Robot World Client Requirements

The client program must connect to a world using the world's IP address and port. The IP address and port must not be hard-coded. You can use a command line argument or configuration file to specify the IP address and port of the server.

Movement and Direction

1. The robot can move forward in the direction it is facing with the command `forward <steps>`.
2. The robot can move backward in the direction opposite to what it is facing. Use the command `back <steps>` to move backwards.
3. The robot can turn left or right with the command `turn <left/right>`.
4. Movements must honor the world artefacts and behaviour such as obstacles, pits and visibility.
5. Use the command `orientation` to find out which direction the robot is facing reported as north, south, east, west.

Looking around

1. The robot can look around for the maximum distance specified by the world. Use the command `look` to look around in all four directions.
2. The response to the command will always return world artefacts in absolute positions (described by compass coordinates such as North, South, East, West). In other words, if the robot is facing East and there's a obstacle to the left of the robot, the world will report that the obstacle is to the North and not "left" of the robot.
3. The world will respond with information within its visible range, honouring all requirements of the game.

IMPORTANT

The world and robots that inhabit the world can only see in straight lines toward the compass directions. In other words, a robot will never be able to see and object "North East" or "South West" of it.

Defend

1. A robot has a shield that can be configured to withstand a configured number of hits from another robot.
2. The strength of the shield is measured in the number of hits it can take before it is destroyed.

For example, if a shield has strength 2 then it can withstand 2 hits with each hit reducing the strength by one unit. On the 3rd hit, the robot is killed.

3. The maximum shield strength allowed is configured by the world.
4. If the robot's shield strength exceeds the maximum set by the world, then the maximum of the world is used.
5. A shield configuration of `0` means the robot has no shield.

Repair mode

Robots can be instructed to rest to repair their shields to its configured maximum strength using the command `repair`.

While shields are being repaired:

1. The world configures how long it takes to repair shields (in seconds).
2. Shields are repaired up to the maximum level or not at all, no matter how long it takes to repair the shields. For example, if the world configures the shield repair time of 3 seconds, then whether the robot has 1 hits left of its shields or 3 hits left, it will still take 3 seconds to repair to the maximum shield strength.
3. Shields cannot be repaired to more than the maximum that the robot is configured for.
4. The robot cannot move while it is being repaired.

Active Attack

A robot has a gun that can be configured with the distance a shot can travel, measured in steps. This distance affects the number of shots a robot can have as follows:

Distance (steps)	Number of Shots
5	1
4	2
3	3
2	4
1	5
0	0 (no active attack available)

Attacking

1. A robot can be instructed to fire the gun using the command `fire`.

2. The robot will fire a shot in the direction it is facing over the distance it is configured.
3. A robot can hit another robot it can't see, like shooting into a dense fog.
4. Shooting an obstacle has no effect on the obstacle.

Reload mode

Robots can be instructed to reload the gun using the command `reload`.

While reloading:

1. The world configures how long it takes to reload the weapon.
2. The robot cannot move.
3. A robot cannot reload more than its maximum number of shots.
4. A robot must reload to its maximum number of shots or not at all. For example, if the robot is configured for 3 shots, and has no shots left and the world has set reload time to 5 seconds; then when it is reloading it must wait 5 seconds to reload 3 shots. If the robot had 2 shots then it must still wait for 5 seconds to reload.

NOTE

The world server needs to keep track of every robot's position and hits. If a robot fires a shot and wants to know if it hit something, the world must respond with the outcome of the shot. See Robot World Protocol for more information.

Passive Attack

1. A robot can be configured to place a mine in an unoccupied coordinate in the world.
2. If a robot steps on the mine, it reduces its shields by 3 hits. A robot can step onto its own mine and will take 3 hits too.
3. A robot that can place mines cannot have a gun/active weapon.
4. A robot can be instructed to place a mine with the command `mine`.

`SetMine` mode

The world configures how long it takes to set a mine.

While placing a mine, the robot shields are temporarily disabled, and it cannot move at all. It's a sitting duck, so to say.

Once the mine is placed, the robot must automatically move forward by 1 step. If the automatic one step forward is blocked by an obstacle, then the robot steps onto its own mine takes hits.

Detecting mines

1. A robot can detect a mine within a quarter of the world's visibility range, rounded down. For ex-

ample, if the visibility range is 10, then robots can see mines that are 2 steps away, in any direction.

Server messages

~~If a robot takes a hit from stepping on a mine or a shot from another robot, then the server world must send a message to the robot with its latest state. This means that the client robot will need to listen for messages from the server. We leave this to you to figure out how to achieve that.~~

If a robot takes a hit or dies, then the client will only know about it when it gets the state back on the next command it issues. See Protocol Specification for details on the state of the robot.

Makes of robots

The client can be programmed to have different makes (or models) of robots with different capability configurations. You can use any name you invent for the make. For example, a `Sniper Robot` is capable of taking a long range shot and has shield strength of 1 and relies on protection from obstacles.

Joining a world

1. The client program first connects to the server world when it starts. However, it has not yet launched a robot into the world.
2. Use the command `launch <make> <name>` to launch the robot into the world where `make` is the name of the make of the robot and `name` is the name of the robot. The world will randomly position the robot in the world in an open space.
3. If there is already another robot in the world with the same name, then the world will deny entry.

The user interface

The client program displays the world whichever way you wish to program it.

It can be a text console view, with a list of obstruction and coordinates, and commands to help explore it via a limited command line interface. Or it can use the turtle graphics library or Zircon or anything else you wish, like **JavaFX**.