CCP 6124 – OBJECT ORIENTED PROGRAMMING & DATA STRUCTURES

ASSINGMENT

Tutorial Section:

T10L

Group No:

07

Student Name	**Student Email**	**Student ID**
IMAN THAQIF BIN NASARUDDIN	**iman.thaqif.nasaruddin@student.mmu.edu.my**	**242UC245G9**
AMIRA RAHEEMA BINTI MOHAMAD KAMAROL	**amira.raheema.mohamad@student.mmu.edu.my**	**242UC244MB**
MOHAMMED IEMAN BIN ZAHARI	**mohammed.ieman.zahari@student.mmu.edu.my**	**242UC244SN**
LIEW ZHI YONG	**liew.zhi.yong@student.mmu.edu.my**	**242UC244TH**

1.0 Task Distribution

No	Students Name	Task Distribution
1	Iman Thaqiff Bin Nasaruddin	Main Base Code, Structure, Battlefield class, ClearDisplay (like a refresh function), Final fixes and Updates. (Border limit, Steps problem,etc)
2	Amira Raheema Binti Mohamad Kamarol	Base code for look. Fixes for fire and move. Flowchart, Reports, border, Finalise Documentation, fixes and updates.
3	Mohammed Ieman Bin Zahari	Base code for move, fire, robot type. Class diagram. Bug fixes and updates (Robot suicide, shell used bug, etc)
4	Liew Zhi Yong	Implement new robot types, Base flowchart, Base code for new robot types, fixes and updates.

2.0 Requirements Fulfilled

No	Requirements	Remark
1	Design documentation (must include a class diagram)	Completed
2	Initialization of a simulation.	Completed
3	Display and logging of the the status of the battlefield at each turn.	Completed
4	Display and logging of the actions and the status of each robot at each turn.	Completed
5	Implementation of the required robot classes with OOP concepts.	Completed
6	The algorithms used to optimize the actions of robots listed in the assignment document.	Completed
7	Implementation of three new robot classes (3 marks for each robot)	Completed

3.0 Code Snippets

3.1 Example of Inheritance

```
class GenericRobot : public Robot, public MovingRobot, public ShootingRobot, public
SeeingRobot, public ThinkingRobot {
private:
    int shells = 10;
    set<UpgradeArea> chosenUpgrades;

    bool hasHide = false;
    bool hasJump = false;
    bool hasLongShot = false;
    bool hasSemiAuto = false;
    bool hasThirtyShot = false;
    bool hasScout = false;
    bool hasTrack = false;

    int hideCount = 3;
    int jumpCount = 3;
    int scoutCount = 3;
    int trackerCount = 3;

public:
    bool isHideBot() const { return hasHide; }
    bool isJumpBot() const { return hasJump; }
    bool isLongShot() const { return hasLongShot; }
    bool isSemiAuto() const { return hasSemiAuto; }
    bool isThirtyShot() const { return hasThirtyShot; }
    bool isScout() const { return hasScout; }
    bool isTrackBot() const { return hasTrack; }

    GenericRobot(string name, int x, int y) : Robot(name, x, y) {}

    void applyUpgrade(const string& upgradeName);

    void move(int dx, int dy, int maxWidth, int maxHeight, Logger* logger) override {
        int oldX = positionX, oldY = positionY;
        positionX = max(0, min(positionX + dx, maxWidth - 1));
        positionY = max(0, min(positionY + dy, maxHeight - 1));

        stringstream ss;
        ss << name << " moved from (" << oldX << "," << oldY << ") to (" << positionX << ","
<< positionY << ")";
        cout << ss.str() << endl;
        if (logger) logger->log(ss.str());
    }

    void fire(int targetX, int targetY, Logger* logger) override {
        if (shells <= 0) {
            cout << name << " is out of ammo!\n";
            if (logger) logger->log(name + " is out of ammo!");
            return;
        }
        shells--;
        stringstream ss;
        ss << name << " fires at (" << targetX << "," << targetY << "). Shells left: " << shells;
        cout << ss.str() << endl;
        if (logger) logger->log(ss.str());
    }
```

```
   void look(int x, int y, Logger* logger) override {
      stringstream ss;
      ss << name << " is looking at (" << x << "," << y << ")";
      cout << ss.str() << endl;
      if (logger) logger->log(ss.str());
   }

   void think(Battlefield* battlefield, int maxWidth, int maxHeight, Logger* logger) override {
      if (logger) logger->log(name + " is thinking...");
      cout << name << " is thinking..." << endl;

      // Use JumpBot
      if (hasJump && jumpCount > 0 && (rand() % 10 == 0)) {
         positionX = rand() % maxWidth;
         positionY = rand() % maxHeight;
         jumpCount--;

         stringstream ss;
         ss << name << " used JumpBot ability to jump to (" << positionX << "," << positionY
<< ")";
         cout << ss.str() << endl;
         if (logger) logger->log(ss.str());
         return;
      }

      // Use HideBot
      if (hasHide && hideCount > 0 && (rand() % 10 == 0)) {
         hideCount--;

         stringstream ss;
         ss << name << " used HideBot ability and is hiding this turn.";
         cout << ss.str() << endl;
         if (logger) logger->log(ss.str());
         return;
      }

      int dx = (rand() % 3) - 1;
      int dy = (rand() % 3) - 1;
      int tx = positionX + dx;
      int ty = positionY + dy;

       //avoid robot suicide
      if (dx == 0 && dy == 0){
         if(logger) logger->log(name + " Skipped firing to avoid shooting itself.");
         cout << name << " Skipped firing to avoid shooting itself." << endl;
         return;
      }
```

```
        // Use ScoutBot
        if (hasScout && scoutCount > 0 && (rand() % 10 == 0)) {
            scoutCount--;

            stringstream ss;
            ss << name << " used ScoutBot to scan the entire battlefield.";
            cout << ss.str() << endl;
            if (logger) logger->log(ss.str());
        }
        // Optional: implement scan display logic here
        if (rand() % 2 == 0) {
            look(positionX + dx, positionY + dy, logger);
        } else {
            look(tx, ty, logger);
        }

        // FIRE logic with upgrade + hit check
        if (shells <= 0) {
            cout << name << " is out of ammo!\n";
            if (logger) logger->log(name + " is out of ammo!");
        } else {
            if (hasSemiAuto) {
                for (int i = 0; i < 3 && shells > 0; ++i) {
                    shells--;
                    stringstream ss;
                    ss << name << " fires at (" << tx << "," << ty << "). Shells left: " << shells;
                    cout << ss.str() << endl;
                    if (logger) logger->log(ss.str());
                    battlefield->checkAndHitRobot(tx, ty, this);
                }
            } else if (hasLongShot) {
                int range = 1 + rand() % 3;
                int lx = positionX + dx * range;
                int ly = positionY + dy * range;
                shells--;
                stringstream ss;
                ss << name << " fires (LongShot) at (" << lx << "," << ly << "). Shells left: " <<
shells;
                cout << ss.str() << endl;
                if (logger) logger->log(ss.str());
                battlefield->checkAndHitRobot(lx, ly, this);

            } else {
                shells--;
                stringstream ss;
                ss << name << " fires at (" << tx << "," << ty << "). Shells left: " << shells;
                cout << ss.str() << endl;
                if (logger) logger->log(ss.str());
                battlefield->checkAndHitRobot(tx,  ty, this);
            }
        }

        //normal move
        if (rand() % 2 == 0) {
            move(dx, dy, maxWidth, maxHeight, logger);
        }
    }
};
```

Explanation :

- The code uses multiple inheritance, where the GenericRobot class inherits from five base classes: Robot, MovingRobot, ShootingRobot, SeeingRobot, and ThinkingRobot.
- The GenericRobot class includes a constructor defined as:
  GenericRobot(string name, int x, int y) : Robot(name, x, y) {}
- The entire code is designed to support robot upgrades, incorporating all inherited virtual methods.
- GenericRobot inherits :
    - Robot : name, positions, lives
    - MovingRobot : Abstract interface for movement
    - ShootingRobot : Abstract interface for shooting
    - SeeingRobot : Abstract interface for looking
    - ThinkingRobot : Abstract interface for AI logic

3.2 Example of Polymorphism

```
class MovingRobot {
public:
    virtual void move(int dx, int dy, int maxWidth, int maxHeight, Logger* logger) = 0;
    virtual ~MovingRobot() = default;
};

// Abstract class for shooting
class ShootingRobot {
public:
    virtual void fire(int targetX, int targetY, Logger* logger) = 0;
    virtual ~ShootingRobot() = default;
};

// Abstract class for vision
class SeeingRobot {
public:
    virtual void look(int offsetX, int offsetY, Logger* logger) = 0;
    virtual ~SeeingRobot() = default;
};

// Abstract class for strategy/thinking
class ThinkingRobot {
public:
    virtual void think(class Battlefield* battlefield, int maxWidth, int maxHeight, Logger* logger) = 0;
    virtual ~ThinkingRobot() = default;
};
```

Explanation :

- This abstract class uses virtual functions to implement polymorphism.
- The line :
    - virtual void move(int dx, int dy, int maxWidth, int maxHeight, Logger* logger) = 0;
    - virtual void fire(int targetX, int targetY, Logger* logger) = 0;
    - virtual void look(int offsetX, int offsetY, Logger* logger) = 0;
    - virtual void think(class Battlefield* battlefield, int maxWidth, int maxHeight, Logger* logger) = 0;

3.3 Example of Operator Overloading

```
class Robot {
protected:
    string name;
    int positionX;
    int positionY;
    char symbol;
    int lives;

public:
    Robot(string name, int x, int y) : name(name), positionX(x), positionY(y), lives(3) {
        symbol = name.empty()  ? 'R' : toupper(name[0]);
    }
    virtual ~Robot() = default;

    string getName() const { return name; }
    int getX() const { return positionX; }
    int getY() const { return positionY; }
    int getLives() const { return lives; }
    void setLives(int l) { lives = 3; } //-- setter
    void loseLife()  { if (lives>0)  --lives; }
    char getSymbol() const { return symbol; }

    bool operator==(const  Robot& other) const {
        return positionX == other.positionX  && positionY == other.positionY;
    }

    friend ostream& operator<<(ostream&  os, const Robot& robot) {
        os << robot.name  << " (" << robot.symbol << ") at ["
          << robot.positionX  << "," << robot.positionY  << "]";
        return os;
    }
};
```

Explanation :

- The Robot class contains 2 Operator Overloading :
    - bool operator==(const Robot& other) const {
    - friend ostream& operator<<(ostream& os, const Robot& robot) {
- The operator == is used to compare two robots based on their positions rather than their memory addresses.
- The operator << is used to output the information of a Robot object.

3.4 Full Code Explanation

```
#include <iostream>
#include <fstream>
#include <vector>
```

```
#include <string>
#include <algorithm>
#include <iomanip>
#include <memory>
#include <sstream>
#include <windows.h>
#include <set>

using namespace std;

enum UpgradeArea { NONE, MOVE, SHOOT, SEE };

class Logger {
private:
    ofstream logFile;

public:
    Logger(const string& filename) {
        logFile.open(filename, ios::out);
        if (!logFile.is_open()) {
            cerr << "Failed to open log file.\n";
        }
    }

    ~Logger() {
        if (logFile.is_open()) {
            logFile.close();
        }
    }

    void log(const string& message) {
        if (logFile.is_open()) {
            logFile << message << endl;
        }
    }
};
```

Explanation :

- Defines an enum for robot upgrade areas. (NONE, MOVE, SHOOT, SEE)
- Implements a Logger class to handle writing messages to a log file.
- The log file will keep track of RobotWar simulation.

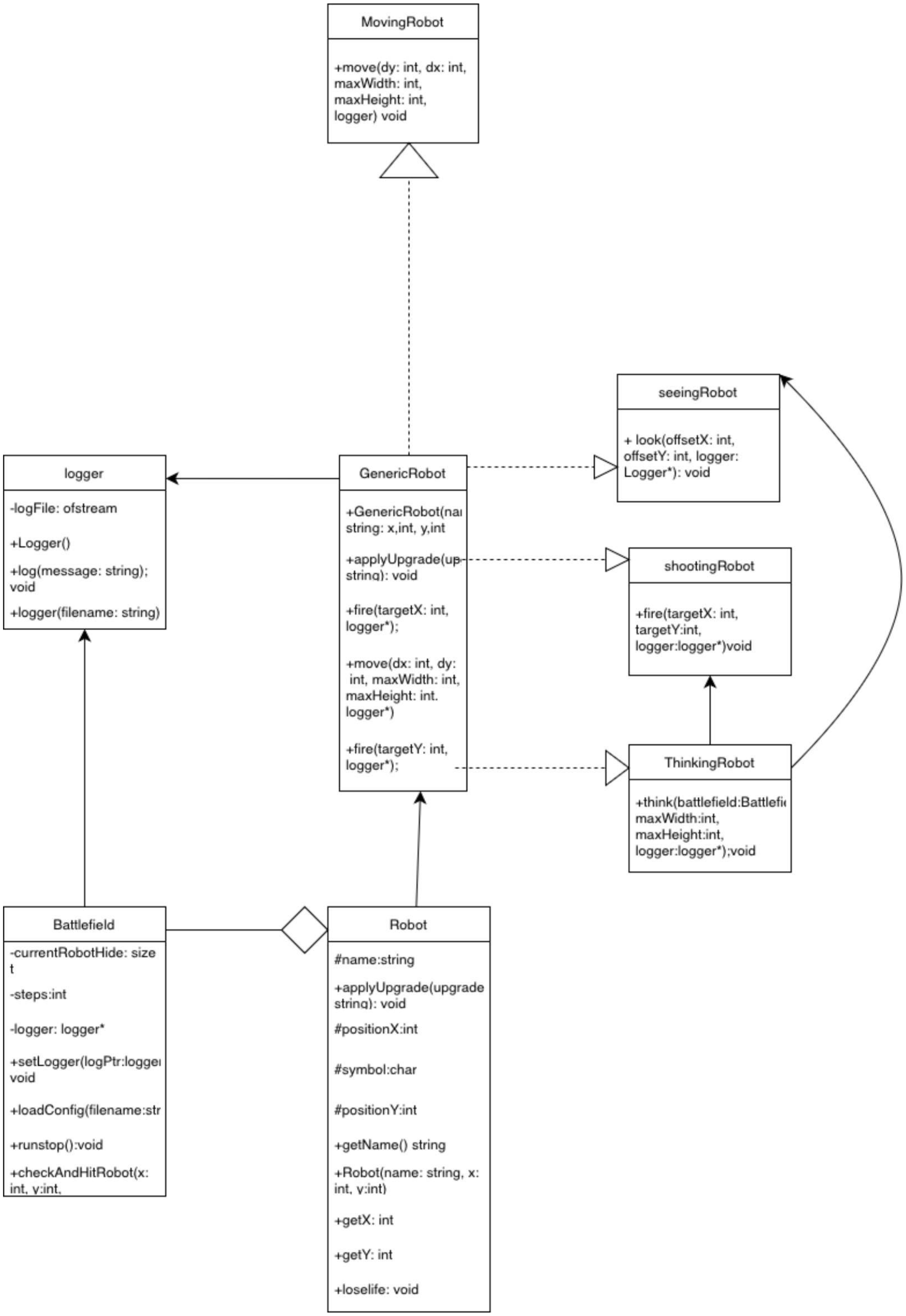
```
void clearScreen() {
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD coord = {0, 0};
    DWORD count;
    CONSOLE_SCREEN_BUFFER_INFO csbi;

    GetConsoleScreenBufferInfo(hStdOut, &csbi);
    FillConsoleOutputCharacter(hStdOut, ' ', csbi.dwSize.X * csbi.dwSize.Y, coord, &count);
    SetConsoleCursorPosition(hStdOut, coord);
}
```

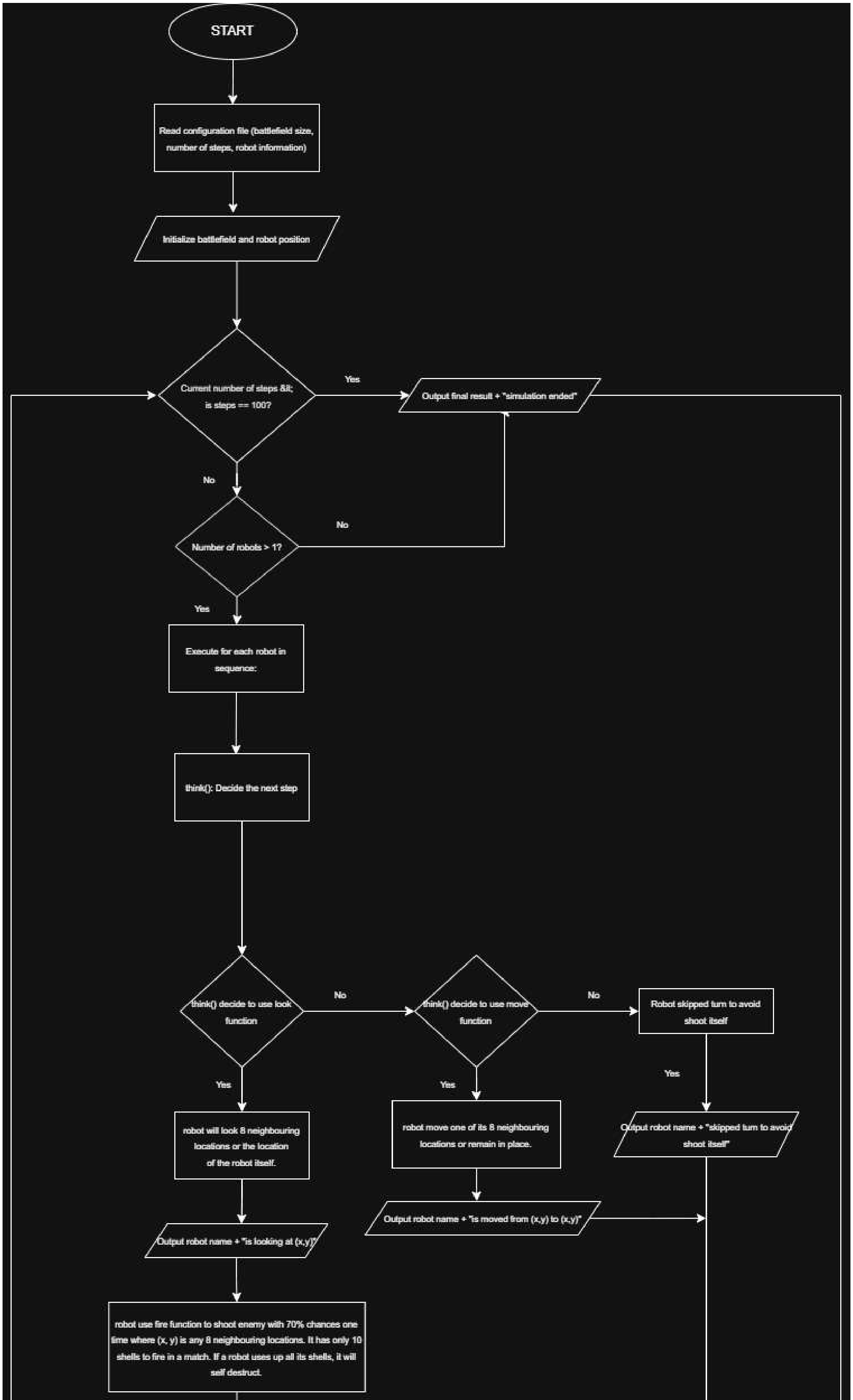
Explanation :

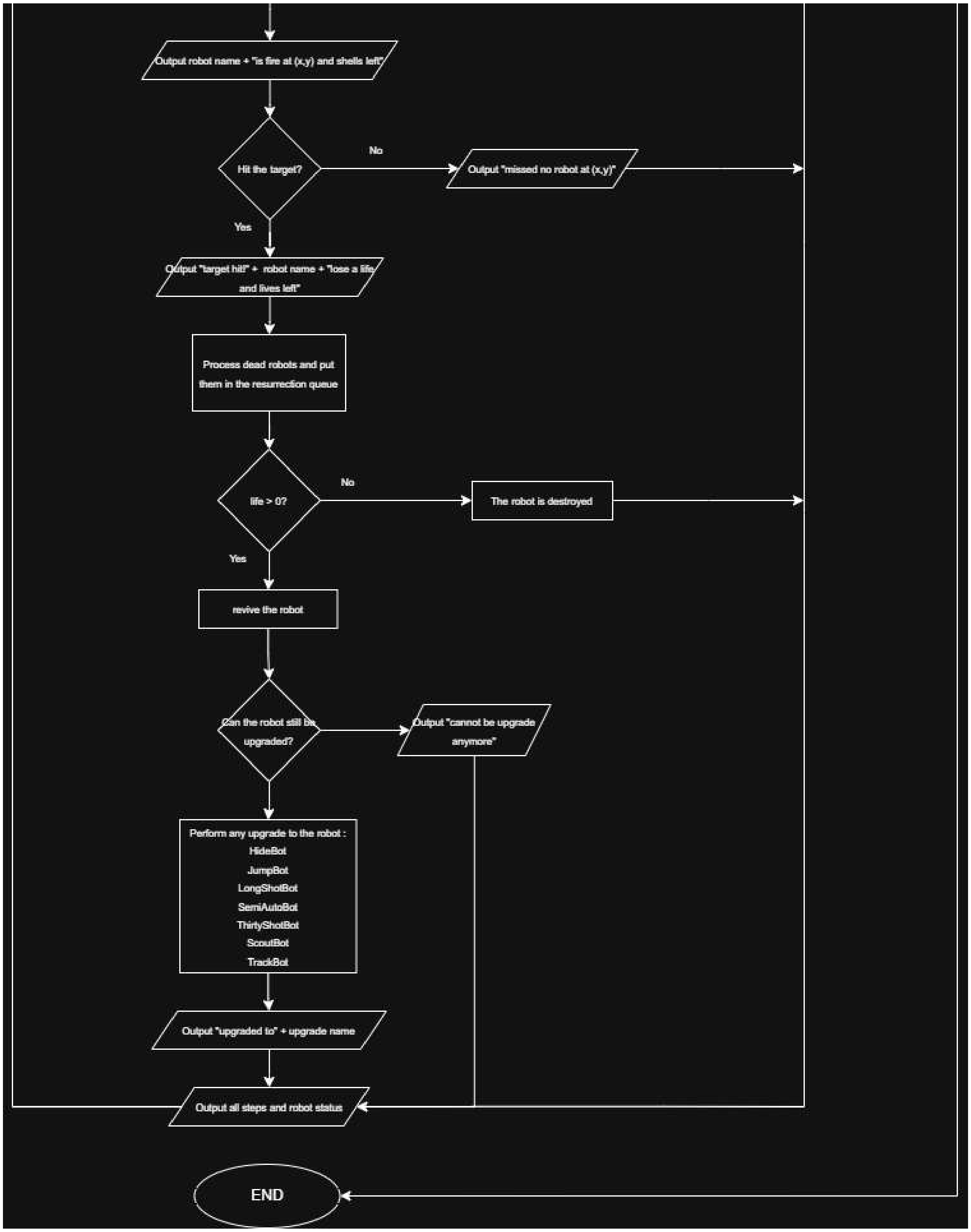
- To clear the console screen in Windows, the screen is cleared at the beginning of each robot's turn. In short, it refreshes the simulation at the interval that has been set.

4.0 Class Diagram


MovingRobot
+move(dy: int, dx: int, maxWidth: int, maxHeight: int, logger) void
seeingRobot
+ look(offsetX: int, offsetY: int, logger: Logger*): void
logger
-logFile: ofstream
+Logger()
+log(message: string); void
+logger(filename: string)
GenericRobot
+GenericRobot(nai string: x,int, y,int
+applyUpgrade(up string): void
+fire(targetX: int, logger*);
+move(dx: int, dy: int, maxWidth: int, maxHeight: int. logger*)
+fire(targetY: int, logger*);
shootingRobot
+fire(targetX: int, targetY:int, logger:logger*)void
ThinkingRobot
+think(battlefield:Battlefie maxWidth:int, maxHeight:int, logger:logger*);void
Battlefield
-currentRobotHide: size t
-steps:int
-logger: logger*
+setLogger(logPtr:logger void
+loadConfig(filename:str
+runstop():void
+checkAndHitRobot(x: int, y:int,
Robot
#name:string
+applyUpgrade(upgrade string): void
#positionX:int
#symbol:char
#positionY:int
+getName() string
+Robot(name: string, x: int, y:int)
+getX: int
+getY: int
+loselife: void

5.0 Flowchart


START
Read configuration file (battlefield size, number of steps, robot information)
Initialize battlefield and robot position
Current number of steps &lt; is steps == 100?
Yes
Output final result + "simulation ended"
No
Number of robots > 1?
No
Yes
Execute for each robot in sequence:
think(): Decide the next step
think() decide to use look function
No
think() decide to use move function
No
Robot skipped turn to avoid shoot itself
Yes
Yes
Yes
robot will look 8 neighbouring locations or the location of the robot itself.
robot move one of its 8 neighbouring locations or remain in place.
Output robot name + "skipped turn to avoid shoot itself"
Output robot name + "is moved from (x,y) to (x,y)"
Output robot name + "is looking at (x,y)"
robot use fire function to shoot enemy with 70% chances one time where (x, y) is any 8 neighbouring locations. It has only 10 shells to fire in a match. If a robot uses up all its shells, it will self destruct.


Output robot name + "is fire at (x,y) and shells left"
Hit the target?
No
Output "missed no robot at (x,y)"
Yes
Output "target hit!" + robot name + "lose a life and lives left"
Process dead robots and put them in the resurrection queue
life > 0?
No
The robot is destroyed
Yes
revive the robot
Can the robot still be upgraded?
Output "cannot be upgrade anymore"
Perform any upgrade to the robot :
HideBot
JumpBot
LongShotBot
SemiAutoBot
ThirtyShotBot
ScoutBot
TrackBot
Output "upgraded to" + upgrade name
Output all steps and robot status
END