MULTIMEDIA UNIVERSITY

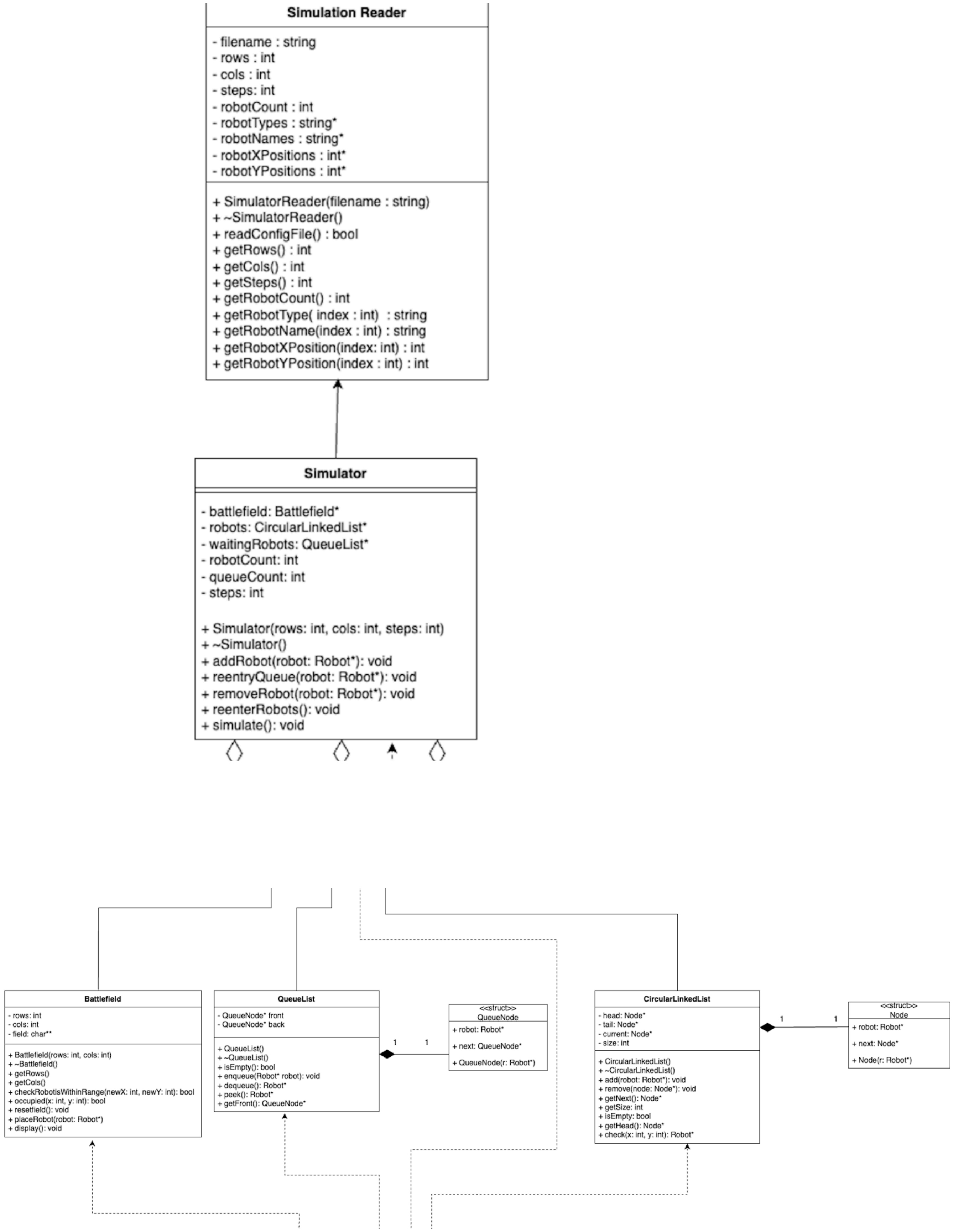
CCP6124 OOPDS

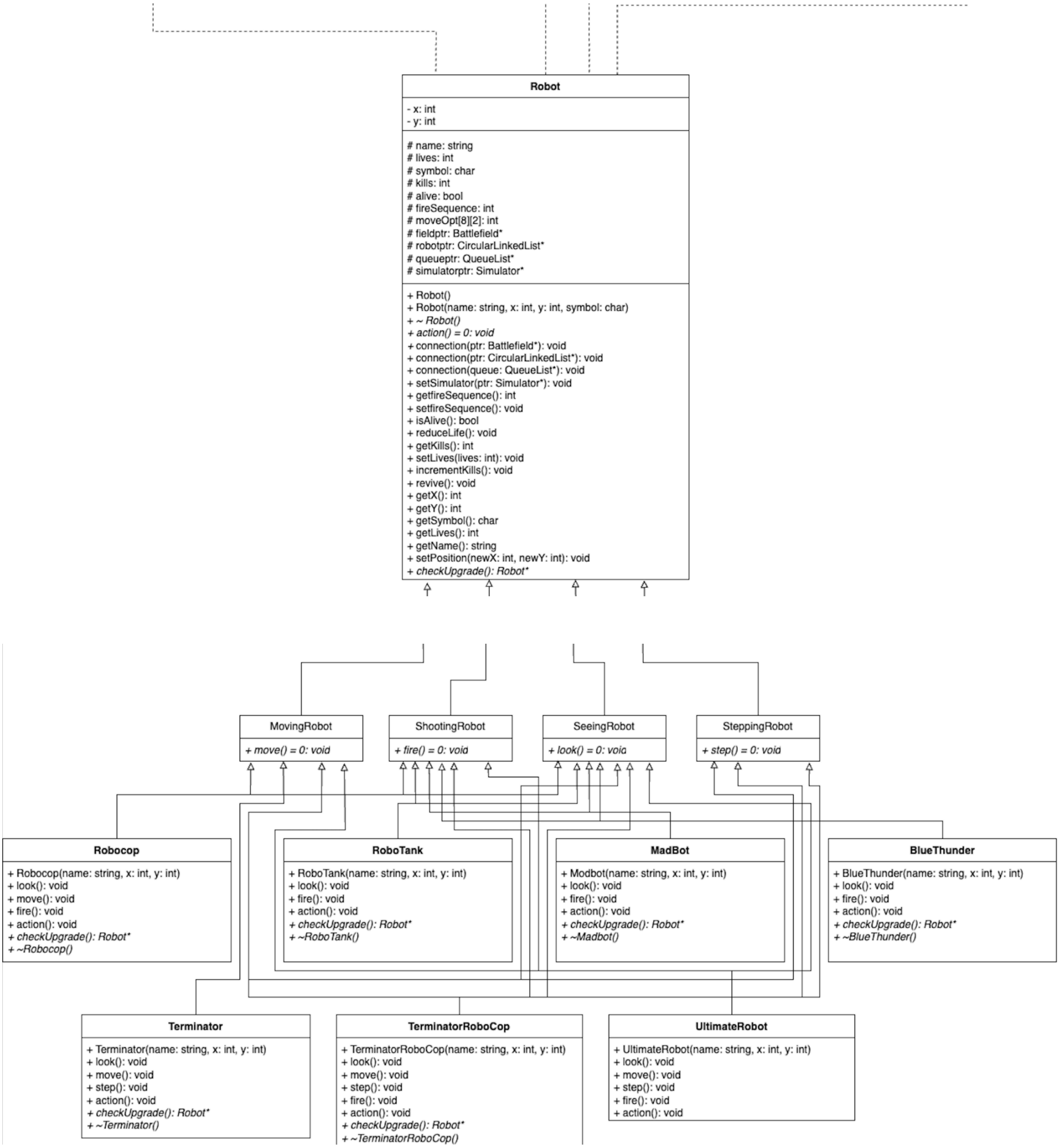
Assignment

Group Member:

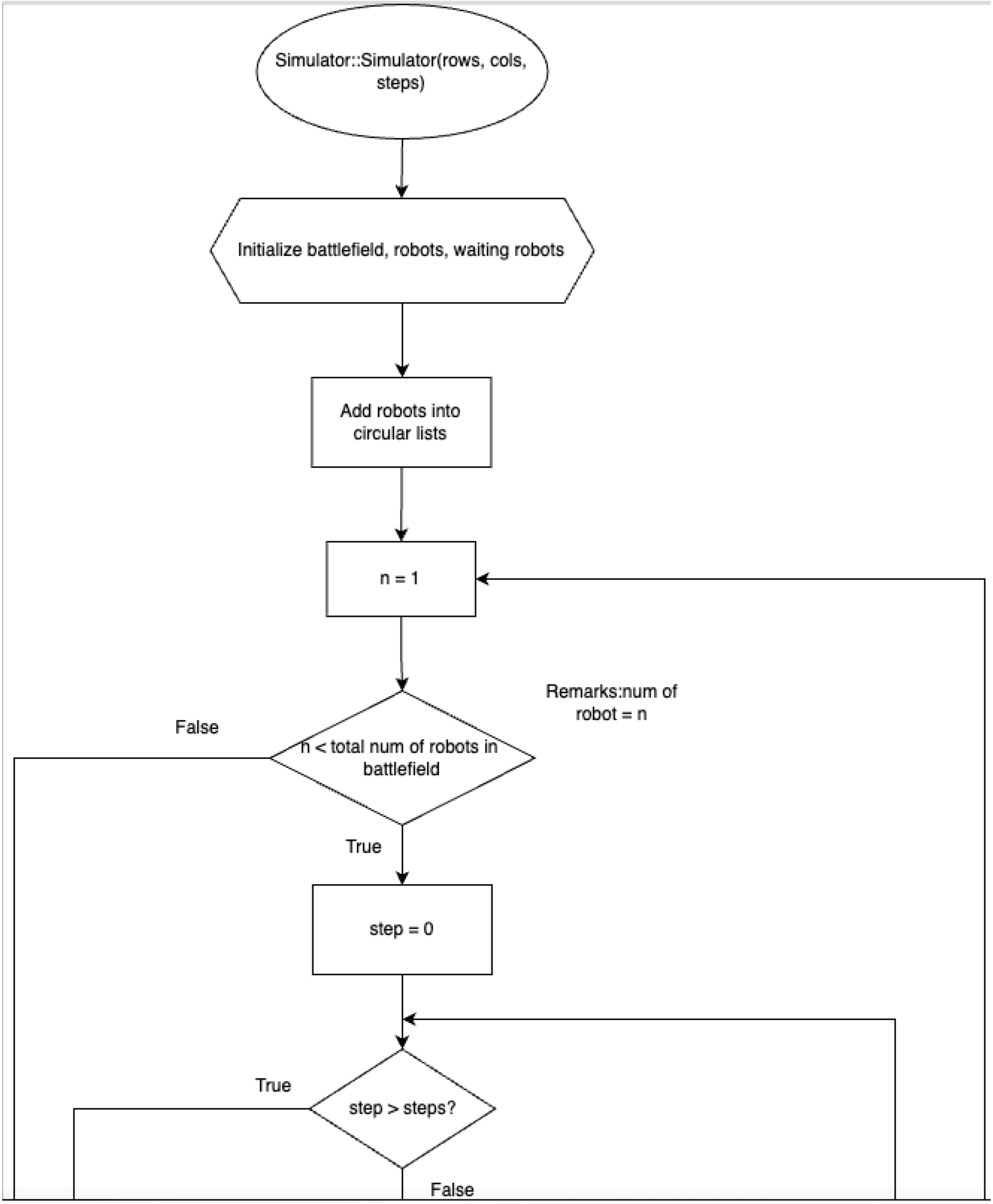
1.	CHAN HOI SIANG	1211111604
2.	SEE JIE SHENG	1211110469
3.	PHANG JUN YUAN	1211110732
4.	WOON WEN TAO	1211108861

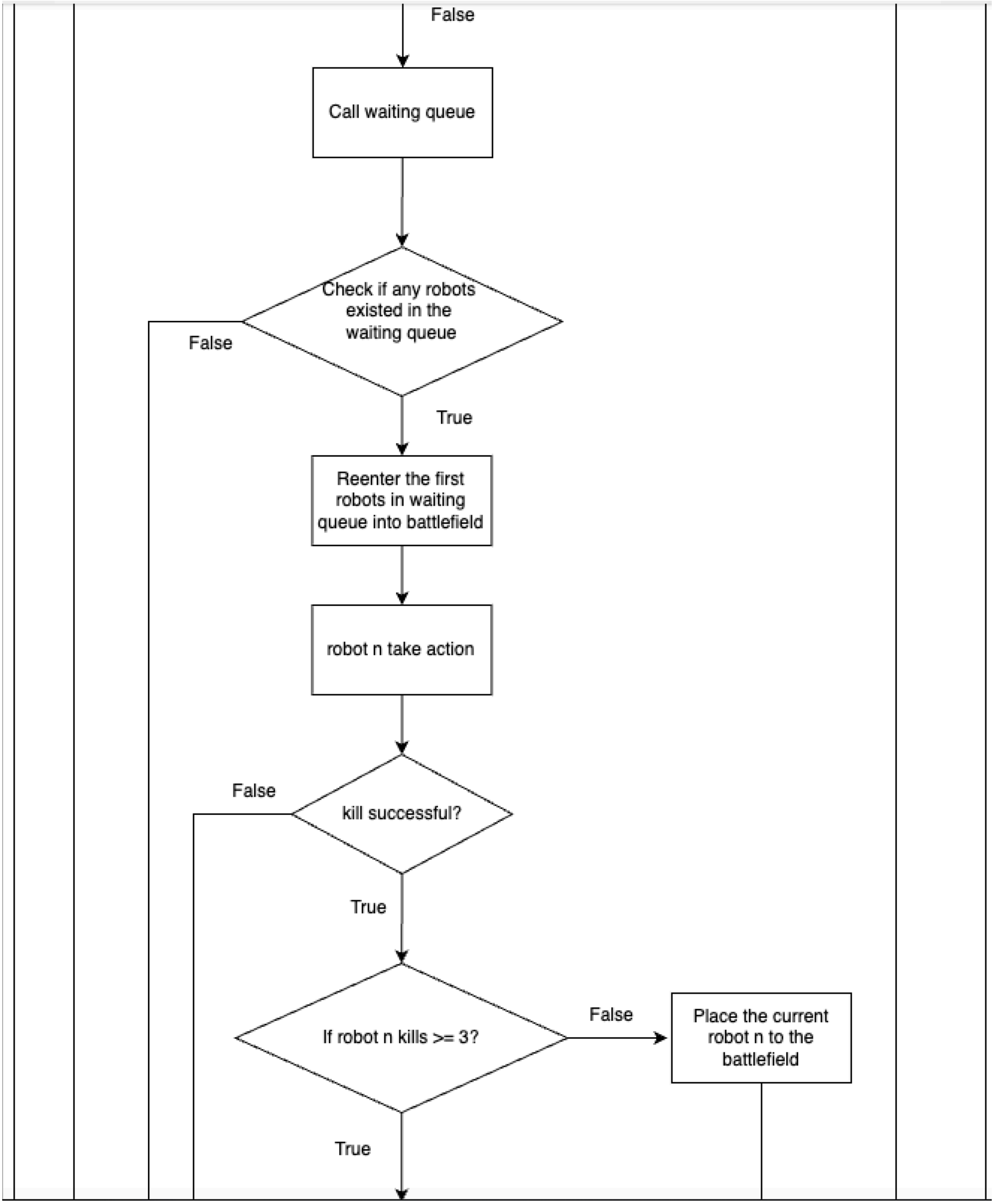
CLASS DIAGRAM

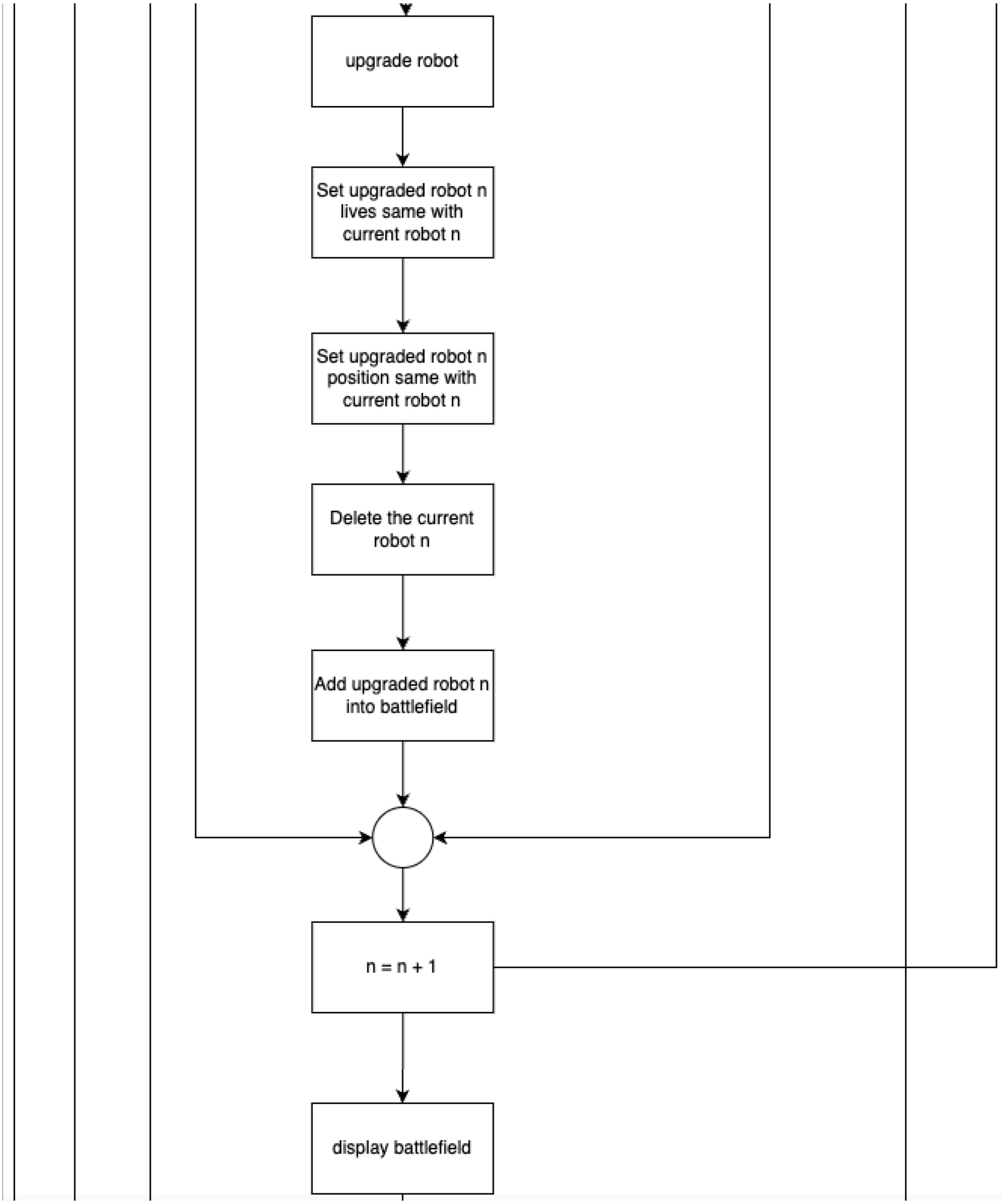

Simulation Reader
- filename : string
- rows : int
- cols : int
- steps: int
- robotCount : int
- robotTypes : string*
- robotNames : string*
- robotXPositions : int*
- robotYPositions : int*
+ SimulatorReader(filename : string)
+ ~SimulatorReader()
+ readConfigFile() : bool
+ getRows() : int
+ getCols() : int
+ getSteps() : int
+ getRobotCount() : int
+ getRobotType(index : int) : string
+ getRobotName(index : int) : string
+ getRobotXPosition(index: int) : int
+ getRobotYPosition(index : int) : int
Simulator
- battlefield: Battlefield*
- robots: CircularLinkedList*
- waitingRobots: QueueList*
- robotCount: int
- queueCount: int
- steps: int
+ Simulator(rows: int, cols: int, steps: int)
+ ~Simulator()
+ addRobot(robot: Robot*): void
+ reentryQueue(robot: Robot*): void
+ removeRobot(robot: Robot*): void
+ reenterRobots(): void
+ simulate(): void
Battlefield
- rows: int
- cols: int
- field: char**
+ Battlefield(rows: int, cols: int)
+ ~Battlefield()
+ getRows()
+ getCols()
+ checkRobotisWithinRange(newX: int, newY: int): bool
+ occupied(x: int, y: int): bool
+ resetfield(): void
+ placeRobot(robot: Robot*)
+ display(): void
QueueList
- QueueNode* front
- QueueNode* back
+ QueueList()
+ ~QueueList()
+ isEmpty(): bool
+ enqueue(Robot* robot): void
+ dequeue(): Robot*
+ peek(): Robot*
+ getFront(): QueueNode*
1
1
<<struct>>
QueueNode
+ robot: Robot*
+ next: QueueNode*
+ QueueNode(r: Robot*)
CircularLinkedList
- head: Node*
- tail: Node*
- current: Node*
- size: int
+ CircularLinkedList()
+ ~CircularLinkedList()
+ add(robot: Robot*): void
+ remove(node: Node*): void
+ getNext(): Node*
+ getSize: int
+ isEmpty: bool
+ getHead(): Node*
+ check(x: int, y: int): Robot*
1
1
<<struct>>
Node
+ robot: Robot*
+ next: Node*
+ Node(r: Robot*)

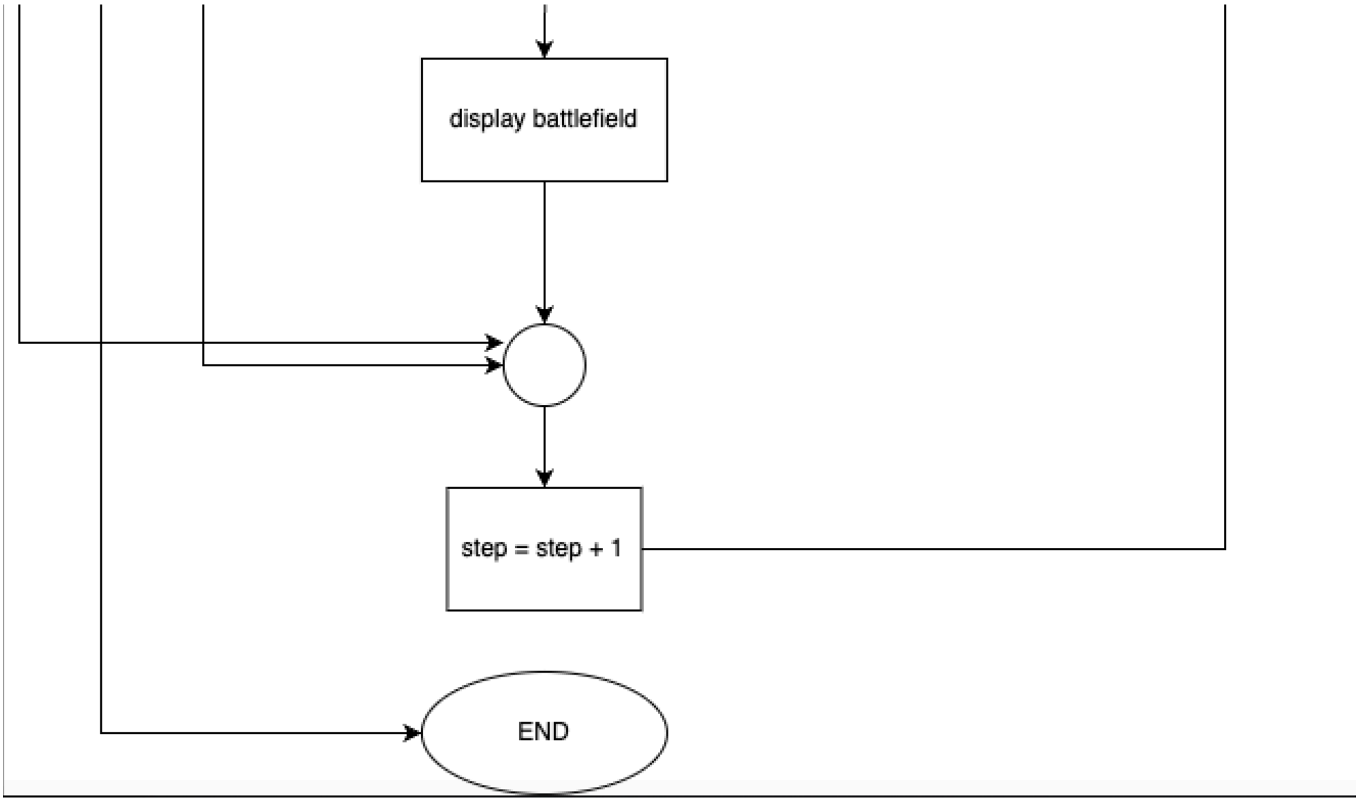

Robot
- x: int
- y: int
# name: string
# lives: int
# symbol: char
# kills: int
# alive: bool
# fireSequence: int
# moveOpt[8][2]: int
# fieldptr: Battlefield*
# robotptr: CircularLinkedList*
# queueptr: QueueList*
# simulatorptr: Simulator*
+ Robot()
+ Robot(name: string, x: int, y: int, symbol: char)
+ ~ Robot()
+ action() = 0: void
+ connection(ptr: Battlefield*): void
+ connection(ptr: CircularLinkedList*): void
+ connection(queue: QueueList*): void
+ setSimulator(ptr: Simulator*): void
+ getfireSequence(): int
+ setfireSequence(): void
+ isAlive(): bool
+ reduceLife(): void
+ getKills(): int
+ setLives(lives: int): void
+ incrementKills(): void
+ revive(): void
+ getX(): int
+ getY(): int
+ getSymbol(): char
+ getLives(): int
+ getName(): string
+ setPosition(newX: int, newY: int): void
+ checkUpgrade(): Robot*
MovingRobot
+ move() = 0: void
ShootingRobot
+ fire() = 0: void
SeeingRobot
+ look() = 0: void
SteppingRobot
+ step() = 0: void
Robocop
+ Robocop(name: string, x: int, y: int)
+ look(): void
+ move(): void
+ fire(): void
+ action(): void
+ checkUpgrade(): Robot*
+ ~Robocop()
RoboTank
+ RoboTank(name: string, x: int, y: int)
+ look(): void
+ fire(): void
+ action(): void
+ checkUpgrade(): Robot*
+ ~RoboTank()
MadBot
+ Modbot(name: string, x: int, y: int)
+ look(): void
+ fire(): void
+ action(): void
+ checkUpgrade(): Robot*
+ ~Madbot()
BlueThunder
+ BlueThunder(name: string, x: int, y: int)
+ look(): void
+ fire(): void
+ action(): void
+ checkUpgrade(): Robot*
+ ~BlueThunder()
Terminator
+ Terminator(name: string, x: int, y: int)
+ look(): void
+ move(): void
+ step(): void
+ action(): void
+ checkUpgrade(): Robot*
+ ~Terminator()
TerminatorRoboCop
+ TerminatorRoboCop(name: string, x: int, y: int)
+ look(): void
+ move(): void
+ step(): void
+ fire(): void
+ action(): void
+ checkUpgrade(): Robot*
+ ~TerminatorRoboCop()
UltimateRobot
+ UltimateRobot(name: string, x: int, y: int)
+ look(): void
+ move(): void
+ step(): void
+ fire(): void
+ action(): void

FLOWCHART


Simulator::Simulator(rows, cols, steps)
Initialize battlefield, robots, waiting robots
Add robots into circular lists
n = 1
Remarks:num of robot = n
False
n < total num of robots in battlefield
True
step = 0
True
step > steps?
False


False
Call waiting queue
Check if any robots existed in the waiting queue
False
True
Reenter the first robots in waiting queue into battlefield
robot n take action
False
kill successful?
True
If robot n kills >= 3?
False
Place the current robot n to the battlefield
True


upgrade robot
Set upgraded robot n lives same with current robot n
Set upgraded robot n position same with current robot n
Delete the current robot n
Add upgraded robot n into battlefield
n = n + 1
display battlefield


display battlefield
step = step + 1
END

IMPORTANT CODE

Circular Linked List

In this assignment, we use a **circular linked list** in implementing robot turn to move. Each robot gets a fair and equal chance to move. The circular list ensures that once the last robot takes its turn, the first robot is up next, thus cycling through all robots indefinitely.

```
void CircularLinkedList::add(Robot *robot)
{
    Node *newNode = new Node(robot);
    if (!head)
    {
        head = newNode;
        tail = newNode;
        tail->next = head;
    }
    else
    {
        tail->next = newNode;
        tail = newNode;
        tail->next = head;
    }
    size++;

void CircularLinkedList::remove(Robot *robot)
{
    if (head == nullptr || robot == nullptr)
        return;

    Node *current = head;
    Node *previous = nullptr;

    while (current->robot != robot)
    {
        previous = current;
        current = current->next;
    }

    if (current == head)
    {
        head = current->next;
        tail->next = head;
    }
    else if (current == tail)
    {
        previous->next = head;
        tail = previous;
    }
    else
    {
        previous->next = current->next;
    }

    delete current;
    size--;
}

Robot *CircularLinkedList::check(int x, int y) const
{
    if (head == nullptr)
    {
        return nullptr;
    }

    Node *current = head;

    do
    {
        if (current->robot->getX() == x && current->robot->getY() ==
y)
        {
            return current->robot;
        }
        current = current->next;
    } while (current != head);

    return nullptr;
}
```

Things to highlight:

A. ADD METHOD

1. `ADD` method inserts a new `Robot` into the list.
2. If the list is empty (`head` is `nullptr`), it initialises `head` and `tail` with the new node and sets `tail->next` to `head`.
3. Otherwise, it adds the new node after the tail, updates the tail to the new node, and makes it circular by setting `tail->next` to `head`.

B. REMOVE METHOD

1. The `REMOVE` method deletes a specific Robot from the list.
2. If the list or the node to be deleted is `nullptr`, it returns immediately.

3. Otherwise, It iterates through the list to find the node.

Updates pointers:

- If removing the head, update head and `tail->next`.
- If removing `tail`, updates `tail` and makes the list circular again.
- Otherwise, update the `previous` node's `next` pointer to skip the `current` node.

C. CHECK METHOD

1. `CHECK` method searches for a `Robot` with specific coordinates `(x, y)` in the list.
2. If the list is empty, it returns `nullptr`.
3. If it finds a `Robot` with matching coordinates, returns that `Robot`.
4. If no matching `Robot` is found, returns `nullptr`.

Queue List:

In this assignment, we use this **queue** to schedule and execute robot tasks sequentially.Each robot might receive commands like move,attack,defend then added to the **queue** and executed in the order they are received.To ensure fair and sequential processing of commands.Thus,this **queue** helps to manage the flow of actions of commands among robots,ensuring that they act in a controlled and ordered manner during the simulated robot war.

Important Code:

```
void QueueList::enqueue(Robot *robot)
{
    QueueNode *newNode = new QueueNode(robot);
    if (isEmpty())
    {
        front = back = newNode;
    }
    else
    {
        back->next = newNode;
        back = newNode;
    }
}

Robot *QueueList::dequeue()
{
    if (isEmpty())
    {
        return nullptr;
    }
    QueueNode *temp = front;
    front = front->next;
    if (front == nullptr)
    {
        back = nullptr;
    }
    Robot *robot = temp->robot;
    delete temp;
    return robot;
}
```

```
Robot *QueueList::peek() const
{
    if (isEmpty())
    {
        return nullptr;
    }
    return front->robot;
}
```

Things to highlight:

A. enqueue METHOD

1. Push a new Robot to the back of the queue.
2. A new QueueNode is created with the given Robot.
3. Both front and back pointers are set to this new node when the queue is empty.
4. Otherwise, it will add the new node to the tail of the queue,and the new node is updated and pointed by the back pointer.

B. dequeue METHOD

1. Pop the Robot from the head of the queue.
2. It will return to nullptr when the queue is empty.
3. Otherwise,it removes the front node then updates the front pointer to the next node and deletes the old front node.
4. It will set the back pointer to nullptr when the queue becomes empty after removing the node.

C. peek METHOD

1. Used to return the Robot at the front of the queue without removing it.If the queue is empty then it will return to nullptr.

Simulation:

The main program that performs the whole operation.

Important Codes:

```
void Simulator::reenterRobots() {
    bool validPosition = false;
    if (!waitingRobots->isEmpty()){
        while (!validPosition){
            Robot* robot = waitingRobots->dequeue();
            int positionX = rand() % battlefield->getRows();
            int positionY = rand() % battlefield->getCols();
            if (battlefield->checkRobotisWithinRange(positionX,
positionY)) {
            Robot* target = robots->check(positionX, positionY);
                if (target) {
                    validPosition = false;
                } else {
                    validPosition = true;
                    robot->setPosition(positionX, positionY);
                    addRobot(robot);
                    battlefield->placeRobot(robot);
                }
            }
        }
    }
}

void Simulator::simulate() {
    Node* current = robots->getHead();
    for (int step = 0; step < steps; ++step) {
        reenterRobots();
        battlefield->resetField();

        if (current->robot->isAlive()) {
            current->robot->action();
            Robot* upgradedRobot =
current->robot->checkUpgrade();
            if (upgradedRobot != current->robot) {
                cout << current->robot->getName() << "upgrade
successfully" << endl;


upgradedRobot->setLives(current->robot->getLives());

upgradedRobot->setPosition(current->robot->getX(),
current->robot->getY());

                removeRobot(current->robot);
                addRobot(upgradedRobot);
            } else {
                battlefield->placeRobot(current->robot);
            }
        }

        Node* temp = robots->getHead();
        do {
            if (temp->robot->isAlive()) {
                battlefield->placeRobot(temp->robot);
          }
            temp = temp->next;
        } while (temp != robots->getHead());

        battlefield->display();

        current = current->next;

        cout << "Step " << step + 1 << " completed." << endl;
    }
}
```

Things to highlight:

A. Reenter METHOD

1. Check if the waitingRobots queue is not empty.
2. If not empty, dequeues robots and tries to place them in random valid positions on the battlefield.
3. If a valid position is found, the robot is added to the battlefield and the robots list.

B. Simulate METHOD

1. Run the simulation for the specified number of steps.
2. Calls reenterRobots() to place robots from the waiting queue back onto the battlefield.
3. For the current robot:
    - If the robot is alive, it performs its action.
    - Checks if the robot needs to upgrade and handles the upgrade if necessary.

OOP CONCEPTS

OOP Concepts Implementation

1. **Inheritance** - We use HIERARCHY inheritance in relation to robot linking. First of all, we have a base class named Robot, under Robot, there are 4 derived classes: SeeingRobot, MovingRobot , ShootingRobot, SteppingRobot. Under these 4 derived classes, there are 7 derived classes named Robocop, RoboTank, MadBot, BlueThunder, Terminator, TerminatorRoboCop, UltimateRobot.

2. **Polymorphism** - In Robot.h file, we have used the virtual function, abstract derived class and virtual destructors as a polymorphism. In this class, '`virtual void action() = 0`' is a pure virtual function, making 'Robot ' an abstract class. Also, '`virtual Robot* checkUpgrade()`' is a virtual function that can be overridden in derived classes. Next, we use '`virtual ~Robot()`' as a virtual destructor to ensure the correct destructor is called for derived classes. For the abstract derived classes, there are some classes further specialize the 'Robot' class and provide their own pure virtual functions. For the example:
   - `class MovingRobot : virtual public Robot`
     - `virtual void move() = 0;`

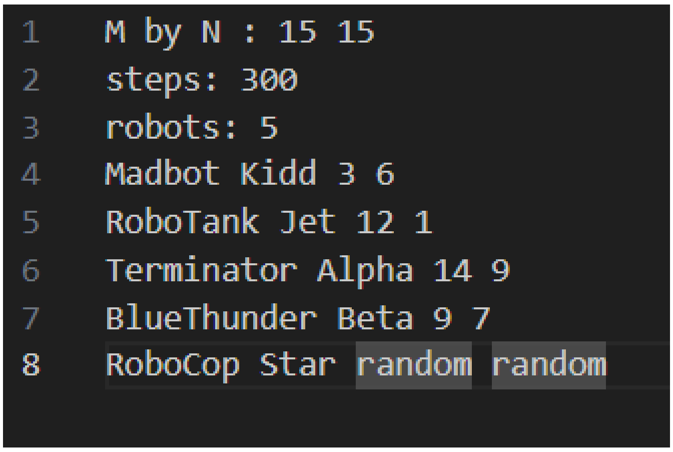
3. **Encapsulation** - In the Simulator.h file,we declared at the beginning of the 'Simulator' class that the private data members('battlefield', 'robots', 'waitingRobots', 'robotCount', 'queueCount', 'steps', 'runFile') are accessible only within the 'Simulator' class itself.Then we declared public data members('getBattlefield()', 'addRobot()', 'reentryQueue()', 'removeRobot()', 'reenterRobots()', 'simulate()', 'record()') and

providing controlled access to manipulate or retrieve the private data members.These methods act as interfaces to interact with the ‘Simulator’ class and its encapsulated data.They enforce how external code can interact with the internal state of the ‘Simulator’ object.

4. **Abstraction** - In `Simulator` class, we abstract the entire simulation process. It hides the complexity of managing robots, battlefield updates, and robot reentry.

SAMPLE OUTPUT

Sample Data:

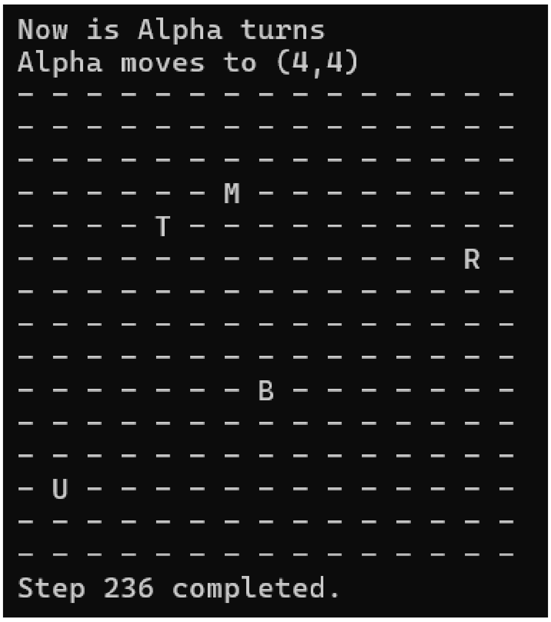


```
M by N : 15 15
steps: 300
robots: 5
Madbot Kidd 3 6
RoboTank Jet 12 1
Terminator Alpha 14 9
BlueThunder Beta 9 7
RoboCop Star random random
```

1. We initialize the battlefield's size to 15 X 15.
2. We set the number of steps to 300 and
   the number of robots to 5.
3. We set 5 robots respectively into coordinates :
   (3, 6), (12, 1), (14, 9), (9,7), (random, random).

Robot Action:

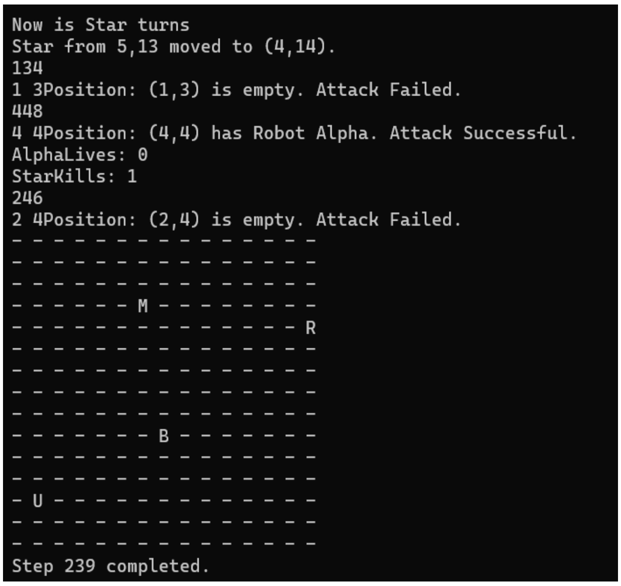
Move:



```
Now is Alpha turns
Alpha moves to (4,4)
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - T - - - - - - - - - -
- - - - - - - - - - - - - R -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - B - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- U - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
Step 236 completed.
```

1. The robot named Alpha as terminator robot T moves to position(4,4).

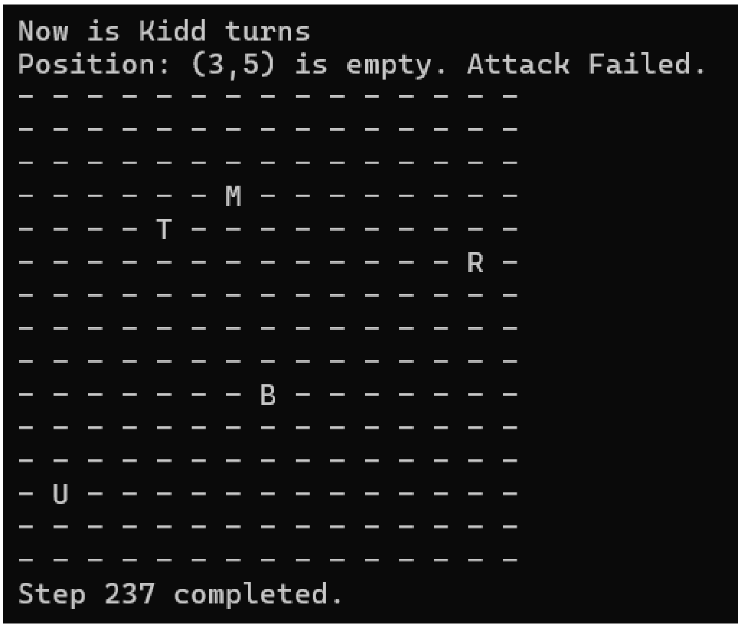
‘Attack success:



```
Now is Star turns
Star from 5,13 moved to (4,14).
134
1 3Position: (1,3) is empty. Attack Failed.
448
4 4Position: (4,4) has Robot Alpha. Attack Successful.
AlphaLives: 0
StarKills: 1
246
2 4Position: (2,4) is empty. Attack Failed.
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - - - - - - - - - - - R
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - B - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- U - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
Step 239 completed.
```

1. Star attacks the position at (4,4) which has Robot Alpha.
2. Thus, the attack successful

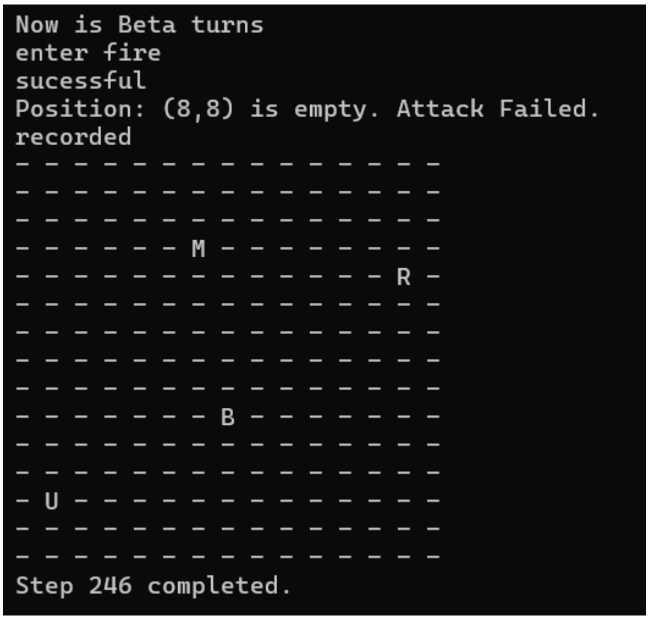
Attack failed:



```
Now is Kidd turns
Position: (3,5) is empty. Attack Failed.
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - T - - - - - - - - - -
- - - - - - - - - - - - - R -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - B - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- U - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
Step 237 completed.
```

1. If the attack position (3, 5) is empty, then it is considered a failed attack.

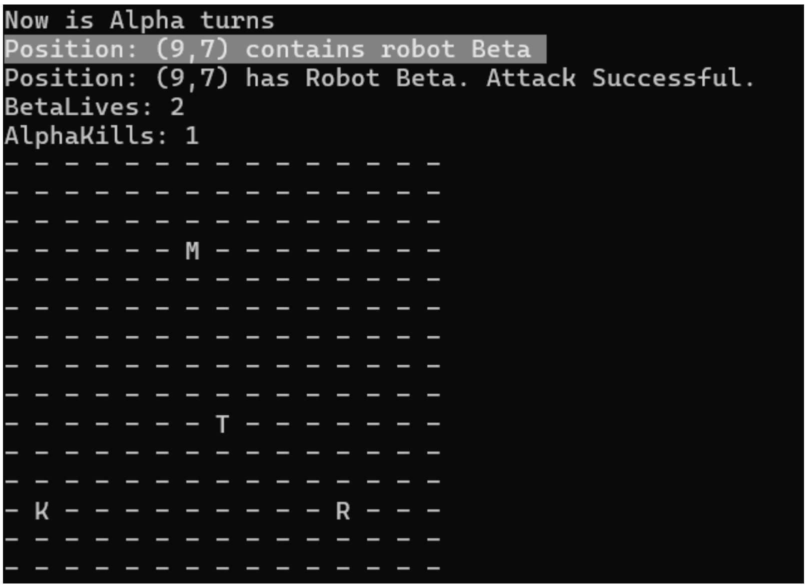
Fire:



```
Now is Beta turns
enter fire
sucessful
Position: (8,8) is empty. Attack Failed.
recorded
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - - - - - - - - - - R -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - B - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- U - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
Step 246 completed.
```

1. Robot Beta fires at position (8,8).

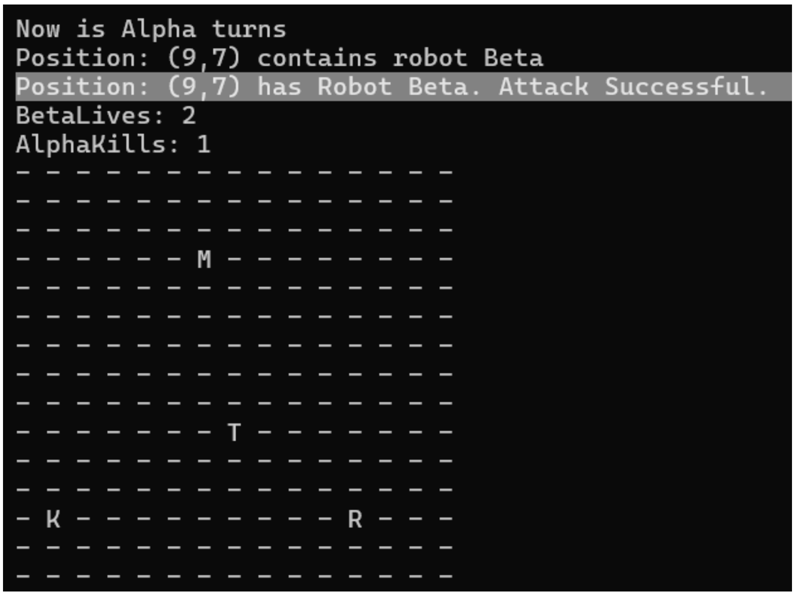
Look:



```
Now is Alpha turns
Position: (9,7) contains robot Beta
Position: (9,7) has Robot Beta. Attack Successful.
BetaLives: 2
AlphaKills: 1
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - T - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- K - - - - - - - - - R - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
```

1. Alpha used the look function and found that (9, 7) contains robot Beta.
2. Thus it will step it.

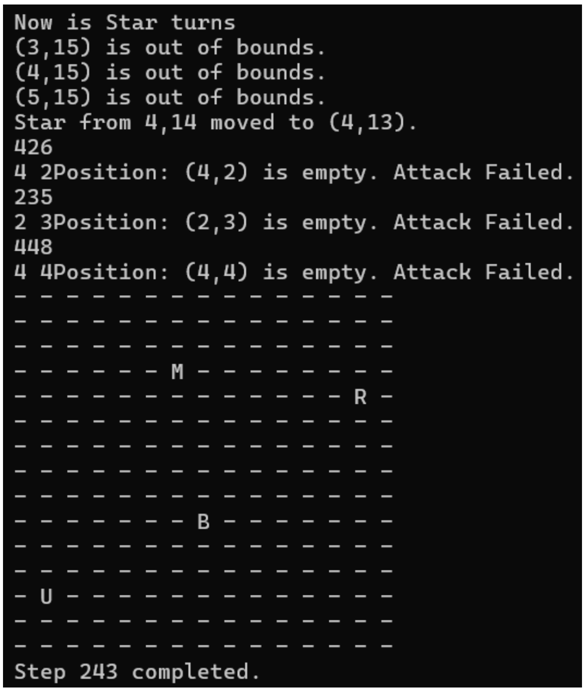
Step:



```
Now is Alpha turns
Position: (9,7) contains robot Beta
Position: (9,7) has Robot Beta. Attack Successful.
BetaLives: 2
AlphaKills: 1
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - T - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- K - - - - - - - - - R - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
```

1. Robot alpha found that position (9,7) has other robots.
2. Robot alpha steps on it.
3. Therefore attack successfully.

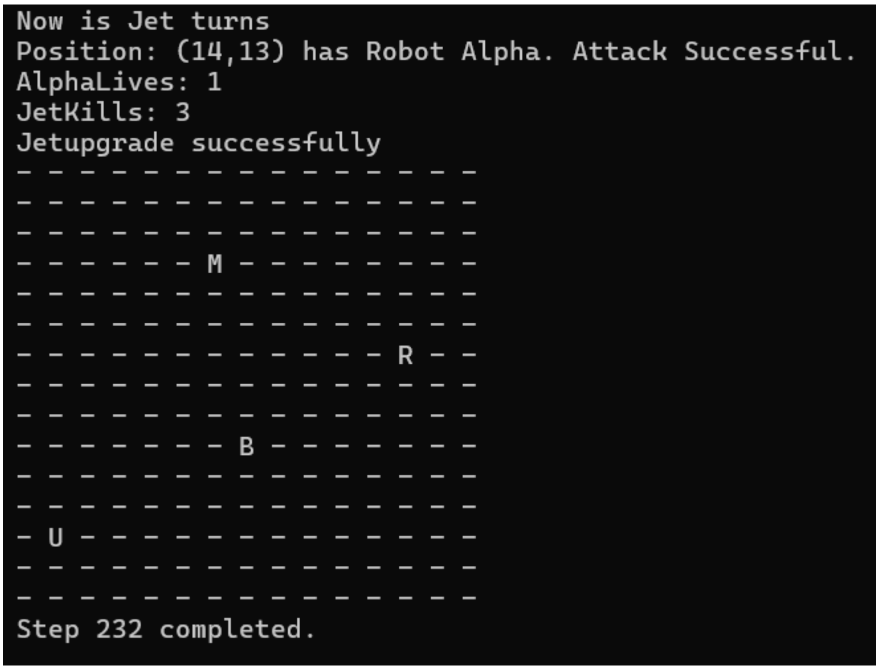
Out of bound:



```
Now is Star turns
(3,15) is out of bounds.
(4,15) is out of bounds.
(5,15) is out of bounds.
Star from 4,14 moved to (4,13).
426
4 2Position: (4,2) is empty. Attack Failed.
235
2 3Position: (2,3) is empty. Attack Failed.
448
4 4Position: (4,4) is empty. Attack Failed.
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - - - - - - - - - - R -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - B - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- U - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
Step 243 completed.
```

1. (3, 15), (4,15), (5, 15) are out of bounds, thus the robot cannot execute the action.

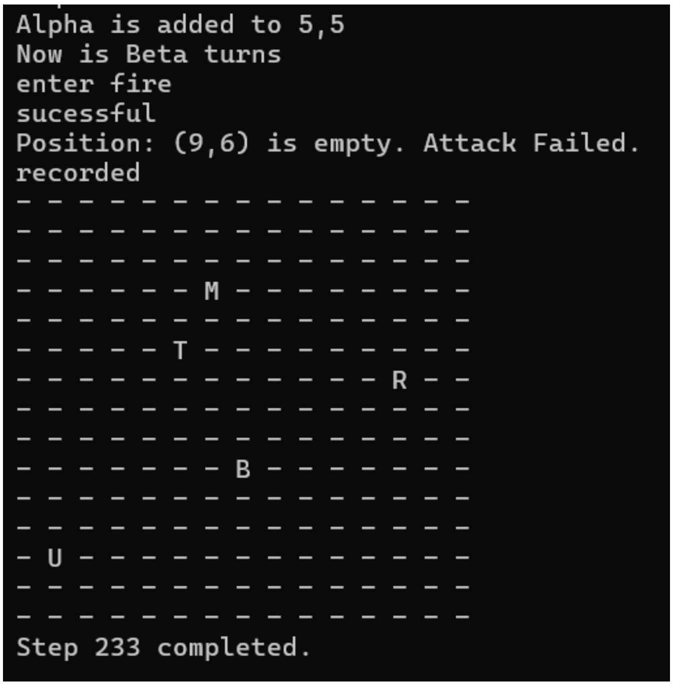
Upgrade:



```
Now is Jet turns
Position: (14,13) has Robot Alpha. Attack Successful.
AlphaLives: 1
JetKills: 3
Jetupgrade successfully
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - R - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - B - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- U - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
Step 232 completed.
```

1. When JetKills (kill) reaches 3, it will upgrade to the next robot.

Add:



```
Alpha is added to 5,5
Now is Beta turns
enter fire
sucessful
Position: (9,6) is empty. Attack Failed.
recorded
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - M - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - T - - - - - - - - -
- - - - - - - - - - - - R - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - B - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- U - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
Step 233 completed.
```

1. The robot named Alpha is added to the battlefield at (5, 5).