Faculty of Computing and Informatics

Multimedia University

Cyberjaya

CCP6124 - OBJECT-ORIENTED PROGRAMMING AND DATA STRUCTURES (OOPDS)

Trimester March/April 2025

Lecture Section: TC1L

Tutorial Section: TT2L

Group Members:

NAME	STUDENT ID
LAW CHIN XUAN	242UC244GC
CHOW YING TONG	242UC244NK
MUI RUI XIN	243UC247CT

Table of Contents

1.0 Implementation

1.1 Inputs

An example text file looks like this:`

```
M by N : 40 50
steps: 300
robots: 5
GenericRobot  Kidd  3    6
GenericRobot  Jet   12   1
GenericRobot  Alpha 35   20
GenericRobot  Beta  20   37
GenericRobot  Star  random    random
```

1.2 Outputs

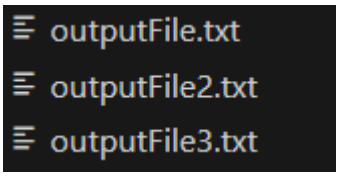
```
C:\Users\SCSM11\Documents\OOPDS Ass_1>g++ main.cpp -o main.exe

C:\Users\SCSM11\Documents\OOPDS Ass_1>main.exe
```

Command: g++ main.cpp -o main.exe

main.exe

Each time will get one outputFile.txt

As requirements, we have provided 3 outputFile.txt as below:



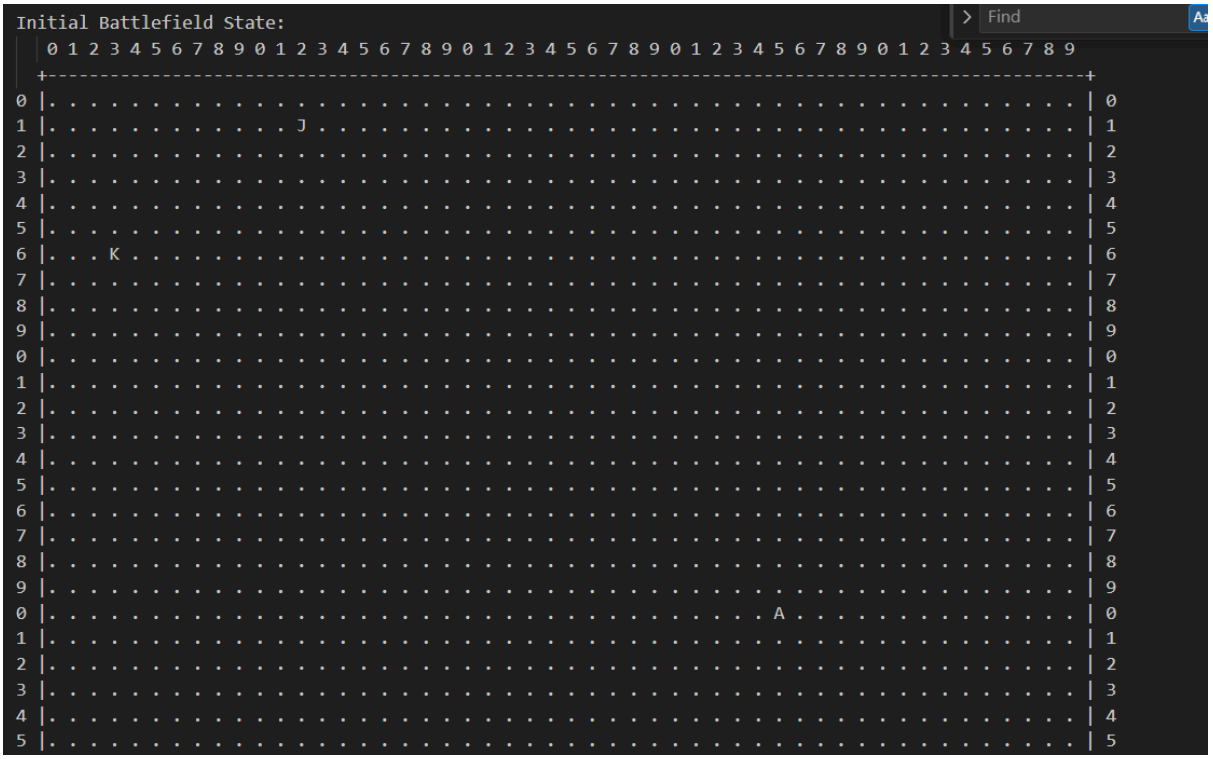
1.3 Outputs Short Explanation

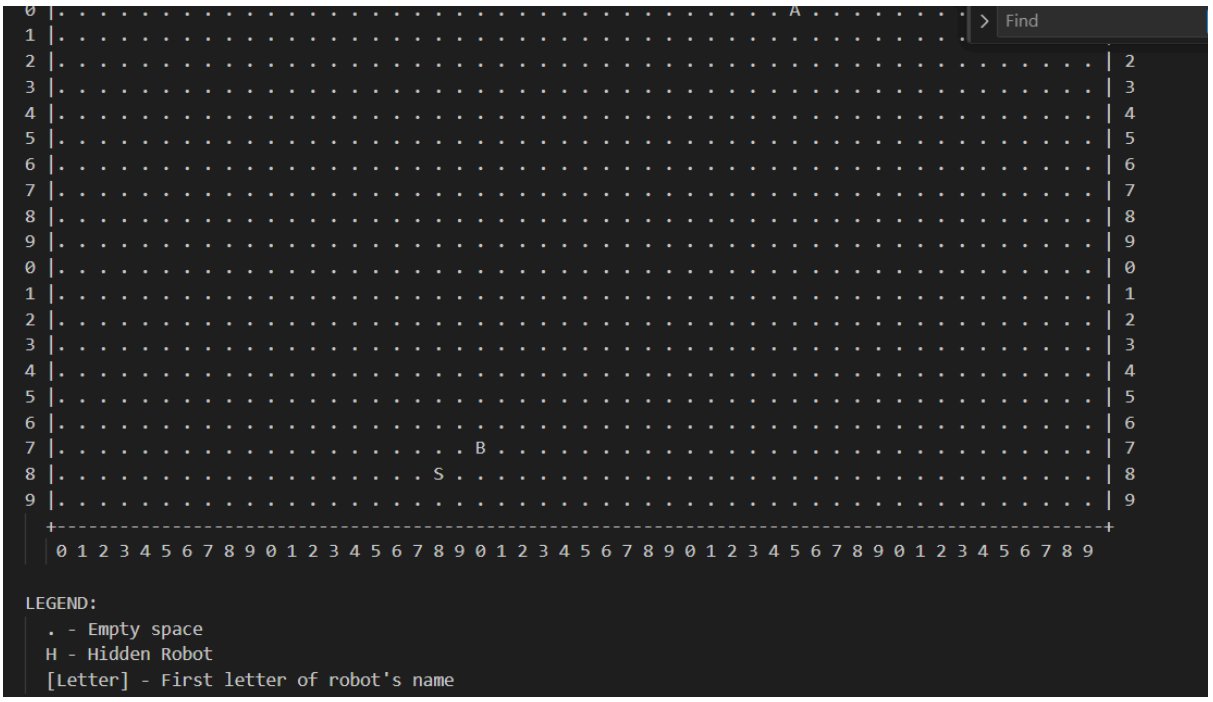
Initial Battlefield Stimulation:

```
Battlefield Dimensions:
Width: 50
Height: 40
Battlefield steps: 300
Battlefield number of robots: 5
Initial Robots on battlefield:
Robot Name: Kidd, Coords: (3,6), Lives: 3
Robot Name: Jet, Coords: (12,1), Lives: 3
Robot Name: Alpha, Coords: (35,20), Lives: 3
Robot Name: Beta, Coords: (20,37), Lives: 3
Robot Name: Star, Coords: (18,38), Lives: 3
```

Will log battlefield dimensions (width, height, steps, number of robots, robots' name, initial robot type, initial coordinates of the robot, initial lives (3))

Initial Battlefield status:





Simulation Started:

1st Step/Round

```
--- Starting Simulation ---

--- Simulation Step 1 ---
Robot Status before Step 1:
  Type: GenericRobot, Name: Kidd, Coords: (3,6), Life: 3, Ammo: 10
  Type: GenericRobot, Name: Jet, Coords: (12,1), Life: 3, Ammo: 10
  Type: GenericRobot, Name: Alpha, Coords: (35,20), Life: 3, Ammo: 10
  Type: GenericRobot, Name: Beta, Coords: (20,37), Life: 3, Ammo: 10
  Type: GenericRobot, Name: Star, Coords: (18,38), Life: 3, Ammo: 10
```

Will log all robot details that are not hurt, alive, not requeue (robot type, robot name, robot's coordinates , robot's life , robot's ammo at that step/round as result as previous step/round)

1st Turn of GenericRobot Kidd

```
--------------------------------------
Kidd's turn:
Revealing (2, 5): Empty space
Revealing (2, 6): Empty space
Revealing (2, 7): Empty space
Revealing (3, 5): Empty space
Revealing (3, 6): Current position
Revealing (3, 7): Empty space
Revealing (4, 5): Empty space
Revealing (4, 6): Empty space
Revealing (4, 7): Empty space
>> Kidd is thinking...
>> Kidd is moving...
Kidd moved to (4, 7)
No shooting as no robots within shooting range.
Kidd is done.
```

And so on

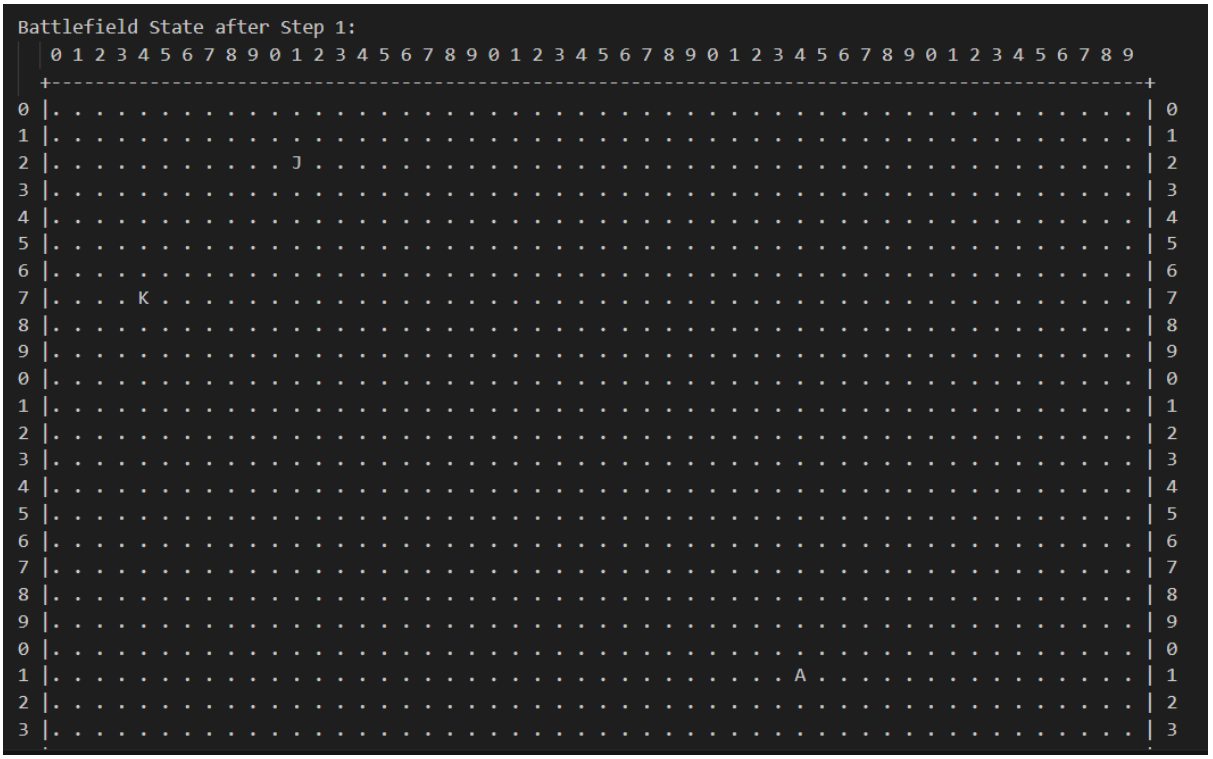
Log robot's perform action

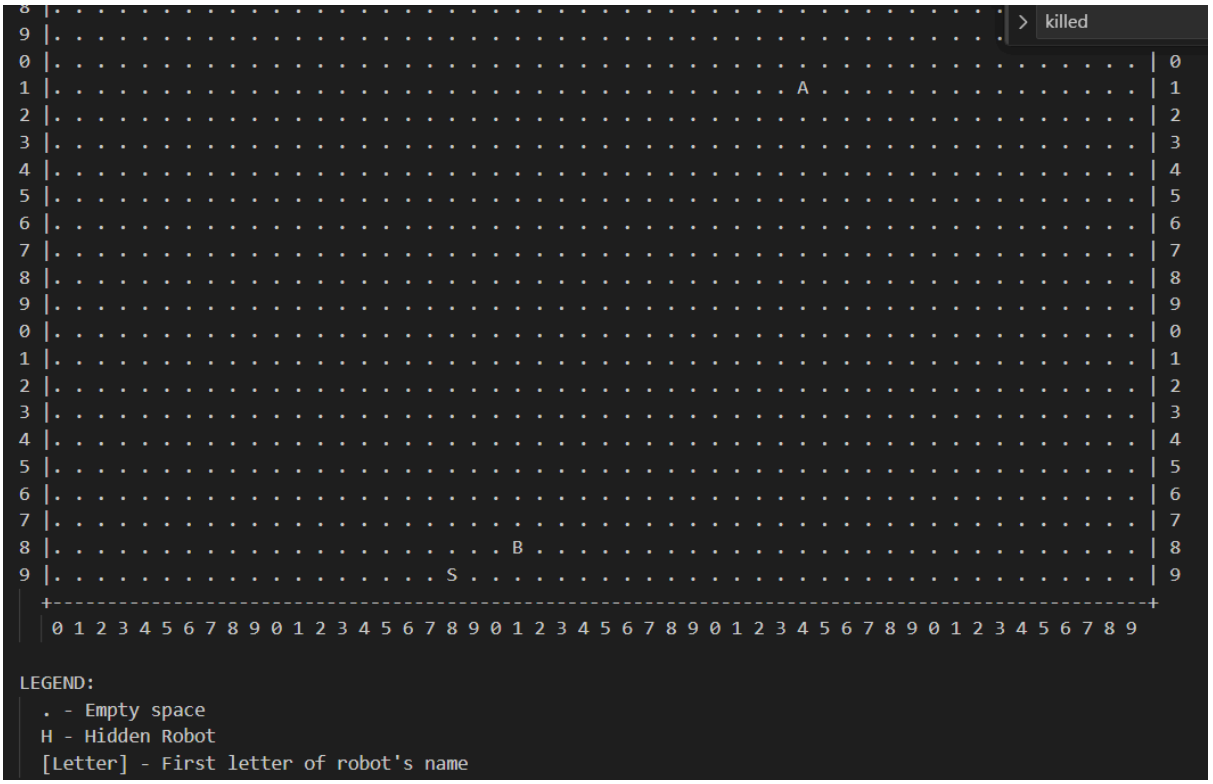
After 1st Round/Step

```
--------------------------------------
Beta has been removed from the battlefield.
--------------------------------------
Reentering Beta
Beta reentered as GenericRobot at (41, 26) with 2 lives and 10 ammo next turn.
```

Log the robot that removed from the battlefield (being hit) at that round, robot's requeue list, robot's reentry details to next round.

Battlefield status after 1st Round/Step





2.0 Class Explanations

2.1 Logger Class Explanation

The Logger class is in charge of outputting the battlefield status and the actions of all robots to the terminal and to a separate .txt file.

Attributes:

outputFile:

An ofstream object to write to the output file

fileName:

A string representing the name of the output file

Constructor:

Default Constructor:

Opens the output file for writing to it later on

Destructor:

~Logger():

Closes the output file

Operator Overloading:

template <typename T>

Logger &operator<<(const T &outputContent) {}

Overloads the << operator to allow Logger to accept any generic data type T as input and forward to cout and output file.

Logger &operator<<(ostream &(*manip)(ostream &))

Overloads the << operator to allow Logger to accept stream manipulators like endl and forward to cout and output file.

2.2 Battlefield Class Explanation

The Battlefield class manages the entire battlefield. It initializes, executes and terminates the simulation, places the robots during each turn, ensuring their actions are processed, and records the simulation results.

Attributes:

Private:

int height:

The height of the battlefield.

int width:

The width of the battlefield

int steps:

The maximum steps of a simulation

int numberOfRobots:

The width of the battlefield

vector<Robot *> listOfRobots:

Stores the list of robots on the battlefield.

vector<vector<Robot *>> battlefieldGrid:

Represents the battlefield grid, storing a vector of vectors of Robot objects

Constructor:

Default Constructor:

Purpose: Initialize the battlefield, set height, width, steps, and numberOfRobots to default value of 0.

Destructor:

~Battlefield() {}:

Purpose: Clear resources by deleting all Robot pointers in listOfRobots and clearing the listOfRobots vector.

Getters:

getWidth(), getHeight():

Methods to retrieve the width and height of the battlefield respectively.

getSteps():

Method to retrieve the maximum number of simulation steps.

getNumberOfRobots():

Method to retrieve the number of robots in the battlefield.

getListOfRobots():

Method to retrieve the list of robots in the battlefield.

getNumberOfAliveRobots():

Method to get the number of alive robots in the battlefield.

getRobotAt(int x, int y):

Method to retrieve the information of a robot at the given coordinates

isPositionAvailable(int x, int y):

Checks if the given coordinates is occupied by any robot.

isPositionWithinGrid(int x, int y):

Checks if the given coordinates is within the boundaries of the battlefield.

Setters:

setDimensions(int h, int w):

Sets the height of the battlefield to h, and the width of the battlefield to w.

setSteps(int s):

Sets the steps member variable to s.

setNumberOfRobots(int n):

Sets the numberOfRobots member variable to n.

addNewRobot (Robot *robot):

Adds the provided robot pointer object to the battlefield.

placeRobot(Robot *robot, int x, int y):

Places the provided robot pointer object to position (x, y) in the battlefield

removeRobotFromGrid(Robot *robot):

Removes the provided robot pointer object from the battlefield.

Other Member Functions:

simulationStep():

Executes one simulation step.

displayBattlefield():

Prints the state of the battlefield.

2.3 Robot Class Explanation

The Robot class serves as the foundational abstract base class for different types of robots in the simulation. It encapsulates common attributes and behaviours that all robot types share, providing a structured framework for inheritance and polymorphism.

Attributes:

Private:

int positionX, int positionY

Integers representing the position of the robot on the battlefield grid.

Protected:

string name

Represents the name of the robot

int lives

Represents the number of lives of the robot

bool hidden

Determines if the robot is currently hidden

bool isDie

Flag to mark a robot as dead (lives = 0 or out of ammo). Initialized as false.

bool isHurt

Flag to mark a robot as hurt (was shot at by another robot in the current turn). Initialized as false.

Constructor:

Robot(string name, int x, int y)

: name(name), positionX(x), positionY(y), lives(3), hidden(false) {}

Purpose: Takes the name, x and y positions as arguments, then sets the name, positionX, positionY values as the provided arguments. Set lives to 3 and hidden to false.

Destructor:

virtual ~Robot() {}

Purpose: A virtual destructor to ensure proper cleanup of resources in derived classes.

Getters:

getName()

Retrieves the name of the robot.

getX(), getY()

Retrieves the position of the robot

getLives()

Retrieves the number of lives of the robot

getIsDie()

Retrieves the value of isDie of the robot

getIsHurt()

Retrieves the value of isHurt of the robot

isHidden()

Retrieves the value of isHidden attribute

Setters:

setLives(int numOfLives)

Sets the lives attribute of the robot to numOfLives

setIsDie(bool val)

Sets the isDie attribute to val

setIsHurt(bool val)

Sets the isHurt attribute to val

setPosition(int x, int y)

Sets positionX attribute to x, and positionY attribute to y.

setHidden(bool state)

Sets isHidden attribute to state

Other Member Functions:

takeDamage()

Decrements lives by 1, sets isHurt to true and sets isDie to true if lives == 0.

Pure Virtual Functions:

Protected:

virtual bool canBeHit() = 0

A pure virtual function that must be overridden in derived classes to return the canBeHit attribute of the robot.

Public:

virtual void think() = 0

A pure virtual function that must be overridden in derived classes to perform the think() action.

virtual void act() = 0

A pure virtual function that must be overridden in derived classes to perform the act() action.

virtual void move() = 0

A pure virtual function that must be overridden in derived classes to perform the move() action.

virtual void fire(int x, int y) = 0

A pure virtual function that must be overridden in derived classes to perform the move() action.

virtual void look(int X, int Y) = 0

A pure virtual function that must be overridden in derived classes to perform the look(int X, int Y) action.

The Robot class forms the backbone of the simulation, providing a template for specific robot types to extend and customize their behaviors. Through inheritance, derived classes can implement the pure virtual

functions to introduce specialized actions and symbols, enabling polymorphic behavior during the simulation.

2.4 MovingRobot Class Explanation

The MovingRobot class is an abstract subclass derived from the Robot class. It is designed to provide a base for robots that have moving capabilities within the simulation.

Purpose:

The MovingRobot class introduces the concept of movement to the robots. It extends the Robot class by adding a pure virtual function specifically for moving actions. This allows derived classes to implement specific movement behaviors.

Attributes:

Protected:

int moveCount

Represents the number of times the robot has moved.

Constructor:

```
MovingRobot(const string &name, int x, int y)
    : Robot(name, x, y), moveCount(0) {}
```

Purpose: Initializes a MovingRobot object with the specified name and coordinates. Uses a member initializer list to call the constructor of the base Robot class, passing in name, x and y. It also sets moveCount to 0.

Destructor:

virtual ~MovingRobot() = default;

Purpose: A virtual destructor to ensure proper cleanup of resources in derived classes.

Setters:

incrementMoveCount()

Increments the moveCount attribute by 1.

Member Functions:

boolIsValidMove(int newX, int newY, const Battlefield &battlefield) const {}

Checks if the intended move to (newX, newY) is valid ((newX, newY) is not occupied and within the boundaries of the battlefield grid.

Pure Virtual Functions:

virtual void move() = 0;

This pure virtual function must be overridden by any concrete subclass derived from MovingRobot. The move function is intended to define the movement behavior of the robot on the battlefield.

The MovingRobot class is an abstract subclass with a pure virtual move() function. This way, it enforces that any specific moving robot type must implement the move function. This allows all moving robots to have a consistent interface for movement, while at the same time allowing for diverse movement behaviours.

2.5 ShootingRobot Class Explanation

The ShootingRobot class is an abstract subclass derived from the Robot class. It is designed to provide a base for robots that have shooting capabilities within the simulation.

Purpose:

The ShootingRobot class introduces the concept of shooting to the robots. It extends the Robot class by adding a pure virtual function specifically for shooting actions. This allows derived classes to implement specific shooting patterns and behaviours.

Attributes:

Protected:

int ammo

Integer representing the remaining number of ammo the robot has.

Constructor:

ShootingRobot(const string &name, int x, int y, int initialAmmo)
: Robot(name, x, y), ammo(initialAmmo) {}

Purpose: Initializes a ShootingRobot object with the specified name and coordinates, and initial ammo. Uses a member initializer list to call the constructor of the base Robot class, passing in name, x and y. It also sets the ammo number to the provided initialAmmo argument.

Destructor:

virtual ~ShootingRobot() = default

Purpose: A virtual destructor to ensure proper cleanup of resources in derived classes.

Setters:

useAmmo()

Decrements the ammo attribute by 1.

Getters:

hasAmmo()

Boolean value that represents if the robot has any ammo remaining.

getAmmo()

Retrieves the value of the ammo attribute

Member Functions:

hitProbability() const

Simulates the probability of a 70% chance of hitting the target when robot shoots.

Pure Virtual Functions:

virtual void fire() = 0;

This pure virtual function must be overridden by any concrete subclass derived from ShootingRobot. The move function is intended to define the shooting behavior of the robot on the battlefield.

The ShootingRobot class is an abstract subclass with a pure virtual fire() function. This way, it enforces that any specific moving robot type must

implement the fire function. This allows all moving robots to have a consistent interface for shooting, while at the same time allowing for diverse shooting behaviours.

2.6 SeeingRobot Class Explanation

The SeeingRobot class is an abstract subclass derived from the Robot class. It is designed to provide a base for robots that have vision or perception capabilities within the simulation.

Purpose:

The SeeingRobot class extends the functionality of robots by introducing vision-related capabilities. It serves as an abstract base class that defines the structure of robots that can "see" or perceive their environment.

Attributes:

Protected:

int visionRange

Represents the range of "seeing" of the robot.

vector<Robot *> detectedTargets

Vector to store other robots that were detected.

bool enemyDetectedNearby

Boolean value to determine if the robot detected any targets from "seeing".

Constructor:

```
SeeingRobot(const string &name, int x, int y, int range)
    : Robot(name, x, y), visionRange(range) {}
```

Purpose: Creates a SeeingRobot object by forwarding name, x, and y to the base Robot class constructor and initializing its own visionRange attribute.

Destructor:

virtual ~SeeingRobot() = default;

Purpose: A virtual destructor to ensure proper cleanup of resources in derived classes.

Getters:

getEnemyDetectedNearby() const

Retrieves the enemyDetectedNearby attribute.

getDetectedTargets() const

Retrieves the detectedTargets vector attribute.

Setters:

setDetectedTargets(const vector<Robot *> &targets)

Sets the detectedTargets vector to the specified vector.

setEnemyDetectedNearby(bool detected)

Sets the enemyDetectedNearby attribute to the specified value

Pure Virtual Functions:

virtual void look(int X, int Y) = 0;

This pure virtual function must be overridden by any concrete subclass derived from SeeingRobot. The look function defines the behavior of the robot when it "looks" or perceives the battlefield. It accepts two parameters: the relative position to the robot’s current position that the look function should be performed at.

By defining the SeeingRobot class as an abstract subclass with a pure virtual function, the design mandates that any specific seeing robot type must implement the look function. This ensures a consistent interface for perception-related actions across all seeing robot implementations while allowing for diverse strategies and behaviors based on their unique characteristics.

2.7 ThinkingRobot Class Explanation

The ThinkingRobot class is an abstract subclass derived from the Robot class. It is designed to provide a base for robots that have thinking capabilities within the simulation.

Purpose:

The ThinkingRobot class extends the functionality of robots by introducing thinking-related capabilities. It serves as an abstract base class that defines the structure of robots that can "think" before performing any other actions.

Attributes:

Protected:

int strategyLevel

Integer representing the strategyLevel the robot thinks at. 1 is the default value.

Constructor:

ThinkingRobot(const string &name, int x, int y, int strategy)

: Robot(name, x, y), strategyLevel(strategy) {}

Purpose: Creates a ThinkingRobot object by forwarding name, x, and y to the base Robot class constructor and initializing its own strategy attribute.

Destructor:

virtual ~ThinkingRobot() = default;

Purpose: A virtual destructor to ensure proper cleanup of resources in derived classes.

Getters:

getStrategyLevel() const

Retrieves the strategyLevel attribute.

Member Functions:

void think() override

Overrides the pure virtual think() function of the base Robot class.

2.8 GenericRobot Class Explanation

The GenericRobot class serves as the third level inherited class. It inherits from MovingRobot class, ThinkingRobot class, SeeingRobot class and ShootingRobot class. At the same time, it encapsulates common attributes and behaviours that all robot types share, providing a structured framework for inheritance and polymorphism for 10 first level upgraded robot class.

Attributes:

Protected Member:

-bool hasUpgraded[3] = {false, false, false};

To track upgrades for moving, shooting, seeing, initialized to all false.

-Battlefield *battlefield = nullptr;

As a pointer to current battlefield

-bool pendingUpgrade = false;

As a determinant of whether a generic robot is upgrade or not. Initially, we set it as false because it haven;t upgrade yet.

-string upgradeType = "";

As a string which store the upgrade type of generic robot

-bool enemyDetectedNearby = false;

As a flag for detecting nearby enemies

-vector<Robot *> detectedTargets;

As a flag to determine which target already detected to prevent repeated detection

-vector<string> upgrades;

As a vector to store the upgrade robot

-vector<pair<int, int>> availableSpaces;

As a vector to store all available empty space which the generic robot can move

-bool hasLooked = false;

bool hasMoved = false;

bool hasThought = false;

bool hasFired = false;

As an action flag to limit the generic robot can only think, move, look, fire one time per turn. Initially, we set it as false because it hasn't performed those actions yet.

Public Member:

-GenericRobot(const string &name, int x, int y)

Constructor that initializes a robot with a name and position, inheriting multiple robot capabilities.

~GenericRobot()

The default destructor for the robot.

-void setBattlefield(Battlefield *bf)

Assigns the battlefield context for the robot.

-void think()

Decides the robot's action sequence (move/fire) based on nearby enemies.

-void act()

Executes the robot's actions (look, think) if alive and uninjured.

-void move()

Relocates the robot to a random adjacent empty cell if available.

-void fire()

Shoots at a random detectable enemy, with a chance to upgrade upon hit.

-void look(int X, int Y)

Scans a 3x3 grid for enemies and empty spaces, updating internal state.

-bool canUpgrade(int area)

Checks if the robot can upgrade in a specified area.

-void setUpgraded(int area)

Marks an area as upgraded.

-bool canBeHit()

Always returns true, indicating the robot can be targeted.

-void setPendingUpgrade(const string &type)

Queues a pending upgrade with a specified type.

-bool PendingUpgrade()

Returns whether an upgrade is pending.

-string getUpgradeType()

Returns the type of pending upgrade.

-void clearPendingUpgrade()

Clears any pending upgrade.

-bool getEnemyDetectedNearby()

Checks if enemies were detected nearby during the last scan.

-void resetActionFlags()

Resets action flags (e.g., hasMoved) at the start/end of a round.

Constructor:

```
GenericRobot::GenericRobot(const string &name, int x, int y)
    : Robot(name, x, y),
      MovingRobot(name, x, y),
      ShootingRobot(name, x, y, 10),
      SeeingRobot(name, x, y, 1),
      ThinkingRobot(name, x, y, 1) {}
```

Purpose: Initializes a GenericRobot object by calling the constructors of its base classes: Robot, MovingRobot, ShootingRobot, SeeingRobot, and ThinkingRobot, with specified values for name and coordinates, and default values for ammo, vision range and thinking strategy level.

Destructor:

virtual ~GenericRobot = default;

Purpose: Cleans up resources when a GenericRobot object is destroyed. It is declared virtual to ensure proper destruction of derived objects.

Move Method

void GenericRobot::move() override

Overrides the pure virtual move() method of the base Robot and MovingRobot classes. Allows GenericRobot to move to a random adjacent available cell.

Look Method

void GenericRobot::look(int X, int Y) override

Overrides the pure virtual look() method of the base Robot and SeeingRobot classes. Allows a 3x3 area centered on (positionX + X, positionY + Y) to be revealed to GenericRobot. X and Y are always set to 0, thus this method effectively scans a 3x3 area centered on the robot's current position.

Fire Method

void GenericRobot::fire(int x, int y) override

Overrides the pure virtual fire() method of the base Robot and ShootingRobot classes. Allows GenericRobot to fire at a random detected enemy with a 70% hit chance. If no enemies were detected, fire at (positionX + x, positionY + y).

Think Method

void GenericRobot::think() override

Overrides the pure virtual think() method of the base Robot class.

Act Method

void GenericRobot::act()

Lets GenericRobot act during its turn if it is not dead or hurt. Performs a look() and think() action.

Mechanics:

During GenericRobot's turn, the act() method is first called. Inside act(), if GenericRobot is not permanently dead or was not hit by another robot, the look(0,0) method is called.

Inside look(), hasLooked flag is set to true, enemyDetectedNearby is set to false and the detectedTargets and availableSpaces vectors are initially empty. GenericRobot performs a scan of the 3x3 area centering on its current position. If an enemy was revealed, enemyDetectedNearby is set to true, and all detected enemies are added to detectedTargets vector. For each cell, if no enemy is detected, add to availableSpaces vector.

After look() is done executing, think() is called. hasThought flag is set to true. It checks if enemyDetectedNearby is true, if yes, fire() and move() are called in succession, otherwise, call move() then fire().

If enemyDetectedNearby is true, fire() is called. hasFired flag is set to true. If GenericRobot still has ammo left, loop through the detectedTargets vector and filter it to only include hittable targets (canBeHit() is true), then add the hittable targets to a new vector validTargets. Then, select a random target to fire at. Call hitProbability() from the ShootingRobot abstract base class, if it returns true then the target is hit. If the target is hit, the GenericRobot can randomly choose an upgrade. After fire() is done

executing, move() is called. hasMoved flag is set to true. Loop through availableSpaces and pick a random cell. If there are no available spaces, GenericRobot does not move.
Otherwise, if enemyDetectedNearby is false, move() is first called then only fire(). In this case, move() is still the same, but for fire(), it will fire at a random spot.

2.9 HideBot Class Explanation

HideBot serves as a fourth level inherited class, that inherits from GenericRobot, featuring stealth capabilities that allow it to periodically hide from attacks and other robots. It can hide up to 3 times during its operation, making it temporarily invulnerable when hidden.

Attributes

Private Members:

-hide_count (int)

Tracks how many times the robot has hidden (max 3) - prevents infinite hiding

-sHidden (bool)

Indicates whether the robot is currently in hidden state - determines if it can be hit

Public Members:

-Constructor HideBot(name, x, y)

Initializes the HideBot with name and position coordinates

-move() override

Controls movement and hiding behavior with a 50% chance to hide if under limit

-getHiddenStatus()

Returns current hidden status - checks if robot is currently hidden

-appear()

Forces the robot to become visible - manually exits hidden state

-act() override

Main decision function combining looking, firing and moving actions

-fire(X, Y) override

Handles shooting logic with special consideration for hidden targets

-canBeHit() override

Determines if robot can be hit based on hidden status - returns false when hidden

2.10 JumpBot Class Explanation

JumpBot is a fourth inheritance level robot inherited from GenericRobot that incorporates teleportation capabilities, allowing it to instantly reposition itself on the battlefield up to three times during combat. This mobility-focused bot combines strategic movement with offensive capabilities.

Attributes

Private Members

-jump_count

Tracks remaining jumps (max 3).

Public Members

-JumpBot(name, x, y)

Initializes bot with name & position.

-move()

Jumps (50% chance) or moves normally if jumps remain.

-act()

Standard turn sequence: look → shoot → move.

-fire(X, Y)

Shoots at valid targets; upgrades on kill.

-getJumpCount()

Returns remaining jumps (0-3).

2.11 LongShotBot Class Explanation

LongShotBot is a fourth inheritance level robot inherited from GenericRobot that incorporates long-range shooting, allowing it to shoot a further distance than other types of robots.

Attributes:

Private:

int fire_count

Track number of successful hits (triggers upgrades), initially set to 0.

vector<string> upgradeTypes

Lists possible evolved forms after kill.

Public:

void fire(int X, int Y) override

Overrides the fire() method from GenericRobot to implement its long-range shooting pattern. It can shoot up to 3 unit distances from its current position.

Constructor:

LongShotBot(const string &name, int x, int y)

: Robot(name, x, y),

GenericRobot(name, x, y) {}

Initializes a LongShotBot object by calling the Robot and GenericRobot constructors, passing in the robot name and its x,y coordinates.

Possible Upgrades:

On successful hit, can upgrade to one of the following:

- HideLongShotBot
- JumpLongShotBot
- LongShotScoutBot
- LongShotTrackBot

2.12 SemiAutoBot Class Explanation

SemiAutoBot is a forth level inheritance robot derived from GenericRobot that specializes in battlefield intelligence shooting. It can fire 3 consecutive shots at a target and can upgrade upon a successful hit.

Attributes

Private Members:

-fire_count

Tracks the number of successful hits.

-upgradeTypes

List of possible upgrades (HideSemiAutoBot, JumpSemiAutoBot, SemiAutoScoutBot, SemiAutoTrackBot).

Public Members:

Constructor

-SemiAutoBot(const string &name, int x, int y)

Initializing the robot with a name and position.

-void fire(int X, int Y) override

Fires 3 shots at the first valid target; if any shot hits, schedules a random upgrade.

-int getFireCount() const

Returns the total successful hits (fire_count).

Upgrade Variants

On successful target hit, transforms into:

HideSemiAutoBot

JumpSemiAutoBot

SemiAutoScoutBot

SemiAutoTrackBot

2.13 ThirtyShotBot Class Explanation

ThirtyShotBot is a fourth level inheritance level robot that inherits from GenericRobot, it can start with 30 shells (ammo) instead of 10. It is an upgrade bot within the shooting category. It can upgrade after a successful kill.

Attributes:

Private:

int shell_count

To store the number of shells remaining.

vector<string> upgradeTypes

Lists possible evolved forms after kill.

Public:

void fire(int X, int Y) override

Overrides the fire() method from GenericRobot to implement its 30 shell firing capabilities.

Constructor:

ThirtyShotBot(const string &name, int x, int y)

: Robot(name, x, y),

GenericRobot(name, x, y) {}

Initializes a ThirtyShotBot object by calling the Robot and GenericRobot constructors, passing in the robot name and its x,y coordinates.

Possible Upgrades:

On successful hit, can upgrade to one of the following:

- HideThirtyShotBot
- JumpThirtyShotBot
- ThirtyShotScoutBot
- ThirtyShotTrackBot

2.14 KnightBot Class Explanation

KnightBot is a fourth level inheritance level robot that inherits from GenericRobot, with additional featuring diagonal-range attacks (5 units) with a chance to upgrade after successful kills. It excels in mid-range engagements with its unique firing pattern.

Private Members

-fire_count

Tracks successful hits (triggers upgrades).

-upgradeTypes

Lists possible evolved forms after kills.

Public Members

-KnightBot(name, x, y)

Initializes with name and position.

-fire(X, Y)

Attacks in a random diagonal line (5 tiles), upgrading on hit.

Upgrade Variants

On successful target hit, transforms into:

HideKnightBot,

JumpKnightBot,

KnightScoutBot,

KnightTrackBot

2.15 QueenBot Class Explanation

QueenBot is a fourth level inheritance level robot that inherits from GenericRobot. Like a Queen in chess, it can fire horizontally, vertically and diagonally across the entire battlefield grid, stopping at the first target it encounters. It is an upgrade bot within the shooting category. It can upgrade after a successful kill.

Attributes:

Private:

vector<pair<int, int>> directions

Lists the 8 possible directions the QueenBot can fire at.

Public:

void fire(int X, int Y) override

Overrides the fire() method from GenericRobot to implement its long-range horizontal, vertical and diagonal firing capabilities.

Constructor:

QueenBot(const string &name, int x, int y)

: Robot(name, x, y),

GenericRobot(name, x, y) {}

Initializes a QueenBot object by calling the Robot and GenericRobot constructors, passing in the robot name and its x,y coordinates.

Possible Upgrades:

On successful hit, can upgrade to one of the following:

- HideQueenBot
- JumpQueenBot
- QueenScoutBot
- QueenTrackBot

2.16 VampireBot Class Explanation

VampireBot is a fourth level inheritance level robot that inherits from GenericRobot. Like a vampire, it can absorb its enemy's blood: It regains one life when it hits an enemy (maximum 3 lives gained this way, cannot gain life when it has 3 lives). It is an upgrade bot within the shooting category. It can upgrade after a successful kill.

Attributes:

Private:

int gainLivesCount

Stores the number of times VampireBot gained life from successful kills. Initially set to 0.

Public:

void fire(int X, int Y) override

Overrides the fire() method from GenericRobot to implement its life-gaining capabilities after successful kills (maximum 3 times per simulation).

Constructor:

VampireBot(const string &name, int x, int y)

: Robot(name, x, y),

GenericRobot(name, x, y) {}

Initializes a VampireBot object by calling the Robot and GenericRobot constructors, passing in the robot name and its x,y coordinates.

Possible Upgrades:

On successful hit, can upgrade to one of the following:

- HideVampireBot
- JumpVampireBot
- VampireScoutBot
- VampireTrackBot

2.17 ScoutBot Class Explanation

ScoutBot is a forth level inheritance robot derived from GenericRobot that specializes in battlefield intelligence gathering. It can scan the entire battlefield up to 3 times and has a chance to upgrade after successful reconnaissance missions.

Private Members

-int scout_count

Tracks the number of scans performed (0-3)

Public Members

Constructor

-ScoutBot(const string &name, int x, int y)

Initializes the ScoutBot with specified name and starting coordinates

Inherits from both Robot and GenericRobot base classes

-void look(int X, int Y)

Scans entire battlefield with 50% chance if scans remain (<3)

Logs positions of all detected robots

Increments scan counter on successful scan

-void fire(int X, int Y) override

Performs reconnaissance-style attack with 50% chance if scans remain

Logs positions of all visible targets

Upgrades to specialized variant after successful target acquisition

Limits to 3 total scans/attacks

Behavioral Control

-void act() override

Executes standard action sequence: scanning, shooting, moving

Currently implements basic logic (to be expanded)

Accessor Method

-int getScoutCount() const

Returns current scan count (0-3)

Provides read-only access to private scout_count

Upgrade Variants

On successful target hit, upgrades into:

HideScoutBot

JumpScoutBot

2.18 TrackBot Class Explanation

TrackBot is a forth level inheritance robot derived from GenericRobot that specializes in battlefield intelligence tracking. It can locates and fires at nearby targets, with limited tracking attempts (tracker) and upgrade potential upon successful hits.

Attributes

Private Members:

-tracker (int)

Remaining tracking attempts (starts at 3).

-track_target (vector<Robot*>)

Stores tracked robots.

Public Members

Constructor

-TrackBot(const string &name, int x, int y)

Initializing the robot with a name and position.

-void look(int X, int Y)

Scans adjacent cells for robots to track (reduces tracker on success).

-void act() override

Default action sequence: look(), fire(), then move().

-void showTrackTarget()

Logs currently tracked robots.

-int getTracker() const

Returns remaining tracking attempts.

-void fire(int X, int Y) override

Fires at tracked targets with a 50% chance; on hit, schedules an upgrade (HideTrackBot or JumpTrackBot).

Upgrade Variants

On successful target hit, transforms into:

1. HideTrackBot
2. JumpTrackBot

2.19 Second Upgrade Robot Class Inheritance Concept Explanation

For the second upgrade robot class , it is inherited from 2 among 10 robot classes (HideBot, JumpBot, SemiAutoBot, LongShotBot, ThirtyShotBot, VampireBot, KnightBot, QueenBot, ScoutBot, TrackBot) which fall at different function class. For example, seeing robot will combine with shooting robot or moving robot. Take one example to explain HideLongShotBot.

HideLongShotBot Explanation

HideLongShotBot is a fifth level inheritance robot derived from Hidebot and LongShotBot. It combines stealth mechanics with long-range attacks, prioritizing hidden sniping and specialized upgrades.

Key Features

1. **Dual Inheritance:**

HideBot → Provides stealth (isHidden), evasion (canBeHit), and ambush movement.

LongShotBot → Adds extended-range targeting (detectedTargets) and sniper-like firing.

2. **Constructor:**

   Initializes all parent classes (Robot, GenericRobot, HideBot, LongShotBot) to avoid diamond problem ambiguity.

3. O**verridden Methods:**

   move(): Uses HideBot's movement (stealth-focused).

   fire(): Executes LongShotBot's fire() for longer-range shots.

Upgrade Variants

On successful target hit, transforms into:

HideLongShotScoutBot

HideLongShotTrackBot

In conclusion, for the second upgrade Robot, we will have total 28 combinations of robot classes. As shown below:

Moving + Shooting

1. HideBot_LongShotBot
2. HideBot_SemiAutoBot
3. HideBot_ThirtyShotBot
4. HideBot_KnightBot
5. HideBot_QueenBot
6. HideBot_VampireBot

7. JumpBot_LongShotBot
8. JumpBot_SemiAutoBot
9. JumpBot_ThirtyShotBot
10. JumpBot_KnightBot
11. JumpBot_QueenBot
12. JumpBot_VampireBot

Moving + Seeing

13. HideBot_ScoutBot
14. HideBot_TrackBot
15. JumpBot_ScoutBot
16. JumpBot_TrackBot

Shooting + Seeing

17. LongShotBot_ScoutBot
18. LongShotBot_TrackBot
19. SemiAutoBot_ScoutBot
20. SemiAutoBot_TrackBot
21. ThirtyShotBot_ScoutBot
22. ThirtyShotBot_TrackBot
23. KnightBot_ScoutBot
24. KnightBot_TrackBot
25. QueenBot_ScoutBot
26. QueenBot_TrackBot
27. VampireBot_ScoutBot
28. VampireBot_TrackBot

2.20 Third Upgrade Robot Class Inheritance Concept Explanation

Triple-upgrade classes inherit from a double-upgrade class (involving moving and shooting upgrades) and another single-upgrade class (from seeing upgrade). Take HideLongShotScoutBot for example.

HideLongShotScoutBot Explanation

HideLongShotScoutBot is a sixth level inheritance robot derived from HideLongShotBot and ScoutBot. It combines stealth mechanics, long-range attacks and long-range vision.

Key Features

1. **Dual Inheritance:**
   HideLongShotBot → Provides stealth, evasion and long-range firing.
   ScoutBot → Adds long-range vision and scanning.
2. **Constructor:**
   Initializes all parent classes (Robot, GenericRobot, HideLongShotBot, ScoutBot) to avoid diamond problem ambiguity.
3. **Overridden Methods:**
   move(): Uses HideBot's move() for stealth.
   fire(): Calls LongShotBot's fire() for longer range.
   act(): Executes ScoutBot's look() for long-range vision and scanning.

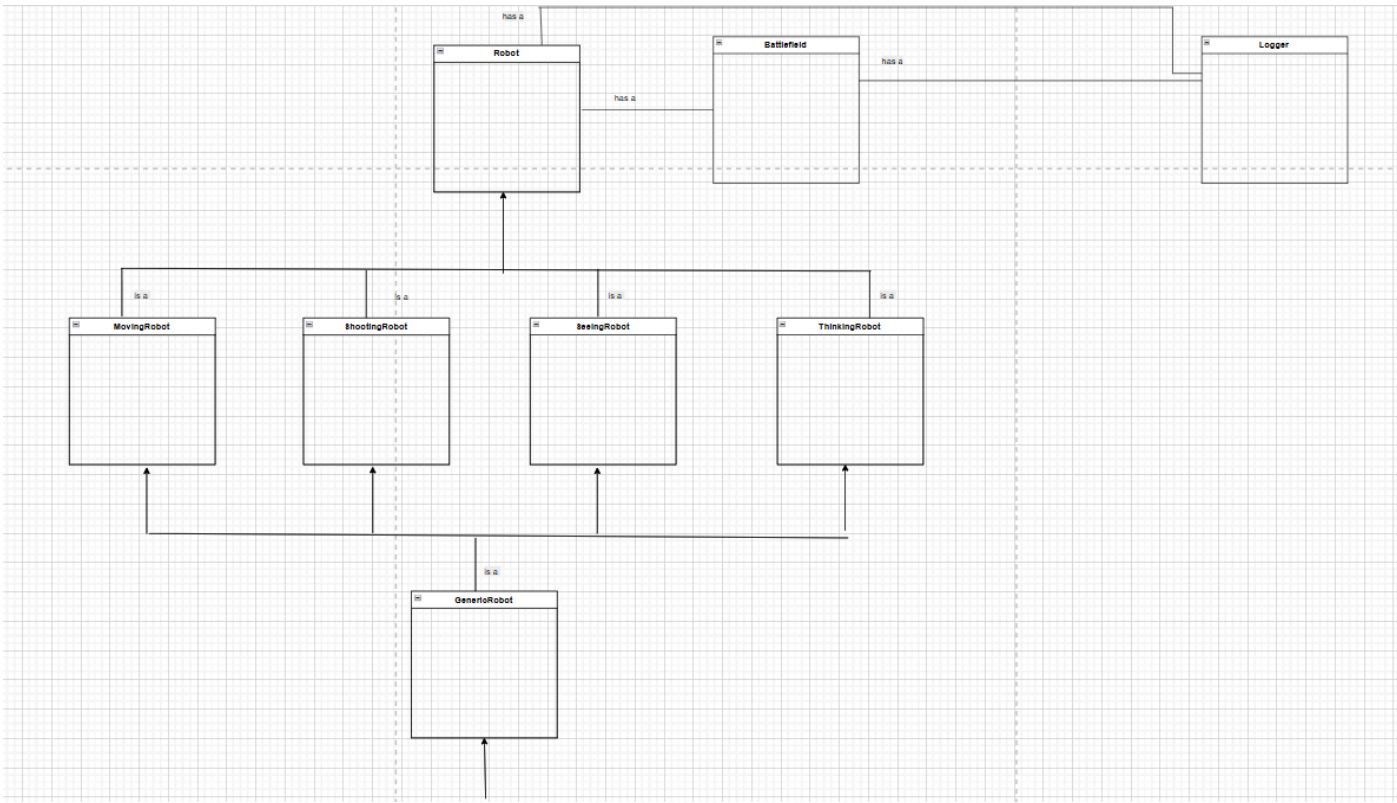
This Robot cannot upgrade anymore after a successful hit.

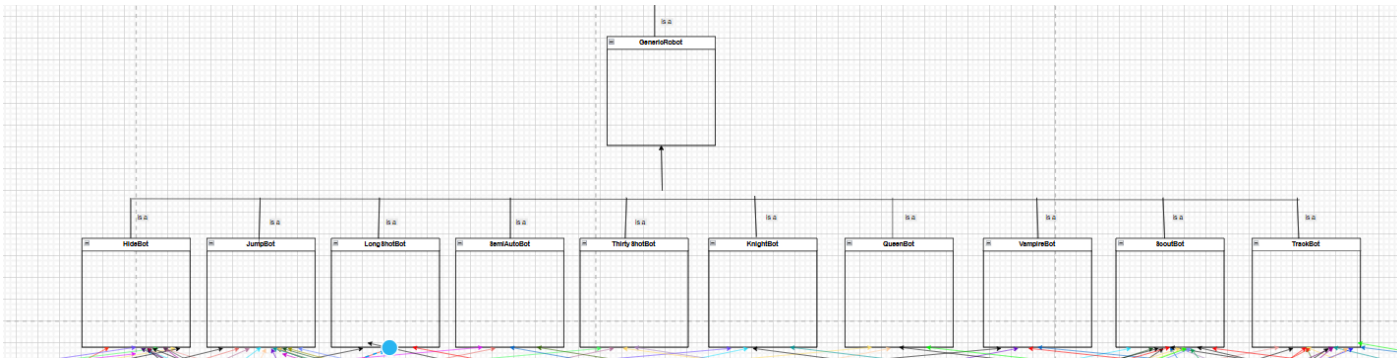
Based on this pattern, there will be a total of 24 third-upgrade Robot combinations, as listed below:

1. HideBot_ LongShotBot_ScoutBot
2. HideBot_ SemiAutoBot_ ScoutBot
3. HideBot + ThirtyShotBot + ScoutBot
4. HideBot + KnightBot + ScoutBot
5. HideBot + QueenBot + ScoutBot
6. HideBot + VampireBot + ScoutBot
7. HideBot + LongShotBot + TrackBot
8. HideBot + SemiAutoBot + TrackBot
9. HideBot + ThirtyShotBot + TrackBot
10. HideBot + KnightBot + TrackBot
11. HideBot + QueenBot + TrackBot
12. HideBot + VampireBot + TrackBot
13. JumpBot + LongShotBot + ScoutBot
14. JumpBot + SemiAutoBot + ScoutBot
15. JumpBot + ThirtyShotBot + ScoutBot
16. JumpBot + KnightBot + ScoutBot
17. JumpBot + QueenBot + ScoutBot
18. JumpBot + VampireBot + ScoutBot
19. JumpBot + LongShotBot + TrackBot
20. JumpBot + SemiAutoBot + TrackBot
21. JumpBot + ThirtyShotBot + TrackBot
22. JumpBot + KnightBot + TrackBot
23. JumpBot + QueenBot + TrackBot
24. JumpBot + VampireBot + TrackBot

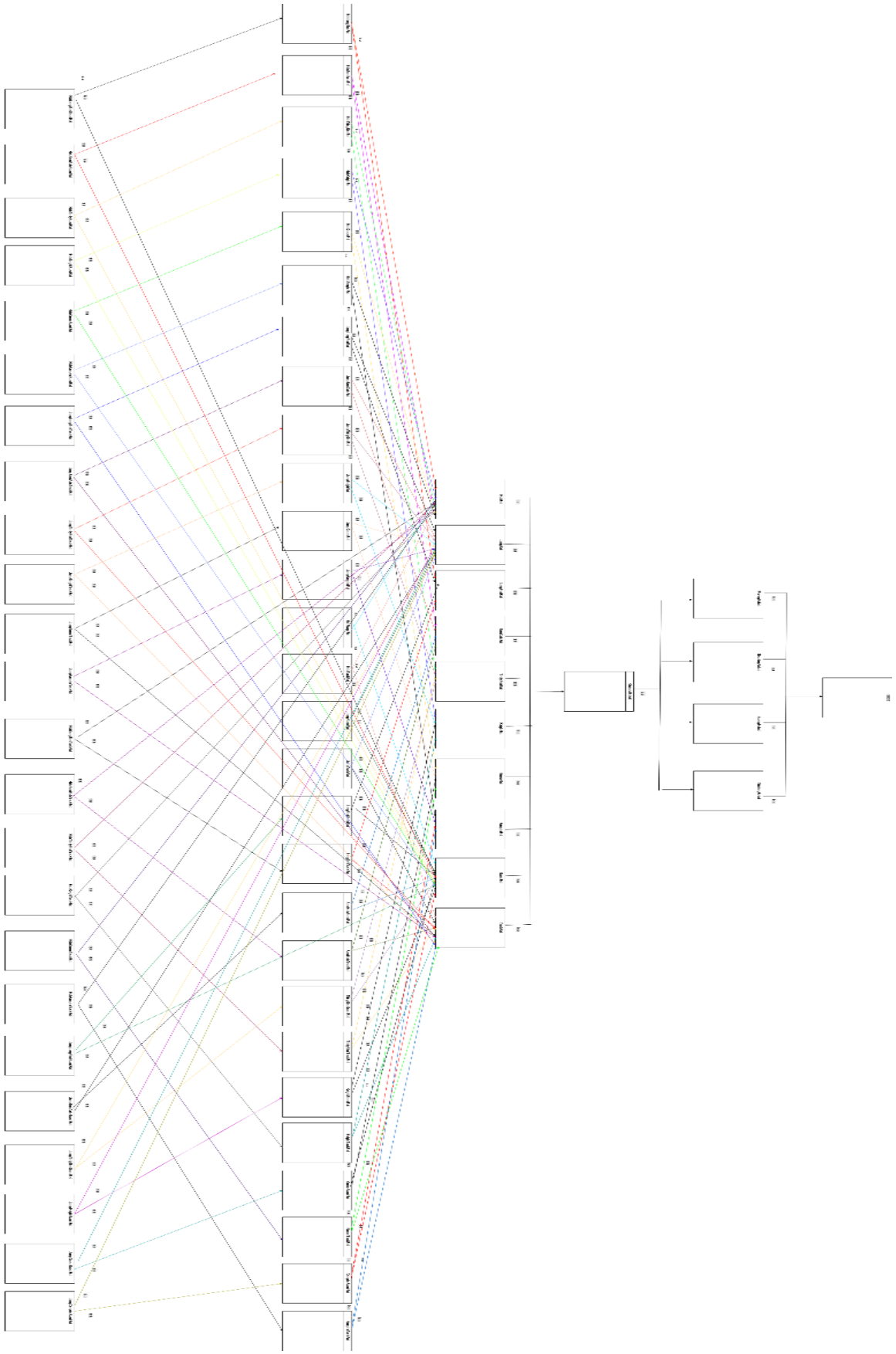
3.0 Class Diagrams

3.01 Overall Class Diagram

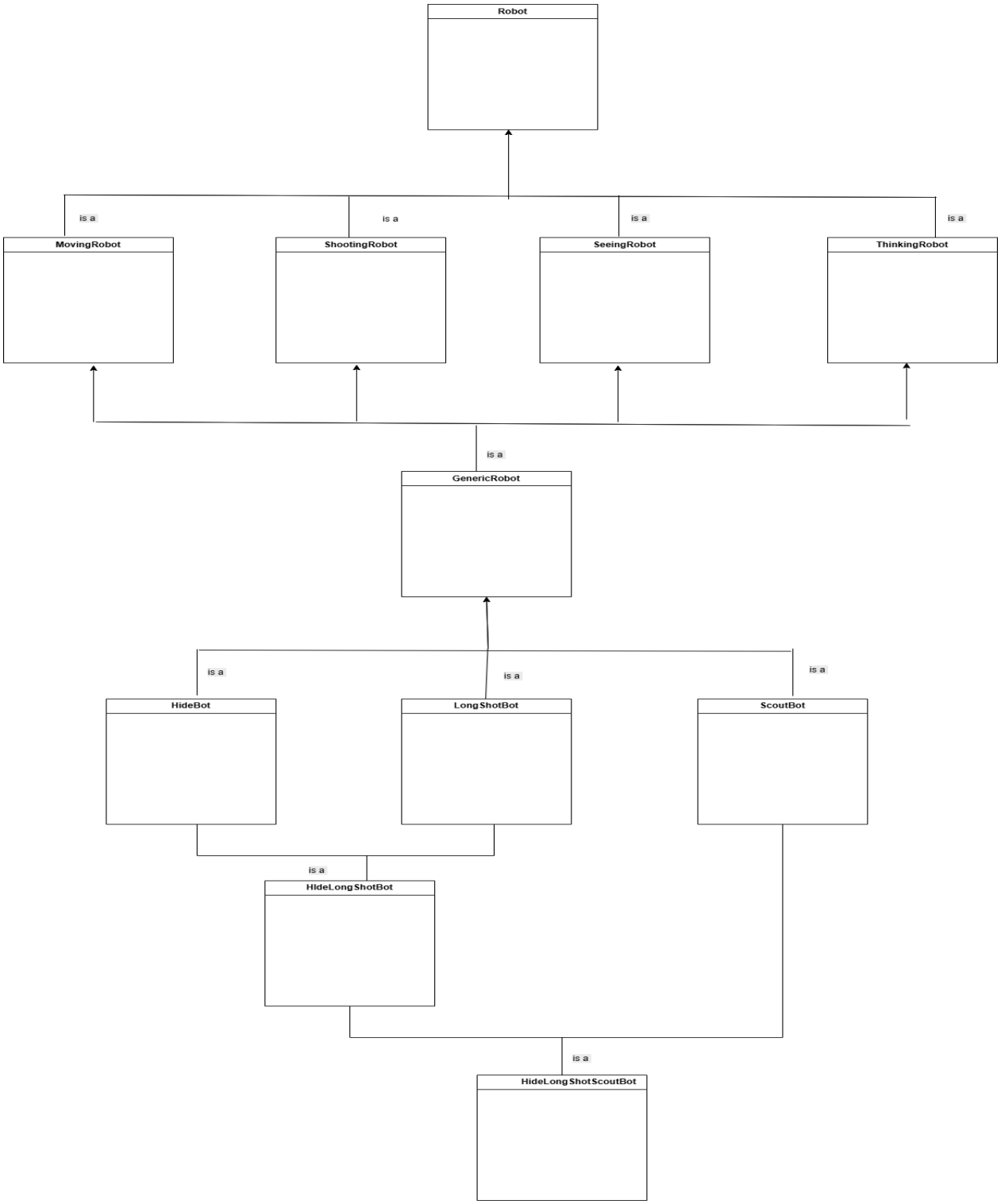




3.02 Class Diagram(Inheritance of Robot Class)



3.03 Class Diagram(Inheritance of Robot Class Breakdown)



3.04 Class Diagram Details (Part 1)

Class Details

Logger

-ofstream outputFile
-string fileName

+Logger
+bool open()
+void close()
+Logger &operator << (const T &outputContent)
+Logger &operator <<(ostream &(*mainip)(ostream &))
~Logger

Robot

- int positionX;
- int positionY;

string name;
int lives;
bool hidden;
bool isDie = false;
bool isHurt = false;
virtual bool isHit() = 0;
bool getEnemyDetectedNearby() const;

+Robot(string name, int x, int y) : name(name), positionX(x), positionY(y), lives(3), hidden(false) {}
+virtual ~Robot() = default;
+virtual void think() = 0;
+virtual void act() = 0;
+virtual void move() = 0;
+virtual void fire() = 0;
+virtual void look(int X, int Y) = 0;
+string getName() const
+ int getX() const
+ int getY() const
+ int getLives() const
+ void setLives(int numOfLives)
+bool isHidden() const
+ bool getIsDie() const
+void setIsDie(bool val)
+ bool getIsHurt() const
+void setIsHurt(bool val)
+void setPosition(int x, int y)
+void takeDamage()
+void setHidden(bool state)

Battlefield

- int height
-int width
-int steps
-int numberOfRbots
-vector <Robot *>listOfRobots
-vector <vector<Root*>>battlefiledGrid
-vector <pair<string, int>> respawnQueue
-map <string,int> respawnCounts
-queue<pair<string,int>> reentryQueue

+Battlefield() : height(0), width(0), steps(0), numberOfRobots(0) {};
~Battlefield();
+void setDimensions(int h, int w)
+void printDimensions() const
+void setSteps(int s)
+void printSteps() const
+ int getSteps() const
+void setNumberOfRobots(int n)
+void printNumberOfRobots() const
+int getNumberOfRobots() const
+const vector<Robot *> getListOfRobots() const
+int getWidth() const { return width; }
+int getHeight() const { return height; }
+void simulationStep();
+ void addNewRobot(Robot *robot);
+int getNumberOfAliveRobots();
+void cleanupDestroyedRobots();
+void respawnRobots();
+void queueForReentry(Robot *robot);
+Robot *getRobotAt(int x, int y) const;
+void placeRobot(Robot *robot, int x, int y);
+void removeRobotFromGrid(Robot *robot);
+ bool isPositionAvailable(int x, int y);
+bool isPositionWithinGrid(int x, int y) const;
+ void displayBattlefield();

MovingRobot

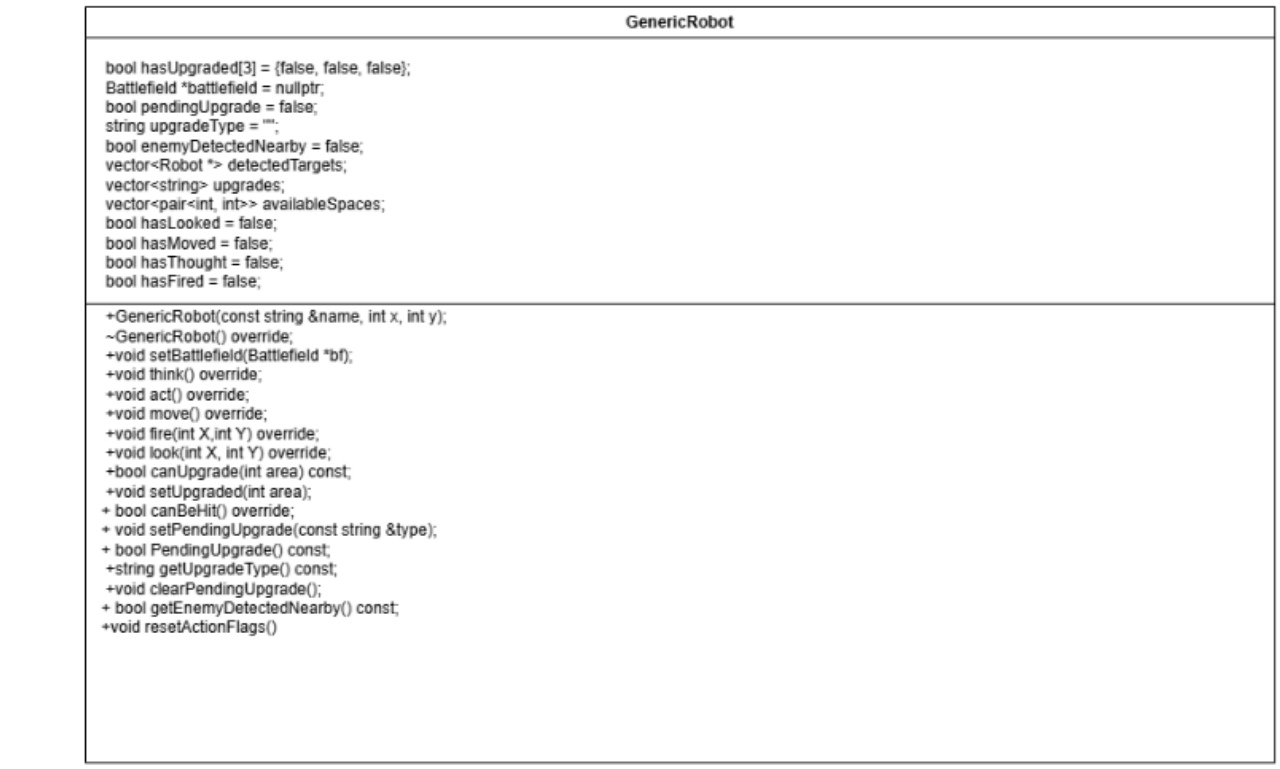
int moveCount;

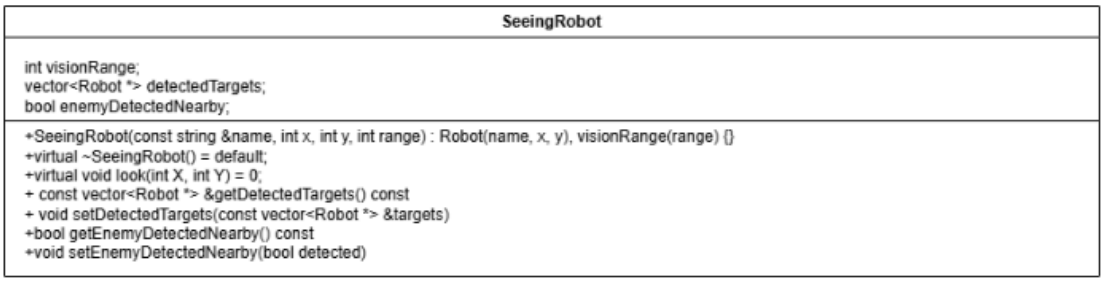
+MovingRobot(const string &name, int x, int y) : Robot(name, x, y), moveCount(0) {}
+ virtual ~MovingRobot() = default;
+virtual void move() = 0
+bool isValidMove(int newX, int newY, const Battlefield &battlefield) const
+void incrementMoveCount()

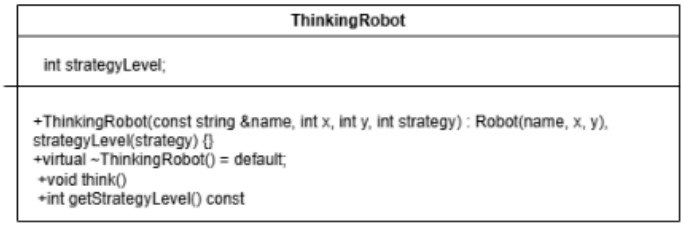
ShootingRobot

int ammo;

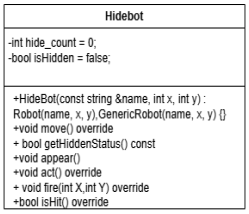
+ShootingRobot(const string &name, int x, int y, int initialAmmo) : Robot(name, x, y), ammo(initialAmmo) {}
+virtual ~ShootingRobot() = default;
+virtual void fire(int X,int Y) = 0;
+bool hasAmmo() const
+ void useAmmo()
+ int getAmmo() const
+bool hitProbability() const

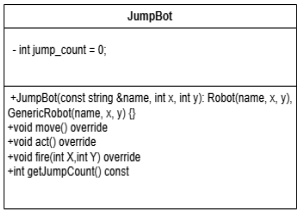

GenericRobot
bool hasUpgraded[3] = {false, false, false};
Battlefield *battlefield = nullptr;
bool pendingUpgrade = false;
string upgradeType = "";
bool enemyDetectedNearby = false;
vector<Robot *> detectedTargets;
vector<string> upgrades;
vector<pair<int, int>> availableSpaces;
bool hasLooked = false;
bool hasMoved = false;
bool hasThought = false;
bool hasFired = false;
+GenericRobot(const string &name, int x, int y);
~GenericRobot() override;
+void setBattlefield(Battlefield *bf);
+void think() override;
+void act() override;
+void move() override;
+void fire(int X,int Y) override;
+void look(int X, int Y) override;
+bool canUpgrade(int area) const;
+void setUpgraded(int area);
+ bool canBeHit() override;
+ void setPendingUpgrade(const string &type);
+ bool PendingUpgrade() const;
+string getUpgradeType() const;
+void clearPendingUpgrade();
+ bool getEnemyDetectedNearby() const;
+void resetActionFlags()

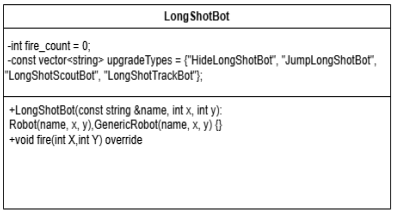

SeeingRobot
int visionRange;
vector<Robot *> detectedTargets;
bool enemyDetectedNearby;
+SeeingRobot(const string &name, int x, int y, int range) : Robot(name, x, y), visionRange(range) {}
+virtual ~SeeingRobot() = default;
+virtual void look(int X, int Y) = 0;
+ const vector<Robot *> &getDetectedTargets() const
+ void setDetectedTargets(const vector<Robot *> &targets)
+bool getEnemyDetectedNearby() const
+void setEnemyDetectedNearby(bool detected)

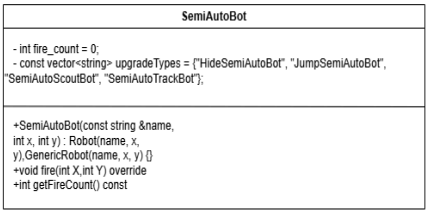

ThinkingRobot
int strategyLevel;
+ThinkingRobot(const string &name, int x, int y, int strategy) : Robot(name, x, y),
strategyLevel(strategy) {}
+virtual ~ThinkingRobot() = default;
+void think()
+int getStrategyLevel() const

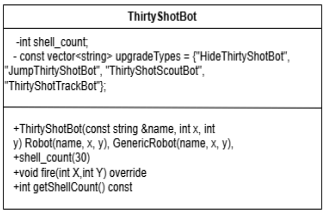
Class Diagram Details (Part 2)

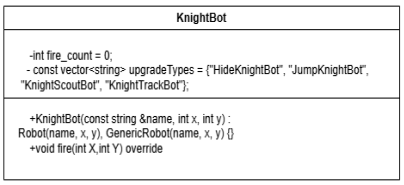


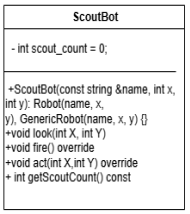


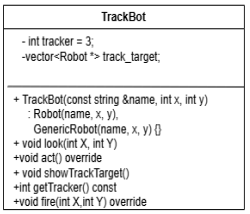


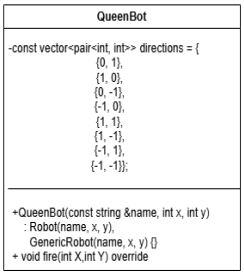


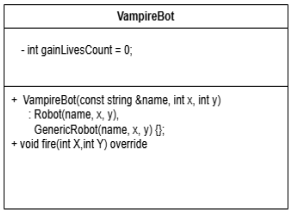




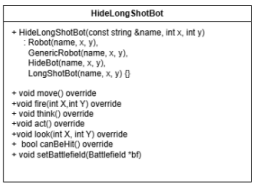


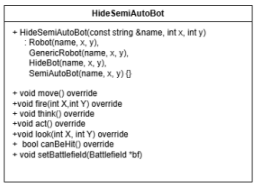


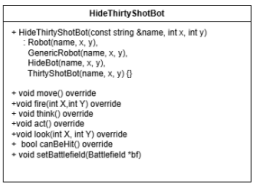


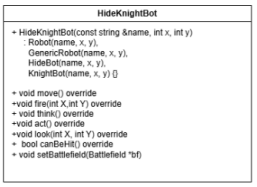


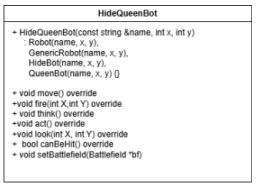
Class Diagram Details (Part 3)

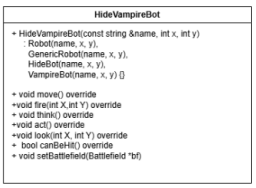


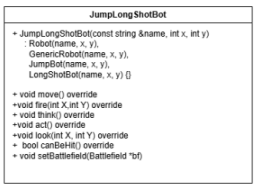


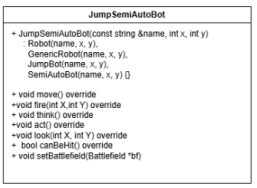


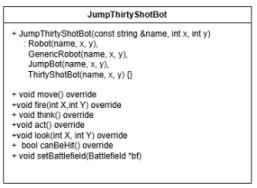


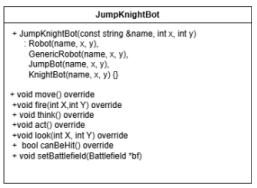


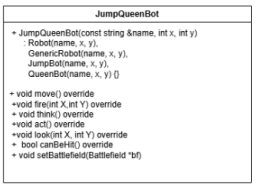


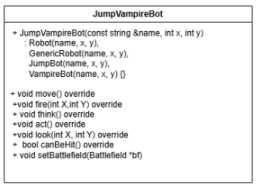


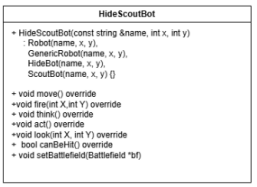


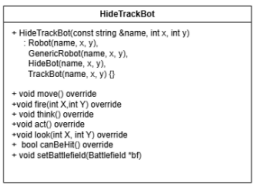


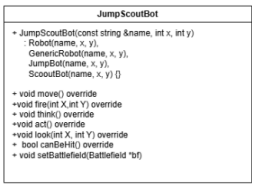


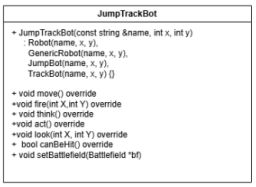


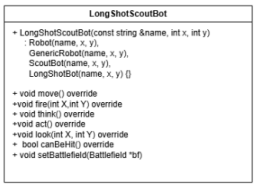


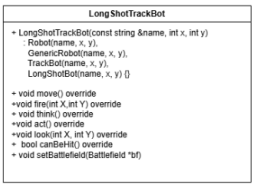


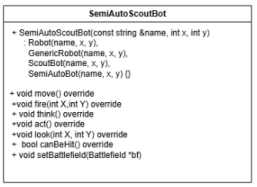


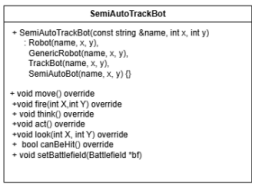


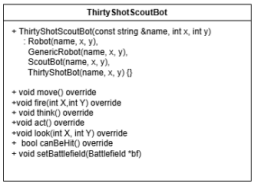


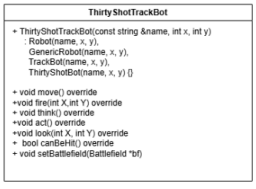


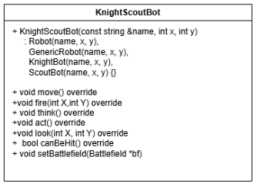


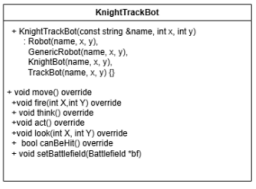


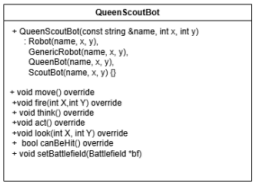


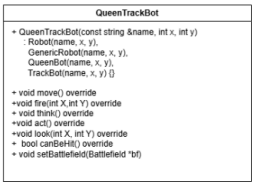


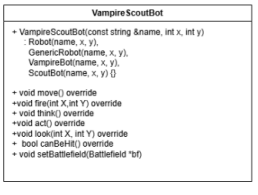


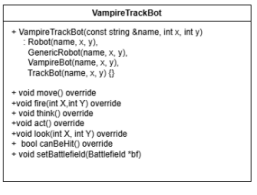




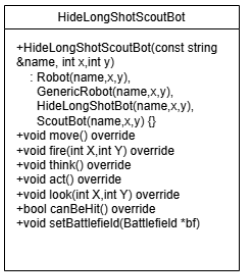


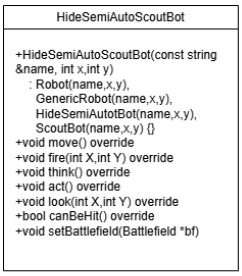


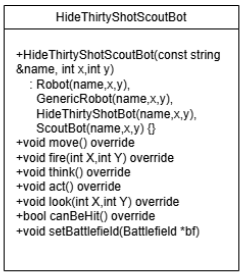


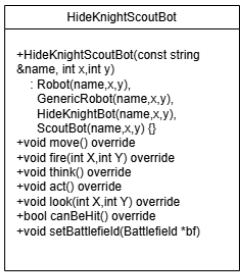


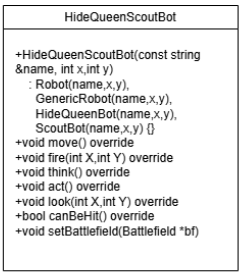
Class Diagram Details (Part 4)

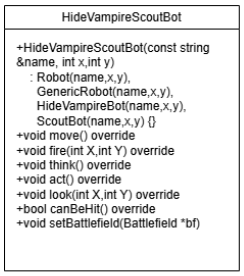


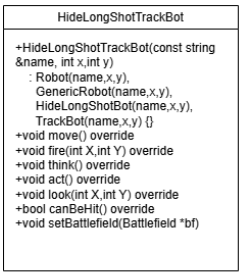


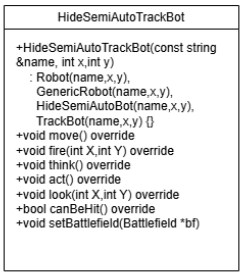


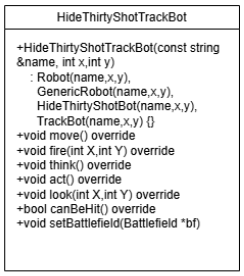


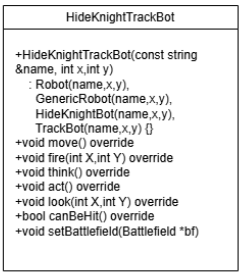


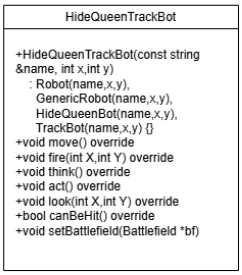


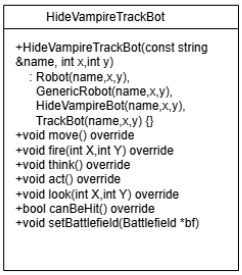


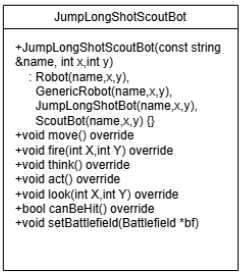


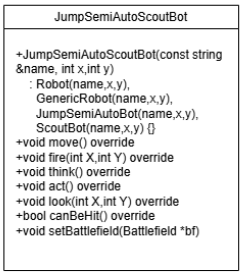


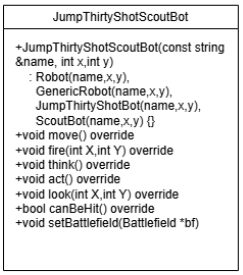


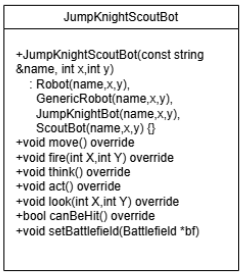


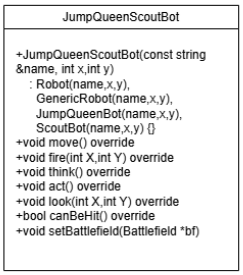


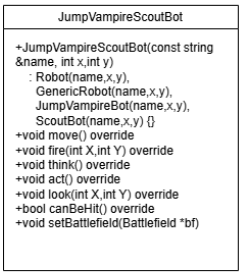


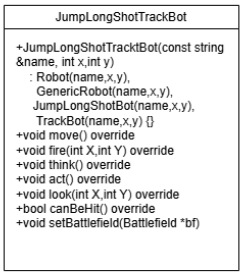


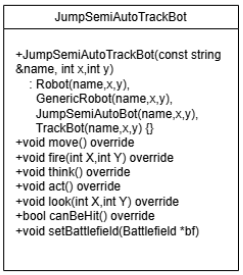


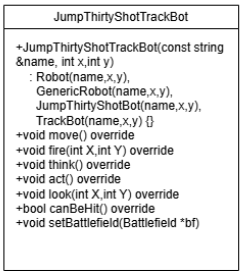


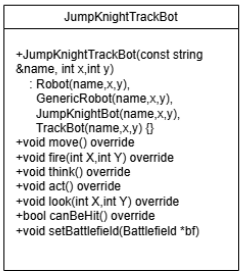


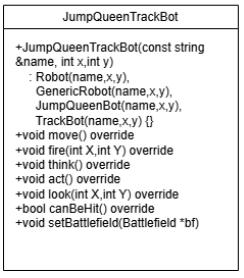


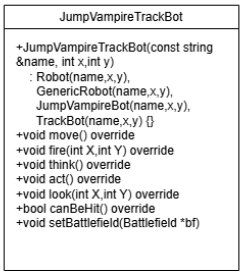












4.0 Pseudocode

Including algorithms used to optimize the actions of robots listed in the assignment document

****

READ INPUT FILE

****

```
START
    OPEN input file
    INITIALIZE battlefield
    FOR each line in file
        IF line start with “M”
            SET battlefield height
            SET battlefield width
        IF line starts with “steps:”
            SET battlefield steps
        IF line starts with “robots:”
            SET battlefield number of robots
        ELSE IF line starts with robot type:
            READ Type, Name, X, Y
                IF X and Y == “random “
                    FIND random empty location
                ENDIF
                CREATE robot type with name,x,y
                ADD robot to robot list
                CALL battlefield place robot
        ENDIF
    ENDFOR
END
```

```
**********************************************************
BATTLE FIELD
**********************************************************
START
        INITIALIZE battlefield grid with HEIGHT and WIDTH
        METHOD Place Robot (Robot , X , Y)
                IF location is available
                        SET battlefield (X,Y) = robot
                        SET robot location X
                        SET robot location Y
                        METHOD Remove Robot (Robot):
                        SET battlefield (X, Y) = “NULL”
                        METHOD Display Battlefield
                        PRINT top border
                        FOR each row and column in battlefield
                                PRINT row number
                                PRINT column number
                                        FOR each cell in row
                                                IF cell == NULL
                                                        PRINT “.”
                                                IF cell is Destroy
                                                        PRINT “X”
                                                IF cell isHidden
                                                        PRINT “H”
                                                ELSE
                                                        PRINT first letter of robot name
                                                ENDIF
                                        ENDFOR
                        ENDFOR
```

```
END

**********************************************************
SIMULATOR
**********************************************************
START
        METHOD Process Turn
                FOR each robot in Robot list
                        IF robot has pending upgrade
                                CREATE new robot of upgrade type
                                COPY function from old robot
                                REPLACE old robot with new robot
                        IF robot still alive
                                CALL robot action
                                CALL clean Destroy robot
                                CALL respawn robot
                        END IF
        METHOD clean Destroy robot
        FOR each robot in list
                IF robot live <= 0
                        REMOVE robot from grid
                IF robot isHurt
                        ADD robot to reentryQueue
                DELETE robot
                END IF
        END FOR
        METHOD respawn robot
        IF reentryQueue  not empty
```

```
            GET robot from queue
            FIND random empty location
            CREATE new GenericRobot
            PLACE on battlefield
        END IF
        METHOD get number of robot alive
        COUNT robot lives >0
END


**********************************************************
GENERICROBOT
**********************************************************
START  $$
        METHOD think
        IF NOT hasThought
             LOG "thinking..."
             SET hasThought = true
             IF detact enermy nearby
                  PRIORITIZE firing then moving
             ELSE
                  PRIORITIZE moving then firing
            END  IF
            METHOD act
            CALL look()
            CALL think()
            CALL fire()
            CALL move()
```

```
END  IF
METHOD move()
IF NOT hasMoved
        SET hasMoved = true
        FIND available adjacent spaces
        IF spaces available
                PICK random space
                UPDATE position
                LOG new position
        END IF
END IF
METHOD fire(X, Y)
IF NOT hasFired and hasAmmo()
        SET hasFired = true
        SELECT valid target
        IF hit 0.7 chance
                CALL target.takeDamage()
                CONSIDER upgrade
                DECREMENT ammo
                        IF ammo == 0
                                SET selfDestruct = true
                        END IF
        END IF
END  IF
METHOD look(X, Y)
 SCAN 3x3 area around current position
RECORD empty spaces and nearby enemies
 SET enemyDetectedNearby flag
```

```
        METHOD takeDamage()
        IF NOT hidden
                DECREMENT lives
                 IF lives <= 0
                        SET isDie = true
                END IF
                 SET isHurt = true
       END IF
      ADD to ReentryQueue
ELSE
        MARK as Dead
ENDIF
END
```

****

HIDEBOT

****

```
START
        INHERIT from GenericRobot
        DECLARE hideCount = 0
        DECLARE isHidden = FALSE
        METHOD move()
           IF hideCount < 3 AND RANDOM(0 or 1) = 0
              SET isHidden = TRUE
              CALL setHidden(TRUE)
              INCREASE hideCount by 1
              PRINT name + hide count
           ELSE
```

```
            SET isHidden = FALSE
            CALL setHidden(FALSE)
            IF hideCount >= 3
               PRINT  finish use hide, keep moving
            ELSE
               PRINT did not hide this turn, keep moving
            ENDIF
         ENDIF
      METHOD canBeHit()
         RETURN NOT isHidden
         END
      METHOD appear()
         SET isHidden = FALSE
         END
      METHOD act()
         PRINT name + " is thinking..."
         CALL look(0, 0)
         CALL fire(0, 0)
         CALL move()
      END
      METHOD getHiddenStatus()
         RETURN isHidden
      END
END
```

****

JUMPBOT (upgrade from robot.move)

****

```
START
    INHERIT from GenericRobot
    DECLARE jumpCount = 0
    METHOD move()
        IF jumpCount < 3 and random
            SET attempts = 0
            SET maxAttempt = 10
            SET positionFound = FALSE
        WHILE positionFound = FALSE and attempts < maxAttempt
            INCREASE attempts by 1
            SET jumpX = random number from 0 to battlefield width
            SET jumpY = random number from 0 to battlefield height
            IF battlefield.getRobotAt(jumpx, jumpy) = NULL
                SET positionFound = TRUE
            ENDIF
        ENDWHILE
        IF positionFound = TRUE
            INCREASE jumpCount by 1
            CALL battlefield.removeRobotFromGrid(this)
            CALL battlefield.placeRobot(this, jumpx, jumpy)
            CALL setPosition(jumpx, jumpy)
            PRINT name +position + jump count
        ELSE
```

```
                PRINT name + could not find empty position to jump
            ENDIF
          ELSE
                IF jumpCount >= 3
                PRINT name +  cannot jump already
          ELSE
                PRINT name + did not jump this turn, keep moving
          ENDIF

    END
    METHOD act()
                PRINT name + " is thinking..."
                CALL look(0, 0)
                CALL fire(0, 0)
                CALL move()
    END
    METHOD getJumpCount()
                RETURN jumpCount
    END
END
```

```
**********************************************************
LONGSHOTBOT (upgrade from fobot.fire)
**********************************************************
START
        INHERIT from GenericRobot
        METHOD Fire(TargetX, TargetY)
                IF Fired
                        RETURN
                 ENDIF
            SET Fired = TRUE
            IF not HasAmmo THEN
                PRINT name + has no ammo left. It will self-destruct!
                SET Lives = 0
                SET IsDead = TRUE
                RETURN
            ENDIF
            FOR dx = -3 to 3
                FOR dy = -3 to 3
                                IF dx == 0 AND dy == 0
                                        CONTINUE
                                IF ABS(dx) + ABS(dy) > 3
                                        CONTINUE
                            SET tx = CurrentX + dx
                            SET ty = CurrentY + dy
                            SET Target = GetRobotAt(tx, ty)
                            IF Target exists and Target != SELF and Target is not
Hurt
                                        PRINT  fires at (tx, ty)
                                        USE Ammo
```

```
                    IF Target is Hidden
                        PRINT  name + is hidden, attack missed.
                    ELSE
                        GENERATE random chance from 1 to 100
                        IF chance <= hit probability
                            CALL Target.TakeDamage()
                            PRINT name is killed
                            INCREASE FireCount by 1
                            CHOOSE random upgrade type
                            SET PendingUpgrade =upgrade type
                            PRINT name + will upgrade into
    UpgradeType next turn
                        ELSE
                            RINT "Missed!"
                        ENDIF
                    ENDIF
                IF not HasAmmo
                    PRINT name + has no ammo left, it will self-destruct
                    SET Lives = 0
                    SET IsDead = TRUE
                 ENDIF
            ENDFOR
            IF no target fired
                PRINT no shooting as no robots within shooting range.
           ENDIF
END
```

```
**************************************************************
SEMIAUTOBOT (upgrade from robot.fire)
**************************************************************
START
    INHERIT from GenericRobot
    METHOD Fire(TargetX, TargetY)
            IF Fired
                RETURN
            ENDIF
            SET Fired = TRUE
            IF NOT HasAmmo THEN
                PRINT name + has no ammo left. It will self-destruct!
                SET Lives = 0
                SET IsDead = TRUE
                RETURN
             ENDIF
            FIND first valid target not self, alive, and not hurt
            IF No valid target THEN
                PRINT "No shooting as no robots within shooting
    range."
                RETURN
            ENDIF
            PRINT name + fires 3 consecutive shots at (TargetX,
    TargetY)
            USE Ammo
        FOR i = 1 to 3
            IF Target is Hidden THEN
                PRINT name + is hidden, attack missed.
            CONTINUE
```

```
                ENDIF
                GENERATE random chance from 1 to 100
                IF chance <= hit probability
                    PRINT shot target
                    CALL Target.TakeDamage()
                    INCREASE FireCount by 1
                    SET Hit = TRUE
                ELSE
                    PRINT "Shot i missed!"
                ENDIF
            IF NOT HasAmmo
                PRINT name + has no ammo left, it will self-destruct!
                SET Lives = 0
                SET IsDead = TRUE
                ENDIF
            ENDFOR
        IF Hit
            CHOOSE random upgrade type
            SET PendingUpgrade =upgrade type
            PRINT name +  will upgrade into UpgradeType next turn
        ENDIF
END
```

```
**************************************************************
THIRTYSHOTBOT (upgrade from robot.fire)
**************************************************************
START
        INHERIT from GenericRobot
        METHOD Fire(TargetX, TargetY)
           IF ShellCount <= 0
              PRINT name + shell is finish
              RETURN
           ENDIF
           IF hasFired
              RETURN
           ENDIF
           SET hasFired = TRUE
           SET Fired = FALSE
           FOR dx = -1 to 1
                FOR dy = -1 TO 1
                 IF dx == 0 and dy == 0
                        SET tx = CurrentX + dx
                        SET ty = CurrentY + dy
                        SET Target = GetRobotAt(tx, ty)
                        IF target exists and target != SELF and target is not hurt
                                IF target canBeHit
                                        GENERATE random chance from 1 to 100
                        IF Chance <= hit probability
                                CALL Target.TakeDamage()
                                DECREASE ShellCount by 1
                                PRINT name + fires at (tx, ty), shell left: ShellCount
```

```
                    PRINT name + is killed!
                    SET Fired = TRUE
                    CHOOSE random upgrade type
                    SET PendingUpgrade = upgrade type
                    PRINT name + will upgrade into UpgradeType next
    turn
                ELSE
                    DECREASE ShellCount by 1
                    PRINT "Missed! Shell left: ShellCount"
                ENDIF
            ENDFOR
END

************************************************************
KNIGHTBOT
************************************************************
START
    INHERIT from GenericRobot
    METHOD Fire(TargetX, TargetY)
    IF hasFired THEN
        RETURN
     ENDIF
     SET hasFired = true
     SET x, y = current position of robot
     SET fired = false
     SET hitSuccessful = false
     INITIALIZE empty list hitRobots
     DEFINE diagonals = [(1,1), (1,-1), (-1,1), (-1,-1)]
     SELECT a random diagonal (dx, dy) from diagonals
```

```
    FOR dist from 1 to 5
        targetX = x + dx * dist
        targetY = y + dy * dist
        IF targetX, targetY is outside the grid
            BREAK
        ENDIF
        SET target TO robot at (targetX, targetY)
        IF target exists AND target != self AND target is alive AND not hurt
THEN
            PRINT "KnightBot fires at (targetX,targetY)"
            useAmmo()
            SET fired = true
            IF target is a GenericRobot and target can be hit
                IF hitProbability() succeeds
                    target.takeDamage()
                    INCREMENT fireCount
                    SET hitSuccessful = true
                    ADD target.name to hitRobots
                    PRINT name + is killed
                ELSE
                    PRINT "KnightBot missed!"
                ENDIF
            ENDIF
        ENDIF
    ENDFOR
    IF not fired
       PRINT "No shooting as no robots in diagonal to fire at"
    ELSE IF hitSuccessful
```

```
            PRINT "KnightBot hit the following robots: hitRobots"
            SELECT a random upgrade type
            SET pending upgrade = upgradeType
            PRINT "KnightBot will upgrade into upgradeType next turn!"
        ENDIF
END

************************************************************
QUEENBOT
************************************************************
START
    INHERIT from GenericRobot
    METHOD Fire(TargetX, TargetY)
    IF hasFired
        RETURN
    ENDIF
    SET hasFired = true
    IF no ammo
        PRINT "QueenBot has no ammo left. It will self destruct!"
        SET lives = 0
        SET isDie = true
        RETURN
      ENDIF
      SET x, y = current position
      SET fired = false
      DEFINE directions = [(0,1),(1,0),(0,-1),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
```

```
    FOR direction (dx, dy)
        SET targetX = x + dx * dist
        SET targetY = y + dy * dist
        IF targetX, targetY is out of bounds
            BREAK
        ENDIF
        IF targetX == x and targetY == y
            CONTINUE
        ENDIF
        SET target = robot at (targetX, targetY)
        IF target exists and target != self and not hurt
            PRINT "QueenBot fires at (targetX, targetY)"
            useAmmo()
            IF hitProbability() succeeds
                target.takeDamage()
                PRINT name + successfully hit
                SELECT random upgrade random
                SET pending upgrade = upgradeType
                    PRINT "QueenBot will upgrade into upgradeType next
turn!"
            ELSE
                LOG "Missed!"
            ENDIF
            SET fired = true
            BREAK
        ENDIF
    ENDFOR
```

```
        IF not fired
            PRINT "No shooting, as sadly, QueenBot found no target in straight
    line"
        ENDIF
END

**************************************************************
VAMPIREBOT
**************************************************************
START
    INHERIT from GenericRobot
    METHOD Fire(TargetX, TargetY)
    IF no ammo
        PRINT name + has no ammo left!
        SET isDie = true
        RETURN
     ENDIF
     IF hasFired
        RETURN
     ENDIF
     SET hasFired = true
      IF detectedTargets is not empty
        SELECT random target
        SET targetX, targetY = target position
        PRINT name + fires at (targetX, targetY)
        useAmmo()
        IF target is hidden
```

```
            PRINT"Target is hidden, attack missed."
        ELSE IF hitProbability()
            target.takeDamage()
            PRINT name + is killed
            IF current lives < 3 and gainLivesCount < 3
                INCREMENT lives
                INCREMENT gainLivesCount
                PRINT “VampireBot gained 1 life from kill!”
            ELSE IF gainLivesCount >= 3
                PRINT "Already gained lives 3 times, cannot gain more."
            ELSE
                PRINT "Already at max lives, cannot gain extra life from this
    kill."
            ENDIF
            SELECT random upgrade type
            SET pending upgrade = upgradeType
            PRINT "VampireBot will upgrade into upgradeType next turn!"
        ENDIF
    ELSE
        PRINT "No shooting as no robots within shooting range."
    ENDIF
END
```

```
**************************************************************
SCOUTBOT (upgrade from robot.look)
**************************************************************
START
    INHERIT from GenericRobot
    DECLARE scoutCount = 0
    METHOD getUpgradeTypes()
            RETURN ["HideScoutBot", "JumpScoutBot"]
    END
    METHOD look(X, Y)
            CLEAR availableSpaces
            IF scoutCount >= 3
                PRINT name + " reach the limit, cannot scan already"
            ELSE IF random(0,1) == 0
                PRINT name + " scan the battlefield"
            FOR y from 0 to battlefield.height - 1
                FOR x from 0 to battlefield.width - 1
                    SET r = battlefield.getRobotAt(x, y)
                    IF r is not NULL
                        PRINT "got robot: " + r.name + " at (" +
    x + "," + y + ")"
                     ENDIF
                ENDFOR
            ENDFOR
            INCREMENT scoutCount
            ELSE
                PRINT name + " try scan it next round"
            ENDIF
```

```
                SET x = current X position
                SET y = current Y position
                FOR dx in [-1, 0, 1]
                    FOR dy in [-1, 0, 1]
                        IF dx == 0 and dy == 0
                            SET newX = x + dx
                            SET newY = y + dy
                        IF battlefield position (newX, newY) is
    available
                            ADD (newX, newY) to
    availableSpaces
                        ENDIF

                    ENDFOR
    END
    METHOD act()
      PRINT name + " is thinking..."
      CALL look(0, 0)
      CALL fire(0, 0)
      CALL move()
    END
    METHOD getScoutCount()
      RETURN scoutCount
END
```

```
**************************************************************
TRACKBOT (Seeing Upgrade)
**************************************************************
START
        INHERIT from GenericRobot
        DECLARE tracker = 3
        DECLARE track_target as list
        METHOD getUpgradeTypes()
           RETURN ["HideTrackBot", "JumpTrackBot"]
        END
        METHOD look(X, Y)
           CLEAR availableSpaces
           IF tracker == 0
              PRINT name + " cannot track robot already"
           ELSE
              SET x = current X position
              SET y = current Y position
              SET plant = false
              FOR dx in [-1, 0, 1] and not plant
                 FOR dy in [-1, 0, 1] and not plant
                    SET targetX = x + dx
                    SET targetY = y + dy
                    SET target = battlefield.getRobotAt(targetX, targetY)
                    IF target is not NULL and target != self
                       ADD target to track_target
                       DECREMENT tracker
                   PRINT name + " track " + target.name + " at (" + targetX + "," +
        targetY + ")"
```

```
                SET plant = true
            ENDIF
        ENDFOR
    ENDFOR
    IF NOT plant THEN
        PRINT name + " no target can track"
    ENDIF
  ENDIF
  FOR dx in [-1, 0, 1]
    FOR dy in [-1, 0, 1]
      IF dx == 0 and dy == 0
            SET newX = current X + dx
            SET newY = current Y + dy
             IF battlefield position (newX, newY) is available
                    ADD (newX, newY) to availableSpaces
            ENDIF
      ENDIF
    ENDFOR
  ENDFOR
END
METHOD act()
  PRINT name + " is thinking..."
  CALL look(0, 0)
  CALL fire(0, 0)
  CALL move()
END
METHOD showTrackTarget()
  IF track_target is empty
```

```
            PRINT name + " didn't track any robot"
        ELSE
            PRINT name + " is tracking:"
            FOR each r in track_target
                PRINT r.name + " at (" + r.X + "," + r.Y + ")"
            ENDFOR
        ENDIF
    END
    METHOD getTracker()
        RETURN tracker
END

**********************************************************************
HIDELONGSHOTBOT (2 UPGRADE BOT)
**********************************************************************
START
    INHERIT from HideBot, LongShotBot and GenericRobot
    METHOD getUpgradeTypes()
            RETURN ["HideLongShotScoutBot", "HideLongShotTrackBot"]
    END
    METHOD move()
            CALL HideBot.move()
    END
    METHOD fire(X, Y)
            CALL LongShotBot.fire(X, Y)
    END
    METHOD think()
            CALL HideBot.think()
```

```
    END
    METHOD act()
            CALL look(0, 0)
            CALL think()
            CALL fire(0, 0)
    END
    METHOD canBeHit()
            RETURN HideBot.canBeHit()
    END
    METHOD setBattlefield(bf)
            CALL GenericRobot.setBattlefield(bf)
    END

**************************************************************
HIDELONGSHOTSCOUTBOT (3 UPGRADE)
**************************************************************
START
    INHERIT from HideLongShotBot , ScoutBot and GenericRobot
    METHOD move()
      CALL HideLongShotBot.move()
    END
   METHOD fire(X, Y)
     CALL HideLongShotBot.fire(0, 0)
   END
   METHOD think()
     CALL HideLongShotBot.think()
   END
```

```
        METHOD act()
            CALL look(0, 0)
            CALL think()
            CALL fire(0, 0)
        END
        METHOD look(X, Y)
            CALL ScoutBot.look(X, Y)
        END
        METHOD canBeHit()
            RETURN HideBot.canBeHit()
         END
        METHOD setBattlefield(battlefield)
            CALL GenericRobot.setBattlefield(battlefield)
         END
END

**************************************************************
BATTLEFIELD SIMULATION STEP
**************************************************************
START
        METHOD simulationStep(stepNumber)
        CLEAR queuedThisRound
         FOR EACH robot IN listOfRobots
            IF robot = GenericRobot and robot has PendingUpgrade
                SET type = robot.getUpgradeType()
                SET upgraded = null
```

```
            IF type == "GenericRobot"
               upgraded = NEW GenericRobot with same name and position
            ELSE IF type == "HideBot"
               upgraded = NEW HideBot ...
            //all robot list
            ELSE IF type == "JumpVampireTrackBot"
               upgraded = NEW JumpVampireTrackBot ...
               IF upgraded IS NOT null
                    SET upgraded.lives = robot.lives
                   CALL upgraded.initializeFrom(robot)
                   SET upgraded.battlefield = this
                    CALL upgraded.clearPendingUpgrade()
                    CALL upgraded.resetActionFlags()
                    CALL removeRobotFromGrid(robot)
                   CALL placeRobot(upgraded, robot.x, robot.y)
                    DELETE robot
                    REPLACE robot IN listOfRobots WITH upgraded
                   PRINT upgraded.name + " has upgraded to " + type + "!"
                ENDIF
         ENDIF
 END FOR
 PRINT "Robot Status before Step " + stepNumber + ":"
FOR EACH robot IN listOfRobots DO
   DETERMINE typeName based on robot type
   SET currentAmmo = 0
   IF robot IS ThirtyShotBot THEN
      currentAmmo = robot.getShellCount()
   ELSE IF robot IS ShootingRobot THEN
```

```
        currentAmmo = robot.getAmmo()
    PRINT "  Type: " + typeName + ", Name: " + robot.name +
          ", Coords: (" + robot.x + "," + robot.y + ")" +
          ", Life: " + robot.lives + ", Ammo: " + currentAmmo
END FOR
SET currentlyAliveRobots = EMPTY LIST
FOR EACH robot IN listOfRobots
    IF robot.lives > 0 AND robot.isDie == FALSE THEN
        ADD robot = currentlyAliveRobots
END FOR
FOR each robot in currentlyAliveRobots
    IF robot = GenericRobot
        CALL robot.setBattlefield()
        CALL robot.resetActionFlags()
    IF robot.getIsHurt() == TRUE
        PRINT "-----------------------------------"
        PRINT robot.name + " was hit and skips this turn."
        CONTINUE TO NEXT ROBOT
    PRINT "-----------------------------------------"
    PRINT robot.name + "'s turn:"
    CALL robot.act()
    PRINT robot.name + " is done."
END FOR
CALL cleanupDestroyedRobots()
PRINT "-----------------------------------------"
CALL respawnRobots()
```

```
    FOR EACH robot IN listOfRobots DO
        SET robot.isDie = FALSE
    END FOR
END
```

```
**********************************************************************
REENTRY QUEUE
**********************************************************************
START
        IF robot is already in queuedThisRound
                RETURN
        ADD robot to queuedThisRound
        IF robot is a ThirtyShotBot
                currentAmmo = robot.shellCount
        ELSE IF robot is a ShootingRobot
                currentAmmo = robot.ammo
        PUSH (robot.name, robot.lives, currentAmmo) into reentryQueue
        PRINT robot is queued for reentry with lives and ammo
END
```

****

RESPAWN ROBOT

****

START

IF reentryQueue is not empty

GET the first robot info from the queue

name = robot. name

lives = robot.lives

ammo = robot.ammo

REMOVE robot from reentryQueue

PRINT reentering robot name

SET attempts = 50

WHILE attempts > 0

GENERATE randomX and randomY

IF position (randomX, randomY) is valid and empty THEN

spotFound = true

Decrement attempts

IF spotFound

CREATE new GenericRobot with same name at random position

SET robot lives to livesLeft

SET robot battlefield pointer

IF robot is a ShootingRobot

SET robot ammo to ammoLeft

PLACE robot on battlefield

ADD robot to listOfRobots

PRINT Robot reentered at (x, y) with lives and ammo

```
            ELSE
                PRINT No spot available. Requeue robot for next turn
                PUSH robot back into reentryQueue
END

**************************************************************
DESTROY ROBOT
**************************************************************
START
    PRINT cleanup section begins
    FOR each robot in listOfRobots
        IF robot is dead or isDie or isHurt
            REMOVE robot from grid
            PRINT Robot removed
            DELETE robot
            REMOVE from listOfRobots
END
**************************************************************
DESTRUCTOR
**************************************************************
START
FOR each robot in listOfRobots
        DELETE robot
CLEAR listOfRobots
END
```