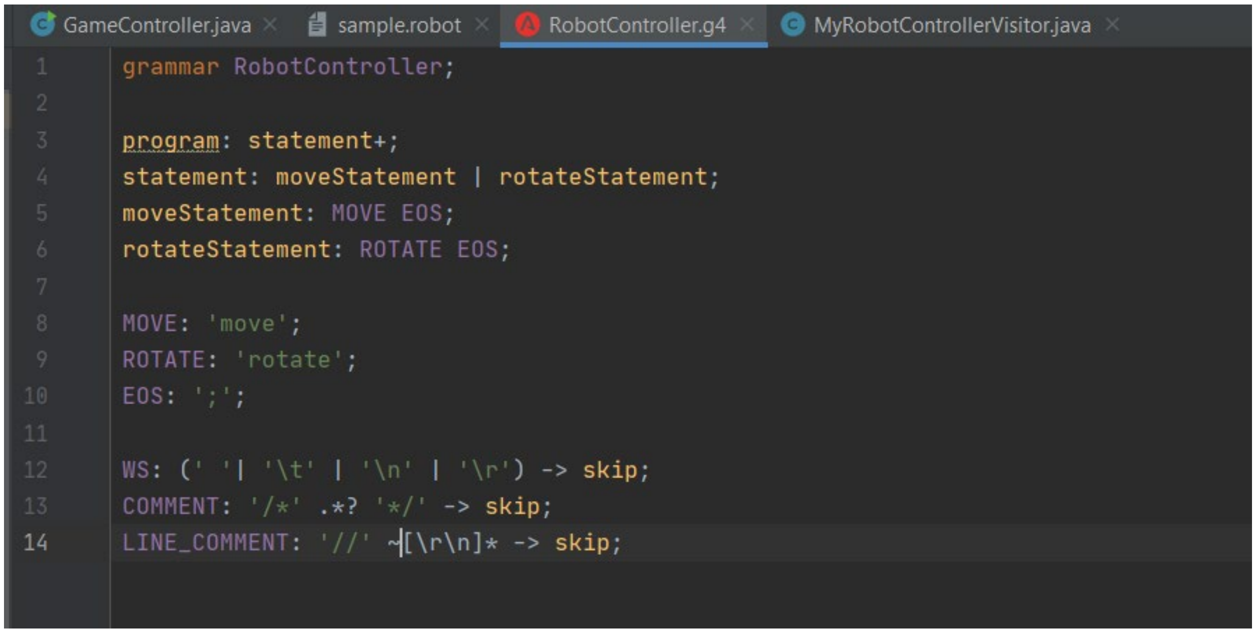
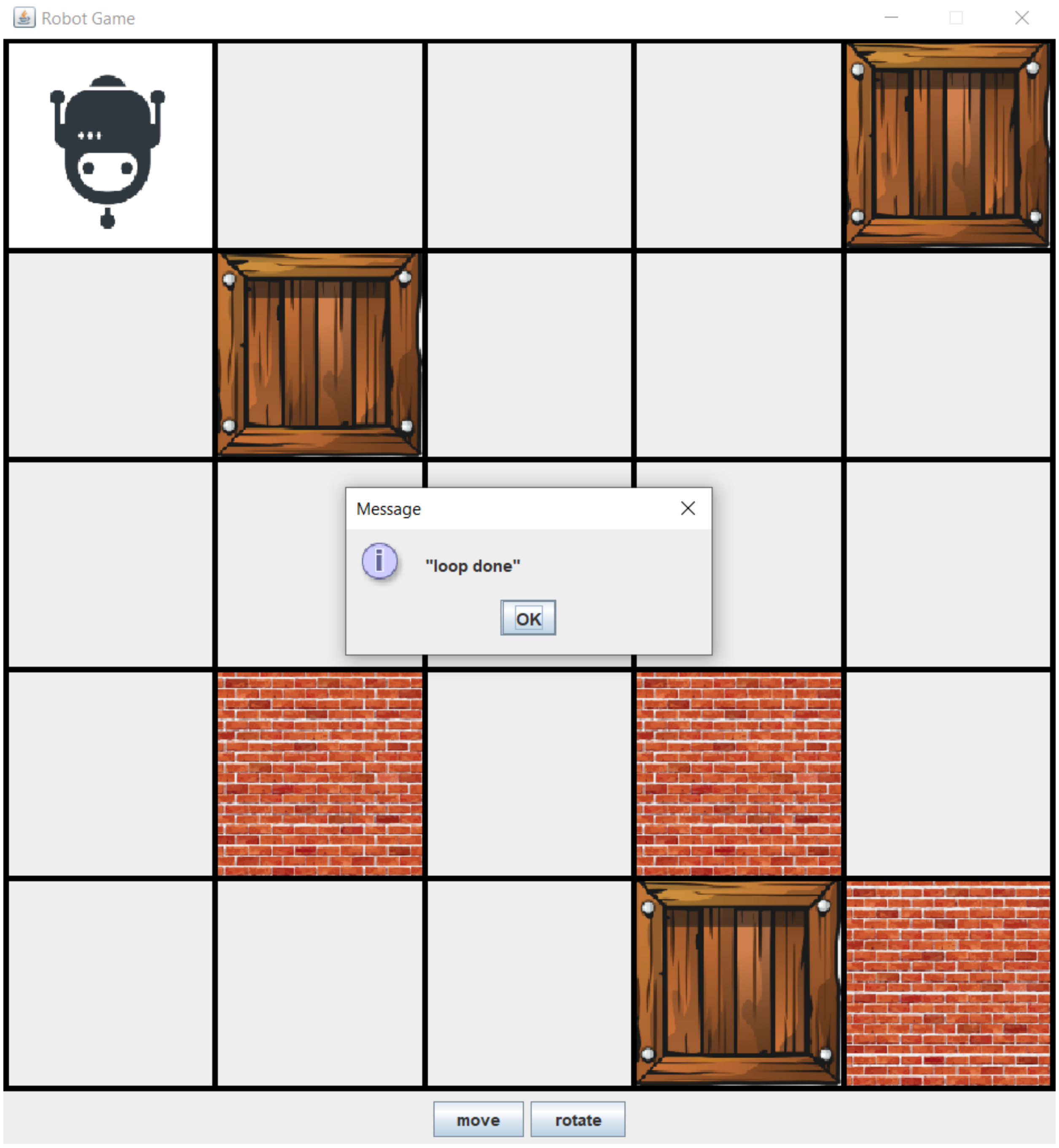


```
grammar RobotController;

program: statement+;
statement: moveStatement | rotateStatement;
moveStatement: MOVE EOS;
rotateStatement: ROTATE EOS;

MOVE: 'move';
ROTATE: 'rotate';
EOS: ';';

WS: (' '| '\t' | '\n' | '\r') -> skip;
COMMENT: '/*' .*? '*/' -> skip;
LINE_COMMENT: '//' ~[\r\n]* -> skip;
```



1. Önálló feladat

Visitor kiegészítéssel:

```
package language.controller;

import game.Direction;
import game.GameController;
```

```java
public class MyRobotControllerVisitor extends
RobotControllerBaseVisitor<Object>{
    private GameController controller;

    public MyRobotControllerVisitor(GameController controller){
        this.controller = controller;
    }

    @Override
    public Object
visitMoveStatement(RobotControllerParser.MoveStatementContext ctx) {
        for (int i = 0; i < Integer.parseInt(ctx.amount().INT().getText());
i++) {
            controller.move();
        }
        return super.visitMoveStatement(ctx);
    }

    @Override
    public Object
visitRotateStatement(RobotControllerParser.RotateStatementContext ctx) {
        int number_of_rotate = 1;
        String goal = (String)ctx.direction().STRING().getText();
        goal = goal.substring(1,goal.length()-1);
        if (controller.getPlayerFacing() == Direction.DOWN){
            switch (goal) {
                case "down":
                    number_of_rotate = 0;
                    break;
                case "left":
                    number_of_rotate = 1;
                    break;
                case "up":
                    number_of_rotate = 2;
                    break;
                case "right":
                    number_of_rotate = 3;
                    break;
            }
        }
        else if (controller.getPlayerFacing() == Direction.LEFT){
            switch (goal) {
                case "down":
                    number_of_rotate = 3;
                    break;
                case "left":
                    number_of_rotate = 0;
                    break;
                case "up":
                    number_of_rotate = 1;
                    break;
                case "right":
                    number_of_rotate = 2;
                    break;
            }
        }
        else if (controller.getPlayerFacing() == Direction.UP){
            switch (goal) {
                case "down":
                    number_of_rotate = 2;
                    break;
                case "left":
                    number_of_rotate = 3;
                    break;
                case "up":
                    number_of_rotate = 0;
                    break;
                case "right":
                    number_of_rotate = 1;
                    break;
            }
        }
        else if (controller.getPlayerFacing() == Direction.RIGHT){
            switch (goal) {
                case "down":
                    number_of_rotate = 1;
                    break;
                case "left":
                    number_of_rotate = 2;
                    break;
                case "up":
                    number_of_rotate = 3;
                    break;
                case "right":
                    number_of_rotate = 0;
                    break;
            }
        }
        for (int i = 0; i < number_of_rotate; i++){
            controller.rotate();
        }

        return super.visitRotateStatement(ctx);
    }
    @Override
    public Object
visitLoopStatement(RobotControllerParser.LoopStatementContext ctx) { Object
result =
            null;
        for (int i = 0; i < Integer.parseInt(ctx.amount().INT().getText());
i++) {
            for (RobotControllerParser.StatementContext stm :
ctx.statement()) {
            result = visit(stm);
        }
        }
        return result;
    }

    @Override
    public Object visitLogStatement(RobotControllerParser.LogStatementContext
ctx) {
        controller.displayMessage(ctx.logMessage().STRING().getText());
        return super.visitLogStatement(ctx);
    }




}
```

Nyelvtan kiegészítéssel:

```
grammar RobotController;

program: statement+;
statement: moveStatement | rotateStatement | loopStatement | logStatement;
moveStatement: MOVE (LPAREN amount RPAREN)? EOS;
rotateStatement: ROTATE (LPAREN direction RPAREN)? EOS;
direction: STRING;
loopStatement: LOOP (LPAREN amount RPAREN) LCURLY statement+ RCURLY;
amount: INT;
logStatement: LOG (LPAREN logMessage RPAREN)? EOS;
logMessage: STRING;


MOVE: 'move';
ROTATE: 'rotate';
EOS: ';';
LOOP: 'loop';
LOG: 'log';
LPAREN: '(';
RPAREN: ')';
LCURLY: '{';
RCURLY: '}';


WS: (' '| '\t' | '\n' | '\r') -> skip;
COMMENT: '/*' .*? '*/' -> skip;
LINE_COMMENT: '//' ~[\r\n]* -> skip;
INT: [0-9]+;
STRING: '"' (~[\r\n])* '"';
```

2. Önálló feladat

RobotLevelMaker.g4:

```
grammar RobotLevelMaker;

program: statement+;
statement: dimStatement | playerStatement | wallStatement | crateStatement;
dimStatement: DIM (LSQUARE XDIM RSQUARE) (LSQUARE XDIM RSQUARE) EOS;
XDIM: INT;
playerStatement: PLAYER (LSQUARE XDIM RSQUARE) (LSQUARE XDIM RSQUARE) EOS;

wallStatement: WALL (LSQUARE XDIM RSQUARE) (LSQUARE XDIM RSQUARE) EOS;
crateStatement: CRATE (LSQUARE XDIM RSQUARE) (LSQUARE XDIM RSQUARE) EOS;
```

```
DIM: 'DIM';
PLAYER: 'PLAYER';
EOS: ';';
WALL: 'WALL';
CRATE: 'CRATE';
LPAREN: '(';
RPAREN: ')';
LCURLY: '{';
RCURLY: '}';
LSQUARE: '[';
RSQUARE: ']';


WS: (' '| '\t' | '\n' | '\r') -> skip;
COMMENT: '/*' .*? '*/' -> skip;
LINE_COMMENT: '//' ~[\r\n]* -> skip;
INT: [0-9]+;
STRING: '"' (~[\r\n])* '"';
```

MyRobotLevelMakerVisitor

```
package language.controller;

import game.Coordinates;
import game.Direction;
import game.GameController;
import game.Model;
import org.stringtemplate.v4.misc.Coordinate;

public class MyRobotLevelMakerVisitor  extends
RobotLevelMakerBaseVisitor<Object>{
    private GameController controller;

    public MyRobotLevelMakerVisitor(GameController controller){
        this.controller = controller;
    }

    @Override
    public Object visitDimStatement(RobotLevelMakerParser.DimStatementContext
ctx) {
        int xdim = Integer.parseInt(ctx.XDIM().get(0).getText());
        int ydim = Integer.parseInt(ctx.XDIM().get(1).getText());
        Model model = controller.getModel();
        model.setBoardCols(xdim);
        model.setBoardRows(ydim);

        return super.visitDimStatement(ctx);
    }

    @Override
    public Object
visitPlayerStatement(RobotLevelMakerParser.PlayerStatementContext ctx) {
        int xdim = Integer.parseInt(ctx.XDIM().get(0).getText());
        int ydim = Integer.parseInt(ctx.XDIM().get(1).getText());
        Model model = controller.getModel();
        model.setPlayerX(xdim);
        model.setPlayerY(ydim);

        return super.visitPlayerStatement(ctx);
    }
    @Override
    public Object
visitWallStatement(RobotLevelMakerParser.WallStatementContext ctx) {
        int xdim = Integer.parseInt(ctx.XDIM().get(0).getText());
        int ydim = Integer.parseInt(ctx.XDIM().get(1).getText());
        Model model = controller.getModel();
        Coordinates coordinates =  new Coordinates(xdim, ydim);
        model.addWalls(coordinates);
        return super.visitWallStatement(ctx);
    }

    @Override
    public Object
visitCrateStatement(RobotLevelMakerParser.CrateStatementContext ctx) {
        int xdim = Integer.parseInt(ctx.XDIM().get(0).getText());
        int ydim = Integer.parseInt(ctx.XDIM().get(1).getText());
        Model model = controller.getModel();
        Coordinates coordinates =  new Coordinates(xdim, ydim);
        model.addCrates(coordinates);
        return super.visitCrateStatement(ctx);
    }
}
```

Model módosítva:

```
package game;
import java.util.ArrayList;
import java.util.List;

public class Model {
    private int boardRows = 5, boardCols = 5, squareSize = 150;
    private Coordinates player = new Coordinates(0, 0);
    private Direction playerFacing = Direction.DOWN;
    private ArrayList<Coordinates> crates = new ArrayList<Coordinates>() {
        {
//            add(new Coordinates(1, 1));
//            add(new Coordinates(3,4));
//            add(new Coordinates(4,0));
        }
    };
    private ArrayList<Coordinates> walls = new ArrayList<Coordinates>() {
        {
//            add(new Coordinates(1, 3));
//            add(new Coordinates(3,3));
//            add(new Coordinates(4,4));
        }
    };

    public int getPlayerX() { return player.x;}
    public int getPlayerY() {return player.y;}
    public void setPlayerX(int x) {player.x = x;}
    public void setPlayerY(int y) {player.y = y;}
    public Direction getPlayerFacing() {return playerFacing;}

    public void movePlayer() {
        Coordinates newCoord = new Coordinates(player.x, player.y);
        switch (playerFacing) {
            case DOWN: newCoord.y += 1; break;
            case UP: newCoord.y -= 1; break;
            case RIGHT: newCoord.x += 1; break;
            case LEFT: newCoord.x -= 1; break;
        }

        if (isValidPlayerCoordinate(newCoord)) {
            player = newCoord;
            checkCratePickup();
        }
    }

    public void rotatePlayer() {
        playerFacing = playerFacing.rotate();
    }

    private void checkCratePickup() {
        if (crates.stream().anyMatch(c -> c.equals(player))) {
            Coordinates coord = crates.stream().filter(c ->
c.equals(player)).findFirst().get();
            crates.remove(coord);
        }
    }

    private boolean isValidPlayerCoordinate(Coordinates newCoord) {
        return newCoord.x >= 0 && newCoord.x < boardRows && newCoord.y >= 0
&& newCoord.y < boardCols && walls.stream().noneMatch(c ->
c.equals(newCoord));
    }

    public int getBoardRows() {return boardRows; }
    public int getBoardCols() {return boardCols; }
    public void setBoardRows(int rows) {boardRows = rows; }
    public void setBoardCols(int cols) {boardCols = cols; }
    public int getSquareSize() {return squareSize; }
    public List<Coordinates> getCrates() {return crates;}
    public List<Coordinates> getWalls() {return walls;}
    public void addCrates(Coordinates crate) {crates.add(crate);}
    public void addWalls(Coordinates wall) {walls.add(wall);}
}
```

GameController módosítva:

```
package game;

import language.controller.*;
import org.antlr.v4.runtime.ANTLRInputStream;
import org.antlr.v4.runtime.CommonTokenStream;
```

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public class GameController {

    private Model model;
    private View view;

    private GameController() {
        model = new Model();
        parseLevelMakerFile();
        view = new View(model);

        view.getMoveButton().addActionListener(e -> move());
        view.getRotateButton().addActionListener(e -> rotate());

        parseControllerFile();
    }

    private void parseControllerFile() {
        try {
            File file = new File("res\\sample.robot");
            InputStream fileStream = new FileInputStream(file);
            ANTLRInputStream inputStream = new ANTLRInputStream(fileStream);
            RobotControllerLexer lexer = new
RobotControllerLexer(inputStream);
            CommonTokenStream tokens = new
                    CommonTokenStream(lexer);
            RobotControllerParser parser = new RobotControllerParser(tokens);
            RobotControllerParser.ProgramContext tree = parser.program();
            MyRobotControllerVisitor visitor = new
MyRobotControllerVisitor(this);
            visitor.visit(tree);
        } catch (IOException ignored) {
        }
    }

    private void parseLevelMakerFile() {
        try {
            File file = new File("res\\level.robot");
            InputStream fileStream = new FileInputStream(file);
            ANTLRInputStream inputStream = new ANTLRInputStream(fileStream);
            RobotLevelMakerLexer lexer = new
RobotLevelMakerLexer(inputStream);
            CommonTokenStream tokens = new
                    CommonTokenStream(lexer);
            RobotLevelMakerParser parser = new RobotLevelMakerParser(tokens);
            RobotLevelMakerParser.ProgramContext tree = parser.program();
            MyRobotLevelMakerVisitor visitor = new
MyRobotLevelMakerVisitor(this);
            visitor.visit(tree);
        } catch (IOException ignored) {
        }
    }
    public void move() {
        model.movePlayer(); view.refresh();
        try { Thread.sleep(500); }
        catch (InterruptedException ignored) {}
        checkWinCondition();
    }


    public void rotate() {
        model.rotatePlayer(); view.refresh();
        try { Thread.sleep(500); }
        catch (InterruptedException ignored) {}
    }


    public Direction getPlayerFacing() {
        return model.getPlayerFacing();
    }

    public void displayMessage(String message) {
        view.displayMessage(message);
    }

    private void checkWinCondition() {
        if (model.getCrates().isEmpty())
            view.win();
    }

    public static void main(String[] args) {
        new GameController();
    }

    public Model getModel(){
        return this.model;
    }
}
```

3. Önálló feladat

Visitor kiegészítései:

```java
package language.controller;

import game.Coordinates;
import game.Direction;
import game.GameController;
import game.Model;
import org.stringtemplate.v4.misc.Coordinate;

public class MyRobotLevelMakerVisitor  extends
RobotLevelMakerBaseVisitor<Object>{
    private GameController controller;

    public MyRobotLevelMakerVisitor(GameController controller){
        this.controller = controller;
    }

    @Override
    public Object visitDimStatement(RobotLevelMakerParser.DimStatementContext
ctx) {
        int xdim = Integer.parseInt(ctx.XDIM().get(0).getText());
        int ydim = Integer.parseInt(ctx.XDIM().get(1).getText());
        Model model = controller.getModel();
        model.setBoardCols(xdim);
        model.setBoardRows(ydim);

        return super.visitDimStatement(ctx);
    }

    @Override
    public Object
visitPlayerStatement(RobotLevelMakerParser.PlayerStatementContext ctx) {
        int xdim = Integer.parseInt(ctx.XDIM().get(0).getText());
        int ydim = Integer.parseInt(ctx.XDIM().get(1).getText());
        Model model = controller.getModel();
        model.setPlayerX(xdim);
        model.setPlayerY(ydim);

        return super.visitPlayerStatement(ctx);
    }
    @Override
    public Object
visitWallStatement(RobotLevelMakerParser.WallStatementContext ctx) {
        Model model = controller.getModel();
        System.out.println(ctx.XDIM());
        String x = ctx.XDIM().get(0).toString();
        String y = ctx.XDIM().get(1).toString();
        if(x.contains("-")){
            if(y.contains("-")){
                for (int i = Integer.parseInt(x.split("-")[0]);
i<=Integer.parseInt(x.split("-")[1]); i++){
                    for (int j = Integer.parseInt(y.split("-")[0]);
j<=Integer.parseInt(y.split("-")[1]); j++){
                        Coordinates coordinates =  new Coordinates(i, j);
                        model.addWalls(coordinates);
                    }
                }
            }
            else {
                for (int i = Integer.parseInt(x.split("-")[0]);
i<=Integer.parseInt(x.split("-")[1]); i++){
                        Coordinates coordinates =  new Coordinates(i,
Integer.parseInt(y));
                        model.addWalls(coordinates);
                }
            }
        }
        else {
            if(y.contains("-")){
                    for (int j = Integer.parseInt(y.split("-")[0]);
j<=Integer.parseInt(y.split("-")[1]); j++){
                        System.out.println(j);
                        Coordinates coordinates =  new
Coordinates(Integer.parseInt(x), j);
                        model.addWalls(coordinates);
                    }
            }
            else {
                    Coordinates coordinates =  new
Coordinates(Integer.parseInt(x), Integer.parseInt(y));
                    model.addWalls(coordinates);
            }
        }

        return super.visitWallStatement(ctx);
    }

    @Override
    public Object
visitCrateStatement(RobotLevelMakerParser.CrateStatementContext ctx) {
        int xdim = Integer.parseInt(ctx.XDIM().get(0).getText());
        int ydim = Integer.parseInt(ctx.XDIM().get(1).getText());
        Model model = controller.getModel();
        Coordinates coordinates =  new Coordinates(xdim, ydim);
        model.addCrates(coordinates);
        return super.visitCrateStatement(ctx);
    }
}
```

nyelvtan kiegészítései:

```
grammar RobotLevelMaker;

program: statement+;
statement: dimStatement | playerStatement | wallStatement | crateStatement;
dimStatement: DIM (LSQUARE XDIM RSQUARE) (LSQUARE XDIM RSQUARE) EOS;
XDIM: INT;
playerStatement: PLAYER (LSQUARE XDIM RSQUARE) (LSQUARE XDIM RSQUARE) EOS;

wallStatement: WALL (LSQUARE (XDIM) RSQUARE) (LSQUARE (XDIM) RSQUARE) EOS;
crateStatement: CRATE (LSQUARE XDIM RSQUARE) (LSQUARE XDIM RSQUARE) EOS;


DIM: 'DIM';
PLAYER: 'PLAYER';
EOS: ';';
WALL: 'WALL';
CRATE: 'CRATE';
LPAREN: '(';
RPAREN: ')';
LCURLY: '{';
RCURLY: '}';
LSQUARE: '[';
RSQUARE: ']';


WS: (' '| '\t' | '\n' | '\r') -> skip;
COMMENT: '/*' .*? '*/' -> skip;
```

```
LINE_COMMENT: '//' ~[\r\n]* -> skip;
INT: [0-9]+|[0-9]'-'[0-9]+;
STRING: '"' (~[\r\n])* '"';
```