coinVisits, coinReward);
        }
        else if (plantType == 'S')
        {
            File >> hornDamage;
            hornPlant.set_properties(plantName, cost, hp, hornDamage);
        }
        else if (plantType == 'B')
        {
            hornDamage = 100;
            bombPlant.set_properties(plantName, cost, hp, hp);
        }
        else if (plantType == 'H')
        {
            File >> healPoints;
            healPlant.set_properties(plantName, cost, hp, healPoints);
        }
    }
}
```

```
void Game::read_game_settings()
{
        string input;
        numLands_=8;
        numZombies_=3;
        cout << "----------------------------" << endl;
        cout << "|          Plants vs. Zombies          |" << endl;
        cout << "----------------------------" << endl;
        cout << "Number of lands on the map (1-10, default: 8)...>";
        getline(cin,input);
        if(!input.empty())
        {
                istringstream iss(input);
                iss>>numLands_;
        }
        cout << "Number of zombies on the map (1-10, default: 3)...>";
        getline(cin,input);
        if(!input.empty())
        {
                istringstream iss(input);
                iss>>numZombies_;
        }
        cout << "=====================================
                 =======================================" << endl;
        cout << "Plants vs. Zombies Rule:\n\n";
        cout << "How to win:\n";
        cout << "    (1) All zombies are dead.\n";
        cout << "    (2) At least one plant is live.\n";
        cout << "    (3) The number of dead bomb plants cannot exceed the number of zombies.\n\n";
        cout << "How to lose:\n";
        cout << "    All plants are dead.\n";
        cout << "=====================================
                =======================================" << endl;
        system("pause");
        system("cls");

        srand(time(NULL));
```

```
    player_.earn_money(150);
    map_ = Map(numLands_,0);
    for(int i=0;i<numZombies_;++i)
    {
        map_.add_zombie(new Zombie(i,rand() % numLands_));
    }
    map_.move_player_to(rand() % numLands_);
}

void Game::move_player()
{
    int place = map_.get_player_position();
    place = (place + (rand() % 6 + 1)) % numLands_;
    map_.move_player_to(place);

    if (map_.get_plant_on(place))
    {
        display();
        map_.get_plant_on(place)->visit(player_, map_.get_all_zombies(), map_.get_all_plants());
    }//if
}

void Game::move_zombies()
{
    std::vector<Zombie*> allZombies = map_.get_all_zombies();

    for (const auto& zombie : allZombies)
    {
        //Move zombie to new position
        int place = zombie->get_position();
        place = (place + (rand() % 3 + 1)) % numLands_;
        map_.move_zombie_to(zombie->get_id(),place);

        display();
        cout<<"Zombie ["<< zombie->get_id() << "] moves to land "<<place<<"."<<endl;

        Plant *plantWithZombie = map_.get_plant_on(zombie->get_position());
```

```
            if (plantWithZombie)
            {
                // Plant attacks zombie
                if (plantWithZombie->is_offensive())
                {
                    std::cout << plantWithZombie->attack(zombie) << '\n';
                    if (zombie->get_health() <= 0)
                    {
                        std::cout << "Zombie is killed!" << std::endl;
                    }//if
                }//if

                // Zombie attacks plant
                if (zombie->get_health() > 0 && plantWithZombie->get_health() > 0)
                {
                    zombie->attack(plantWithZombie);
                }//if

                if (plantWithZombie->get_health() <= 0)
                {
                    std::cout << "Plant " << plantWithZombie->get_type_name() << " is dead!" << std::endl;
                }//if
            }//if

            map_.remove_dead_plants();
            system("pause");
        }
        //std::cerr << "asdfasf\n";
        //system("pause");
        map_.remove_dead_zombies();
}
```

```
void Game::display() const
{
        system("cls");
        map_.display(numZombies_);
        cout << "----------------------------------------------\n"
             << "Zombie information:" << endl;
        std::vector<Zombie*> allZombies = map_.get_all_zombies();

        sort(allZombies.begin(), allZombies.end(),
                [](Zombie *a, Zombie *b) { return a->get_id() < b->get_id(); });

        for (const auto& zombie : allZombies)
        {
                string zombieStats = zombie->get_stats();
                cout << zombieStats << endl;
        }
        cout<<"=============================================="<<endl;
}//Game::display_zombies_stats

void Game::round()
{
        string input;
        display();

        if(player_.get_money() > 0 && map_.get_plant_on(map_.get_player_position()) == nullptr)
        {

                cout<<"[0] "<<CoinPlant::get_info()<<endl;
                cout<<"[1] "<<HornPlant::get_info()<<endl;
                cout<<"[2] "<<BombPlant::get_info()<<endl;
                cout<<"[3] "<<HealPlant::get_info()<<endl;
                cout<<endl;

                int choice=lastChoice_;
                cout<<"player $"<<player_.get_money()
                        <<":        Enter your choice (4 to give up, default: " << lastChoice_ << ")...>";
                getline(cin,input);
```

```
    if(!input.empty())
    {
        istringstream iss(input);
        iss>>choice;
        if (choice < 0 || choice > 3)
        {
            choice = 4;
        }
        lastChoice_ = choice;
    }
    int place = map_.get_player_position();
    Plant* plantPtr=nullptr;
    switch (choice)
    {
        case 0:
            plantPtr = new CoinPlant();
            map_.add_plant(plantPtr);
            player_.buy(*plantPtr);
            cout<<"You have planted "<<CoinPlant::get_name()<<" at land "<<place<<" !"<<endl;
            break;

        case 1:
            plantPtr= new HornPlant();
            map_.add_plant(plantPtr);
            player_.buy(*plantPtr);
            cout<<"You have planted "<<HornPlant::get_name()<<" at land "<<place<<" !"<<endl;
            break;
        case 2:
            plantPtr= new BombPlant();
            map_.add_plant(plantPtr);
            player_.buy(*plantPtr);
            cout<<"You have planted "<<BombPlant::get_name()<<" at land "<<place<<" !"<<endl;
            break;
        case 3:
            plantPtr= new HealPlant();
            map_.add_plant(plantPtr);
            player_.buy(*plantPtr);
            cout<<"You have planted "<<HealPlant::get_name()<<" at land "<<place<<" !"<<endl;
            break;
```

```
                case 4:
                    cout<<"You give up!"<<endl;
                    break;
            }
            system("pause");
        }
        else if (map_.get_plant_on(map_.get_player_position()) == nullptr)
        {
            cout<<"You do not have enough money to plant anything!"<<endl;
            system("pause");
        }

        int numPlants = map_.get_all_plants().size();
        int numZombies = map_.get_all_zombies().size();

        if(numPlants ==0)
        {
            gameEnd_ = true;
            winned_ = false;
            bombUsed_ = false;
            return;
        }//if
        else if(numZombies == 0)
        {
            gameEnd_ = true;
            winned_ = true;
            bombUsed_ = false;
            return;
        }//else if

        move_zombies();
        move_player();

        numPlants = map_.get_all_plants().size();
```

```
    if(numPlants == 0)
    {
        gameEnd_=1;
        winned_ = false;
        bombUsed_ = false;
        return;
    }
    else if(BombPlant::get_dead_bomb_cnt()>(numZombies_/2))
    {
        gameEnd_ = true;
        winned_ = false;
        bombUsed_ = true;
        return;
    }
    // system("pause");
    system("cls");
};
```

```
/* -----------------------------------------------------------
// Land.hpp
// ---------------------------------------------------------- */
#ifndef LAND_HPP
#define LAND_HPP

#include <vector>

class Zombie;
class Plant;

class Land
{
public:
	Land(){};
	~Land();
	void display(const int landId, const int numZombies) const;
	std::vector<Zombie*> get_zombies()const;
	Plant* get_plant()const;
	bool has_plant()const;
	bool has_player()const;
	bool has_zombie()const;
	void add_plant(Plant *plant);
	void add_zombie(Zombie *zombie);
	void remove_dead_plant();
	void remove_dead_zombies();
private:
	bool hasPlayer_ = false;
	Plant* plant_ = nullptr;
	std::vector<Zombie*> zombies_;

	friend class Map;
};

#endif
```

```
/* ------------------------------------------------------------
// Land.cpp
// ------------------------------------------------------------ */
#include "Land.hpp"
#include "Zombie.hpp"
#include "Plant.hpp"
#include <iostream>
#include <string>
#include <algorithm>

Land::~Land()
{
	delete plant_;

	for (Zombie *z : zombies_)
	{
		delete z;
	}//for z
	zombies_.clear();
}
void Land::display(const int landId, const int numZombies) const
{
	std::string land = "[" + std::to_string(landId) + "]{";

	//player
	if(hasPlayer_)
	{
		land += "*";
	}//if
	else
	{
		land += " ";
	}//else
```

```
    //zombies
    for(int i = 0, cnt = 0; i < numZombies; ++i)
    {
        bool hasZombie = false;
        for (size_t j = cnt; j < zombies_.size(); ++j)
        {
            if (zombies_[j]->get_id() == i)
            {
                land += std::to_string(zombies_[j]->get_id());
                hasZombie = true;
                ++cnt;
                break;
            }//if
        }//for j
        if (!hasZombie)
        {
            land += " ";
        }//if
    }//for i

    land += "}";

    //plant
    if(plant_ != nullptr)
    {
        land += plant_->get_state();
    }
    else
    {
        land += "Empty";
    }
    std::cout << land << std::endl;
}
std::vector<Zombie*> Land::get_zombies()const
{
    return zombies_;
}
```

```
Plant* Land::get_plant()const
{
	return plant_;
}
bool Land::has_plant()const
{
	return plant_ != nullptr;
}
bool Land::has_player()const
{
	return hasPlayer_;
}
bool Land::has_zombie()const
{
	return zombies_.empty();
}
void Land::add_plant(Plant *plant)
{
	if(plant_ == nullptr)
	{
		plant_ = plant;
	}
}
void Land::add_zombie(Zombie *zombie)
{
	zombies_.push_back(zombie);
	std::sort(zombies_.begin(), zombies_.end(), [](Zombie *a, Zombie *b) { return (a->get_id()) <
(b->get_id()); });
}
void Land::remove_dead_plant()
{
	if (plant_ == nullptr) return;

	if (plant_->get_health() <= 0)
	{
		delete plant_;
		plant_ = nullptr;
	}//if
}
```

```
void Land::remove_dead_zombies()
{
	for (size_t i = 0; i < zombies_.size(); ++i)
	{
		if (zombies_[i]->get_health() <= 0)
		{
			delete zombies_[i];
			zombies_.erase(zombies_.begin() + i);
			--i;
		}//if
	}//for it
}
/* ------------------------------------------------------------
// Map.hpp
// ------------------------------------------------------------ */
#ifndef MAP_HPP
#define MAP_HPP
#include<vector>
#include "Land.hpp"
class Zombie;
class Plant;
class Land;

class Map
{
public:
	Map(){};
	Map(const int landNum,const int position);
	void display(const int numZombies) const;
	void move_player_to(const int newPosition);
	void move_zombie_to(const int id, const int newPosition);
	int get_player_position() const;
	std::vector<Zombie*> get_all_zombies() const;
	std::vector<Plant*> get_all_plants();
	Plant* get_plant_on(int landId);
	void add_plant(Plant *plant);
	void add_zombie(Zombie *zombie);
	void remove_dead_plants();
	void remove_dead_zombies();
private:
	std::vector<Land> lands_;
	int playerPosition_ = 0;
};

#endif
/* ------------------------------------------------------------
// Map.cpp
// ------------------------------------------------------------ */
#include "Map.hpp"
#include "Zombie.hpp"
#include "Plant.hpp"
#include "Land.hpp"

Map::Map(const int landNum, const int position): lands_(std::vector<Land>(landNum)),
playerPosition_(position) {}

void Map::display(const int numZombies) const
{
	for(size_t i = 0; i < lands_.size(); ++i)
	{
		lands_[i].display(i, numZombies);
	}
}

void Map::move_player_to(int newPosition)
{
	lands_[playerPosition_].hasPlayer_ = false;
	playerPosition_ = newPosition;
	lands_[playerPosition_].hasPlayer_ = true;
}
```

```
void Map::move_zombie_to(int id, int newPosition)
{
    for (Land &l : lands_)
    {
        for (size_t i = 0; i < l.zombies_.size(); ++i)
        {
            if (l.zombies_[i]->get_id() == id)
            {
                l.zombies_[i]->move(newPosition);
                lands_[newPosition].add_zombie(l.zombies_[i]);
                l.zombies_.erase(l.zombies_.begin() + i);
                return;
            }//if
        }//for z
    }//for l
}

int Map::get_player_position()const
{
    return this->playerPosition_;
}
std::vector<Zombie*> Map::get_all_zombies() const
{
    std::vector<Zombie*> all_zombies;
    for(size_t i = 0; i < lands_.size(); ++i)
    {
        std::vector<Zombie*> zombies = lands_[i].get_zombies();
        all_zombies.insert(all_zombies.end(), zombies.begin(), zombies.end());
    }
    return all_zombies;
}
```

```
std::vector<Plant*> Map::get_all_plants()
{
    std::vector<Plant*> all_plants;
    for(size_t i = 0; i < lands_.size(); ++i)
    {
        Plant *plant = lands_[i].get_plant();
        if (plant != nullptr)
            all_plants.push_back(plant);
    }
    return all_plants;
}

Plant* Map::get_plant_on(int landId)
{
    return lands_[landId].plant_;
}//Map::get_plant_on

void Map::add_plant(Plant *plant)
{
    lands_[playerPosition_].add_plant(plant);
}
void Map::add_zombie(Zombie *zombie)
{
    lands_[zombie->get_position()].add_zombie(zombie);
}

void Map::remove_dead_plants()
{
    for (Land &l : lands_)
    {
        l.remove_dead_plant();
    }//for l
}
```

```
void Map::remove_dead_zombies()
{
	for (Land &l : lands_)
	{
		l.remove_dead_zombies();
	}//for l
}
/* ------------------------------------------------------------
// Plant.hpp
// ------------------------------------------------------------ */
#ifndef PLANT_HPP
#define PLANT_HPP

#include <string>
#include <vector>
#include "Zombie.hpp"
#include "Player.hpp"
class Zombie;
class Player;

class Plant
{
public:
	Plant(){};
	Plant(int health, int position = 0, std::string typeName = "", bool isOffensive = false);
	virtual ~Plant(){};
	int get_position() const;
	int get_health() const;
	bool is_offensive() const;
	const std::string& get_type_name() const;
	void move_to(int position);
	void increse_health(int heal_points, int maxHealth);
	void decrease_health(int damage);
	virtual std::string get_state() const = 0;
	virtual std::string attack(Zombie *Zombie) = 0;
	virtual int get_cost() const = 0;
	virtual int get_maxHealth() const = 0;
	virtual void visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants) = 0;
protected:
        int health_ = 0;
        int position_ = 0;
        std::string typeName_ = "";
        bool isOffensive_ = false;
};

class CoinPlant : public Plant
{
public:
        CoinPlant();

        static void set_properties(std::string name, int cost, int maxHealth, int visitNeed, int reward);
        static std::string get_info();
        virtual std::string get_state() const override;
        virtual std::string attack(Zombie *zombie) override;
        static std::string get_name();
        virtual int get_maxHealth() const override;
        int get_cost() const override;
        void visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants) override;

private:
        int visitLeft_ = 0;
        static std::string name_;
        static int cost_;
        static int reward_;
        static int visitNeed_;
        static int maxHealth_;
};
```

```
class HornPlant : public Plant
{
public:
        HornPlant();
        static void set_properties(std::string name, int cost, int maxHealth, int damage);
        static std::string get_info();
        virtual std::string get_state() const override;
        virtual std::string attack(Zombie *zombie) override;
        static std::string get_name();
        virtual int get_maxHealth() const override;
        int get_cost() const override;
        void visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants) override;

private:
        static std::string name_;
        static int cost_;
        static int maxHealth_;
        static int damage_;
};

class BombPlant : public Plant
{
public:
        BombPlant();
        ~BombPlant();
        static void set_properties(std::string name, int cost, int maxHealth, int damage);
        static std::string get_info();
        virtual std::string get_state() const override;
        virtual std::string attack(Zombie *zombie) override;
        static std::string get_name();
        virtual int get_maxHealth() const override;
        int get_cost() const override;
        void visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants) override;
        static int get_dead_bomb_cnt();
private:
        static std::string name_;
        static int cost_;
        static int maxHealth_;
        static int damage_;
        static int deadBombCnt_;
};

class HealPlant : public Plant
{
public:
        HealPlant();
        static void set_properties(std::string name, int cost, int maxHealth, int healPoints);
        static std::string get_info();
        virtual std::string get_state() const override;
        virtual std::string attack(Zombie *zombie) override;
        static std::string get_name();
        virtual int get_maxHealth() const override;
        int get_cost() const override;
        void visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants) override;

private:
        static std::string name_;
        static int cost_;
        static int maxHealth_;
        static int healPoints_;
};

#endif
```

```
/* ------------------------------------------------------------
// Plant.cpp
// ------------------------------------------------------------ */
#include <iostream>

#include "Plant.hpp"

std::string CoinPlant::name_ = "";
int CoinPlant::cost_ = 0;
int CoinPlant::maxHealth_ = 0;
int CoinPlant::visitNeed_ = 0;
int CoinPlant::reward_ = 0;

std::string HornPlant::name_ = "";
int HornPlant::cost_ = 0;
int HornPlant::maxHealth_ = 0;
int HornPlant::damage_ = 0;

std::string BombPlant::name_ = "";
int BombPlant::cost_ = 0;
int BombPlant::maxHealth_ = 0;
int BombPlant::damage_ = 0;
int BombPlant::deadBombCnt_ = 0;

std::string HealPlant::name_ = "";
int HealPlant::cost_ = 0;
int HealPlant::maxHealth_ = 0;
int HealPlant::healPoints_ = 0;

//**************************PLANT**************************//
Plant::Plant(int health, int position, std::string typeName, bool isOffensive): health_(health),
position_(position), typeName_(typeName), isOffensive_(isOffensive) {}

int Plant::get_position() const
{
	return position_;
} // get_position
```

```
int Plant::get_health() const
{
        return health_;
}

bool Plant::is_offensive() const
{
        return isOffensive_;
}//Plant::is_offensive

const std::string& Plant::get_type_name() const
{
        return typeName_;
}//get_type_name

void Plant::move_to(int position)
{
        position_ = position;
} // move_to

void Plant::increse_health(int heal_points, int maxHealth)
{
        health_ += heal_points;
        if(health_ > maxHealth)health_ = maxHealth;
} // increse_health

void Plant::decrease_health(int damage)
{
        health_ -= damage;
} // decrese_health

//*************************COINPLANT*************************//
CoinPlant::CoinPlant(): Plant(maxHealth_, 0, name_, false), visitLeft_(visitNeed_) {}

void CoinPlant::set_properties(std::string name, int cost, int maxHealth, int visitNeed, int reward)
{
        CoinPlant::name_ = name;
        CoinPlant::cost_ = cost;
        CoinPlant::maxHealth_ = maxHealth;
```

```
    CoinPlant::visitNeed_ = visitNeed;
    CoinPlant::reward_ = reward;
} // set_properties

std::string CoinPlant::get_info()
{
    std::string info = name_;
    info += " $" + std::to_string(CoinPlant::cost_);
    info += " HP: " + std::to_string(CoinPlant::maxHealth_);
    info += " - gives $" + std::to_string(CoinPlant::reward_);
    info += " every " + std::to_string(CoinPlant::visitNeed_) + " rounds";
    return info;
} // get_info

std::string CoinPlant::get_state() const
{
    std::string state = name_;
    state += " HP: " + std::to_string(health_);
    state += " (" + std::to_string(visitLeft_);
    state += " more visit";
    if (visitLeft_ < 2)
    {
        state += ")";
    } // if
    else
    {
        state += "s)";
    } // else
    return state;
} // get_state

std::string CoinPlant::attack(Zombie *zombie)
{
    return "";
}

std::string CoinPlant::get_name()
{
    return CoinPlant::name_;
```

```
} // get_name

int CoinPlant::get_cost() const
{
        return CoinPlant::cost_;
} // get_cost

void CoinPlant::visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants)
{
        std::string action = "";
        visitLeft_ -= 1;

        if (visitLeft_ > 0)
        {
                action = "You still need " + std::to_string(visitLeft_) + " visit to earn money.";
        } // if
        else
        {
                player.earn_money(reward_);
                action = "You have earned $" + std::to_string(CoinPlant::reward_) + "! ";
                action += "Now you have $" + std::to_string(player.get_money()) + ".";
                visitLeft_ = 2;
        } // else

        std::cout << action << std::endl;
        system("pause");
} // visit
int CoinPlant::get_maxHealth() const
{
        return CoinPlant::maxHealth_;
}
//**************************HORNPLANT *************************** //
HornPlant::HornPlant(): Plant(maxHealth_, 0, name_, true) {}

void HornPlant::set_properties(std::string name, int cost, int maxHealth, int damage)
{
        HornPlant::name_ = name;
        HornPlant::cost_ = cost;
        HornPlant::maxHealth_ = maxHealth;
```

```
	HornPlant::damage_ = damage;
} // set_properties

std::string HornPlant::get_info()
{
	std::string info = HornPlant::name_;
	info += " $" + std::to_string(HornPlant::cost_);
	info += " HP: " + std::to_string(HornPlant::maxHealth_);
	info += " - gives " + std::to_string(HornPlant::damage_);
	info += " damage points";
	return info;
} // get_info

std::string HornPlant::get_state() const
{
	std::string state = HornPlant::name_;
	state += " HP: " + std::to_string(health_);
	return state;
} // get_state

std::string HornPlant::attack(Zombie *zombie)
{
	zombie->decrease_health(damage_);
	return name_ + " gives " + std::to_string(damage_) + " damage to the zombie!";
}

std::string HornPlant::get_name()
{
	return HornPlant::name_;
} // get_name

int HornPlant::get_cost() const
{
	return HornPlant::cost_;
} // get_cost
```

```
void HornPlant::visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants)
{
	std::string action = "Your " + name_ + " is guarding your land.";
	std::cout << action << std::endl;
	system("pause");
} // visit

int HornPlant::get_maxHealth() const
{
	return HornPlant::maxHealth_;
}
//*************************BOMBPLANT*************************//
BombPlant::BombPlant(): Plant(maxHealth_, 0, name_, true)
{

}//BombPlant::BombPlant

BombPlant::~BombPlant()
{
	++deadBombCnt_;
}//BombPlant::~BombPlant

void BombPlant::set_properties(std::string name, int cost, int maxHealth, int damage)
{
	BombPlant::name_ = name;
	BombPlant::cost_ = cost;
	BombPlant::maxHealth_ = maxHealth;
	BombPlant::damage_ = damage;
} // set_properties

std::string BombPlant::get_info()
{
	std::string info = BombPlant::name_;
	info += " $" + std::to_string(BombPlant::cost_);
	info += " HP: " + std::to_string(BombPlant::maxHealth_);
	info += " - gives " + std::to_string(BombPlant::damage_);
	info += " damage points";
	return info;
} // get_info
```

```
std::string BombPlant::get_state() const
{
	std::string state = BombPlant::name_;
	state += " HP: " + std::to_string(health_);
	return state;
} // get_state

std::string BombPlant::attack(Zombie *zombie)
{
	zombie->decrease_health(damage_);
	health_ = 0;
	return name_ + " gives " + std::to_string(damage_) + " damage to the zombie!";
}

std::string BombPlant::get_name()
{
	return BombPlant::name_;
} // get_name

int BombPlant::get_cost() const
{
	return BombPlant::cost_;
} // get_cost

void BombPlant::visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants)
{
	std::string action = "Your " + name_ + " is guarding your land.";
	std::cout << action << std::endl;
	system("pause");
} // visit

int BombPlant::get_dead_bomb_cnt()
{
	return deadBombCnt_;
}//BombPlant::get_dead_bomb_num
```

```
int BombPlant::get_maxHealth() const
{
	return BombPlant::maxHealth_;
}
//*************************HEALPLANT*************************//
HealPlant::HealPlant(): Plant(maxHealth_, 0, name_, false) {}

void HealPlant::set_properties(std::string name, int cost, int maxHealth, int healPoints)
{
	HealPlant::name_ = name;
	HealPlant::cost_ = cost;
	HealPlant::maxHealth_ = maxHealth;
	HealPlant::healPoints_ = healPoints;
} // set_properties

std::string HealPlant::get_info()
{
	std::string info = HealPlant::name_;
	info += " $" + std::to_string(HealPlant::cost_);
	info += " HP: " + std::to_string(HealPlant::maxHealth_);
	info += " - gives all your plants " + std::to_string(HealPlant::healPoints_);
	info += " HP back";
	return info;
} // get_info

std::string HealPlant::get_state() const
{
	std::string state = HealPlant::name_;
	state += " HP: " + std::to_string(health_);
	return state;
} // get_state

std::string HealPlant::attack(Zombie *zombie)
{
	return "";
}
```

```
std::string HealPlant::get_name()
{
	return HealPlant::name_;
} // get_name

int HealPlant::get_cost() const
{
	return HealPlant::cost_;
} // get_cost

void HealPlant::visit(Player &player, std::vector<Zombie *> zombies, std::vector<Plant *> plants)
{
	std::string action = "";
	for (auto plant : plants)
	{
		plant->increse_health(HealPlant::healPoints_, plant->get_maxHealth());
	}
	action = "All your plants have recovered " + std::to_string(HealPlant::healPoints_) + " HP !\n";
	std::cout << action << std::endl;
	system("pause");
} // visit
int HealPlant::get_maxHealth() const
{
	return HealPlant::maxHealth_;
}
/* ------------------------------------------------------------
// Player.hpp
// ------------------------------------------------------------ */
#ifndef PLAYER_HPP
#define PLAYER_HPP
#include "Plant.hpp"
class Plant;
class Player
{
public:
	int get_money() const;
	// int get_position() const;
	void earn_money(int reward);
	void buy(const Plant &newPlant);
protected:
	int money_ = 0;
	// int position_ = 0;
};

#endif
/* ----------------------------------------------------------------
// Player.cpp
// ---------------------------------------------------------------- */
#include "Player.hpp"
#include "Map.hpp"

int Player::get_money() const
{
	return money_;
}

// int Player::get_position() const
// {
//	return position_;
// }

void Player::earn_money(int reward)
{
	money_ += reward;
}

void Player::buy(const Plant &newPlant)
{
	money_ -= newPlant.get_cost();
	// Map::add_plant(newPlant);
}
```

TPP2023-HW5-40822017L 吳謹言 40847029S 王麒翔 40847056S 何芷倩 40847009S 王芷鈴 **（請記得修改這裡！未填學號與姓名將會扣很大！若為多人作業，請列出所有成員。）**

```
/* ------------------------------------------------------------
// Zombie.hpp
// ------------------------------------------------------------ */
#ifndef ZOMBIE_HPP
#define ZOMBIE_HPP
#include <string>
class Plant;

class Zombie
{
public:
	static constexpr int damage = 15;
	static constexpr int maxHealth = 40;
	Zombie() = default;
	Zombie(const int id, const int position): id_(id), position_(position){};
	int get_position()const;
	int get_id()const;
	int get_health()const;
	void decrease_health(const int damage);
	void move(int newPosition);
	std::string get_stats() const;
	void attack(Plant *plant);
private:
	int Health_ = maxHealth;
	int id_ = 0;
	int position_ = 0;
};
#endif
/* ------------------------------------------------------------
// Zombie.cpp
// ------------------------------------------------------------ */
#include "Zombie.hpp"
#include "Plant.hpp"
#include <iostream>
#include <string>
int Zombie::get_position()const
{
	return position_;
}
```

```
int Zombie::get_id()const
{
    return id_;
}
int Zombie::get_health()const
{
    return Health_;
}
void Zombie::decrease_health(const int damage)
{
    Health_ -= damage;
}
void Zombie::move(int newPosition)
{
    position_ = newPosition;
}
std::string Zombie::get_stats() const
{
    std::string id = std::to_string(id_), hp = std::to_string(Health_);
    std::string stats = "[" + id + "]" + " Damage: 15 HP:";
    for(int i = 0; i < Health_; ++i)
    {
        stats += "*";
    }
    return stats;
}
void Zombie::attack(Plant *plant)
{
    std::string name = plant->get_type_name();
    plant->decrease_health(damage);
    std::cout << "Zombie eats plant " << name << " and causes damage 15." << std::endl;
}
```

TPP2023-HW5-40822017L 吳謹言 40847029S 王麒翔 40847056S 何芷倩 40847009S 王芷鈴 **（請記得修改這裡！未填學號與姓名將會扣很大！若為多人作業，請列出所有成員。）**

```
/* ---------------------------------------------------------------
// main.cpp
// ------------------------------------------------------------- */
#include <iostream>
#include <vector>

#include "Game.hpp"
#include "Plant.hpp"
#include "Player.hpp"
#include "Map.hpp"
#include "Land.hpp"
#include "Zombie.hpp"

int main()
{
	CoinPlant coinPlant;
	HornPlant hornPlant;
	BombPlant bombPlant;
	HealPlant HealFlower;

	Game game(coinPlant,hornPlant,bombPlant,HealFlower);
	game.start(coinPlant,hornPlant,bombPlant,HealFlower);

	return 0;
} // main
```