TPP2021-HW4-Team1
40647043S 楊紹玄, 40747028S 盧昭華, 40747023S 劉洧聿, 90998011Y 洪軾凱

```
/* ----------------------------------------------------------------
// main.cpp
// ------------------------------------------------------------ */

#include <iostream>
#include <algorithm>
#include "game.h"
using namespace std;

int main()
{
    Game agent;
    PAUSE_AND_CLEAR
    agent.gameloop();
    cout << "\nmade by 2021_TPP_Group1" << endl;
    cout << "Member:\n\tskps2010\n\t
               skyhong2002\n\tdennisyong\n\tLiuwy-matt" << endl;
    return 0;
}

/* ----------------------------------------------------------------
// game.h
// ------------------------------------------------------------ */

#ifndef GAME_H_INCLUDED
#define GAME_H_INCLUDED

#include <vector>
#include "map.h"
#include "land.h"
#include "zombie.h"

#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
defined(__NT__)
    #define PAUSE_AND_CLEAR system("pause");system("cls");
#else
    #define PAUSE_AND_CLEAR system("read -n 1");system("clear");
#endif

class Game
{
private:
    const int DEFAULTLAND = 8;
    const int MAXLAND = 10;
    const int DEFAULTZOMBIE = 3;
    const int MAXZOMBIE = 10;
    int lastDecision = 0;
    int gamestatus = 0;
    int plantUnderthePlayer = -1;
    std::vector<char> dictionary;
    map *m = nullptr;
    void logo(void);
    void rule(void);
    void showMap(void);
    void showPlants(void);
    void makeDecision(void);
    const int setNumberOfLand(void);
    const int setNumberOfZombie(void);
    const int rolldice(const int minimum, const int maximum);
```

```
public:
    Game();
    ~Game();
    void gameloop(void);
};

#endif

/* --------------------------------------------------------------
// game.cpp
// ------------------------------------------------------------ */

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include "game.h"
#include "map.h"
#include "land.h"
#include "zombie.h"

Game::Game() : DEFAULTLAND(8), MAXLAND(10), DEFAULTZOMBIE(3),
MAXZOMBIE(10), lastDecision(0), gamestatus(0),
plantUnderthePlayer(-1)
{
    srand(time(NULL));
    // init the map
    logo();
    m = new map(setNumberOfLand(), setNumberOfZombie());
    rule();
    m->PlayerMove(rolldice(1, m->Landcnt()));
    for (size_t i = 0; i < m->Zombiecnt(); ++i)
    {
        m->ZombieMove(i, rolldice(1, m->Landcnt()));
    }

    // read file
    std::ifstream ifs;
    ifs.open("plants.txt");
    char input = 0;
    while (ifs >> input)
    {
        dictionary.push_back(input);
        std::string name;
        ifs >> name;
        char dollarsign = 0;
        ifs >> dollarsign;
        if (input == 'B')
        {
            int cost = 0, hp = 0;
            ifs >> cost >> hp;
            m->B_Init(name, cost, hp);
        }
        else if (input == 'C')
        {
            int cost = 0, hp = 0, times = 0, coin = 0;
            ifs >> cost >> hp >> times >> coin;
            m->C_Init(name, cost, hp, times, coin);
        }
        else if (input == 'H')
        {
            int cost = 0, hp = 0, healpoint = 0;
            ifs >> cost >> hp >> healpoint;
            m->H_Init(name, cost, hp, healpoint);
        }
        else if (input == 'S')
        {
            int cost = 0, hp = 0, atk = 0;
            ifs >> cost >> hp >> atk;
            m->S_Init(name, cost, hp, atk);
        }
    }
    ifs.close();
    lastDecision = dictionary.size();
}

Game::~Game()
{
    delete m;
}

void Game::logo(void)
{
    std::cout << "=--------------------------------------------=" <<
std::endl;
    std::cout << "| TPP_2021 Hw05 Group1 Plants vs. Zombies |" <<
std::endl;
    std::cout << "=--------------------------------------------=" <<
std::endl;
    return;
}

void Game::rule(void)
{
    std::cout <<
"==================================================================
============" << std::endl;
    std::cout << "Plants vs. Zombies Rule:" << std::endl;
    std::cout << "\nHow to win:\n\t(1) All zombies are dead.\n\t(2) At
least one plant is live.\n\t(3) The number of dead bomb plants cannot
exceed the number of zombies." << std::endl;
    std::cout << "\nHow to lose:\n\tAll plants are dead." << std::endl;
    std::cout <<
"==================================================================
============" << std::endl;
    return;
}

void Game::showMap(void)
{
    std::cout << *m;
    return;
}

void Game::showPlants(void)
{
    for (size_t i = 0; i < dictionary.size(); ++i)
    {
        std::cout << "[" << i << "]";
        m->PrintPlant(dictionary[i]);
        std::cout << std::endl;
    }
    return;
}

void Game::makeDecision(void)
{
    while (1)
    {
        showPlants();
        std::cout << std::endl;
        m->PrintPlayer();
        std::cout << "Enter your choice (" << dictionary.size() << " to
give up, default: " << lastDecision << ")...>";
        int decision = 0;
        std::cin >> decision;
        std::cin.clear();
        std::cin.ignore();
        if (decision > dictionary.size() || decision < 0)
        {
            std::cout << "Wrong input! Set your decision as default!"
<< std::endl;
            decision = lastDecision;
        }
        else
        {
            lastDecision = decision;
        }
        if (decision != dictionary.size())
        {
            if (m->PlantShowCost(dictionary.at(decision)) >
m->PlayerCoin())
            {
                std::cout << "Not enough money! Please input again!" <<
std::endl;
                system("pause");
            }
            else
            {
                m->SetPlant(m->PlayerPosition(),
dictionary.at(decision));
                std::cout << "You have planted " <<
m->PlantName(m->PlayerPosition()) << " at land " << m->PlayerPosition()
<< " !" << std::endl;
                break;
            }
        }
        else
        {
            std::cout << "You give up!" << std::endl;
            break;
        }
    }
    return;
}
```

```
const int Game::setNumberOfLand(void)
{
    int NumOfLand = 0;
    std::cout << "Number of lands on the map (1-" << MAXLAND << ", default:
" << DEFAULTLAND << ")...>";
    std::cin >> NumOfLand;
    if (NumOfLand > MAXLAND || NumOfLand <= 0)
    {
        NumOfLand = DEFAULTLAND;
    }
    return NumOfLand;
}

const int Game::setNumberOfZombie(void)
{
    int NumOfZombie = 0;
    std::cout << "Number of zombies on the map (1-" << MAXZOMBIE << ",
default: " << DEFAULTZOMBIE << ")...>";
    std::cin >> NumOfZombie;
    if (NumOfZombie > MAXZOMBIE || NumOfZombie <= 0)
    {
        NumOfZombie = DEFAULTZOMBIE;
    }
    return NumOfZombie;
}

const int Game::rolldice(const int minimum, const int maximum)
{
    return (rand() % (maximum - minimum + 1)) + minimum;
}

void Game::gameloop(void)
{
    while (1)
    {
        showMap();
        if (m->LandisEmpty(m->PlayerPosition()))
        {
            if (m->PlayerCoin() > 0)
            {
                makeDecision();
            }
            else
            {
                std::cout << "You do not have enough money to plant
anything!" << std::endl;
            }
        }
        else if (plantUnderthePlayer == 0)
        {
            std::cout << m->PlantName(m->PlayerPosition()) << " is
guarding your land." << std::endl;
        }
        else if (plantUnderthePlayer == 1)
        {
            if (m->PlantisReady(m->PlayerPosition()))
            {
                std::cout << "You have earned $" <<
```

```
m->PlantStatus(m->PlayerPosition()) << "! Now you have $" <<
m->PlayerCoin() << "." << std::endl;
            }
            else
            {
                std::cout << "You still need " <<
m->PlantStatus(m->PlayerPosition()) << " visit to earn money." <<
std::endl;
            }
        }
        else if (plantUnderthePlayer == 2)
        {
            std::cout << "All your plants have recovered " <<
m->PlantStatus(m->PlayerPosition()) << " HP!" << std::endl;
        }
        PAUSE_AND_CLEAR

        if (!(m->Plantcnt()))
        {
            std::cout << "Oh no... You have no plant on the map ...."
<< std::endl;
            break;
        }

        // zombie move
        int deadzombie = 0;
        for (size_t i = 0; i < m->Zombiecnt(); ++i)
        {
            if (!(m->ZombieisDead(i)))
            {
                showMap();
                m->ZombieMove(i, rolldice(1, 3));
                std::cout << "Zombie [" << i << "] moves to land " <<
m->ZombiePosition(i) << "." << std::endl;
                if (!(m->LandisEmpty(m->ZombiePosition(i))))
                {
                    std::string NowPlantName =
m->PlantName(m->ZombiePosition(i));
                    if (m->PlantisOffensive(m->ZombiePosition(i)))
                    {
                        std::cout << m->PlantName(m->ZombiePosition(i))
<< " gives " << m->PlantDP(m->ZombiePosition(i)) << " damage to the
zombie!" << std::endl;
                        m->PAttackZ(m->ZombiePosition(i), i);
                    }
                    if (!(m->ZombieisDead(i)))
                    {
                        if (!(m->LandisEmpty(m->ZombiePosition(i))))
                        {
                            std::cout << "Zombie eats plant " <<
m->PlantName(m->ZombiePosition(i)) << " and causes damage 15." <<
std::endl;
                            m->ZAttackP(m->ZombiePosition(i), i);
                        }
                    }
                    else
                    {
                        std::cout << "Zombie is killed!" << std::endl;
                        ++deadzombie;
```

```
                 }

                 if (m->LandisEmpty(m->ZombiePosition(i)))
                 {
                     std::cout << "Plant " << NowPlantName << " is dead!"
<< std::endl;
                     if (m->Plantcnt() == 0)
                     {
                         gamestatus = 1;
                         break;
                     }
                 }
                 if (m->B_Used() >= m->Zombiecnt())
                 {
                     gamestatus = 2;
                     break;
                 }
             }
             PAUSE_AND_CLEAR
         }
         else
         {
             ++deadzombie;
         }
     }

     if (gamestatus == 1)
     {
         std::cout << "Oh no... You have no plant on the map ...."
<< std::endl;
         break;
     }
     else if (gamestatus == 2)
     {
         std::cout << "You lose the game since you cannot use that
many bomb plants!" << std::endl;
         break;
     }

     if (deadzombie == m->Zombiecnt())
     {
         std::cout << "Congratulations! You have killed all zombies!"
<< std::endl;
         break;
     }
     plantUnderthePlayer = m->PlayerMove(rolldice(1, 6));
   }
   return;
}

/* ---------------------------------------------------------------
// map.h
// ------------------------------------------------------------- */

#ifndef map__
#define map__
#include <iostream>
#include "land.h"
#include "zombie.h"
```

```
#include "player.h"
#include "plant.h"

class map
{
public:
   map() : row_(8), zombiecnt_(3), land_(new land[row_]()),
zombie_(new zombie[zombiecnt_]()){};
   map(const int row, const int zombiecnt) : row_(row),
zombiecnt_(zombiecnt), land_(new land[row_]()), zombie_(new
zombie[zombiecnt_]()) {}
   ~map()
   {
      delete[] land_;
      delete[] zombie_;
   }
   void PrintPlayer() const;
   void PrintRow(const int i) const;
   void PrintZombie(const int id) const;
   void PrintPlant(const char T) const { show(T); }

   int Landcnt() const { return row_; }
   int Zombiecnt() const { return zombiecnt_; }

   bool ZombieisDead(const int id) const { return
zombie_[id].isDead(); }
   int ZombieMove(const int id, const int step);
   int ZombiePosition(const int id);

   int PlayerMove(const int step);
   int PlayerPosition();
   int PlayerCoin() const { return player_.Coin(); }
   void Cost(const int m) { player_.Cost(m); }

   void S_Init(std::string name, int cost, int hp, int atk)
{ ShootPlant::set(name, cost, hp, atk); }
   void B_Init(std::string name, int cost, int hp)
{ BombPlant::set(name, cost, hp); };
   void C_Init(std::string name, int cost, int hp, int time, int coin)
{ CoinPlant::set(name, cost, hp, time, coin); };
   void H_Init(std::string name, int cost, int hp, int healpoint)
{ HealPlant::set(name, cost, hp, healpoint); }
   int B_Used() { return BombPlant::showused(); } //炸彈炸過幾次

   std::string SetPlant(int row, const char T)
   {
      land_[row].SetPlant(T);
      Cost(land_[row].PlantCost());
      return land_[row].PlantName();
   }
   bool LandisEmpty(int row) const { return land_[row].isEmpty(); }
   std::string PlantName(int row) const { return
land_[row].PlantName(); }
   char PlantType(int row) const { return land_[row].PlantName()[0]; }
   int PlantCost(int row) const { return land_[row].PlantCost(); }
   int PlantMHP(int row) const { return land_[row].PlantMHP(); }
   int PlantHP(int row) const { return land_[row].PlantHP(); }
   int PlantDP(int row) const { return land_[row].PlantDP(); }
   void PlantHurt(int row, int hp) { land_[row].PlantHurt(hp); }
```

```
    bool PlantisReady(int row) const { return isready_; }
    void PlantUpdate(int row)
    {
        isready_ = land_[row].PlantisReady();
        status_ = land_[row].PlantGet();
    }
    int PlantVisit(int row) const { return land_[row].PlantVisit(); }
    int PlantStatus(int row) { return status_; }
    // HealPlant: Heal point
    // CoinPlant: 剩餘回合(NotReady) or 拿到的錢(isReady)
    bool PlantisOffensive(int row) const { return
land_[row].isOffensive(); }

    void PAttackZ(int row, int zid);
    void ZAttackP(int row, int zid);
    int Plantcnt() const;
    int PlantShowCost(char T) { return showcost(T); }

private:
    int row_ = 8;
    int zombiecnt_ = 3;
    bool isready_ = false;
    int status_ = -1;
    player player_;
    zombie *zombie_;
    land *land_;
};

std::ostream &operator<<(std::ostream &os, const map &out);

#endif

/* -------------------------------------------------------------
// map.cpp
// ----------------------------------------------------------- */

#include <iostream>
#include "map.h"
using namespace std;

void map::PrintPlayer() const
{
    cout << player_;
}

void map::PrintRow(const int i) const
{
    cout << '[' << i << ']';
    cout << '<' << (player_.Position() == i ? '*' : ' ');
    for (int j = 0; j < Zombiecnt(); j++)
    {
        if (!zombie_[j].isDead())
        {
            if (zombie_[j].Position() == i)
                cout << j;
            else
                cout << ' ';
        }
    }
    cout << '>';
    cout << land_[i] << "\n";
}

void map::PrintZombie(const int id) const
{
    if (!zombie_[id].isDead())
        cout << zombie_[id] << "\n";
}

int map::ZombieMove(const int id, const int step)
{
    zombie_[id].Step(step, row_);
    return zombie_[id].Position();
}

int map::ZombiePosition(const int id)
{
    return zombie_[id].Position();
}

int map::PlayerMove(const int step)
{
    player_.Step(step, row_);
    if (!land_[player_.Position()].isEmpty())
    {
        int tmp = land_[player_.Position()].PlantVisit();
        PlantUpdate(player_.Position());
        if (tmp == 1) // Coin
        {
            if (isready_)
            {
                Cost(-status_);
            }
        }
        else if (tmp == 2) // Heal
        {
            for (int i = 0; i < row_; i++)
            {
                if (!land_[i].isEmpty())
                {
                    land_[i].PlantHeal();
                }
            }
        }
        return tmp;
    }
    else
    {
        return -1;
    }
}

int map::PlayerPosition()
{
    return player_.Position();
}

void map::PAttackZ(int row, int zid)
```

```
{
    int tmp = land_[row].PlantDP();
    if (!zombie_[zid].isDead())
        zombie_[zid].Hurt(tmp);
    land_[row].CheckAlive();
}

void map::ZAttackP(int row, int zid)
{
    int tmp = zombie_[zid].DamagePoint();
    if (!land_[row].isEmpty())
    {
        land_[row].PlantHurt(tmp);
    }
}

int map::Plantcnt() const
{
    int tmp = 0;
    for (int i = 0; i < row_; ++i)
    {
        if (!land_[i].isEmpty())
            ++tmp;
    }
    return tmp;
}

std::ostream &operator<<(std::ostream &os, const map &out)
{
    for (int i = 0; i < out.Landcnt(); i++)
        out.PrintRow(i);
    cout << string(50, '-') << "\n"
         << "Zombie information:\n";
    for (int i = 0; i < out.Zombiecnt(); i++)
        out.PrintZombie(i);
    cout << string(50, '=') << "\n";
    return os;
}

/* -------------------------------------------------------------
// player.h
// ------------------------------------------------------------ */

#ifndef player__
#define player__
#include <iostream>

class player
{
public:
    int Coin() const { return coin_; }
    void Cost(const int i) { coin_ -= i; }
    int Position() const { return position_; }
    void Step(const int i, const int row) { position_ = (position_ +
i) % row; }

private:
    int coin_ = 150;
    int position_ = 0;
};

std::ostream &operator<<(std::ostream &os, const player &out);

#endif

/* -------------------------------------------------------------
// player.cpp
// ----------------------------------------------------------- */

#include <iostream>
#include "player.h"
using namespace std;

std::ostream &operator<<(std::ostream &os, const player &out)
{
   cout << "Player coin: " << out.Coin() << "    ";

   return os;
}

/* -------------------------------------------------------------
// zombie.h
// ----------------------------------------------------------- */

#ifndef zombie__
#define zombie__
#include <iostream>

class zombie
{
public:
   zombie()
   {
      id_ = idcount_;
      count_ += 1;
      idcount_ += 1;
   }
   ~zombie() { count_ -= 1; }
   static int Count() { return count_; }
   int ID() const { return id_; }
   int DamagePoint() const { return damage_point_; }
   int LifePoint() const { return life_point_; }
   int Position() const { return position_; }
   bool isDead() const { return (life_point_ <= 0); }
   void Step(const int i, const int row) { position_ = (position_ +
i) % row; }
   void Hurt(const int hp);

private:
   static int count_;
   static int idcount_;
   int id_ = -1;
   const int damage_point_ = 15;
   int life_point_ = 40;
   int position_ = 0;
};

std::ostream &operator<<(std::ostream &os, const zombie &out);
```

```
#endif

/* -----------------------------------------------------------
// zombie.cpp
// --------------------------------------------------------- */

#include <iostream>
#include "zombie.h"
using namespace std;

int zombie::count_ = 0;
int zombie::idcount_ = 0;

void zombie::Hurt(const int hp)
{
    life_point_ -= hp;
    if (life_point_ < 0)
        life_point_ = 0;
}

std::ostream &operator<<(std::ostream &os, const zombie &out)
{
    cout << "[" << out.ID() << "] Damage:" << out.DamagePoint()
         << " HP:"
         << string(out.LifePoint(), '*');
    return os;
}

/* -----------------------------------------------------------
// land.h
// --------------------------------------------------------- */

#ifndef land__
#define land__
#include <iostream>
#include "plant.h"

class land
{
public:
    void PrintPlant() const { std::cout << *p; }

    void SetPlant(const char T) { p = create_p(T); }
    bool isEmpty() const { return (p == nullptr); }
    int PlantCost() const { return p->showcost(); }
    int PlantMHP() const { return p->showmaxhp(); }
    int PlantHP() const { return p->showhp(); }
    int PlantDP() { return p->attack(); }
    void PlantHeal() { p->healed(); }
    void PlantHurt(int hp);
    void CheckAlive();
    std::string PlantName() { return p->showname(); }
    int PlantVisit() { return p->visit(); } //1: Coin 2: Heal
    bool PlantisReady() const { return p->isready(); }
    int PlantGet() { return p->get(); } //1:
    bool isOffensive() const { return p->isoffensive(); }

private:
    plant *p = nullptr;
    //plant
};

std::ostream &operator<<(std::ostream &os, const land &out);

#endif

/* -------------------------------------------------------------
// land.cpp
// ----------------------------------------------------------- */

#include <iostream>
#include "land.h"
#include "plant.h"
using namespace std;

void land::PlantHurt(const int hp)
{
    p->attacked(hp);
    CheckAlive();
}

void land::CheckAlive()
{
    if (!alive(*p))
    {
        delete p;
        p = nullptr;
    }
}

std::ostream &operator<<(std::ostream &os, const land &out)
{
    if (out.isEmpty())
    {
        cout << "Empty";
    }
    else
    {
        out.PrintPlant(); //print plant
    }
    return os;
}

/* -------------------------------------------------------------
// plant.h
// ----------------------------------------------------------- */

          #ifndef plant__
#define plant__
#include<iostream>
class plant
{
public:
    plant(std::string name,int cost,int maxhp,int atk);
    virtual ~plant();
    std::string showname() const;
    int showcost() const;
    int showmaxhp() const;
    int showhp() const;
    int showatk() const;
    virtual void showskill() const;
    virtual int visit();
    virtual bool isready() const;
    virtual int get();
    virtual bool isoffensive()  const = 0;
    int attack();
    void attacked(int atk);
    void healed();
protected:
    static void sethealpoint(int healpoint);
    virtual void  atk_impl();

private:
    std::string name_;
    int cost_;
    int maxhp_;
    int remainhp_;
    int atk_;
    static int healpoint_;
};

class ShootPlant : public plant
{
public:
    static void set(std::string name, int cost, int hp, int atk);
    static int showcost();
    ShootPlant();
    virtual bool isoffensive() const override;
    static void print();
private:
    static std::string name_;
    static int cost_;
    static int maxhp_;
    static int atk_;
};

class BombPlant : public plant
{
public:
    static void set(std::string name,int cost, int hp);
    static int showcost();
    BombPlant();
    ~BombPlant() ;
    virtual bool isoffensive() const override;
    static int showused() ;
    static void print();
protected:
    virtual void  atk_impl() override;
private:
    static std::string name_;
    static int cost_;
    static int maxhp_;
    static int used_;
};

class CoinPlant : public plant
```

```
{
public:
    static void set(std::string name, int cost, int hp, int time, int
coin);
    static int showcost();
    CoinPlant();
    virtual int visit() override;
    virtual bool isready() const override;
    virtual int get() override;
    virtual void showskill() const override;
    virtual bool isoffensive() const override;
    static void print();
private:
    static std::string name_;
    static int cost_;
    static int maxhp_;
    static int maxtime_;
    static int reward_;
    int time_ = -1;
};

class HealPlant : public plant
{
public:
    static void set(std::string name, int cost, int hp, int healpoint);
    static int showcost();
    HealPlant();
    virtual int visit() override;
    virtual int get() override;
    virtual bool isoffensive() const override;
    static void print();
private:
    static std::string name_;
    static int cost_;
    static int maxhp_;
    static int healpoint_;
};

plant* create_p(char type);

plant& create_r(char type);

bool alive (const plant& pl);

std::ostream & operator << (std::ostream &os, const plant &rhs);

void show(char type);

int showcost(char type);


#endif

/* -------------------------------------------------------------
// plant.cpp
// ----------------------------------------------------------- */

#include<string>
#include<iostream>
```

```
#include"plant.h"

using namespace std;

//static*****************************************************************
***************

int plant::healpoint_ = 20;

string ShootPlant::name_ = "Hornflower";

int ShootPlant::cost_ = 50;

int ShootPlant::maxhp_ = 50;

int ShootPlant::atk_ = 10;

string BombPlant::name_ = "Bombflower";

int BombPlant::cost_ = 50;

int BombPlant::maxhp_ = 30;

int BombPlant::used_ = 0;

string CoinPlant::name_ = "Mushroom";

int CoinPlant::cost_ = 25;

int CoinPlant::maxhp_ = 50;

int CoinPlant::maxtime_ = 2;

int CoinPlant::reward_ = 25;

string HealPlant::name_ = "Healflower";

int HealPlant::cost_ = 25;

int HealPlant::maxhp_ = 50;

int HealPlant::healpoint_ = 20;

//plant******************************************************************
**************************************************************

plant::plant(string name,int cost,int maxhp,int atk) :name_(name),
cost_(cost), maxhp_(maxhp), remainhp_(maxhp), atk_(atk)
{

}

plant::~plant()
{

}

string plant::showname() const
{
    return name_;
}

int plant::showcost() const
{
    return cost_;
}

int plant::showmaxhp() const
{
    return maxhp_;
}

int plant::showhp() const
{
    return remainhp_;
}

int plant::showatk() const
{
    return atk_;
}


void plant::showskill() const
{

}

int plant::visit()
{
    return 0;
}

bool plant::isready() const
{
    return false;
}

int plant::get()
{
    return -1;
}

int plant::attack()
{
    atk_impl();
    return plant::showatk();
}

void plant::attacked(int atk)
{
    remainhp_ -= atk;
}

void plant::healed()
{
    remainhp_ += healpoint_;
    if(remainhp_>maxhp_)
    {
        remainhp_ = maxhp_;
    }
}


void plant::sethealpoint(int healpoint)
{
    healpoint_ = healpoint;
}

void  plant::atk_impl()
{

}

//Shoot*****************************************************************
****************

void ShootPlant::set(string name, int cost, int hp, int atk)
{
    name_ = name;
    cost_ = cost;
    maxhp_ = hp;
    atk_ = atk;
}

int ShootPlant::showcost()
{
    return cost_;
}

ShootPlant::ShootPlant(): plant(name_,cost_,maxhp_,atk_)
{

}

bool ShootPlant::isoffensive() const
{
    return true;
}

void ShootPlant::print()
{
    cout << name_  << " $"<< cost_ << " HP: " << maxhp_ << " - gives
" << atk_ << " damage points" ;
}

//Bomb******************************************************************
*************************************

void BombPlant::set(string name,int cost, int hp)
{
    name_ = name;
    cost_ = cost;
    maxhp_ = hp;
}

int BombPlant::showcost()
```

```
{
    return cost_;
}

BombPlant::BombPlant(): plant(name_,cost_,maxhp_,maxhp_)
{

}

BombPlant::~BombPlant()
{
    used_+=1;
}

bool BombPlant::isoffensive() const
{
    return true;
}

int BombPlant::showused()
{
    return used_;
}

void BombPlant::print()
{
    cout << name_  << " $"<< cost_ << " HP: " << maxhp_ << " - gives
" << maxhp_ << " damage points" ;
}

void BombPlant::atk_impl()
{
    this->attacked(this->showhp());
}

//Coin*************************************************************
*****************************************

void CoinPlant::set(string name, int cost, int hp, int time, int coin)
{
    name_ = name;
    cost_ = cost;
    maxhp_ = hp;
    maxtime_ = time;
    reward_ = coin;
}

int CoinPlant::showcost()
{
    return cost_;
}

CoinPlant::CoinPlant() : plant(name_,cost_,maxhp_,0)
{
    time_ = maxtime_;
}

int CoinPlant::visit()
{
    time_-=1;
    return 1;
}

bool CoinPlant::isready() const
{
    return !time_;
}

int CoinPlant::get()
{
    if(time_)
    {
        return time_;
    }
    time_ = maxtime_;
    return reward_;
}
void CoinPlant::showskill() const
{
    cout << " (" << time_ << " more visits)";
}

bool CoinPlant::isoffensive() const
{
    return false;
}

void CoinPlant::print()
{
    cout << name_  << " $"<< cost_ << " HP: " << maxhp_ << " - gives
$" << reward_ << " every " << maxtime_ << " rounds";
}

//Heal************************************************************
**********************************************************

void HealPlant::set(string name, int cost, int hp, int healpoint)
{
    name_ = name;
    cost_ = cost;
    maxhp_ = hp;
    healpoint_ = healpoint;
    sethealpoint(healpoint);
}

int HealPlant::showcost()
{
    return cost_;
}

HealPlant::HealPlant() : plant(name_,cost_,maxhp_,0)
{

}

int HealPlant::visit()
{
    return 2;
```

```
}

int HealPlant::get()
{
    return healpoint_;
}

bool HealPlant::isoffensive() const
{
    return false;
}

void HealPlant::print()
{
    cout << name_ << " $"<< cost_ << " HP: " << maxhp_ << " - gives
all your plants " << healpoint_ << " HP back";
}

//non-class*******************************************************
*******************************************************

plant* create_p(char type)
{
    plant* temp = nullptr;
    switch(type)
    {
    case 'S':
        temp = new ShootPlant;
        break;
    case 'B':
        temp = new BombPlant;
        break;
    case 'C':
        temp = new CoinPlant;
        break;
    case 'H':
        temp = new HealPlant;
        break;
    }
    return temp;
}

plant& create_r(char type)
{
    plant* temp = nullptr;
    switch(type)
    {
    case 'S':
        temp = new ShootPlant;
        break;
    case 'B':
        temp = new BombPlant;
        break;
    case 'C':
        temp = new CoinPlant;
        break;
    case 'H':
        temp = new HealPlant;
        break;
    }
    return *temp;
}

bool alive (const plant& pl)
{
    if(pl.showhp()<=0)
    {
        return false;
    }
    return true;
}

ostream & operator << (ostream &os, const plant &rhs)
{
    os << rhs.showname() << " HP: " << rhs.showhp() ;
    rhs.showskill();
    return os;
}

void show(char type)
{
    switch(type)
    {
    case 'S':
        ShootPlant::print();
        break;
    case 'B':
        BombPlant::print();
        break;
    case 'C':
        CoinPlant::print();
        break;
    case 'H':
        HealPlant::print();
        break;
    }
}

int showcost(char type)
{
    int cost = 0;
    switch(type)
    {
    case 'S':
        cost = ShootPlant::showcost();
        break;
    case 'B':
        cost = BombPlant::showcost();
        break;
    case 'C':
        cost = CoinPlant::showcost();
        break;
    case 'H':
        cost = HealPlant::showcost();
        break;
    }
    return cost;
}
```