Faculty of Computing and Informatics (FCI)

Multimedia University

Cyberjaya

CCP6124 – Object-Oriented Programming & Data Structures

Trimester 2410

Assignment Report

Submitted to:

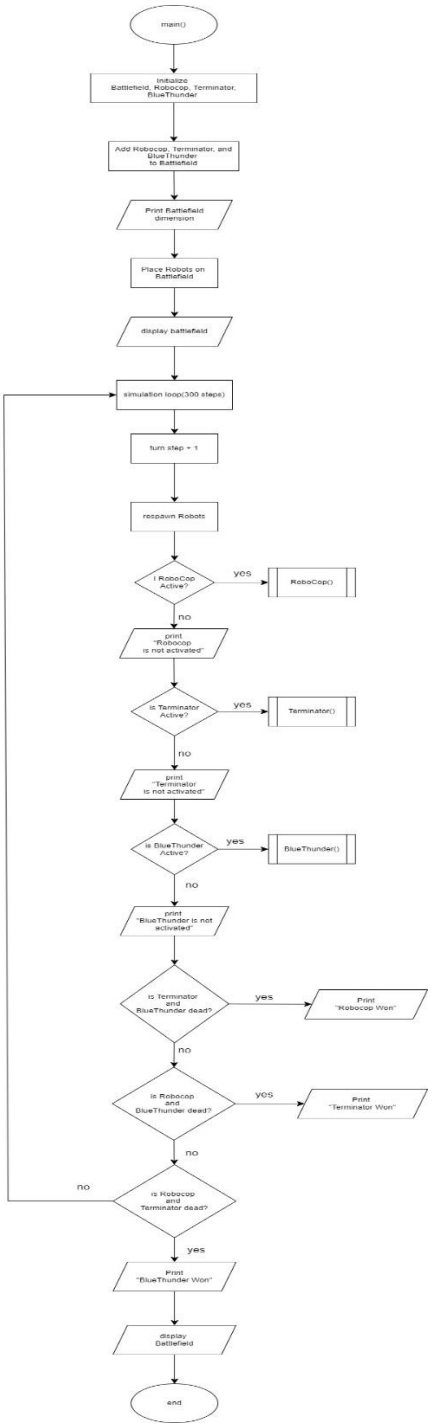
Mr. Sharaf El-Deen Sami Mohammed Al- Horani

Group Members:

NAME	STUDENT ID
ABDUL ADZEEM ABDUL RASYID	1211109773
AIN NUR YASMIN BINTI MUHD ZAINI	1211109582
MOHAMMAD HAZMAN BIN KHAIRIL ANWAR	1211110444
IZZA NELLY BINTI MOHD NASIR	1211111583

MAIN

This main program simulates a battle between three robots — RoboCop, Terminator, and BlueThunder—on a 10x10 battlefield. Each robot starts at a specified position, and their actions are simulated in a loop for up to 300 steps. In each step, the program updates the battlefield, makes each robot perform an action if they are still active, and checks if the game has ended based on the robots' statuses. The simulation stops when only one robot remains active, declaring that robot as the winner.



BATTLEFIELD

Explanation:
The Battlefield class simulates a battlefield where robots can be placed, moved, and respawned. It manages a 2D grid and keeps track of the robots and their states. The class has several attributes and methods to facilitate these operations.

The class contains the following attributes:

- *'rows'* and *'cols'* which define the dimensions of the battlefield.
- *'field[11][11]'* which is a 2D array representing the battlefield grid.
- *'robots[3]'* which is an array that can hold up to 3 robots currently on the battlefield.
- *'robotCount'* which keeps track of the number of robots on the battlefield.
- 'respawnQueue' which is a queue of robots waiting to be respawned.

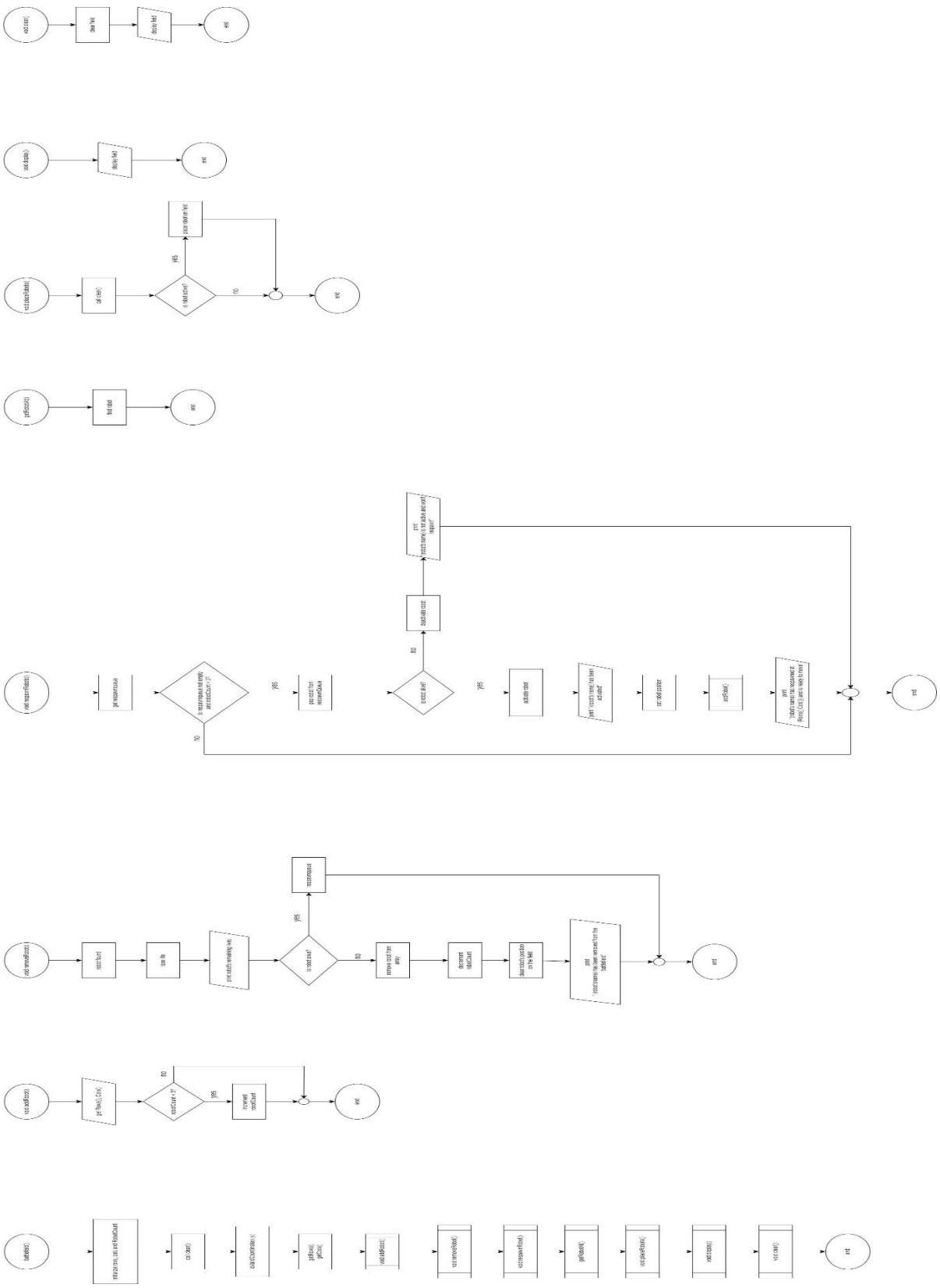
The constructor initializes the battlefield with the given dimensions and clears the field to prepare it for robot placement. The *'isValidCoordinate'* method checks if given coordinates are within the battlefield boundaries.

The *'getRows'* and *'getCols'* methods return the dimensions of the battlefield. The *'addRobot'* method adds a robot to the battlefield if there is space, while the *'removeRobot'* method removes a robot from the battlefield, reduces its lives, and queues it for respawn if it still has lives left.

The *'respawnRobots'* method respawns robots from the queue if there is space on the battlefield. It activates the robot, finds an empty spot on the battlefield, sets the robot's position, and adds it back to the battlefield.

The *'getRobotAt'* method returns the robot at specified coordinates if one exists. The *'placeRobots'* method clears the battlefield and places all active robots in their respective positions. The *'display'* method prints the current state of the battlefield, showing where each robot is placed. The *'clear'* method clears the battlefield, setting all grid positions to empty.

Flowchart:



BASE ABSTRACT CLASS

ROBOT

Explanation:

The *'Robot'* class serves as the base class for various types of robots. It defines common attributes and methods that all derived robot types will share. These include attributes for the robot's name, position on the battlefield, number of lives, kills, death count, and active status. The class also provides several methods to manipulate these attributes and to define robot behavior.

The constructor initializes the robot with a name, initial position, three lives, zero kills, zero deaths, and an active status. The destructor is virtual, allowing for proper cleanup in derived classes.

The class provides getter methods for the robot's name, position, lives, kills, and deaths. There are methods to decrease the robot's lives *('loseLife')*, check if the robot is alive or dead *('isAlive', 'isDead')*, increment the kill and death counts *('incrementKills', 'incrementDeaths')*, set the robot's position *('setPosition')*, and check or change the active status *('isActive', 'deactivate', 'activate')*. Additionally, there is a method to set the number of kills *('setKills')*.

The *'Robot'* class declares several pure virtual methods: *'getDisplayChar', 'look', 'move', 'fire', 'stepping'*, and *'action'*. These methods must be implemented by any derived class, providing specific functionality for different types of robots.

4 BASIC ABSTRACT SUBCLASSES

1. MOVE ROBOT

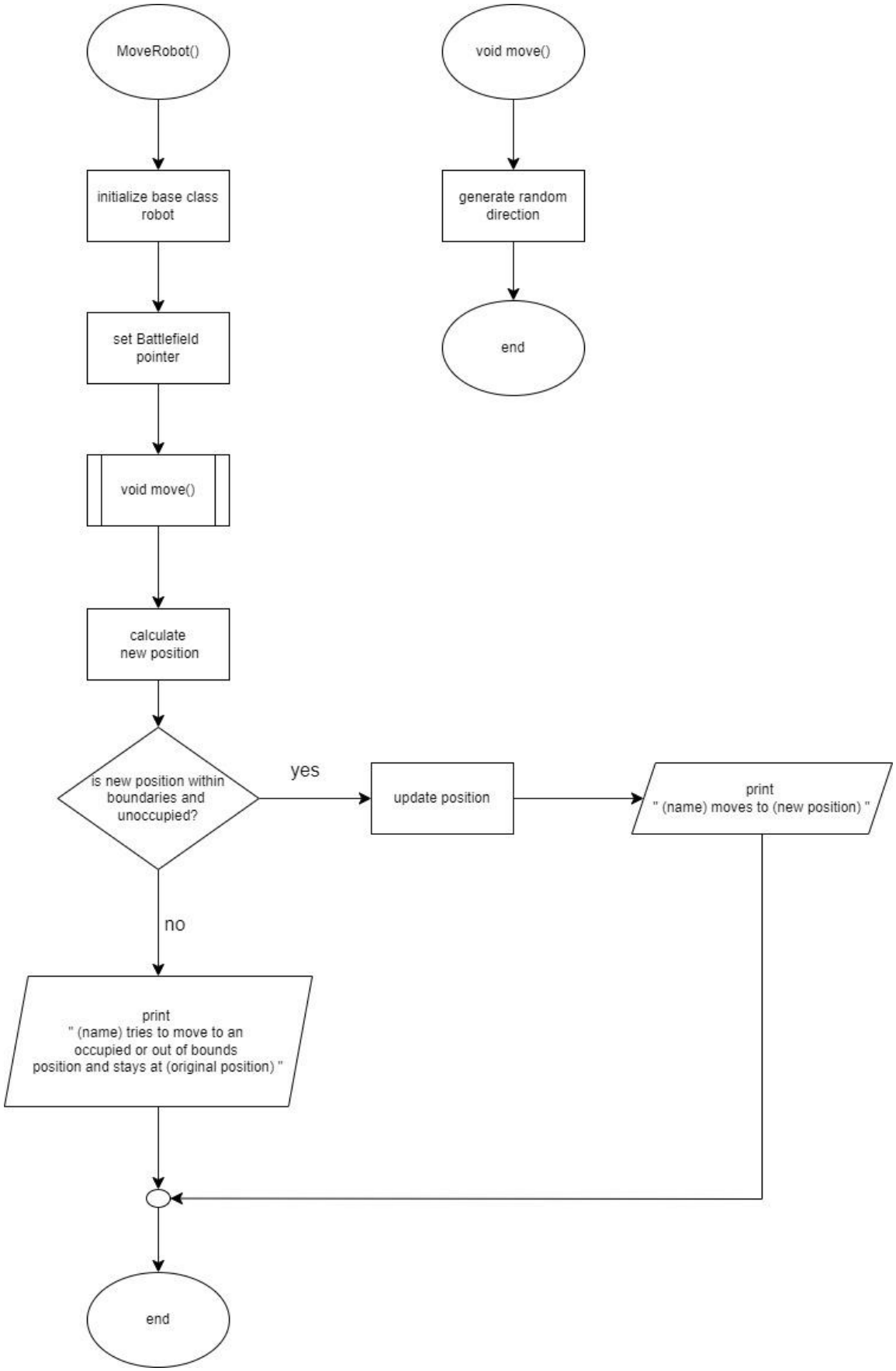
Explanation:

The *'MoveRobot'* class is a specialized robot that inherits from the base *'Robot'* class. This class represents a robot capable of moving randomly within a battlefield. The *'MoveRobot'* class uses virtual inheritance from Robot to allow multiple inheritance scenarios where *'MoveRobot'* can be combined with other robot types (e.g., shooting, seeing).

The constructor *'MoveRobot'* initializes the robot's name, position, and battlefield reference. It calls the base *'Robot'* constructor to set up the name and position and sets the battlefield pointer.

The *'move'* method is overridden to provide the movement capability. It randomly selects one of the eight possible directions (up, down, left, right, and diagonals) to move. The new coordinates are calculated based on the chosen direction. The method then checks if the new position is within the battlefield boundaries and not occupied by another robot. If the move is valid, it updates the robot's position and prints the new coordinates. If the move is invalid (out of bounds or occupied), it prints a message indicating the robot stays in its current position.

Flowchart:


MoveRobot()
initialize base class robot
set Battlefield pointer
void move()
calculate new position
is new position within boundaries and unoccupied?
yes
update position
print " (name) moves to (new position) "
no
print " (name) tries to move to an occupied or out of bounds position and stays at (original position) "
end
void move()
generate random direction
end

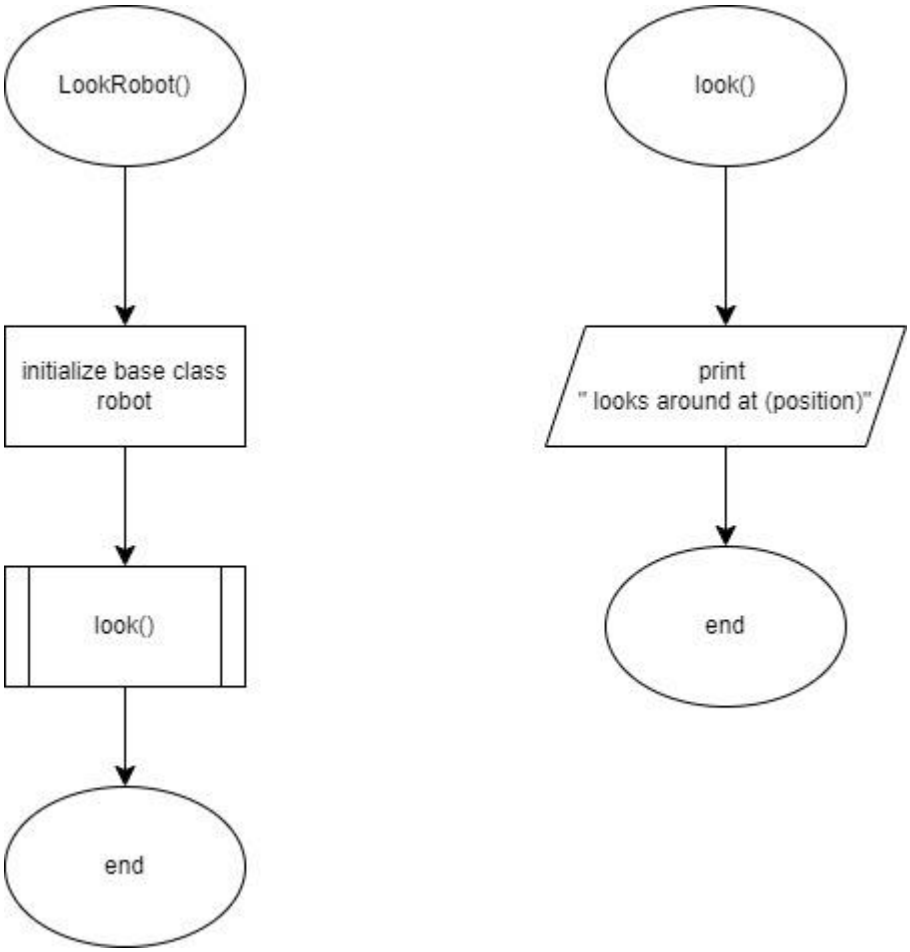
2. LOOK ROBOT

Explanation:

The *'LookRobot'* class is a specialized type of *'Robot'* that has the ability to "look" around. It inherits from the *'Robot'* class virtually, allowing it to be combined with other robot types without ambiguity or redundancy in the inheritance structure. The *'look'* method is implemented to output a message indicating that the robot is looking around its current position.

Upon construction, a *'LookRobot'* object initializes with its name and initial position by calling the base *'Robot'* class constructor.

Flowchart:



3. STEP ROBOT

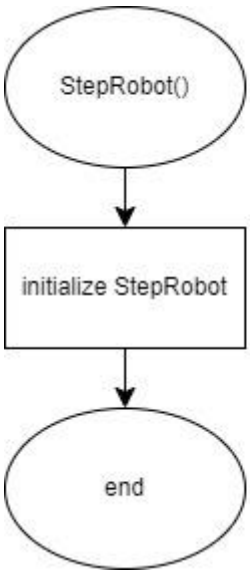
Explanation:

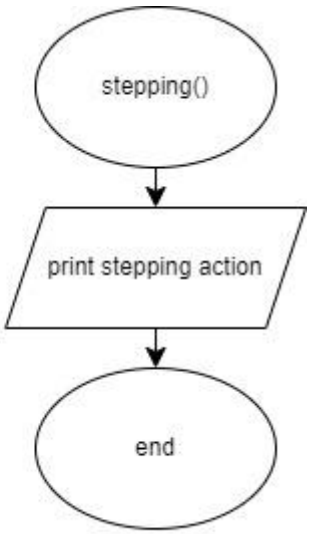
The *'StepRobot'* class extends the 'Robot' class to introduce the functionality of "stepping" on other robots. This class is designed to be used with other robot types through virtual inheritance, allowing the combination of different robot functionalities without ambiguity.

When a *'StepRobot'* object is created, it initializes the robot's name and position using the *'Robot'* class constructor.

The *'stepping'* method in *'StepRobot'* outputs a message indicating that the robot is stepping on another robot at its current position. Note that this method does not perform any actual logic beyond printing a message; it's essentially a placeholder for more complex behavior that might be added in the future.

Flowchart:





4. SHOOT ROBOT

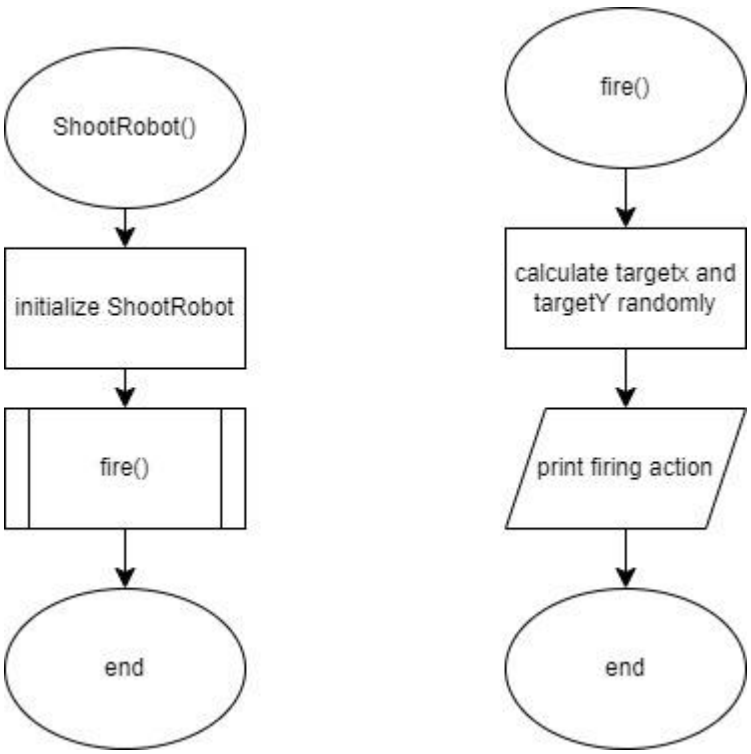
Explanation:

The *'ShootRobot'* class extends the *'Robot'* class and introduces the capability to fire at targets. This class is designed to be used in combination with other robot types through virtual inheritance, which prevents ambiguity in the inheritance hierarchy.

Upon construction, a *'ShootRobot'* object is initialized with its name and position by invoking the *'Robot'* class constructor.

The *'fire'* method in *'ShootRobot'* calculates a target position around the robot's current location. The target coordinates are determined by adding a random value from -1 to 1 to the robot's current x and y positions. This random approach means the robot can fire in any of the eight surrounding squares, or stay in the same square.

Flowchart:



ROBOTS

1. ROBOCOP

Explanation:

The *'RoboCop'* class is a multifunctional robot that inherits from *'MovingRobot'*, *'ShootingRobot'*, and *'SeeingRobot'*. This class represents a robot capable of moving, shooting, and "seeing" (though the seeing functionality is not explicitly implemented here). The *'RoboCop'* operates within a Battlefield and has the potential to upgrade to a *'TerminatorRoboCop'* after achieving a certain number of kills.

The constructor initializes the robot's name, position, and battlefield reference. It calls the base constructors of *'Robot'*, *'MovingRobot'*, *'ShootingRobot'*, and *'SeeingRobot'* to set up the necessary properties and adds the robot to the battlefield.

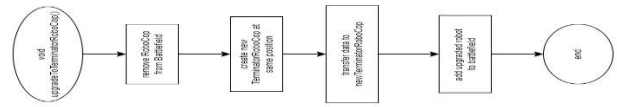
The 'fire' method handles the firing mechanism. It randomly selects a target within a specified range (up to 5 units away in any direction). If the target coordinates are valid and there is a robot at that position, it fires at the robot. If the robot is hit, it is removed from the battlefield, and the firing robot's kill count is incremented. If the target coordinates are out of range or the shot misses, appropriate messages are printed.

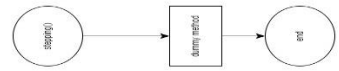
The *'action'* method performs a sequence of actions: it calls the look method, moves the robot to a new position, and fires three times in succession. It then prints the robot's current kill count. If the *'RoboCop'* achieves three or more kills, it indicates an upgrade to *'TerminatorRoboCop'*.

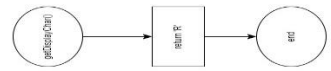
The *'getDisplayChar'* method returns 'R' to represent *'RoboCop'* on the battlefield.

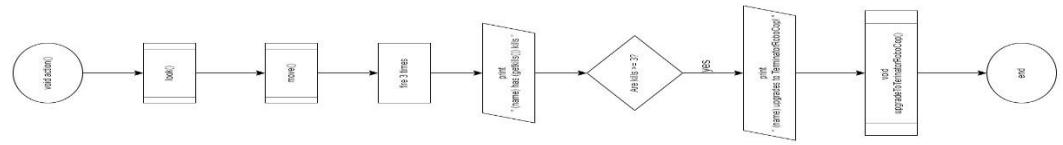
The *'stepping'* method is overridden but not implemented, serving as a dummy method.

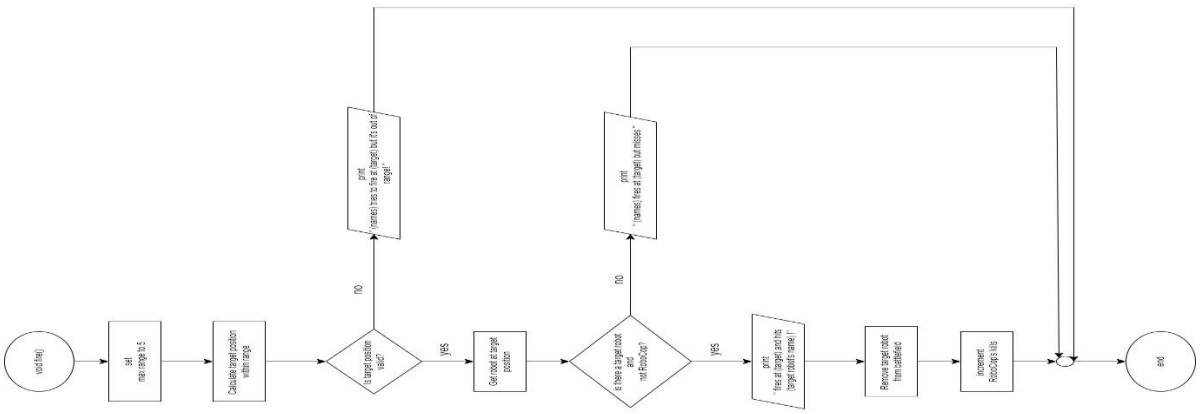
Flowchart:

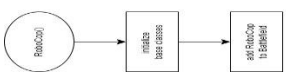

void upgradeToTerminatorRoboCop()
remove RoboCop from Battlefield
create new TerminatorRoboCop at same position
transfer data to newTerminatorRoboCop
add upgraded robot to battlefield
end


sleeping()
dummy method
end


getClassIn.Chair()
return R
end


void action()
look()
move()
hits 5 times
print "(Name) has upgraded his hits"
Are Hits >= 10
yes
print "(Name) upgrades to TerminatorRoboCop!"
void upgradeToTerminatorRoboCop()
end


void fire()
set max range to 5
Calculate target position within range
is target position valid?
no
print "(Name) tries to fire at (target), but it's out of range!"
yes
Get robot at target position
is there a target robot and not RoboCop?
no
print "(Name) fires at (target) but misses."
yes
print "fires at (target) and hits (target.robotName)!"
Remove target robot from battlefield
increment RoboCops hits
end


RoboCop()
initalize bases classes
add RoboCop to battlefield

2. TERMINATOR

Explanation:

The *'Terminator'* class inherits from *'MovingRobot'*, *'SteppingRobot'*, and *'SeeingRobot'*, integrating these functionalities into a single robot type. Upon creation, the 'Terminator' is positioned on a *'Battlefield'* where it can observe its surroundings, move to new locations, and step on enemies that happen to be at the same position.

When the *'Terminator'* performs its primary action, it first observes the battlefield to understand the state of its environment. Following this, it moves to a new location on the battlefield. The *'Terminator'* then engages in the "stepping" action, where it steps on any enemies currently at its position, effectively removing them from the battlefield and increasing its kill count.

The *'action'* method encapsulates these steps and, after performing them, checks if the *'Terminator'* has achieved enough kills to trigger an upgrade. If the *'Terminator'* has accumulated three or more kills, it prints a message about the upgrade and is supposed to evolve into a *'TerminatorRoboCop'*.

The *'stepping'* method specifically handles the logic of stepping on enemies. It checks the *'Battlefield'* for any robots at the current position and if an enemy robot is found, it removes that robot from the battlefield and updates the *'Terminator'*'s kill count. If no enemies are found, it simply reports that there was no enemy to step on.

The *'getDisplayChar'* method returns a character representing the *'Terminator'* on the battlefield map. In this case, the character 'T' is used to denote the *'Terminator'*.

3. TERMINATOR ROBOCOP

Explanation:

The *'TerminatorRoboCop'* class inherits from *'MovingRobot'*, *'ShootingRobot'*, *'SeeingRobot'*, and *'SteppingRobot'*. This inheritance means that *'TerminatorRoboCop'* possesses a combination of features from all these base classes, allowing it to move around the battlefield, observe its environment, attack enemies from a distance, and step on nearby foes.

Upon creation, a *'TerminatorRoboCop'* instance is placed on a *'Battlefield'* where it can perform its various actions. The *'fire'* method in *'TerminatorRoboCop'* allows it to shoot at a target within a larger range compared to the *'Terminator'* class. The *'stepping'* method is also implemented to handle the action of stepping on enemies.

In the *'action'* method, *'TerminatorRoboCop'* observes the battlefield, moves to a new location, fires at enemies multiple times, and steps on any enemy at its current position. After these actions, it reports its number of kills. The 'action' method thus orchestrates the *'TerminatorRoboCop'*"s activities in the battlefield and demonstrates its enhanced capabilities.

The *'getDisplayChar'* method returns a character that represents the *'TerminatorRoboCop'* on the battlefield map, using 'T' to denote this advanced robot form.

Flowchart:

4. BLUE THUNDER

Explanation:

The *'BlueThunder'* class is a specialized robot that inherits from *'ShootingRobot'* and *'SeeingRobot'*. This class represents a robot that can fire in four directions but does not move or use the seeing capability. It operates within a 'Battlefield' and keeps track of its firing direction.

The constructor *'BlueThunder'* initializes the robot's name, position, and battlefield, and sets the initial firing direction to upward. The constructor also registers the robot with the battlefield.

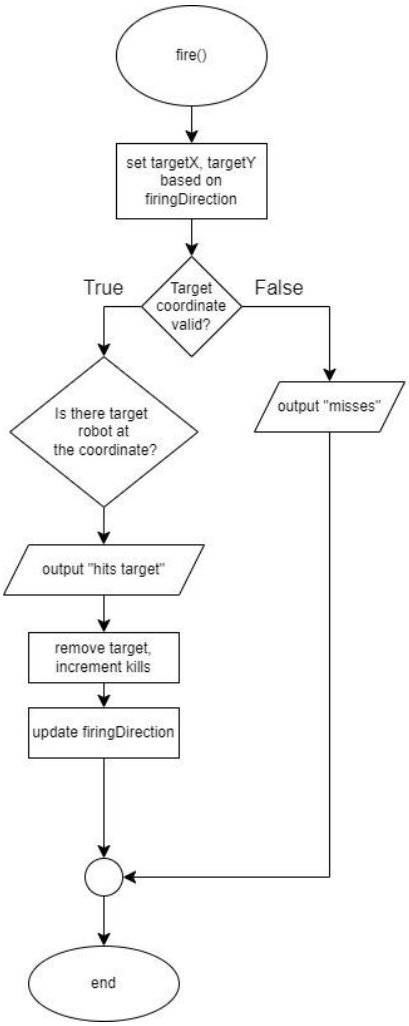
The *'fire'* method handles the firing mechanism of the robot. Depending on the current firing direction (up, right, down, left), it calculates the target coordinates. If the target coordinates are valid and there is a robot at that position, it fires at the robot. If the robot is hit, it is removed from the battlefield, and the firing robot's kill count is incremented. The firing direction is then updated to the next direction in a clockwise manner.

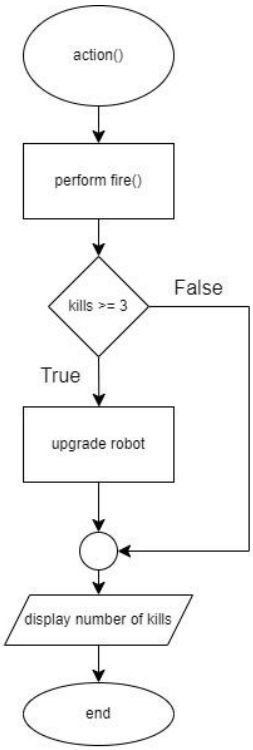
The *'look'* and *'move'* methods are overridden but not implemented since *'BlueThunder'* does not use these capabilities. The *'stepping'* method is also a dummy implementation.

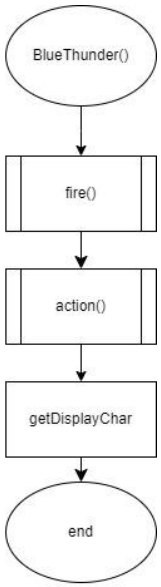
The 'action' method calls the *'fire'* method once per action cycle and checks if the robot has achieved at least three kills, potentially changing the robot type (although this feature is not implemented here). It also prints the robot's current kill count.

The *'getDisplayChar'* method returns 'B' to represent *'BlueThunder'* on the battlefield.

Flowchart:


fire()
set targetX, targetY based on firingDirection
True
Target coordinate valid?
False
Is there target robot at the coordinate?
output "misses"
output "hits target"
remove target, increment kills
update firingDirection
end


action()
perform fire()
kills >= 3
False
True
upgrade robot
display number of kills
end


BlueThunder()
fire()
action()
getDisplayChar
end

5. MAD BOT

Explanation:

The *'MadBot'* class is a specialized robot that inherits from both *'ShootingRobot'* and *'SeeingRobot'*. This robot can shoot in any of the eight directions (up, down, left, right, and diagonals) but does not move. It operates within a *'Battlefield'* and has the ability to upgrade to a *'RoboTank'* after achieving a certain number of kills.

The constructor initializes the robot's name, position, and battlefield. It also registers the robot with the battlefield.

The *'move'* method is overridden but not implemented since *'MadBot'* does not move.

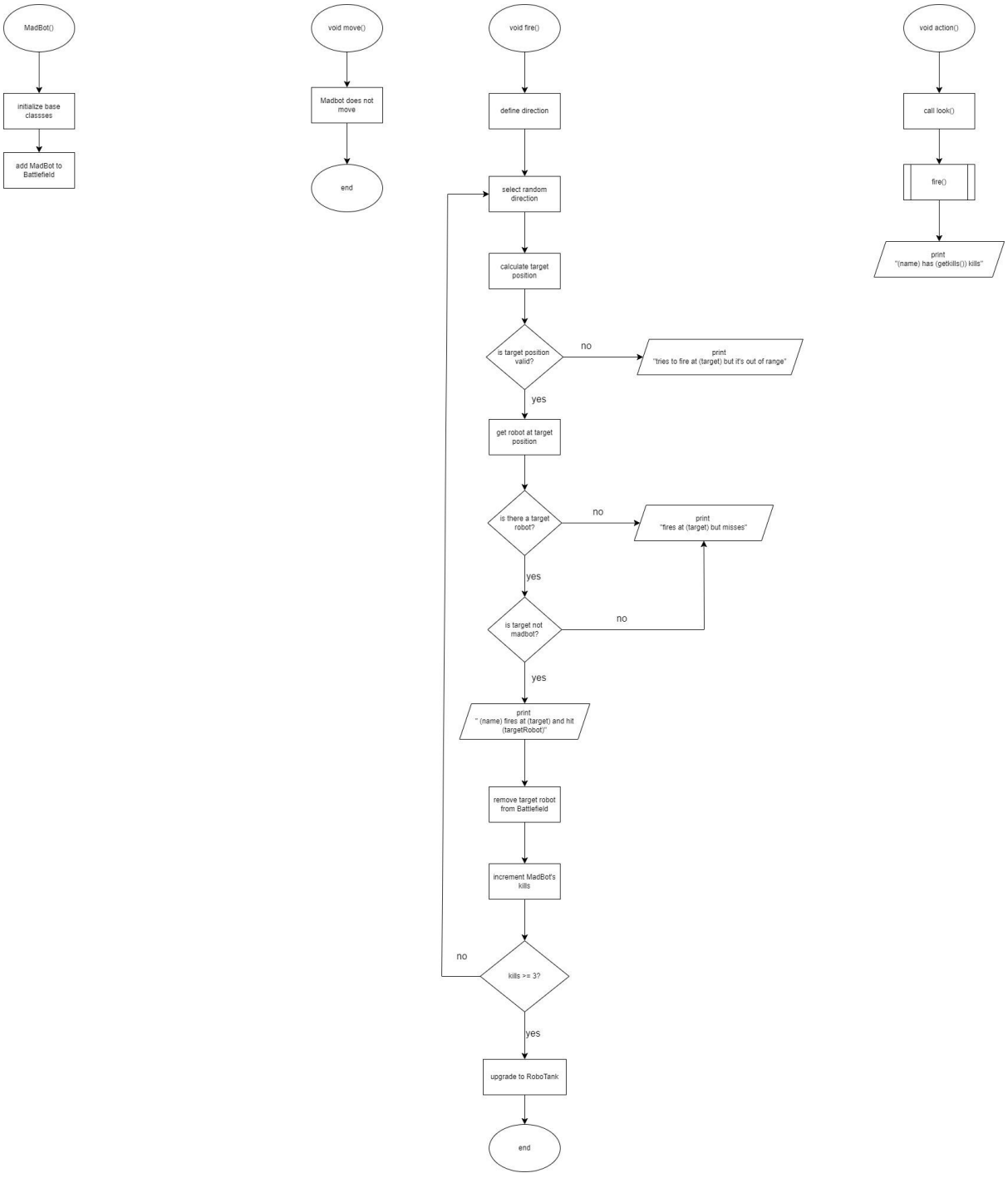
The *'fire'* method handles the firing mechanism. It randomly selects one of the eight possible directions to fire. If the target coordinates are valid and there is a robot at that position, it fires at the robot. If the robot is hit, it is removed from the battlefield, and the firing robot's kill count is incremented. If the *'MadBot'* achieves three or more kills, it upgrades to a *'RoboTank'*.

The *'action'* method calls the *'look'* and *'fire'* methods and prints the robot's current kill count.

The *'getDisplayChar'* method returns 'M' to represent *'MadBot'* on the battlefield.

The *'upgradeToRoboTank'* method handles the upgrade process. It removes the current *'MadBot'* from the battlefield, creates a new *'RoboTank'* at the same position, transfers the kill count to the new robot, and adds the upgraded robot to the battlefield.

Flowchart:


MadBot()
initialize base classses
add MadBot to Battlefield
void move()
Madbot does not move
end
void fire()
define direction
select random direction
calculate target position
is target position valid?
no
print "tries to fire at (target) but it's out of range"
yes
get robot at target position
is there a target robot?
no
print "fires at (target) but misses"
yes
is target not madbot?
no
yes
print " (name) fires at (target) and hit (targetRobot)"
remove target robot from Battlefield
increment MadBot's kills
kills >= 3?
no
yes
upgrade to RoboTank
end
void action()
call look()
fire()
print "(name) has (getkills()) kills"

6. ROBOTANK

Explanation:

The *'RoboTank'* class is a specialized robot inherits from both *'ShootingRobot'* and *'SeeingRobot'*. It does not move but has enhanced firing capabilities, allowing it to target any coordinate on the battlefield. The class includes mechanisms for upgrading to an even more advanced robot, *'UltimateRobot'*, once it achieves a certain number of kills.

Upon construction, a *'RoboTank'* object initializes with its name, initial position, and a reference to the battlefield. It then adds itself to the battlefield.

The *'move'* method is overridden to do nothing, reflecting that *'RoboTank'* does not move.

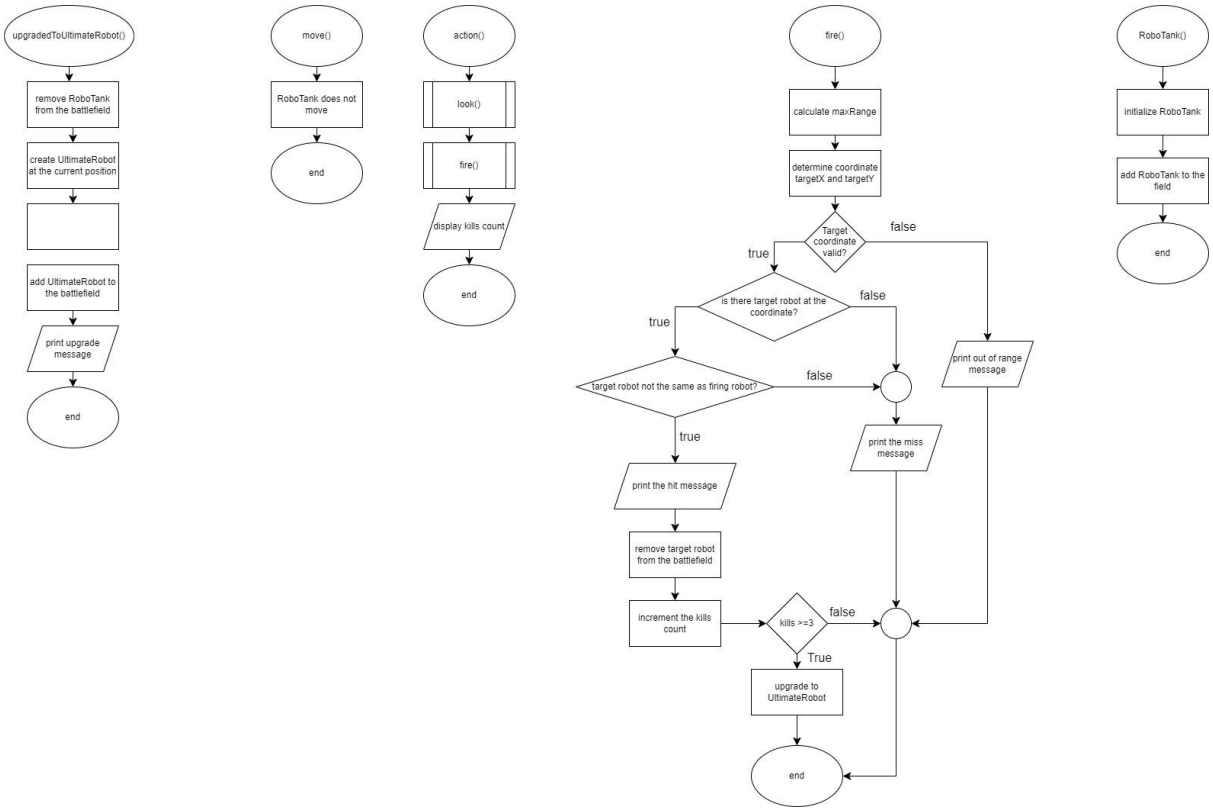
The *'fire'* method allows *'RoboTank'* to shoot at a random target within the entire battlefield range. If a valid target robot is hit, it removes the target from the battlefield and increments its kill count. When the kill count reaches 3, it upgrades to an *'UltimateRobot'*.

The *'action'* method makes *'RoboTank'* look and fire, and it outputs the current number of kills.

The *'getDisplayChar'* method returns 'R', representing the *'RoboTank'* on the battlefield.

The private method *'upgradeToUltimateRobot'* handles the transformation of *'RoboTank'* into an *'UltimateRobot'*. It removes the current robot from the battlefield, creates a new *'UltimateRobot'* with the same position and kill count, and adds the new robot to the battlefield.

Flowchart:


upgradedToUltimateRobot()
remove RoboTank from the battlefield
create UltimateRobot at the current position
add UltimateRobot to the battlefield
print upgrade message
end
move()
RoboTank does not move
end
action()
look()
fire()
display kills count
end
fire()
calculate maxRange
determine coordinate targetX and targetY
Target coordinate valid?
false
true
is there target robot at the coordinate?
false
true
target robot not the same as firing robot?
false
true
print out of range message
print the miss message
print the hit message
remove target robot from the battlefield
increment the kills count
kills >=3
false
True
upgrade to UltimateRobot
end
RoboTank()
initialize RoboTank
add RoboTank to the field
end

7. ULTIMATE ROBOT

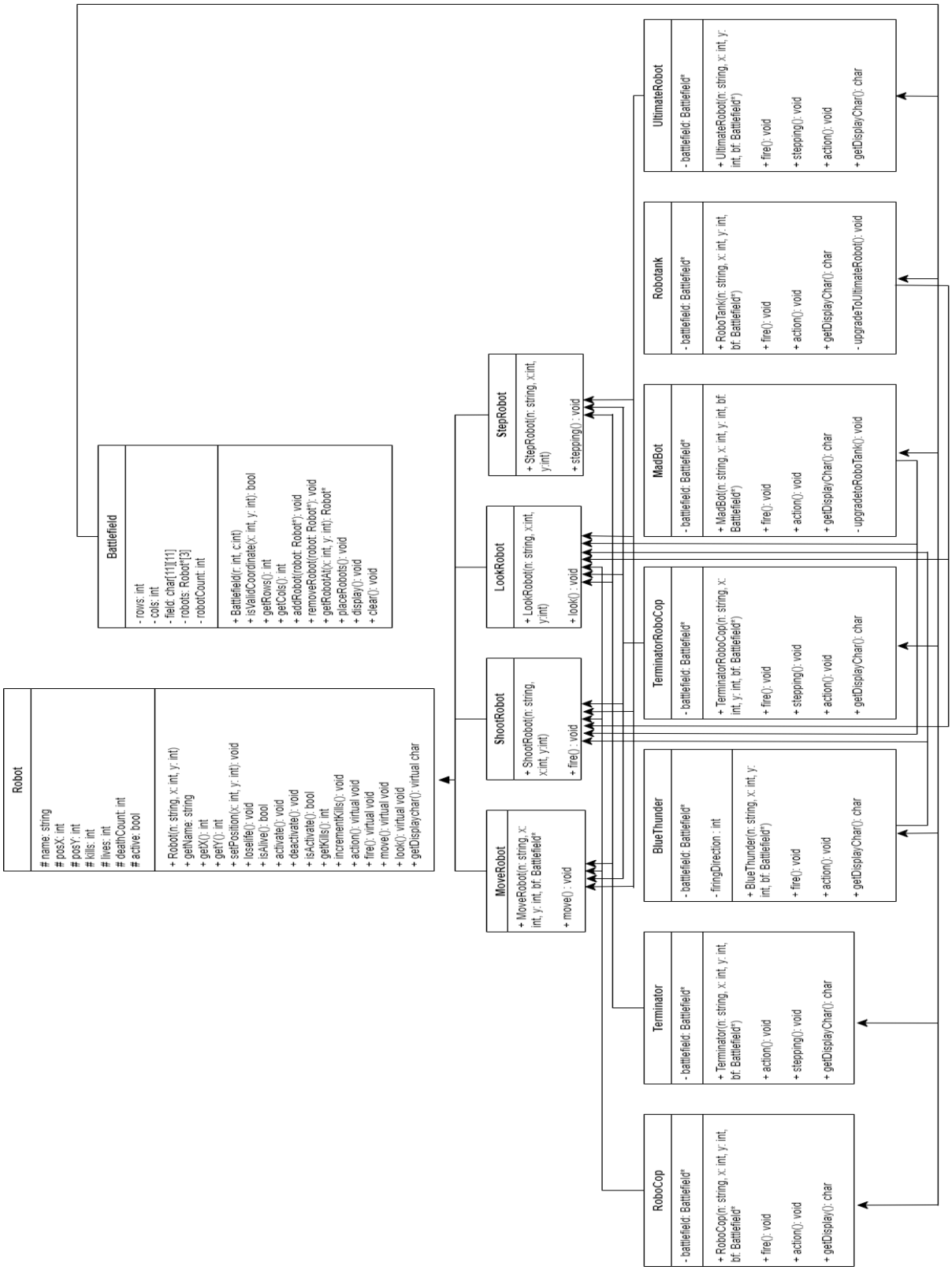
Explanation:

The *'UltimateRobot'* class builds upon the functionalities of *'TerminatorRoboCop'* by providing more refined and effective methods for combat and battlefield navigation. It integrates a comprehensive set of features that enable the robot to engage in multiple forms of attacks and maneuvers.

When an *'UltimateRobot'* is created, it is placed on a *'Battlefield'* where it can perform its actions. The *'fire'* method in *'UltimateRobot'* allows it to execute a series of attacks, targeting random coordinates within a specified range. The *'stepping'* method enables it to step on enemies that are in the same position. The *'action'* method orchestrates the robot's behavior during a turn, making it observe the battlefield, move to a new location, attack enemies, and step on any foes at its current position.

The *'getDisplayChar'* method returns 'U' for the *'UltimateRobot'* to represent it on the battlefield map.

CLASS DIAGRAM



RUNNER DEMO

- Initialization of a simulation

```
The battlefield dimension is 10 x 10.
Steps: 50
This is the starting position of the robots.
. . . . . .
. . . . . .
. . R . . . .
. . . T . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B .
. . . . . .
. . . . . .
```

- Display and logging of the status of the battlefield, action of the robots and the status of the robots at each turn

```
Turn 1:
RoboCop looks around at (2, 2)
RoboCop moves to (1, 1)
RoboCop tries to fire at (-1, 6) but it's out of range!
RoboCop fires at (6, 5) but misses!
RoboCop tries to fire at (2, -2) but it's out of range!
RoboCop has 0 kills.
Terminator looks around at (3, 3)
Terminator moves to (2, 3)
Terminator tries to step but finds no enemy at (2, 3)!
Terminator has 0 kills.
BlueThunder fires at (7, 6) but misses!
BlueThunder has 0 kills.
. . . . . .
. R . . . . . .
. . . T . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B .
. . . . . .
. . . . . .
```

```
Turn 2:
RoboCop looks around at (1, 1)
RoboCop moves to (2, 2)
RoboCop fires at (1, 4) but misses!
RoboCop fires at (5, 7) but misses!
RoboCop tries to fire at (-1, 2) but it's out of range!
RoboCop has 0 kills.
Terminator looks around at (2, 3)
Terminator tries to move to an occupied or out of bounds position and stays at (2, 3)
Terminator tries to step but finds no enemy at (2, 3)!
Terminator has 0 kills.
BlueThunder fires at (8, 7) but misses!
BlueThunder has 0 kills.
. . . . . .
. . . . . .
. . R T . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B . .
. . . . . .
. . . . . .
```

```
Turn 3:
RoboCop looks around at (2, 2)
RoboCop moves to (3, 1)
RoboCop tries to fire at (-1, 0) but it's out of range!
RoboCop tries to fire at (1, -4) but it's out of range!
RoboCop fires at (4, 0) but misses!
RoboCop has 0 kills.
Terminator looks around at (2, 3)
Terminator moves to (1, 2)
Terminator tries to step but finds no enemy at (1, 2)!
Terminator has 0 kills.
BlueThunder fires at (7, 8) but misses!
BlueThunder has 0 kills.
. . . . . .
. . T . . . . .
. . . . . .
. R . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B . .
. . . . . .
. . . . . .
```

```
Turn 4:
RoboCop looks around at (3, 1)
RoboCop moves to (2, 0)
RoboCop fires at (2, 3) but misses!
RoboCop tries to fire at (-2, -2) but it's out of range!
RoboCop tries to fire at (7, -4) but it's out of range!
RoboCop has 0 kills.
Terminator looks around at (1, 2)
Terminator moves to (0, 1)
Terminator tries to step but finds no enemy at (0, 1)!
Terminator has 0 kills.
BlueThunder fires at (6, 7) but misses!
BlueThunder has 0 kills.
. T . . . . . .
. . . . . .
R . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B . .
. . . . . .
. . . . . .
```

```
Turn 5:
RoboCop looks around at (2, 0)
RoboCop moves to (1, 0)
RoboCop tries to fire at (4, -1) but it's out of range!
RoboCop tries to fire at (3, -2) but it's out of range!
RoboCop tries to fire at (-2, 5) but it's out of range!
RoboCop has 0 kills.
Terminator looks around at (0, 1)
Terminator tries to move to an occupied or out of bounds position and stays at (0, 1)
Terminator tries to step but finds no enemy at (0, 1)!
Terminator has 0 kills.
BlueThunder fires at (7, 6) but misses!
BlueThunder has 0 kills.
. T . . . . . .
R . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B . .
. . . . . .
. . . . . .
```

```
Turn 6:
RoboCop looks around at (1, 0)
RoboCop tries to move to an occupied or out of bounds position and stays at (1, 0)
RoboCop fires at (5, 1) but misses!
RoboCop tries to fire at (-2, -5) but it's out of range!
RoboCop tries to fire at (-1, -1) but it's out of range!
RoboCop has 0 kills.
Terminator looks around at (0, 1)
Terminator moves to (1, 2)
Terminator tries to step but finds no enemy at (1, 2)!
Terminator has 0 kills.
BlueThunder fires at (8, 7) but misses!
BlueThunder has 0 kills.
. . . . . .
R . T . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B . .
. . . . . .
. . . . . .
```

```
Turn 7:
RoboCop looks around at (1, 0)
RoboCop tries to move to an occupied or out of bounds position and stays at (1, 0)
RoboCop tries to fire at (3, -1) but it's out of range!
RoboCop tries to fire at (0, -5) but it's out of range!
RoboCop fires at (2, 5) but misses!
RoboCop has 0 kills.
Terminator looks around at (1, 2)
Terminator moves to (0, 3)
Terminator tries to step but finds no enemy at (0, 3)!
Terminator has 0 kills.
BlueThunder fires at (7, 8) but misses!
BlueThunder has 0 kills.
. . . T . . . . .
R . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B . .
. . . . . .
. . . . . .
```

```
Turn 8:
RoboCop looks around at (1, 0)
RoboCop moves to (1, 1)
RoboCop tries to fire at (3, -2) but it's out of range!
RoboCop fires at (6, 2) but misses!
RoboCop fires at (6, 6) but misses!
RoboCop has 0 kills.
Terminator looks around at (0, 3)
Terminator moves to (0, 4)
Terminator tries to step but finds no enemy at (0, 4)!
Terminator has 0 kills.
BlueThunder fires at (6, 7) but misses!
BlueThunder has 0 kills.
. . . . T . . . . .
. R . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . B . .
. . . . . . . . . .
. . . . . . . . . .
```

```
Turn 9:
RoboCop looks around at (1, 1)
RoboCop moves to (2, 0)
RoboCop fires at (6, 4) but misses!
RoboCop tries to fire at (-2, -2) but it's out of range!
RoboCop tries to fire at (-1, -5) but it's out of range!
RoboCop has 0 kills.
Terminator looks around at (0, 4)
Terminator tries to move to an occupied or out of bounds position and stays at (0, 4)
Terminator tries to step but finds no enemy at (0, 4)!
Terminator has 0 kills.
BlueThunder fires at (7, 6) but misses!
BlueThunder has 0 kills.
. . . . T . . . . .
. . . . . . . . . .
R . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . B . .
. . . . . . . . . .
. . . . . . . . . .
```

```
Turn 10:
RoboCop looks around at (2, 0)
RoboCop moves to (3, 0)
RoboCop tries to fire at (-2, 1) but it's out of range!
RoboCop tries to fire at (-1, -3) but it's out of range!
RoboCop tries to fire at (-2, 1) but it's out of range!
RoboCop has 0 kills.
Terminator looks around at (0, 4)
Terminator moves to (0, 5)
Terminator tries to step but finds no enemy at (0, 5)!
Terminator has 0 kills.
BlueThunder fires at (8, 7) but misses!
BlueThunder has 0 kills.
. . . . . T . . . .
. . . . . .
. . . . . .
R . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . B . .
. . . . . .
. . . . . .
```

```
Turn 174:
RoboCop looks around at (5, 9)
RoboCop moves to (5, 8)
RoboCop tries to fire at (7, 11) but it's out of range!
RoboCop fires at (3, 8) but misses!
RoboCop fires at (3, 9) and hits Terminator!
Terminator has 0 lives left!
Terminator has been removed from the battlefield.
RoboCop has 6 kills.
RoboCop upgrades to TerminatorRoboCop!
Terminator is not activated!
BlueThunder is not activated!
Terminator and BlueThunder have been eliminated.
RoboCop won!
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . R
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

IMPLEMENTATION OF OOP FEATURES

BlueThunder:

```
#include "MoveRobot.h"
#include "ShootRobot.h"
#include "StepRobot.h"
#include "LookRobot.h"
#include "Battlefield.h"

using namespace std;

class BlueThunder : public ShootRobot, public LookRobot {
    Battlefield* battlefield;
    int firingDirection; // Tracks the direction for firing (0: Up, 1: Right, ...)
public:
    BlueThunder(string n, int x, int y, Battlefield* bf)
        :Robot(n, x, y), battlefield(bf), firingDirection(0), ShootRobot(n, x, y), LookRobot(n, x, y) {
            bf->addRobot(this);
        }
```

inheritance:

- class BlueThunder : public ShootRobot, public LookRobot
- BlueThunder(string n, int x, int y, Battlefield* bf)

MadBot:

```
#include "ShootRobot.h"
#include "LookRobot.h"
#include "Battlefield.h"
#include "RoboTank.h"

class MadBot : public ShootRobot, public LookRobot {
    Battlefield* battlefield;
public:
    MadBot(string n, int x, int y, Battlefield* bf)
        : Robot(n, x, y), ShootRobot(n, x, y), LookRobot(n, x, y), battlefield(bf) {
            bf->addRobot(this);
        }
```

inheritance:

- class MadBot : public ShootRobot, public LookRobot
- BlueThunder(string n, int x, int y, Battlefield* bf)

RoboCop:

```
#include "MoveRobot.h"
#include "ShootRobot.h"
#include "StepRobot.h"
#include "LookRobot.h"
#include "Battlefield.h"
#include "TerminatorRoboCop.h"

class RoboCop : public MoveRobot, public ShootRobot, public LookRobot {
    Battlefield* battlefield;
public:
    RoboCop(string n, int x, int y, Battlefield* bf)
        : Robot(n, x, y), MoveRobot(n, x, y, bf), ShootRobot(n, x, y), LookRobot(n, x, y), battlefield(bf) {
            bf->addRobot(this);
        }
```

inheritance:
-class RoboCop : public MoveRobot, public ShootRobot, public LookRobot
-RoboCop(string n, int x, int y, Battlefield* bf

Robot.h:

```
public:
    Robot(string n, int x, int y) : name(n), posX(x), posY(y), lives(3), kills(0), deathCount(0), active(true) {}
    virtual ~Robot() {}
```

```
virtual void look() = 0;
virtual void move() = 0;
virtual void fire() = 0;
virtual void stepping() = 0;
virtual void action() = 0;
```

Virtual Destructor:
- virtual ~Robot()

Virtual Methods:
- virtual void look() = 0
- virtual void move() = 0
- virtual void fire() = 0
- virtual void stepping() = 0
- virtual void action() = 0

RoboTank:

```
class RoboTank : public ShootRobot, public LookRobot {
    Battlefield* battlefield;
public:
    RoboTank(string n, int x, int y, Battlefield* bf)
        : Robot(n, x, y), ShootRobot(n, x, y), LookRobot(n, x, y), battlefield(bf) {
            bf->addRobot(this);
        }
```

inheritance:

- class RoboTank : public ShootRobot, public LookRobot
- RoboTank(string n, int x, int y, Battlefield* bf)

ShootRobot:

```
#include "Robot.h"

class ShootRobot : virtual public Robot {
public:
    ShootRobot(string n, int x, int y) : Robot(n, x, y) {}
    void fire() override {
        int targetX = posX + (rand() % 3 - 1);
        int targetY = posY + (rand() % 3 - 1);
        cout << name << " fires at (" << targetX << ", " << targetY << ")\n";
    }
```

inheritance:

-class ShootRobot : virtual public Robot
-ShootRobot(string n, int x, int y) : Robot (n, x, y)

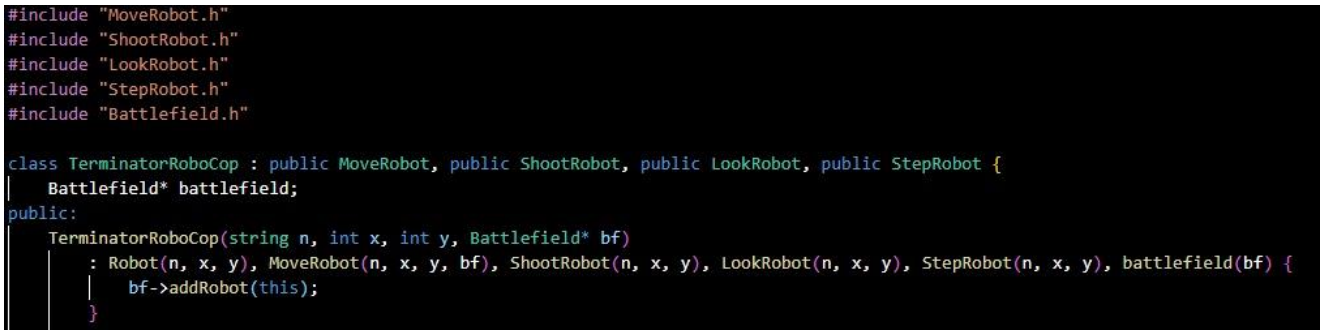
Terminator:

```
#include "MoveRobot.h"
#include "StepRobot.h"
#include "LookRobot.h"
#include "Battlefield.h"
#include "TerminatorRoboCop.h"

class Terminator : public MoveRobot, public StepRobot, public LookRobot {
    Battlefield* battlefield;
public:
    Terminator(string n, int x, int y, Battlefield* bf)
        : Robot(n, x, y), MoveRobot(n, x, y, bf), StepRobot(n, x, y), LookRobot(n, x, y), battlefield(bf) {
            bf->addRobot(this);
        }
```

-class Terminator : public MoveRobot, public StepRobot, public LookRobot
-Terimator(string n, int x, int y, Battlefield* bf)

TerminatorRoboCop:



```
#include "MoveRobot.h"
#include "ShootRobot.h"
#include "LookRobot.h"
#include "StepRobot.h"
#include "Battlefield.h"

class TerminatorRoboCop : public MoveRobot, public ShootRobot, public LookRobot, public StepRobot {
    Battlefield* battlefield;
public:
    TerminatorRoboCop(string n, int x, int y, Battlefield* bf)
        : Robot(n, x, y), MoveRobot(n, x, y, bf), ShootRobot(n, x, y), LookRobot(n, x, y), StepRobot(n, x, y), battlefield(bf) {
        bf->addRobot(this);
    }
```

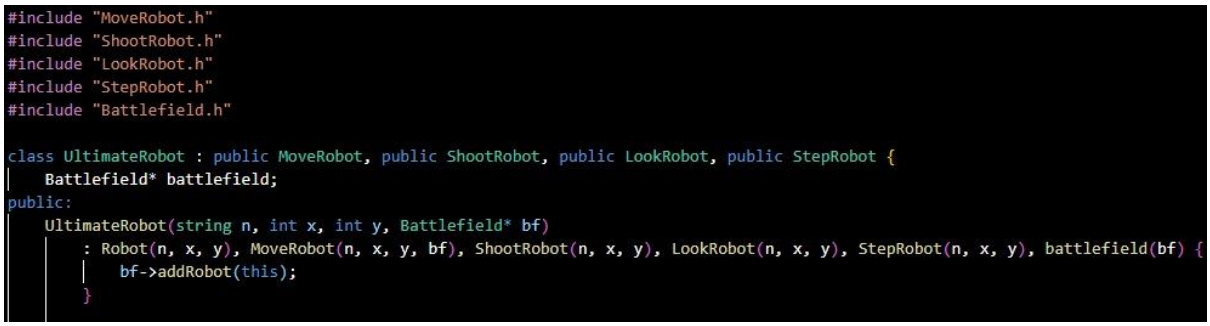
inheritance:

-class TerminatorRoboCop : public MoveRobot, public StepRobot, public LookRobot, public StepRobot
-TerimatorRoboCop(string n, int x, int y, Battlefield* bf)

inheritance:

-class TerminatorRoboCop : public MoveRobot, public StepRobot, public LookRobot, public StepRobot
-TerimatorRoboCop(string n, int x, int y, Battlefield* bf)

UltimateRobot:



```
#include "MoveRobot.h"
#include "ShootRobot.h"
#include "LookRobot.h"
#include "StepRobot.h"
#include "Battlefield.h"

class UltimateRobot : public MoveRobot, public ShootRobot, public LookRobot, public StepRobot {
    Battlefield* battlefield;
public:
    UltimateRobot(string n, int x, int y, Battlefield* bf)
        : Robot(n, x, y), MoveRobot(n, x, y, bf), ShootRobot(n, x, y), LookRobot(n, x, y), StepRobot(n, x, y), battlefield(bf) {
        bf->addRobot(this);
    }
```

inheritance:
-class UltimateRoboot : public MoveRobot, public ShootRobot, public LookRobot, public StepRobot
-UltimateRoobot(string n, int x, int y, Battlefield* bf)