**Class House:**

**members (private):**

Matrix<char> floor

… all the rest

**functions:**

intGetDirtLevel(point position)

void SetDirtLevel(point position, int dirtLevel)

// set dirtlevel to be dirtlevel --if its 1+

//returns 1 if cleaned 0 if not cleaned

int Clean(point position)

bool IsWall(point position)

bool IsDockingStation(point position )//also battery charge

bool IsPositionValid(Point position)

bool IsClean()

Point GetDockingStation()

int SumOfDirtInTheHouse

ReadHouseFromFile(std∷string fileName)

**Class Sensor implement AbstractSensor**

**members (private):**

const House & currentHousePointer

const Point & currentPositionPointer

**functions:**

Sensor( const House & h, const Point & p) // constuctor – can't change

void MoveCurrentPoint(Direction d) // assume move is possible

SensorInfromation Sense() //use house and point

**Class RandomRobotAlgorithm implements algorithm – includes random step()**

**members (private):**

const AbstractSensor robotSensor;

map<string,int> configs;

const battery & robotBattery;

**functions:**

void SetSensor(const AbstractSensor & sesor)

void SetConfiguration(map<string,int> configs)

Direction step()

void AboutToFinish(StepTillFinishing)

**Class Battery**

**members**

int batteryLevel

int rechargeRate

int consumptionRate

functions:

bool IsEmpty()

GetBatteryLevel()

void Recharge() //fill the battery

void Consume() //eats the battery

**Class AlgorithmSingleRun**

**members(private):**

int simulationSteps; // steps commited in the run

Algorithm currentAlgorithm;

Battery robotBattery;

House currentHouse;

Point currentPosition;

bool valid;

int dirtColleIsWall(pointsAfterMoving[i])cted;

**functions:**

AlgorithmSingleRun(house h, algorithm a, battery b, const map〈string,int〉 & configs) //constuctor

{

currentPosition = house. GetDockingStation();

currentAlgorithm = a;

currentAlgorithm.SetSensor(new Sensor(house , currentPosition) );

currentAlgorithm. SetConfiguration(map〈string,int〉 configs))

robotBattery = b;

…

}

void DoStep() //Pre: IsStillValid(), ! robotBattery.IsEmpty()

Direction d = algorithm.step();

SetCurrentPosition(d);

robotBattery.consume();

simulationSteps++;

valid = currentHouse. IsPositionValid(currentPosition)

If(valid)

{

if(house.IsDockingStation(currentPosition)

robotBattery.Recharge()

else

dirtCollected += currentHouse.clean(currentPosition );

}

bool CanDoStep()

{

return IsStillValid() && ! robotBattery.IsEmpty() && ! IsHouseCleaned())

}

bool IsHouseCleaned()

bool IsStillValid() // case of hitting a wall

void SetCurrentPosition(Direction d)

int GetScore()

bool IsBackInDocking()

**Class Simulator**

**members(private):**

// battery according to the config file – should be copied to each **AlgorithmSingleRun**

Battery startingBattery

std∷vector<House> houses

std∷vector< algorithm> algorithms

std∷map<string, int> configs

std∷vector< AlgorithmSingleRun> runs

**functions:**

// when initiallize algorithm, the algorithm initalize it's field sensor

void initialize(std∷string configFilePath, std∷string houseFolderPath)

{

foreach(fileName in houseFolderPath){

house \* h = new house()

h.ReadHouseFromFile(fileName)

houses.push\_back(h);

}

algorithms.push\_back(new **RandomRobotAlgorithm())**

ReadConfigFromFile(configs , configFilePath)

}

void ReadConfigFromFile(std∷map<string, int> & configMap, const string & configFilePath);

void SimulateAll()

{

foreach(house h, algorithm a)

{

runs.pushback(new AlgorithmSingleRun(h, a, startingBattery, configs)

}

int iterations = 0;

int maxsteps = GetMaxSteps();

while( ! AnyAlgorithmFinsihed() && iterations < maxsteps)

{

forach(r in runs)

{

if(r.CanDoStep())

{

r.DoStep();

}

}

}

if(iterations != maxsteps)

{

iterations = 0;

int maxStepsAfterWinner = GetMaxStepsAfterWinner();

}

}

int GetMaxStepsAfterWinner();

int GetMaxSteps();

void AnyAlgorithmFinished()