# Title page

**Multimedia University**

**CCP6124 Object Oriented Programming and Data Structures 2510**

**Lecture section: TC2L**

**Tutorial section: TT5L**

**Group number: G11**

**Group leader student name: Chong Meng Hang**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num** | **Student**  **ID** | **Student Name by**  **alphabetical order** | **Task Descriptions** | **Percentage (100%)** |
| 1 | 242UC244JT | Chong Meng Hang | FUNCTIONS : starting base struture, printing battlefield, print only current robot, get set and clear cells, getter and setter fucntions  code logic : upgrade class, pointer related, turn-based  class : scoutbot, trackbot, longshotbot  Extra task: debugging | 25 |
| 2 | 242UC244PF | Tan Yong Xin | FUNCTIONS: actionLook  Code logic: let robot check 3x3 area around itself, show [R] for itself, [ ] for empty, [E] for enemy, log everything to file  Class: HealthBot, GodBot  Extra task: UML design | 25 |
| 3 | 242UC244S6 | Tham Mei Ting | FUNCTIONS: actionMove, handle random jump coordinate  Code logic: jump limit check using jumpCount and MAXjump, turnCounter to control when JukeBot teleport  Class: JumpBot, JukeBot  Extra task: UML design | 25 |
| 4 | 242UC244KX | Yap Chi Yi | FUNCTIONS : readfile, placeRobots, updateWaitingRobots, returnbattlefield\_, waitingrobots, destroyedrobots, returnrobots,  code logic : actionShoot 's general shooting logic( not upgrading ) , outputting to a txt file , game ending requirements  class : thirtyshotbot, semiautobot, hidebot  extra task : change code to non user input, debugging upgrade class | 25 |

# 

# 

**Assignment Evaluation Criteria**

# Assignment Programming, Assignment Report (40m)

You are required to present your assignment in group to your respective tutor.

Also report and document all your assignment tasks with this marking table that contain cover page, table of contents, page numbering, inputs, outputs, figures, screenshots, explanations, descriptions, classes design, and others.

|  |  |  |
| --- | --- | --- |
| Criteria | Mark | Max |
| Q1. Design documentation or -40  [] UML classes diagram, explanations  [] screenshots, screen OUTPUT, figures, explanations  [] two sample input files, two sample txt log output files, correct, explanations  [] code segments, pseudocodes, explanations  [] great | ? | 5 |
| Q2. Initialization of a simulation, explanations  [] read input files  [] screen outputs initial battlefield of all robots  [] fixed seed integer for random functions | ? | 3 |
| Q3. OUTPUT and logging of the status of the battlefield of robots at each turn, explanations  [] OUTPUT battlefields to screen  [] battlefields in txt log output files  battlefield two end games  [] fileInput2a.txt: left one robot  [] fileInput2b.txt: reached max number of turns  [] great  Q4. OUTPUT and logging of the actions and the status of each robot at each turn, explanations  ()()() accept status of robots & screen OUTPUT & has txt log output files, good, great | ?  ? | 5  3 |
| Q5. Implementation of the 7 additional required robot classes with OOP concepts i.e. I&P&OO +3, explanations  Seeing: []ScoutBot []TrackBot  Shooting: []LongShotBot []SemiAutoBot []ThirtyShotBot  Moving: []HideBot []JumpBot  []inheritance []polymorphism []operator overloading | ? | 10 |
| Q6. The algorithms used to optimize the actions of robots listed in the assignment document, explanations  () vector or queue  () vector and queue  ()()() accept algorithms & has explanations, good, great | ? | 5 |
| Q7. Implementation of three new robot classes  (3 marks for each robot), explanations  New robot name 1: GodBot  New robot name 2: JukeBot  New robot name 3: HealthBot  ()()() accept & has explanations, good, great | ? | 9 |
|  |  |  |
| Total |  | 40 |

Additional comments

|  |
| --- |
|  |

You are required to fill in your task percentage and task descriptions.  
 Every student is responsible for 100% (task percentage) of this group assignment work.

Student 1

|  |  |
| --- | --- |
| Student ID | 242UC244JT |
| Student name | CHONG MENG HANG |
| Task percentage | 25 |
| Task descriptions | FUNCTIONS : starting base struture, printing battlefield, print only current robot, get set and clear cells, getter and setter fucntions  code logic : upgrade class, pointer related, turn-based  class : scoutbot, trackbot, longshotbot |
| Total score (40m) | ? |

Student 2

|  |  |
| --- | --- |
| Student ID | 242UC244PF |
| Student name | TAN YONG XIN |
| Task percentage | 25 |
| Task descriptions | FUNCTIONS: actionLook  Code logic: let robot check 3x3 area around itself, show [R] for itself, [ ] for empty, [E] for enemy, log everything to file  Class: HealthBot, GodBot  Extra task: UML design |
| Total score (40m) | ? |

Student 3

|  |  |
| --- | --- |
| Student ID | 242UC244S6 |
| Student name | THAM MEI TING |
| Task percentage | 25 |
| Task descriptions | FUNCTIONS: actionMove, handle random jump coordinate  Code logic: jump limit check using jumpCount and MAXjump, turnCounter to control when JukeBot teleport  Class: JumpBot, JukeBot |
| Total score (40m) | ? |

Student 4

|  |  |
| --- | --- |
| Student ID | 242UC244KX |
| Student name | YAP CHI YI |
| Task percentage | 25 |
| Task descriptions | FUNCTIONS : readfile, placeRobots, updateWaitingRobots, returnbattlefield\_, waitingrobots, destroyedrobots, returnrobots,  code logic : actionShoot 's general shooting logic( not upgrading ) , outputting to a txt file , game ending requirements  class : thirtyshotbot, semiautobot, hidebot  extra task : change code to non user input, debugging upgrade class |
| Total score (40m) | ? |

Each feature will be evaluated based on fulfilment of requirements, correctness, compilation without warnings and errors, error free during runtime, basic error handling, quality of comments, user friendliness, good coding format and style.

**Table of contents with page numbers and links**

[Title page 1](#_Toc197344085)

[Assignment Programming, Assignment Report (40m) 2](#_Toc197344086)

[Delete this information section 6](#_Toc197344087)

[Questions Section 7](#_Toc197344088)

[Q1. [5] Design documentation or -40, UML classes diagram, explanations, screenshots, screen display, figures, explanations, two sample input files, two sample txt log output files, correct, explanations, code segments, pseudocodes, explanations 7](#_Toc197344089)

[Q2. [3] Initialization of a simulation, read input file, screen output initial battlefield of all robots, fixed seed integer for random functions 10](#_Toc197344090)

[Q3. [5] Display and logging of the status of the battlefield of robots at each turn 11](#_Toc197344091)

[Q4. [3] Display and logging of the robot actions and the status of each robot at each turn 11](#_Toc197344092)

[Q5. [10] Implementation of the 7 additional required robot classes with OOP concepts i.e. I&P&OO +3 14](#_Toc197344093)

[Q6. [5] The algorithms used to optimize the actions of robots, vector, queue 23](#_Toc197344094)

[Q7. [9] Implementation of three new robot classes 25](#_Toc197344095)

[Question: How to compile, how to run the program 28](#_Toc197344096)

# 

# Delete this information section

Insert the comment below at the beginning of your source code files:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Program: YOUR\_FILENAME.cpp

// Course: CCP6124 OOPDS

// Lecture Class: TC2L

// Tutorial Class: ?TT5L

// Trimester: 2510

// Member\_1: ID | NAME | EMAIL | PHONE

// Member\_2: ID | NAME | EMAIL | PHONE

// Member\_3: ID | NAME | EMAIL | PHONE

// Member\_4: ID | NAME | EMAIL | PHONE

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Task Distribution

// Member\_1:

// Member\_2:

// Member\_3:

// Member\_4:

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Filename examples

{

TC2L\_G04\_main.cpp

fileInput1.txt, fileInput2.txt, fileOutput1.txt, fileOutput2.txt  
TC2L\_G04\_report.docx

}

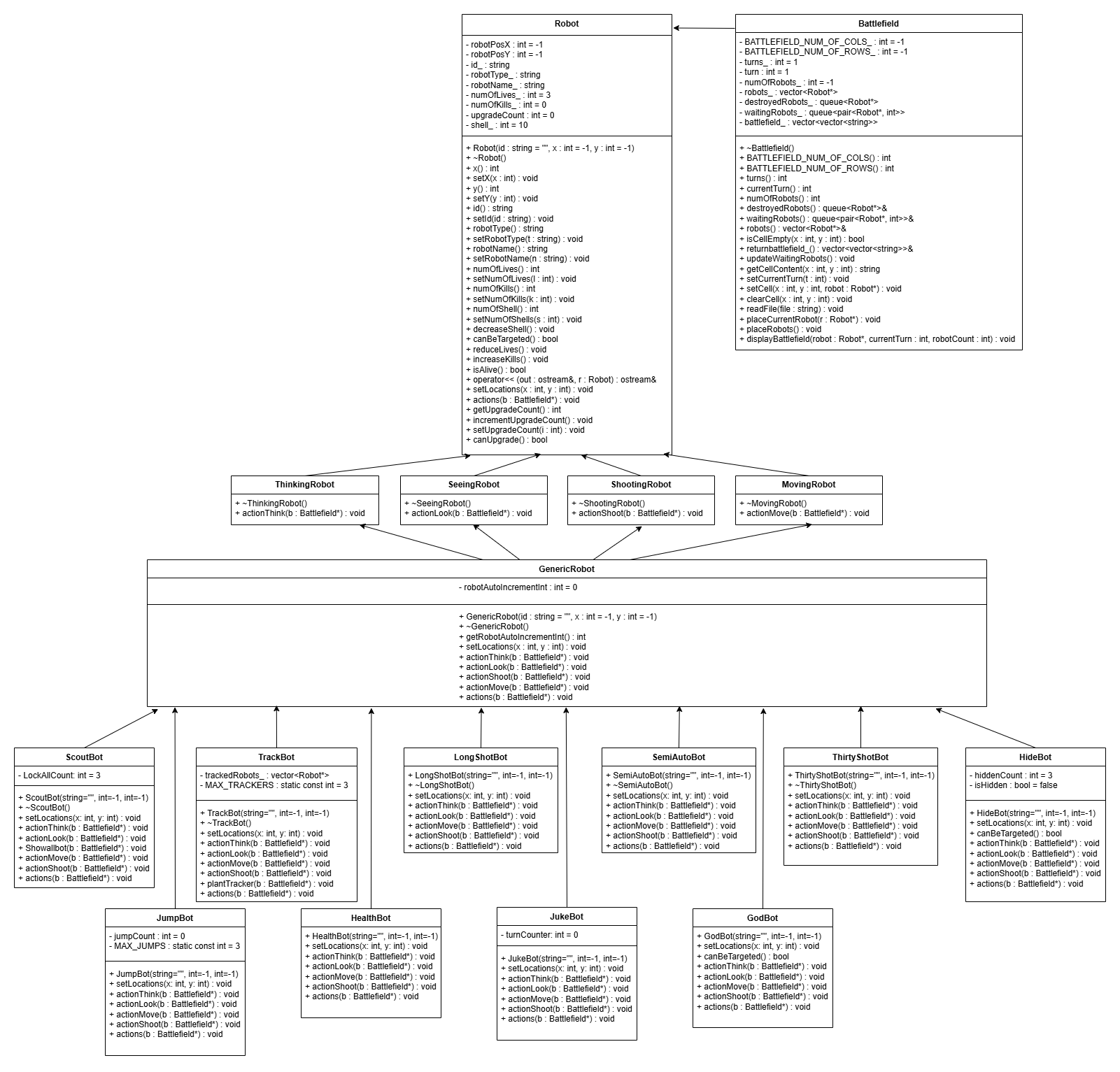
# Questions Section

# Q1. [5] Design documentation or -40, UML classes diagram, explanations, screenshots, screen OUTPUT, figures, explanations, two sample input files, two sample txt log output files, correct, explanations, code segments, pseudocodes, explanations

UML classes diagram, explanations

Screenshots (inputs, outputs), text screen OUTPUT, figures, explanations

UML Class Diagram



Class Explanations

1. Robot (Base Class)

Attributes:

* robotPosX, robotPosY (position)
* id\_, robotType\_, robotName\_ (identification)
* numofLives\_, numOfKills\_, shell\_ (combat stats)
* UpgradeCount (upgrade tracking)

Methods:

* setLocations(), actions() (pure virtual)
* reduceLives(), increaseKills(), isAlive()

1. ThinkingRobot, SeeingRobot, ShootingRobot, MovingRobot (Interfaces)

* Define role-specific actions (actionThink(), actionLook(), etc.)
* Inherited by concrete robot types.

1. GenericRobot (Basic Robot)

* Default implementation of all actions.
* Can be upgraded into specialized robots.

1. Can be upgraded into specialized robots.

* Each has unique abilities:
* ScoutBot: Can reveal the entire battlefield.
* TrackBot: Can track enemy movements.
* LongShotBot: Can shoot at long range.
* HideBot: Can temporarily become invisible.

UML Code segments, pseudocode parts, explanations

Two sample input files, two sample txt log output files, correct, explanations

Sample 1 file input

file input name: fileInput1.txt

/////

M by N: 10 15

turns: 300

robots: 5

GenericRobot GR01\_Kidd 3 6

GenericRobot GR02\_Jet 1 1

GenericRobot GR03\_Alpha 5 2

GenericRobot GR04\_Beta 2 7

GenericRobot GR05\_Star random random

/////

Sample 1 file output  
filename: fileOutput1.txt

step by step screenshot outputs, output file, explanations

/////

///

Sample 2 file input

file input name: fileInput2.txt

/////

M by N: 5 5

turns: 300

robots: 5

GenericRobot GR01\_Kidd 3 4

GenericRobot GR02\_Jet 1 1

GenericRobot GR03\_Alpha 4 2

GenericRobot GR04\_Beta 3 3

GenericRobot GR05\_Star random random

/////

Sample 2 file output  
filename: fileOutput2.txt

step by step screenshot outputs, output file, explanations

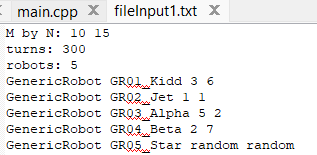
/////

?

/////

# Q2. [3] Initialization of a simulation, read input file, screen output initial battlefield of all robots, fixed seed integer for random functions

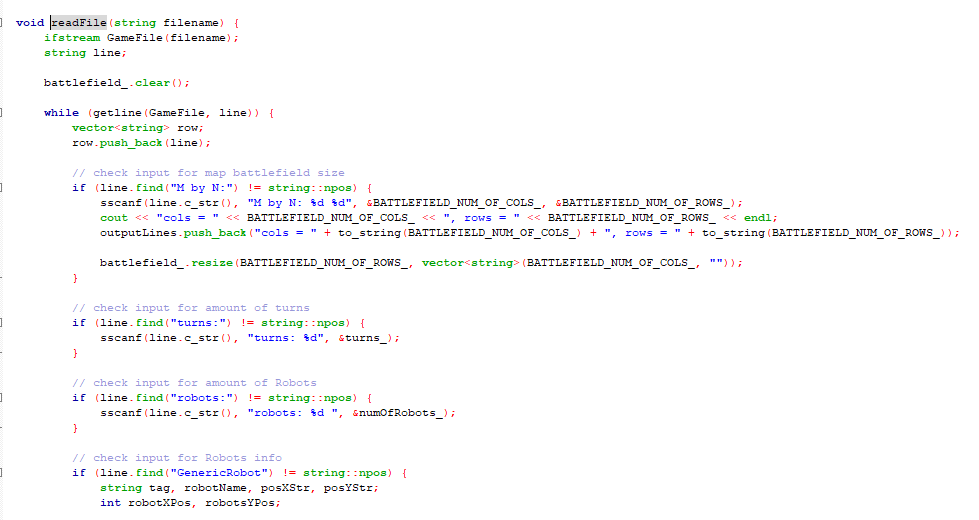
Screenshots (inputs, outputs), text screen OUTPUT, figures, explanations

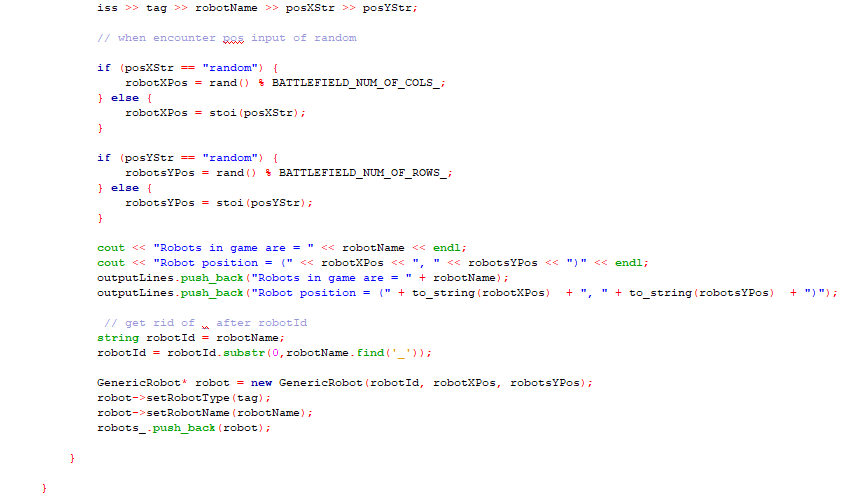


A screen shot of a computer

AI-generated content may be incorrect.

Code segments, initial of a simulation, read input file, fixed seed integer right at first line in the main function, pseudocode parts, explanations





* Read in file, and check line using .find() to find string that correlates to their assigning value, and c string style function sscanf to extract data and push into their assigned data



* Srand(id/1000) for fixed seed integer

# Q3. [5] OUTPUT and logging of the status of the battlefield of robots at each turn

a. OUTPUT battlefields of robots at each turn to screen

Screenshots (inputs, outputs), text screen OUTPUT, figures, explanations

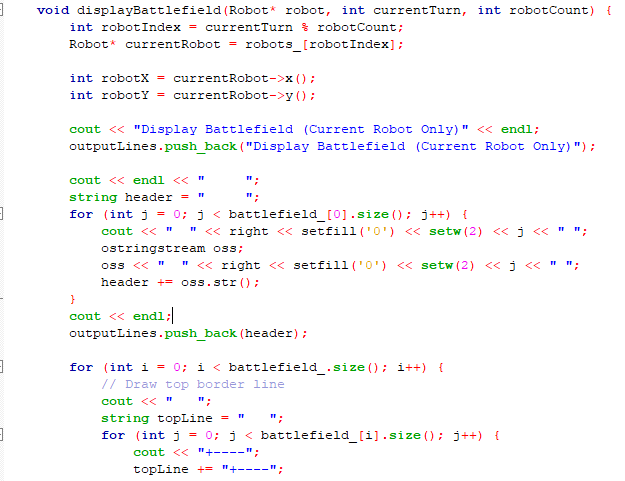
A grid of lines with white dots

AI-generated content may be incorrect.

A computer screen shot of a robot

AI-generated content may be incorrect.

Code segments, fixed seed integer right at first line in the main function, pseudocode parts, explanations



DisplayBattlefield first takes in the a pointer to robots, the currentTurn and how many robots are there. It find the X and Y positions of said robots and then prints out the battlefield for that current robot only, all the rrobots are actually on the same grid and some bots later down the upgrades path can see all the robots on the battlefield, they are just hidden as default.

PSUEDOCODE:

FUNCTION displayBattlefield(robot, currentTurn, robotCount):  
  
 robotIndex = currentTurn MOD robotCount  
 currentRobot = robots\_[robotIndex]  
   
 robotX = currentRobot's X position  
 robotY = currentRobot's Y position  
   
 PRINT "Display Battlefield (Current Robot Only)"  
 ADD message to outputLines  
   
  
 PRINT " " (5 spaces for row labels)  
 header = " "  
   
 FOR each column j in battlefield:  
 PRINT formatted column number (e.g., " 02 ")  
 ADD to header string  
 PRINT newline  
 ADD header to outputLines  
   
   
 FOR each row i in battlefield:  
 // Draw top border (+----+----+...)  
 PRINT " " (3 spaces for alignment)  
 topLine = " "  
   
 FOR each column in battlefield:  
 PRINT "+----"  
 ADD "+----" to topLine  
 PRINT "+"  
 ADD "+" to topLine  
 ADD topLine to outputLines  
   
   
 PRINT formatted row number (e.g., " 02")  
 rowLine = formatted row number  
   
 FOR each column j in battlefield:  
 IF current cell (i,j) is robot's position:  
 PRINT "|" + robot's ID (left-aligned, 4 chars)  
 ADD "|" + ID to rowLine  
 ELSE:  
 PRINT "| " (empty cell)  
 ADD "| " to rowLine  
 PRINT "|"  
 ADD "|" to rowLine  
 ADD rowLine to outputLines  
   
   
 PRINT " " (3 spaces for alignment)  
 bottomLine = " "  
   
 FOR each column in battlefield:  
 PRINT "+----"  
 ADD "+----" to bottomLine  
 PRINT "+"  
 ADD "+" to bottomLine  
 ADD bottomLine to outputLines

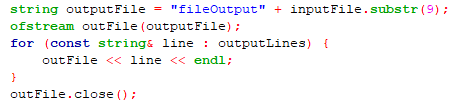
b. Battlefields of robots at each turn in txt log output file

Screenshots (inputs, outputs), text screen OUTPUT, figures, explanations

A screenshot of a computer

AI-generated content may be incorrect.

Code segments, fixed seed integer right at first line in the main function, pseudocode parts, explanations



PSUEDOCODE:

FUNCTION WriteOutputToFile(inputFile, outputLines):  
 // Generate output filename by replacing "fileInput" with "fileOutput"  
 outputFile = "fileOutput" + inputFile.substring(starting from index 9)  
   
 // Open the output file for writing  
 outFile = open outputFile for writing  
   
 // Write all accumulated output lines  
 FOR EACH line IN outputLines:  
 WRITE line to outFile  
 WRITE newline to outFile  
   
 // Close the file  
 CLOSE outFile

# Q4. [3] OUTPUT and logging of the robot actions and the status of each robot at each turn

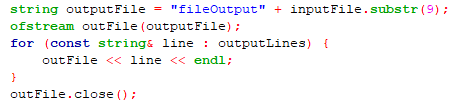
c. OUTPUT and logging of the actions and the status of each robot at each turn

Screenshots (inputs, outputs), text screen OUTPUT, figures, explanations

A screenshot of a computer program

AI-generated content may be incorrect.

Code segments, fixed seed integer right at first line in the main function, pseudocode parts, explanations



PSUEDOCODE:

FUNCTION WriteOutputToFile(inputFile, outputLines):  
 // Generate output filename by replacing "fileInput" with "fileOutput"  
 outputFile = "fileOutput" + inputFile.substring(starting from index 9)  
   
 // Open the output file for writing  
 outFile = open outputFile for writing  
   
 // Write all accumulated output lines  
 FOR EACH line IN outputLines:  
 WRITE line to outFile  
 WRITE newline to outFile  
   
 // Close the file  
 CLOSE outFile

d. Battlefield two end games

fileInput2a.txt: left one robot

{

Screenshots (inputs, outputs), text screen OUTPUT, figures, explanations

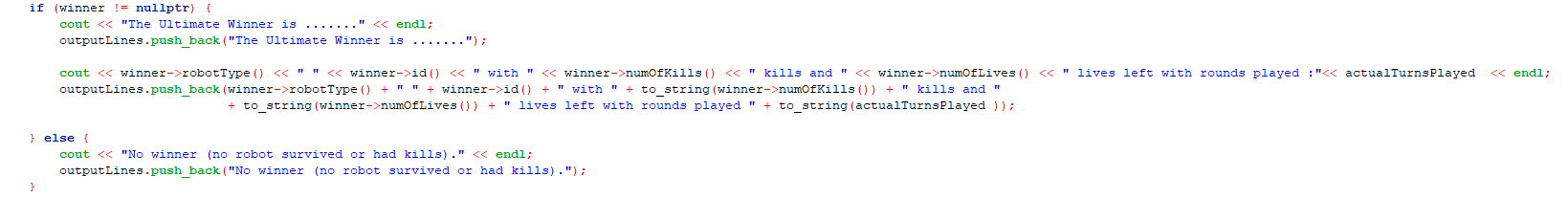
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Code segments, fixed seed integer right at first line in the main function, pseudocode parts, explanations



PSUEDOCOEDE:

FUNCTION AnnounceWinner(winner, actualTurnsPlayed, outputLines):  
 IF winner is not null:  
 PRINT "The Ultimate Winner is ......."  
 ADD "The Ultimate Winner is ......." to outputLines  
  
 winnerStats = winner's type + " " + winner's ID +   
 " with " + winner's kill count + " kills and " +  
 winner's remaining lives + " lives left" +  
 " with rounds played: " + actualTurnsPlayed  
  
 PRINT winnerStats  
 ADD winnerStats to outputLines  
 ELSE:  
 PRINT "No winner (no robot survived or had kills)"  
 ADD "No winner (no robot survived or had kills)" to outputLines

e. Battlefield two end games

fileInput2b.txt: reached max number of turns

{

Screenshots (inputs, outputs), text screen OUTPUT, figures, explanations

A screenshot of a computer

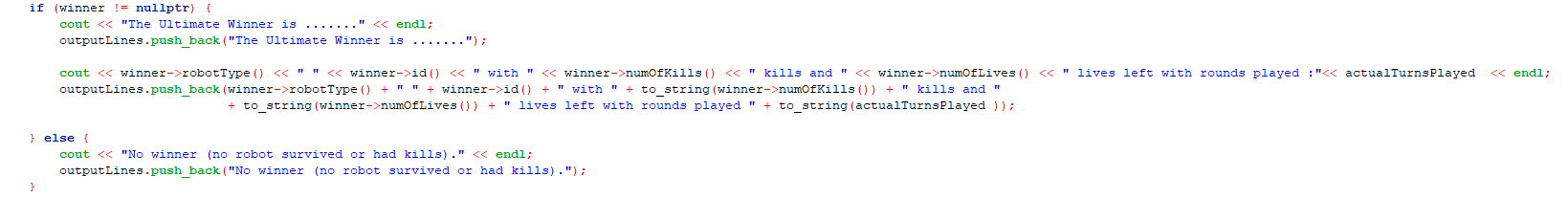
AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

* HB02 won with the most kills and 1 life left after all 100 rounds played.

Code segments, fixed seed integer right at first line in the main function, pseudocode parts, explanations



PSUEDOCODE:

FUNCTION AnnounceWinner(winner, actualTurnsPlayed, outputLines):  
 IF winner is not null:  
 PRINT "The Ultimate Winner is ......."  
 ADD "The Ultimate Winner is ......." to outputLines  
  
 winnerStats = winner's type + " " + winner's ID +   
 " with " + winner's kill count + " kills and " +  
 winner's remaining lives + " lives left" +  
 " with rounds played: " + actualTurnsPlayed  
  
 PRINT winnerStats  
 ADD winnerStats to outputLines  
 ELSE:  
 PRINT "No winner (no robot survived or had kills)"  
 ADD "No winner (no robot survived or had kills)" to outputLines

# Q5. [10] Implementation of the 7 additional required robot classes with OOP concepts i.e. I&P&OO +3

GenericRobot

- actionThink(): random order either actionFire() actionMove() or actionMove() actionFire()

- actionLook(): looks 8 neighbors from its current position

- actionFire(): fires at one random of its 8 neighboring locations, it has only 10 shells to fire in a match, if a robot uses up all its shells then it will self-destruct

- actionMove(): moves to one of its 8 neighboring locations or remain in place

A robot can change only to one robot upgrade after killed an enemy robot.

Screenshots (inputs, outputs), figures, explanations

A screenshot of a computer program

AI-generated content may be incorrect.

Code segments, pseudocode parts, explanations

ActionThink was suppose to be where the sequence of actions is dertermined, but the reason ours just prints 1 line is because

1. We have a separate function that already does it
2. Only 1 sequence is allowed (think look move shoot), the reason (think look shoot move) wouldn't work is because the bot would change classes by the time it moves and its hard to keep track for that bot to move while its changing classes

A screenshot of a computer program

AI-generated content may be incorrect.

PSEUDOCODE:

FUNCTION actionThink  
 PRINT generic robot is thinking...

ADD “generic robot is thinking...” to outputLines

ActionLook allows the bot to look at the neighbouring 8 squares by printing the 3 x 3 grid, if there is a bot it will print the first initial of that bot, if its out of bound it will print a #.

A computer code with text

AI-generated content may be incorrect.

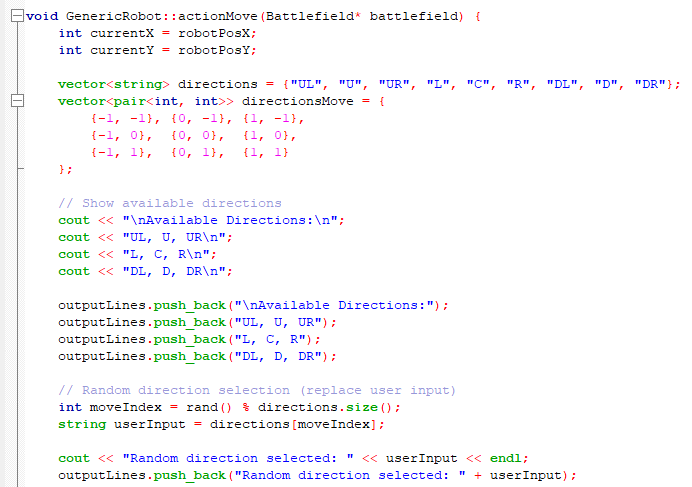
A screen shot of a computer code

AI-generated content may be incorrect.

PSEUDOCODE :  
FUNCTION actionLook(battlefield)

currentX = robot's X position  
 currentY = robot's Y position  
   
 PRINT "=== Robot View (3x3 Grid) ==="  
 ADD "=== Robot View (3x3 Grid) ===" to outputLines  
   
 FOR (int dy = -1; dy <= 1; ++dy):  
 row = ""  
 FOR (int dx = -1; dx <= 1; ++dx):  
 x = currentX + dx  
 y = currentY + dy  
   
 IF position (x,y) is within battlefield boundaries:  
 IF position is robot's current position:  
 PRINT "[R] "  
 ADD "[R] " to row  
 ELSE:  
 content = get cell content at (x,y)  
 IF content is empty:  
 PRINT "[ ] "  
 ADD "[ ] " to row  
 ELSE:  
 PRINT first character of content in brackets  
 ADD first character of content in brackets to row  
 ELSE:  
 PRINT "[#] " (out of bounds)  
 ADD "[#] " to row  
  
 PRINT newline  
 ADD row to outputLines

ActionMove it records the current robots location and randomly choses a location to move to in the 8 neigbouring squares, if it picked a spot out of bounds or a robot is occupying it, it will not count.



A computer screen shot of a program code

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A computer code with text

AI-generated content may be incorrect.

PSEUDOCODE :

FUNCTION actionLook

INT currentX = robotPosX;

INT currentY = robotPosY;

VECTOR directions = {"UL", "U", "UR", "L", "C", "R", "DL", "D", "DR"}

VECTOR PAIR directionsMove = {  
 {-1, -1}, {0, -1}, {1, -1},  
 {-1, 0}, {0, 0}, {1, 0},  
 {-1, 1}, {0, 1}, {1, 1}

PRINT "\nAvailable Directions:\n"  
 PRINT"UL, U, UR\n L, C, R\n DL, D, DR\n”  
  
 ADD ("\nAvailable Directions:") to outputLines  
 ADD ("UL, U, UR\n L, C, R\n DL, D, DR") to outputLines  
  
 INT moveIndex = rand() % directions.size();  
 STRING userInput = directions[moveIndex];  
  
 PRINT "Random direction selected: " << userInput << endl  
 ADD ("Random direction selected: " + userInput) to outputLines  
  
 BOOL validInput = false;  
 FOR (int i = 0; i < directions.size(); i++) {  
 IF (directions[i] == userInput) {  
 moveIndex = i;  
 validInput = true;  
 BREAK;  
  
 If (!validInput)   
 PRINT “nvalid move direction\n"  
 ADD("Invalid move direction") to outputLines

RETURN  
  
 INT newX = currentX  
 INT newY = currentY  
  
 IF (userInput != "C") {  
 newX += directionsMove[moveIndex].first  
 newY += directionsMove[moveIndex].second  
  
 IF (newX < 0 | | newX > battlefield columns   
 PRINT "Cannot move outside battlefield\n"  
 PRINT("Cannot move outside battlefield") to outputLines  
 RETURN  
  
 FOR (Robot\* robot : battlefield->robots())  
 INT PotentialRobotX = robot->x();  
 INT PotentialRobotY = robot->y();  
  
 IF(newX == PotentialRobotX && newY == PotentialRobotY)  
 PRINT"Destination occupied"  
 ADD("Destination occupied") to outputLines

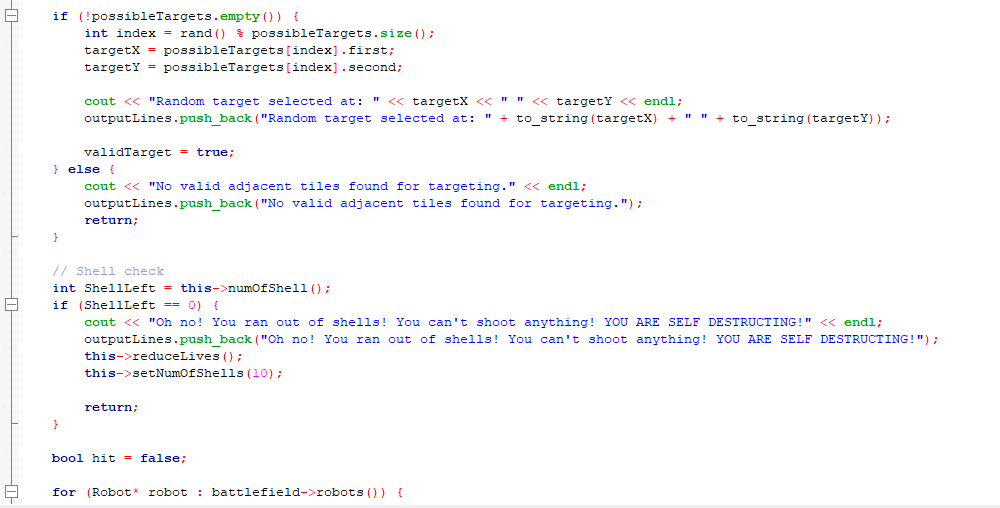
RETURN

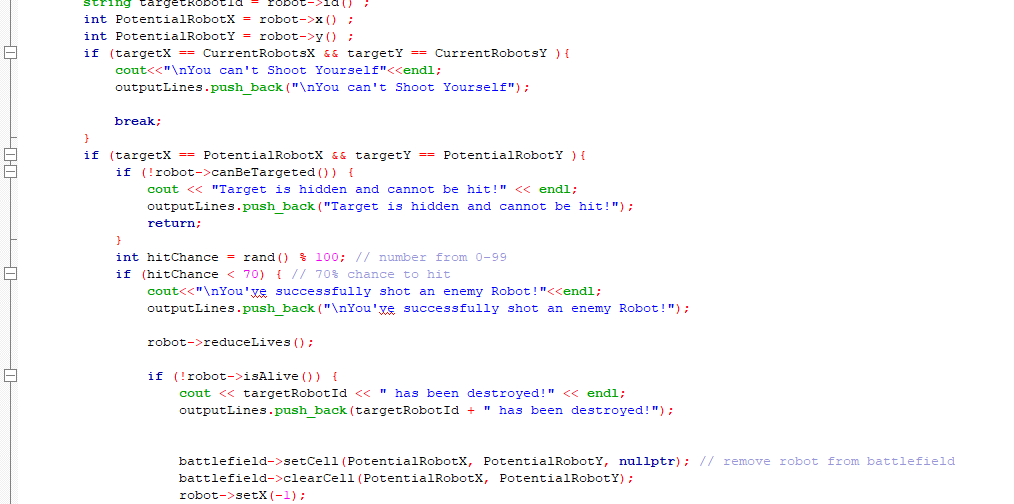
robotPosX = newX;  
 robotPosY = newY;  
  
 set old cell to null pointer  
 set new cell with this robot and new X and Y  
  
 PRINT"Moved to (" << newX << "," << newY << ")"  
 ADD("Moved to (" + to\_string(newX) + "," + to\_string(newY) + ")") to outputLines

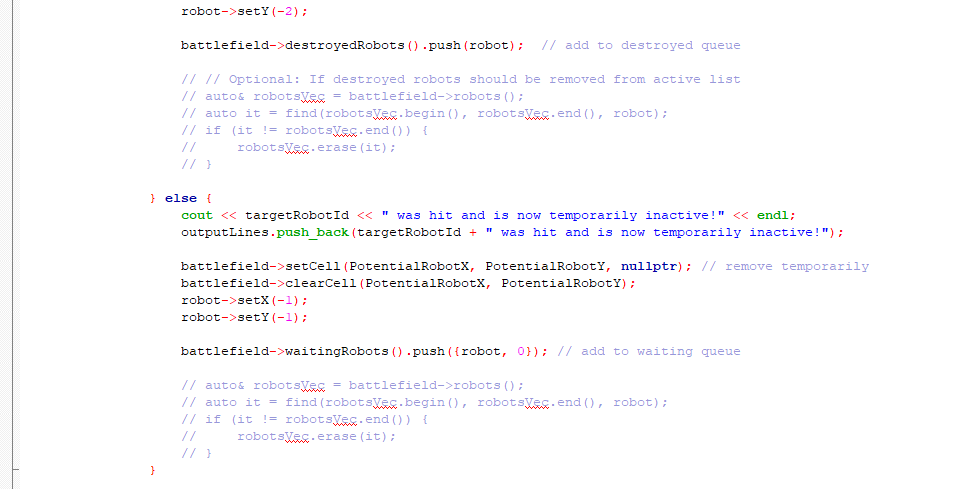
ActionShoot lets a robot randomly shoot a shot at a neighbouring 8 squares , if it hits a taget robot, it will be able to upgrade and randomly picks the upgrade. But if the robot runs out of shells it will self destruct.

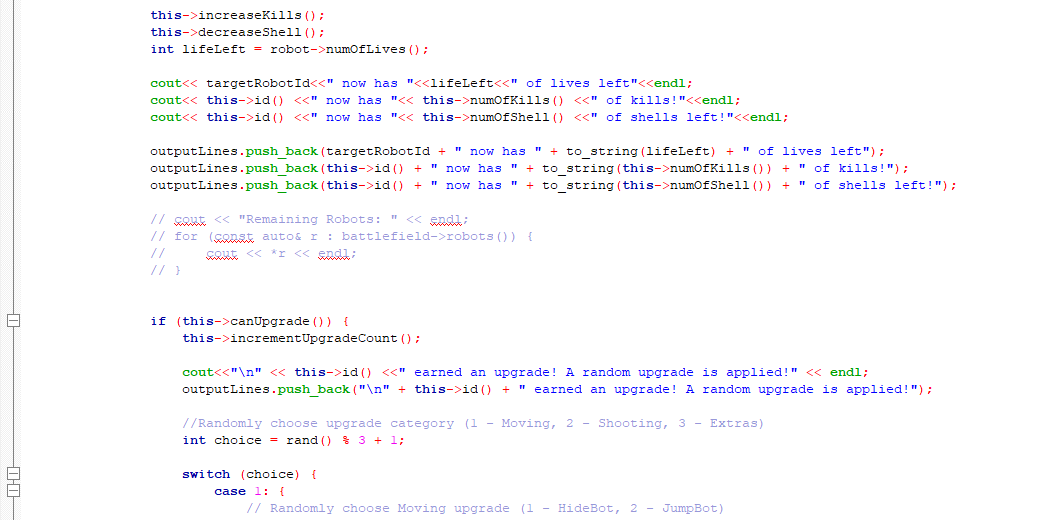
A screenshot of a computer code

AI-generated content may be incorrect.



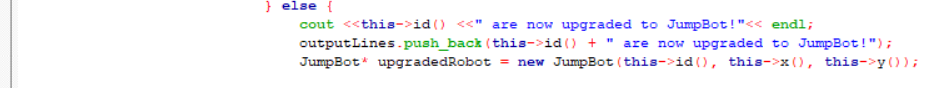




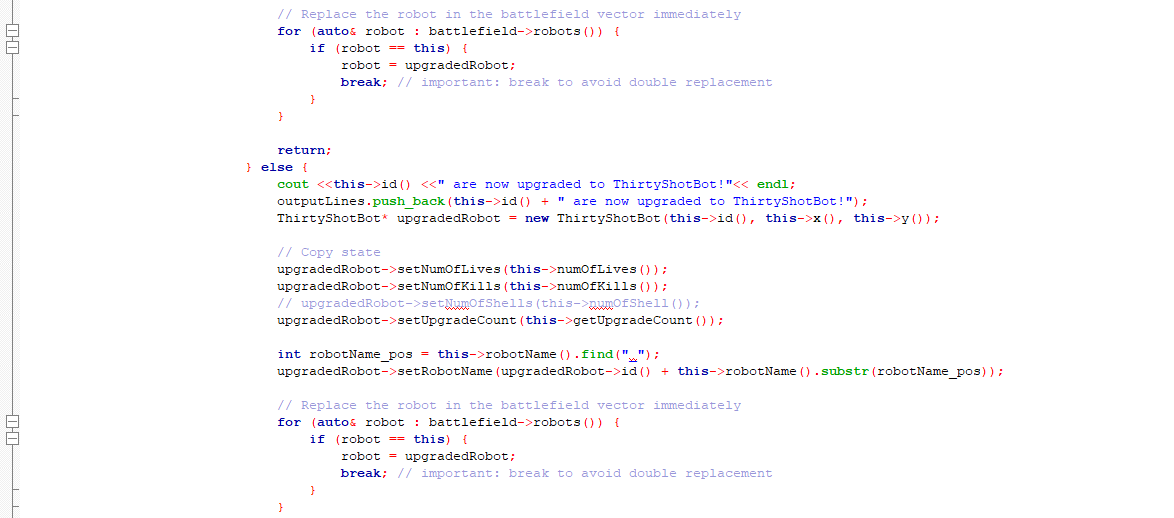


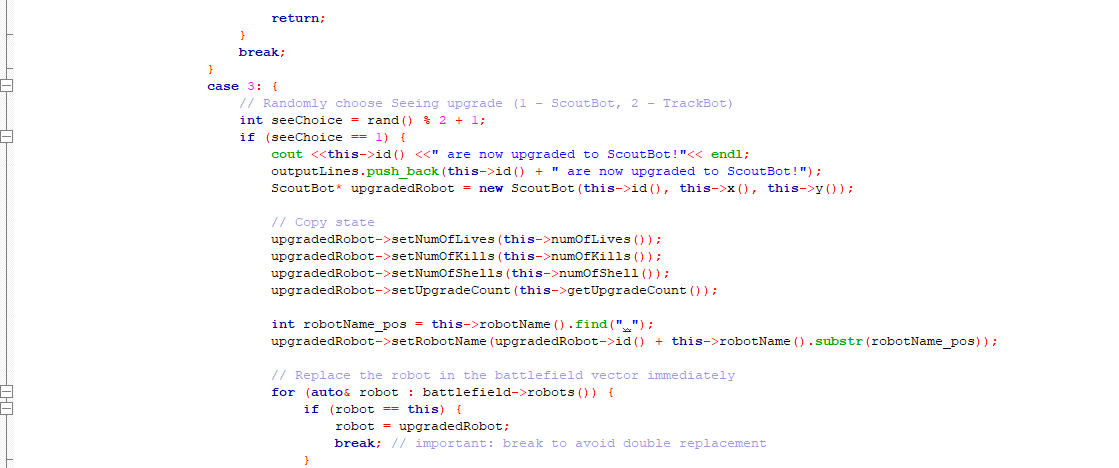
A screenshot of a computer code

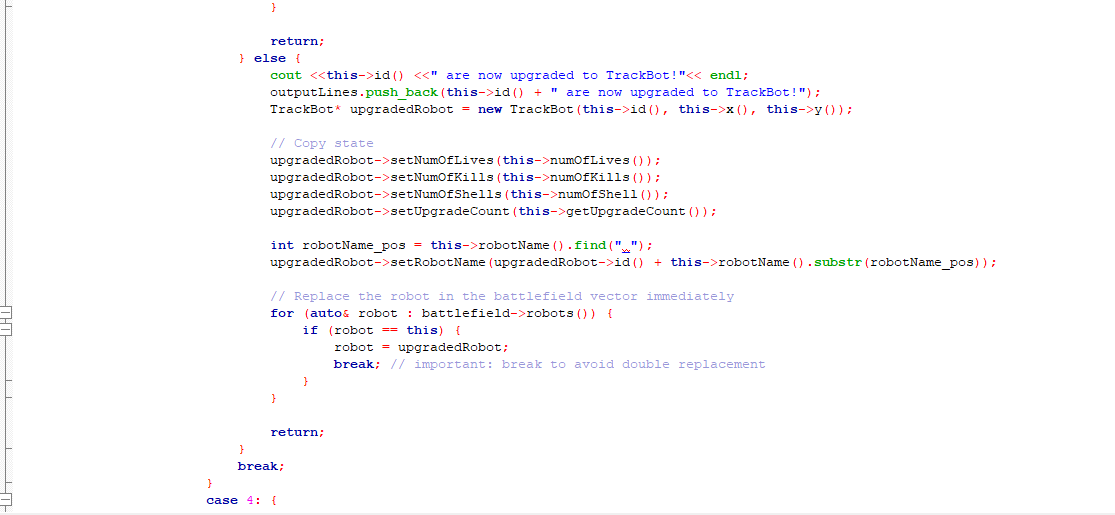
AI-generated content may be incorrect.

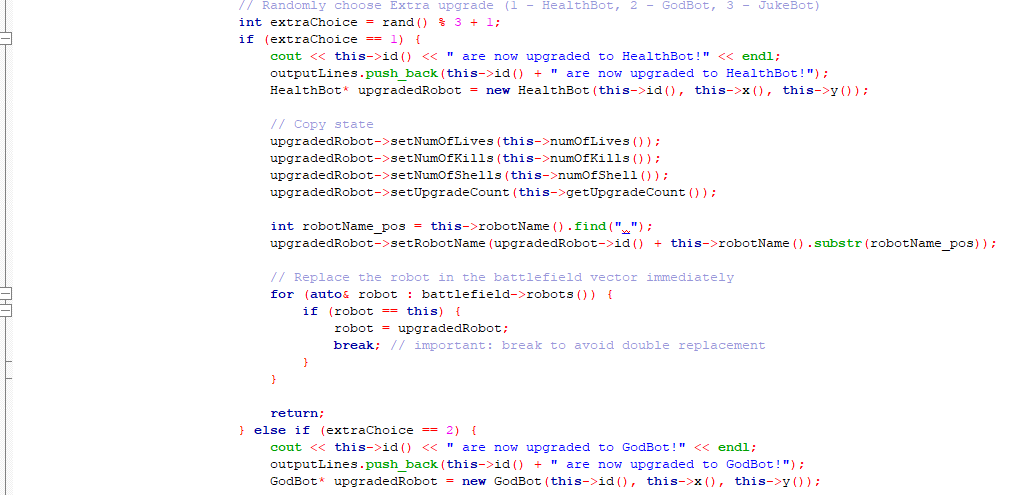




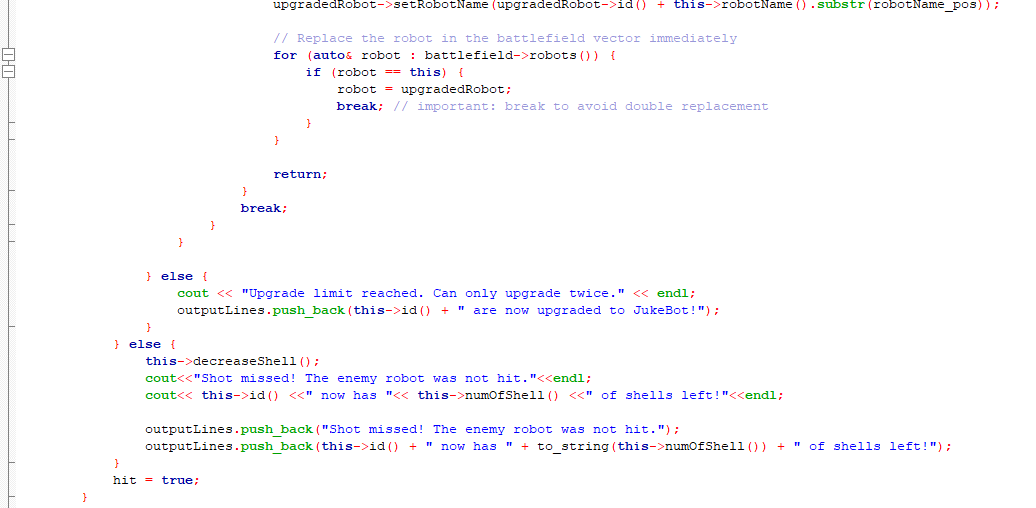


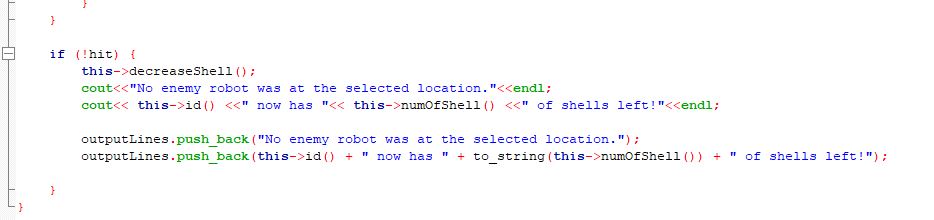












PSEUDOCODE:

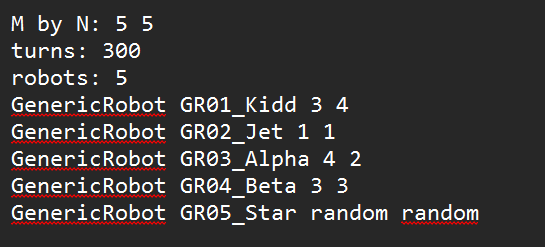
FUNCTION actionShoot(battlefield)  
 PRINT current robot name

ADD current robot name to outputLines  
   
 currentX = robot's X position  
 currentY = robot's Y position  
   
 possibleTargets = empty list  
 FOR (int dx = -1; dx <= 1; dx++):  
 FOR (int dy = -1; dy <= 1; dy++):  
 IF dx = 0 AND dy = 0: CONTINUE (skip self)  
   
 newX = currentX + dx  
 newY = currentY + dy  
   
 IF (newX, newY) is within battlefield:  
 ADD (newX, newY) to possibleTargets  
   
 IF possibleTargets is not empty:  
 targetX, targetY = random target from possibleTargets  
 PRINT selected target  
 ADD target to outputLines  
 ELSE:  
 PRINT no targets message  
 ADD message to outputLines  
 RETURN  
   
 IF robot has no shells left:  
 PRINT out of shells message  
 ADD message to outputLines  
 reduce robot's lives  
 reset shells to 10  
 RETURN  
   
 hit = false  
   
 FOR EACH robot in battlefield:  
 IF target is current robot:  
 PRINT can't shoot self message  
 ADD message to outputLines  
 BREAK  
   
 IF target matches robot's position:  
 IF robot can't be targeted:  
 PRINT target hidden message  
 ADD message to outputLines  
 RETURN  
   
 hitChance = random number (0-99)  
   
 IF hitChance < 70: // 70% hit chance  
 PRINT hit success message  
 ADD message to outputLines  
   
 reduce target's lives  
   
 IF target is dead:  
 PRINT destroyed message  
 ADD message to outputLines  
 remove from battlefield  
 add to destroyed queue  
 ELSE:  
 PRINT hit but alive message  
 ADD message to outputLines  
 temporarily remove from battlefield  
 add to waiting queue  
   
 increase shooter's kill count  
 decrease shooter's shells  
   
 PRINT remaining lives, kills, shells  
 ADD stats to outputLines  
   
  
 IF shooter can upgrade:  
 increment upgrade count  
 PRINT upgrade message  
 ADD message to outputLines  
   
 choice = random upgrade category (1-4)  
   
 CASE choice:  
 1: // Moving upgrades (HideBot or JumpBot)  
 2: // Shooting upgrades (LongShotBot, SemiAutoBot, ThirtyShotBot)  
 3: // Seeing upgrades (ScoutBot or TrackBot)  
 4: // Extra upgrades (HealthBot, GodBot, JukeBot)  
   
 Create new upgraded robot  
 Copy all stats from current robot  
 Replace in battlefield  
 RETURN  
 ELSE:  
 PRINT upgrade limit message  
 ADD message to outputLines  
 ELSE:   
 decrease shells  
 PRINT miss message and shell count  
 ADD messages to outputLines  
 hit = true  
   
 IF no robot was hit:  
 decrease shells  
 PRINT no target message and shell count  
 ADD messages to outputLines

a. ScoutBot SBXX

- actionLook(): The robot can look the entire battlefield.

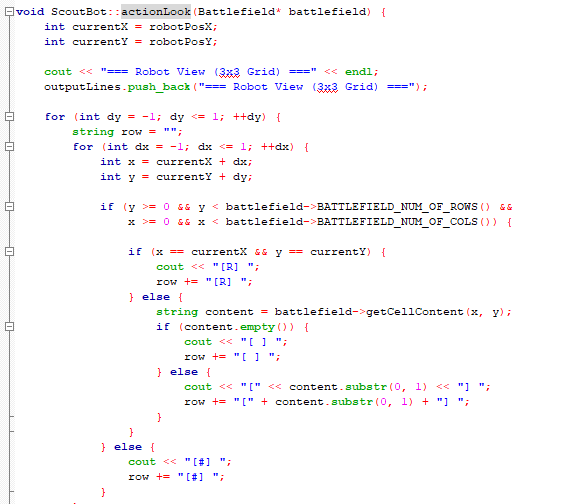
Screenshots (inputs, outputs), figures, explanations



A computer screen shot of a robot

AI-generated content may be incorrect.

Code segments, pseudocode parts, explanations



A computer code with text

AI-generated content may be incorrect.

It has a function that is almost like LookAll that enables it to see the entire battlefield when executed, it is tied to a rand() variable of lookChoice, when actionLook is called it could look normally and then randomly choose between looking the entire battlefield or not.

PSEUDOCODE:  
 FUNCTION actionLook (Battlefield\* Battlefield)

INT currentX = robotPosX;  
 INT currentY = robotPosY;  
  
 PRINT "=== Robot View (3x3 Grid) ==="  
 ADD "=== Robot View (3x3 Grid) ===" to outputLines  
  
 FOR (int dy = -1; dy <= 1; ++dy)   
 STRING row = "";  
 FOR (int dx = -1; dx <= 1; ++dx)  
 INT x = currentX + dx;  
 INT y = currentY + dy;  
  
 IF y > 0 & y < battlefield rows & x > 0 & x < battlefield coloumns  
  
 IF(x == currentX && y == currentY)   
 PRINT "[R] "  
 ADD "[R] " to rows  
 ELSE   
 STRING content = battlefield->getCellContent(x, y);  
 if (content.empty()) {  
 PRINT "[ ] "  
 ADD row "[ ] " to outputLines  
 ELSE  
 PRINT first letter of content  
 ADD first letter of content to rows

ELSE

PRINT"[#] "  
 ADD row "[#] "

b. TrackBot TBXX

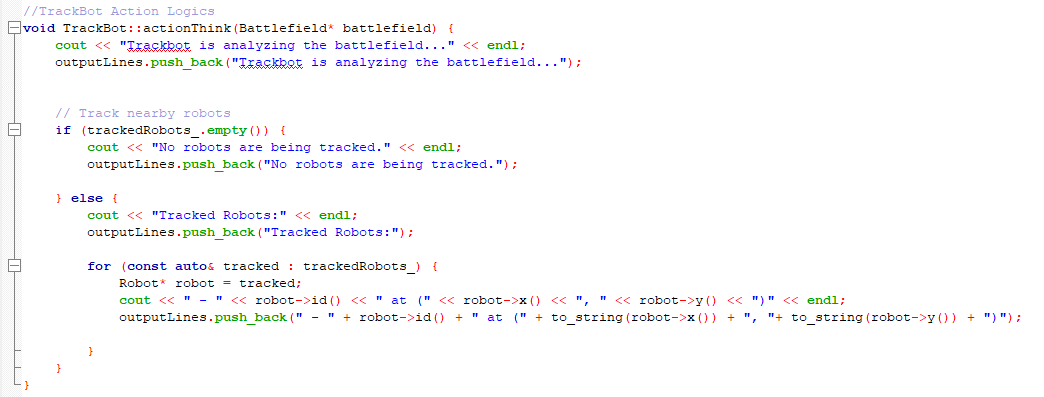
- actionLook(): The robot can plant a tracker on another enemy robot so that it can look. The robot has maximum of three trackers to look on maximum of three enemy robots.

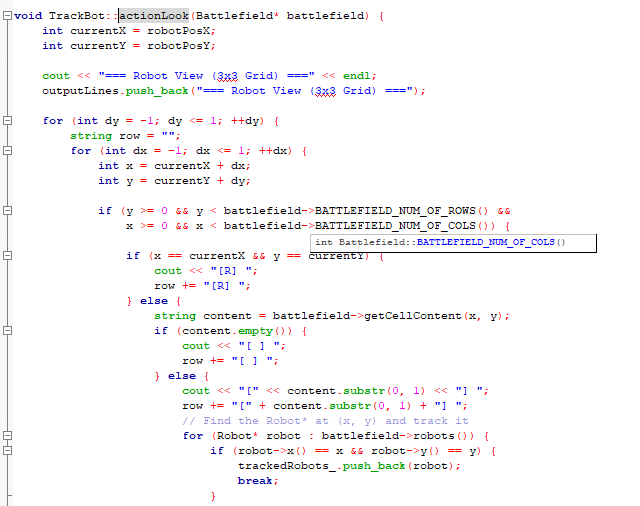
Screenshots (inputs, outputs), figures, explanations

A screenshot of a computer screen

AI-generated content may be incorrect.

Code segments, pseudocode parts, explanations





A computer code with text

AI-generated content may be incorrect.

TrackBot when using actionThink can see currently tracking robots it has placed a tracker on. It places a tracker when it uses actionLook and sees a robot in its neighbouring 8 squares.

PSEUDOCODE:

ActionThink

FUNCTION actionThink(battlefield)  
 PRINT "Trackbot is analyzing the battlefield..."  
 ADD message to outputLines  
  
 IF trackedRobots\_ list is empty:  
 PRINT "No robots are being tracked."  
 ADD message to outputLines  
 ELSE:  
 PRINT "Tracked Robots:"  
 ADD header to outputLines  
   
 FOR EACH robot in trackedRobots\_:  
 PRINT robot's ID and position (x,y)  
 ADD robot info to outputLines

ActionLook

FUNCTION actionLook(battlefield)  
 currentX = robot's X position  
 currentY = robot's Y position  
   
 PRINT "=== Robot View (3x3 Grid) ==="  
 ADD grid header to outputLines  
   
 FOR dy FROM -1 TO 1:  
 row = empty string  
 FOR dx FROM -1 TO 1:  
 x = currentX + dx  
 y = currentY + dy  
   
 IF (x,y) is within battlefield boundaries:  
 IF position is robot's current position:  
 PRINT "[R] "  
 ADD "[R] " to row  
 ELSE:  
 content = get cell content at (x,y)  
 IF content is empty:  
 PRINT "[ ] "  
 ADD "[ ] " to row  
 ELSE:  
 PRINT first character of content in brackets  
 ADD first character in brackets to row  
   
 FOR EACH robot in battlefield:  
 IF robot is at (x,y):  
 ADD robot to trackedRobots\_ list  
 BREAK  
 ELSE: // Out of bounds  
 PRINT "[#] "  
 ADD "[#] " to row  
   
 PRINT newline  
 ADD row to outputLines

c. LongShotBot LBXX

- actionFire(): The robot can fire up to three units distance away from its location. It means the robot can fire(x, y) where dx + dy <= 3.

Screenshots (inputs, outputs), figures, explanations

Input

A screenshot of a computer

AI-generated content may be incorrect.

Code segments, pseudocode parts, explanations

A screen shot of a computer code

AI-generated content may be incorrect.

* Create all possible target that maximum 3 units away and emplace into vector<pair<int,int>> possibleTargets;
* If possibleTargets is not empty than it use rand() to get randomise targetX and targetY
* Then actionShoot is the same as GenericRobot where it check if the targetX and targetY contain another robot.

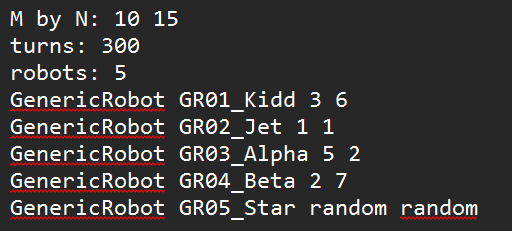
PSEUDOCODE:

FUNCTION actionShoot(battlefield):  
 PRINT "The Current Robot controlled is [robotName]"  
 ADD message to outputLines  
   
 currentX = robot's X position  
 currentY = robot's Y position  
   
 // Initialize target variables  
 targetX, targetY = 0  
 validTarget = false  
   
 battlefieldWidth = battlefield's width  
 battlefieldHeight = battlefield's height  
   
 // Generate possible targets (Manhattan distance ≤ 3)  
 possibleTargets = empty list  
   
 FOR dx FROM -3 TO 3:  
 FOR dy FROM -3 TO 3:  
 IF dx = 0 AND dy = 0:  
 CONTINUE // Skip self  
   
 IF |dx| + |dy| > 3:  
 CONTINUE // Skip if beyond range  
   
 newX = currentX + dx  
 newY = currentY + dy  
   
 IF (newX, newY) is within battlefield bounds:  
 ADD (newX, newY) to possibleTargets  
   
 // Select a random target  
 IF possibleTargets is not empty:  
 index = random index in possibleTargets  
 targetX, targetY = possibleTargets[index]  
   
 PRINT "Random target selected at: (targetX, targetY)"  
 ADD message to outputLines  
   
 validTarget = true  
 ELSE:  
 PRINT "No valid tiles found for targeting."  
 ADD message to outputLines  
 RETURN  
   
 // (Rest of the shooting logic would go here...)

d. SemiAutoBot SAXX

- actionFire(): Each shot the robot fires is now considered as three consecutive shells into one location and each fire of three shells to destroy another robot if any.

Screenshots (inputs, outputs), figures, explanations



Code segments, pseudocode parts, explanations

PSEUDOCODE:

IF targetX is equal to PotentialRobotX AND targetY is equal to PotentialRobotY THEN

IF robot cannot be targeted THEN

OUTPUT "Target is hidden and cannot be hit!"

ADD "Target is hidden and cannot be hit!" TO outputLines

RETURN from the function

ENDIF

SET threeShotSuccess TO FALSE

INITIALIZE hitChance as INTEGER

FOR i FROM 0 TO 2 DO

SET chance TO random number between 0 and 99

IF chance < 70 THEN

SET threeShotSuccess TO TRUE

BREAK the loop

ENDIF

ENDFOR

IF threeShotSuccess IS TRUE THEN

SET hitChance TO 69 // make it successful

ELSE

SET hitChance TO 80 // make it unsuccessful

ENDIF

IF hitChance < 70

OUTPUT "You’ve successfully shot an enemy Robot!"

ADD " You’ve successfully shot an enemy Robot! " TO outputLines

CONTINUE ON SHOOTACTION SAME AS GENERICROBOT • • •

END

EXPLAIN

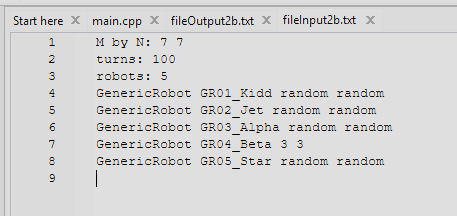
* SemiAutoBot instead of doing 1 random hit chance after aiming at a target with robot, it has 3 tries to be successful
* If any turn is successful, it sets threeShotSuccess to true and break the loop
* Then as threeShotSuccess is true hitChance is set to 69, if false it is set to 80 which is unsuccessful and won’t hit the enemy

e. ThirtyShotBot TSXX

- actionFire(): The robot now has a fresh load of 30 shells replacing its current load of shells.

Screenshots (inputs, outputs), figures, explanations

Input:



Output:

* After turning into ThirtyShotBot, its shells is set to for 30 next round

Code segments, pseudocode parts, explanations

PSEUDOCODE:

FUNCTION Constructor ThirtyShotBot(id = "", x = -1, y = -1)

SET id\_ TO "TS" + substring of id from position 2 to 3

SET robotType\_ TO "ThirtyShotBot"

SET robotPosX TO x

SET robotPosY TO y

CALL setNumOfShells WITH argument 30

END FUNCTION

EXPLAIN

-We didn’t do it actionFire() instead after upgrading into ThrityShotBot, its shell is set to 30 instantly.

f. HideBot HBXX- actionMove(): The robot can hide three turns in a match. When the robot hides, it cannot be hit by other enemy robots.

Screenshots (inputs, outputs), figures, explanations

Input :

Output:

Code segments, pseudocode parts, explanations

Pseudocode 1

FUNCTION actionMove(battlefield):

INITIALIZE variable isHidden

SET hiddenChoice to 0

IF hiddenChoice is 0 AND hiddenCount is greater than 0:

SETi sHidden to true

DECREASEhiddenCount by 1

OUTPUT "HideBot [id] has decided to hide!"

OUTPUT "Number of Hides left: [hiddenCount]"

ADD message "HideBot [id] has decided to hide!" TO output log

ADD message "Number of Hides left: [hiddenCount]" TO output log

ELSE

SET isHidden TO FASLE

CONTINUE ON actionMove that is same as GENEREICBOT • • • • •

Pseudocode 2

IF targetX equals PotentialRobotX AND targetY equals PotentialRobotY:

IF robot cannot be targeted:

OUTPUT "Target is hidden and cannot be hit!"

ADD "Target is hidden and cannot be hit!" to output log

EXIT

CONTINUE ON actionShoot that is same as GENEREICBOT • • • • •

Explanation

* Implement a bool function in HideBot class called canBeTargeted() which returns !isHidden
* In ActionMove, create boolean isHidden and hiddenChoice is selected with rand()
* If hiddenChoice is 0 then it hides and isHidden is True ELSe isHidden is False
* In all ActionShoot of all robots , after checking the pos contain a robot, it thens check if (!robot->canTargeted()) , when hidden is true it runs as (!(!(hidden)) == (hidden).
* If true it can’t be shot.

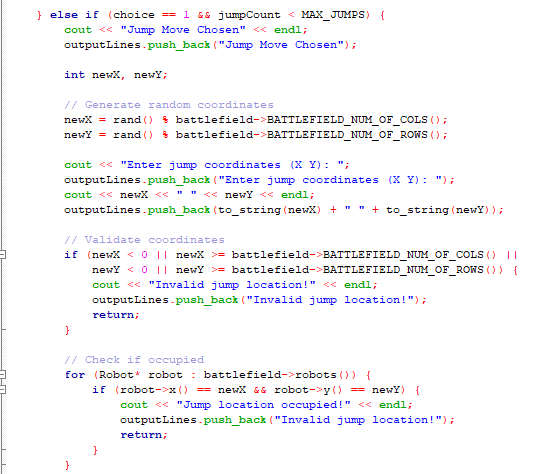
g. JumpBot JBXX

- actionMove(): The robot can jump to a new location anywhere in the battlefield. It can jump three times in a match.

Screenshots (inputs, outputs), figures, explanations

* All robots are taking turns during each game round.
* When its JumpBot’s turn it automatically run its own action methods
* JumpBot is not manually selected
* During its move, it randomly chosen 0=regular move while 1 =jump
* If 1=jump is chosen
* Random jump coordinate will generated
* Position will be checked for validility
* If valid then jump will perform
* Jump count will increased by 1
* A GenericRobot can be upgraded to JumpBot after earning a kill
* Once upgrade it will behave like a JumpBot in future turn

Code segments, pseudocode parts, explanations



**Pseudocode**

Part 1

DISPLAY "1. Regular Move\n2. Jump\nChoose action:"

ADD message to outputLines

SET choice to a random number between 0 and 1

Part 2

IF choice is 1 AND jumpCount is less than MAX\_JUMPS THEN

DISPLAY "Jump Move Chosen"

ADD message to outputLines

GENERATE random coordinates newX and newY within battlefield bounds

DISPLAY "Enter jump coordinates (X Y):"

DISPLAY newX and newY

ADD messages to outputLines

IF newX or newY is out of battlefield bounds THEN

DISPLAY "Invalid jump location!"

ADD message to outputLines

EXIT the function

ENDIF

FOR each robot on battlefield

IF another robot exists at (newX, newY) THEN

DISPLAY "Jump location occupied!"

ADD message to outputLines

EXIT the function

ENDIF

END FOR

Part 3

SET current cell (robotPosX, robotPosY) to empty

UPDATE robotPosX to newX

UPDATE robotPosY to newY

SET new cell (robotPosX, robotPosY) to this robot

INCREMENT jumpCount by 1

DISPLAY "[id\_] jumped to (newX, newY)!"

ADD message to outputLines

ELSE

DISPLAY "Jump limit reached or invalid choice!"

ADD message to outputLines

Explanation

* The robot shows 2 option Regular Move or Jump and randomly choose 1 action.
* If Jump is chosen and the robot have jumps left:
* I t picks random position on the battlefield
* It checks iof the position is valid
* If is valid and free, the robot jumps to the new position and the jump count increases
* A success message is printed
* If the jump limit is reached:
* An error message is printed adn the robot stays in places

Inheritance, figures, explanations

code segments, pseudocode parts, explanations

GenericRobot including the other upgrade bots all inherit the base class of Thinking, Seeing, Shooting, Moving Robot

Polymorphism, figures, explanations

code segments, pseudocode parts, explanations

ThinkingRobot is a base abstract class that only has a Pure Virtual Function that all other classes inherit and redefine differently

Operator overloading, figures, explanations

code segments, pseudocode parts, explanations

Here we overload the << function as a friend with the robot class, so the cout is able to access private members of the class.

# Q6. [5] The algorithms used to optimize the actions of robots, vector, queue

Algorithm 1 using optimize the action, figures, explanations

Vector: keep track robot actions during the simulation

Algorithm 1 screenshots (inputs, outputs), explanations

Algorithm 1 code segments, pseudocode parts, explanations

PSEUDOCODE

Part1

INITIALIZE GLOBALLY vector<string> outputLines;

Part2

SET outputFile TO "fileOutput" + SUBSTRING OF inputFile STARTING FROM index 9

OPEN outputFile FOR writing AS outFile

FOR EACH line IN outputLines:

WRITE line TO outFile WITH newline

CLOSE outFile

END

Explaination

* Every line of code that contain cout is followed by outputLines.push\_back() that push string into the vector to be output to a txt file at the end of the program.

Algorithm 2 optimize the action, figures, explanations

Two queues: keep track of the respective destroyed and waiting robots

Algorithm 2 screenshots (inputs, outputs), explanations

Updatewaitingrobots() uses waitingrobots\_, it let them wait for a certain round before letting them return to battlefield

PlaceRobots() check for whether a robot is in waitingRobots\_ or destroyedRobots\_ before placing

In actionShoot() of every class we set pos to –1 , -1 to for waiting robots, then push into waitingRobots() and –1 , -2 for destroyed robots and push into waitingRobots()

In main we used the pos for destroyed and waiting robot to skip turns

Algorithm 2 code segments, pseudocode parts, explanations

PSEUDOCODE

FUNCTION updateWaitingRobots: SET size TO SIZE OF waitingRobots\_

FOR i FROM 0 TO size - 1:  
 REMOVE the front ELEMENT OF waitingRobots\_ AND UNPACK INTO robot AND turnsWaited  
  
 IF turnsWaited + 1 IS GREATER THAN OR EQUAL TO 5 THEN:  
 SET positionFound TO FALSE  
  
 DO:  
 SET newX TO RANDOM INTEGER MODULO BATTLEFIELD\_NUM\_OF\_COLS\_  
 SET newY TO RANDOM INTEGER MODULO BATTLEFIELD\_NUM\_OF\_ROWS\_  
 IF cell AT (newX, newY) IS empty THEN:  
 SET positionFound TO TRUE  
 WHILE positionFound = FALSE  
  
 IF positionFound IS FALSE THEN:  
 FOR newY FROM 0 TO BATTLEFIELD\_NUM\_OF\_ROWS\_ - 1:  
 FOR newX FROM 0 TO BATTLEFIELD\_NUM\_OF\_COLS\_ - 1:  
 IF cell AT (newX, newY) IS empty THEN:  
 SET positionFound TO TRUE  
 GOTO position\_found  
 LABEL position\_found  
  
 IF positionFound IS TRUE THEN:  
 SET robot X coordinate TO newX  
 SET robot Y coordinate TO newY  
 PLACE robot ON battlefield cell AT (newX, newY)  
  
 OUTPUT message: robot has waited 5 turns and is returning  
 ADD message TO outputLines  
  
 ADD robot TO robots\_ list  
 ELSE:  
 OUTPUT message: no empty space found  
 ADD message TO outputLines  
 ADD (robot, turnsWaited + 1) BACK TO waitingRobots\_  
 ELSE:  
 ADD (robot, turnsWaited + 1) BACK TO waitingRobots\_  
END FOR

END FUNCTION

2. FUNCTION updateWaitingRobots: SET size TO SIZE OF waitingRobots\_

FOR i FROM 0 TO size - 1 DO:  
 EXTRACT robot AND turnsWaited FROM FRONT OF waitingRobots\_  
 REMOVE FRONT OF waitingRobots\_  
  
 IF turnsWaited + 1 IS GREATER THAN OR EQUAL TO 5 THEN:  
 SET positionFound TO FALSE  
  
 DO:  
 SET newX TO RANDOM NUMBER MODULO BATTLEFIELD\_NUM\_OF\_COLS\_  
 SET newY TO RANDOM NUMBER MODULO BATTLEFIELD\_NUM\_OF\_ROWS\_  
  
 IF isCellEmpty(newX, newY) IS TRUE THEN:  
 SET positionFound TO TRUE  
 WHILE positionFound IS EQUAL TO FALSE  
  
 IF positionFound IS FALSE THEN:  
 FOR newY FROM 0 TO BATTLEFIELD\_NUM\_OF\_ROWS\_ - 1 DO:  
 FOR newX FROM 0 TO BATTLEFIELD\_NUM\_OF\_COLS\_ - 1 DO:  
 IF isCellEmpty(newX, newY) IS TRUE THEN:  
 SET positionFound TO TRUE  
 GOTO position\_found  
 END FOR  
 END FOR  
 LABEL position\_found:  
 END IF  
  
 IF positionFound IS TRUE THEN:  
 CALL robot->setX(newX)  
 CALL robot->setY(newY)  
 CALL setCell(newX, newY, robot)  
  
 OUTPUT robot ID AND RETURN POSITION MESSAGE  
 ADD MESSAGE TO outputLines  
  
 ADD robot TO robots\_  
 ELSE:  
 OUTPUT "No empty space found" MESSAGE  
 ADD MESSAGE TO outputLines  
 ADD {robot, turnsWaited + 1} TO waitingRobots\_  
 END IF  
 ELSE:  
 ADD {robot, turnsWaited + 1} TO waitingRobots\_  
 END IF  
END FOR

END FUNCTION

3. FUNCTION placeRobots: FOR EACH robot IN robots\_ DO: SET robotID TO robot's ID

SET isDestroyed TO FALSE  
 COPY destroyedRobots\_ TO tempDestroyed  
  
 WHILE tempDestroyed IS NOT EMPTY DO:  
 SET r TO FRONT OF tempDestroyed  
 REMOVE FRONT FROM tempDestroyed  
 IF r's ID EQUALS robotID THEN:  
 SET isDestroyed TO TRUE  
 BREAK  
 END WHILE  
  
 SET isWaiting TO FALSE  
 COPY waitingRobots\_ TO tempWaiting  
  
 WHILE tempWaiting IS NOT EMPTY DO:  
 SET r TO FIRST ELEMENT’S ROBOT IN tempWaiting  
 REMOVE FRONT FROM tempWaiting  
 IF r's ID EQUALS robotID THEN:  
 SET isWaiting TO TRUE  
 BREAK  
 END WHILE  
  
 IF isDestroyed IS FALSE AND isWaiting IS FALSE THEN:  
 SET x TO robot's X position  
 SET y TO robot's Y position  
  
 IF y < battlefield\_ SIZE AND x < battlefield\_[y] SIZE THEN:  
 CALL setCell(x, y, robot)  
 ELSE:  
 OUTPUT "Robot " + robotID + " is out OF bounds!"  
 TERMINATE PROGRAM  
 END IF  
END FOR

END FUNCTION

4.

IF robot IS NOT alive THEN

DISPLAY targetRobotId + " has been destroyed!"

ADD targetRobotId + " has been destroyed!" TO outputLines

SET cell at (PotentialRobotX, PotentialRobotY) TO NULL IN battlefield

CLEAR cell at (PotentialRobotX, PotentialRobotY) IN battlefield

SET robot X TO -1

SET robot Y TO -2

PUSH robot TO destroyedRobots QUEUE IN battlefield

ELSE

DISPLAY targetRobotId + " was hit and is now temporarily inactive!"

ADD targetRobotId + " was hit and is now temporarily inactive!" TO outputLines

SET cell at (PotentialRobotX, PotentialRobotY) TO NULL IN battlefield

CLEAR cell at (PotentialRobotX, PotentialRobotY) IN battlefield

SET robot X TO -1

SET robot Y TO -1

PUSH (robot, 0) TO waitingRobots QUEUE IN battlefield

END IF

5.

WHILE (currentTurn < totalTurns)

• • • •

IF robot's position is (-1, -1) THEN

PRINT "Skipping Turns, Robot [robotName] is still in waiting queue."

ADD to outputLines "Skipping Turns, Robot [robotName] is still in waiting queue."

INCREMENT currentTurn

CONTINUE to next robot

IF robot is not alive (isAlive() == false) THEN

PRINT "Skipping Turns, Robot [robotName] is destroyed and out of the game!"

ADD to outputLines "Skipping Turns, Robot [robotName] is destroyed and out of the game!"

INCREMENT currentTurn

CONTINUE to next robot

IF robot's position is (-1, -2) THEN

PRINT "Skipping Turns, Robot [robotName] is destroyed and out of the game!"

ADD to outputLines "Skipping Turns, Robot [robotName] is destroyed and out of the game!"

INCREMENT currentTurn

CONTINUE to next robot

CONTINUITAION OF WHILE LOOP CODE• • • •

EXPLANATION

1. Updatewaitingrobots() can be called using battlefield->waitingRobots().push({robot, 0});
   1. It push in robots and at the same time and an int 0 which mean it start to count down from 0
   2. Updatewaitingrobots() accept an int to count down from it to 5.
   3. After 5, it randomise an unoccupied location for the robot to be respawned at.
   4. The robot is then .pop out of the queue waitingRobots\_ and respawned at random location.
   5. \*\*\*\* Updatewaitingrobots() \*\*\*\* is called every turn in main to update the reamaing turn the robots is stunned for.
2. PlaceRobots() check first if current robot about to be placed is in waitingRobots\_ or is in destroyedRobots\_
3. After a robot is pushed into destroyedRobots() it wont be able to respawn again and thus out of this game
4. A robot that is destroyed is setX to –1 and setY to –2 then pushed intoqueue<Robot\*> destroyedRobots\_, while a robot that is killed and pushed into waitingRobots is setX to –1 and setY to –1 then push into queue<pair<Robot\*, int>> waitingRobots\_
5. In main, we cheked if x() == -1 && y() == -1 which is the condition for waitingRobots\_ where it will skip to next turn
   1. We also cheked if x() == -1 && y() == -2 which is the condition for destroyedRobots\_ where it will skip to next turn
   2. OR check if isAlive() if return false then it also means its a destroyedRobots\_ thus skipping turn

# Q7. [9] Implementation of three new robot classes

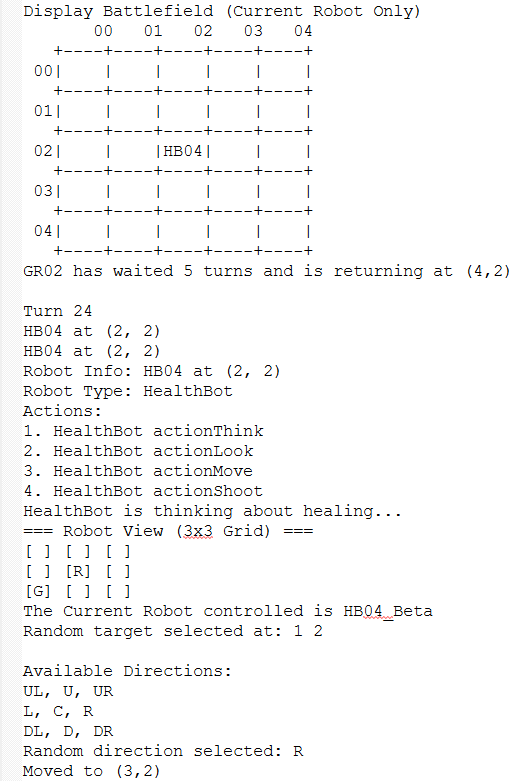
New robot name 1: HealthBot HBXX

- The robot starts with 5 lives, making it harder to destroy than a normal robot.

Screenshots (inputs, outputs), figures, explanations

Input:

Output:



- SB04 has been upgraded to a HealthBot, now identified as HB04, and is currently located at position (2,2).

- HealthBot functions like a GenericRobot but starts with 5 lives instead of 3, making it more durable in battle.

- The robot’s name format changes to "HBxx" to reflect the upgraded type. For example, HB04 refers to the upgraded version of SB04.

Code segments, pseudocode parts, explanations

**Pseudocode**

FUNCTION Constructor HealthBot(id = "", x = -1, y = -1)

SET id\_ TO "HB" + substring of id from position 2 to 2

SET robotType\_ TO "HealthBot"

SET robotPosX TO x

SET robotPosY TO y

CALL setNumOfLives WITH argument 5

END FUNCTION

**Explanation**

The HealthBot constructor initializes a robot with the given ID and position, then sets its robot type and gives it 5 lives using setNumOfLives(5). This is more than the default 3 lives of a GenericRobot, allowing it to survive longer in battle. The ID is prefixed with "HB" to distinguish it from other robot types.

New robot name 2: GodBot GBXX

- canBeTargeted(): The robot cannot be targeted or destroyed by other robots (invincible).

Screenshots (inputs, outputs), figures, explanations

Input:

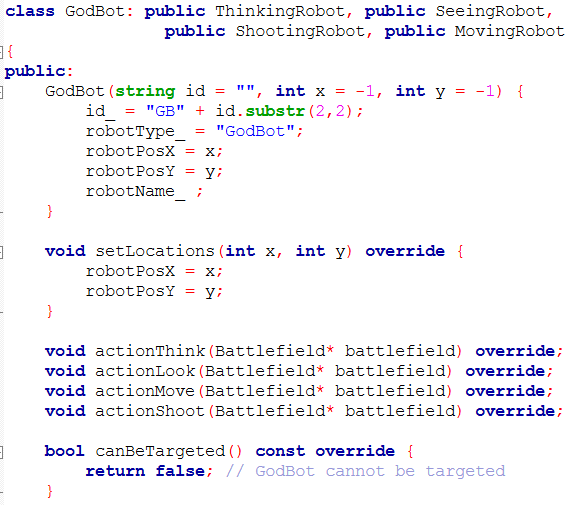
Output:

- GR02\_Jet has been upgraded to a GodBot, now identified as GB02, and is currently located at position (1,1).

- GodBot behaves like a normal robot but is invincible, as its canBeTargeted() function always returns false.

- In Turn 42, GB02 executed its standard actions: think, look, move, and shoot. It fired at (2,3) but missed. The jump action failed due to a limit or invalid choice.

Code segments, pseudocode parts, explanations



**Pseudocode**

FUNCTION Constructor GodBot(id = "", x = -1, y = -1)

SET id\_ TO "GB" + substring of id from 2 to 2

SET robotType\_ TO "GodBot"

SET robotPosX TO x

SET robotPosY TO y

END FUNCTION

FUNCTION canBeTargeted() RETURNS bool

RETURN false

END FUNCTION

**Explanation**

The GodBot constructor sets a unique robot ID starting with "GB" and initializes its position. While it behaves like a normal robot in movement and attack, it becomes invincible by overriding canBeTargeted() to always return false, meaning it cannot be hit or destroyed. This makes it ideal for testing or acting as a powerful boss character in the battlefield.

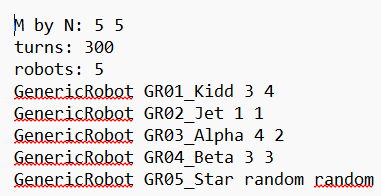
New robot name 3: JukeBot

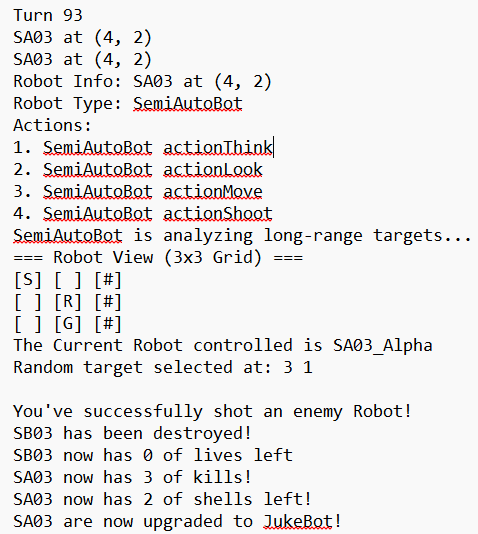
Explanations, choose one.

Instaed of putting it at actionMove() we decided to put it at actions()

- actions(): Every 5 turns, the JukeBot will instantly teleport to a random empty spot on the batterfield, and this happens automatically, no player input needed.

Screenshots (inputs, outputs), figures, explanations





* Each time it takes turn the turnCounter increases
* Every 5 turn, it attempts to “juke” (teleport) to a random empty position
* It tries up to 100 random position to find an empty one
* It will print out successful message when success
* It will also print unsuccessful message when not success
* A robot gets an upgrade after earning a kill
* The upgrade type is chosen randomly
* If JukeBot selected, the robot will become JukeBot immediately

Code segments, pseudocode parts, explanations

# Pseudocode

INCREMENT turnCounter by 1

IF turnCounter is divisible by 5 THEN

SET positionFound to FALSE

REPEAT up to 100 times

SET newX to a random column within battlefield bounds

SET newY to a random row within battlefield bounds

IF the cell at (newX, newY) is empty THEN

SET positionFound to TRUE

BREAK the loop

ENDIF

END REPEAT

IF positionFound is TRUE THEN

// Move the robot to the new position

SET current cell (robotPosX, robotPosY) to null in battlefield

UPDATE robotPosX to newX

UPDATE robotPosY to newY

SET new cell (robotPosX, robotPosY) to this robot in battlefield

PRINT message: "[robot ID] juked to (newX, newY)!"

ADD message to outputLines list

ELSE

PRINT message: "No empty space found for juke!"

ADD message to outputLines list

ENDIF

ENDIF

**Explanation**

* Every 5 turns, the robot tries to perfor, a ‘juke’ (sudden move)
* It randomly searches for an empty position on the batterfield (up to 100 tries)
* If an empty spot was found:
* The robot will move to the new postition
* A message is printed
* If no empty spot is found:

A failure message is printed

# Question: How to compile, how to run the program

Go to int main() , write ur prefered inputfile to be ran, and run main.cpp