Design Explanation:

The flow of our design is such that main function gets as argument the house struct and the rest of the required argument, add wall to the house struct and parse it to a matrix with entries containing the dirt level, and possible directions.

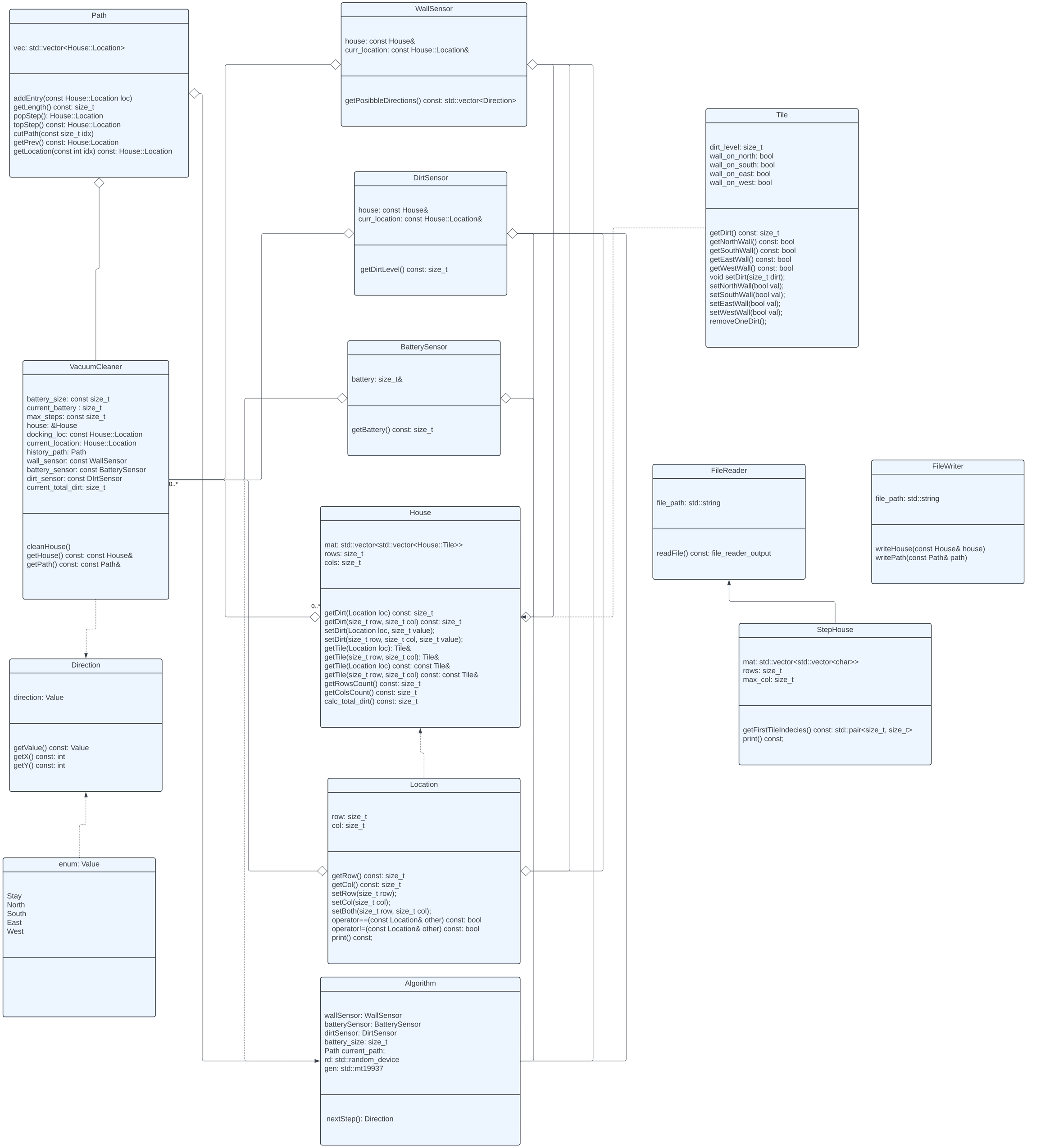
Then we call “cleanHouse()” function. We run this function for a maximum steps received as an argument or until the house is completely clean. On each iteration we call the decider algorithm used as an API that uses the sensors to queries the amount of dirt, the battery level and possible directions. By calculating this data, the algorithm then decides if the vacuum cleaner should move and to which direction and the algorithm execute this move. The algorithm also takes in consideration the amount of battery and return to the docking station for charging if needed.

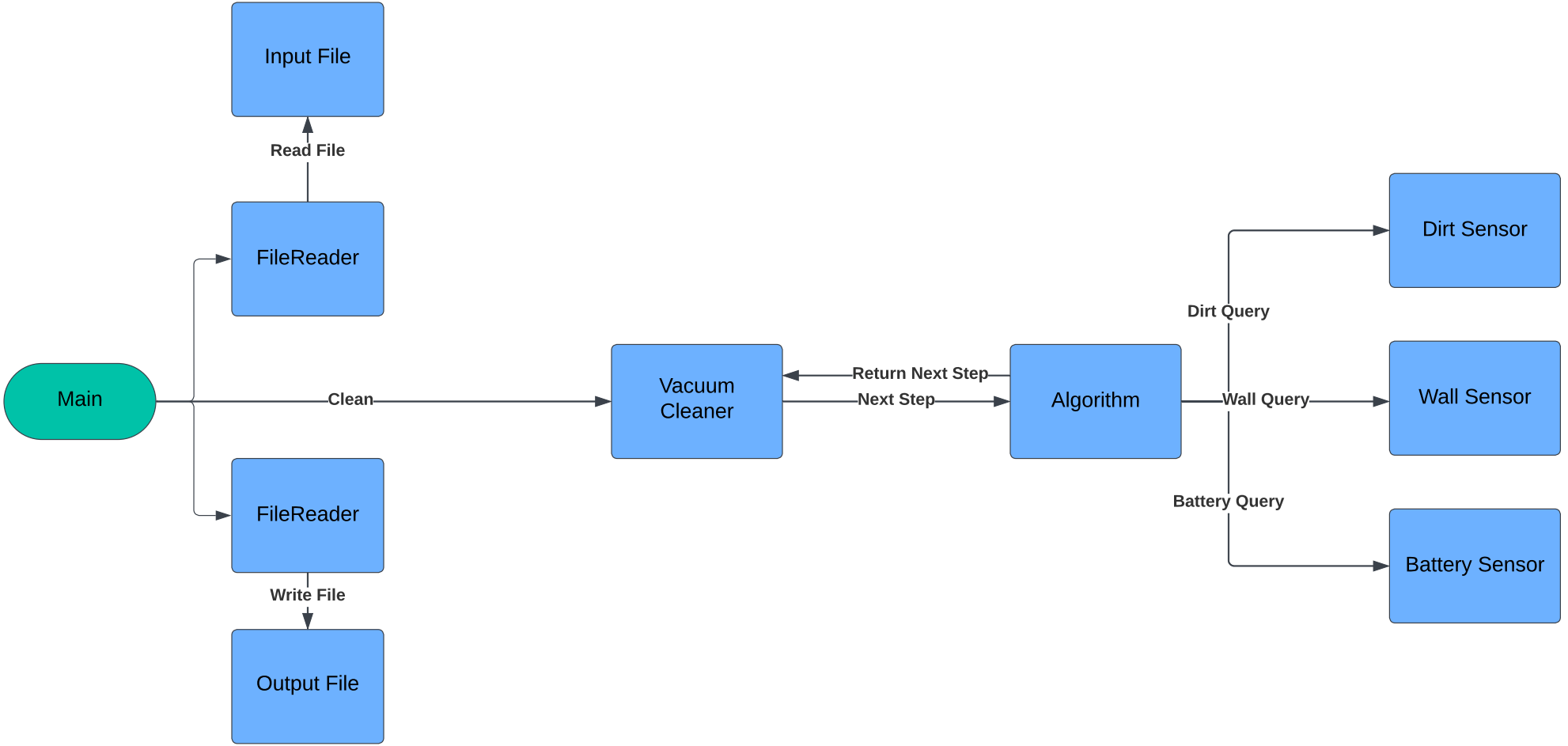
After the function has done, we write all the required results to an output file.

Testing Approach:

We used unit tests for each class, during the development cycle.

At the end we tested E2E with several use cases.


Path
vec: std::vector<House::Location>
addEntry(const House::Location loc)
getLength() const: size_t
popStep(): House::Location
topStep() const: House::Location
cutPath(const size_t idx)
getPrev() const: House:Location
getLocation(const int idx) const: House::Location
WallSensor
house: const House&
curr_location: const House::Location&
getPosibbleDirections() const: std::vector<Direction>
Tile
dirt_level: size_t
wall_on_north: bool
wall_on_south: bool
wall_on_east: bool
wall_on_west: bool
getDirt() const: size_t
getNorthWall() const: bool
getSouthWall() const: bool
getEastWall() const: bool
getWestWall() const: bool
void setDirt(size_t dirt);
setNorthWall(bool val);
setSouthWall(bool val);
setEastWall(bool val);
setWestWall(bool val);
removeOneDirt();
DirtSensor
house: const House&
curr_location: const House::Location&
getDirtLevel() const: size_t
BatterySensor
battery: size_t&
getBattery() const: size_t
VacuumCleaner
battery_size: const size_t
current_battery : size_t
max_steps: const size_t
house: &House
docking_loc: const House::Location
current_location: House::Location
history_path: Path
wall_sensor: const WallSensor
battery_sensor: const BatterySensor
dirt_sensor: const DIrtSensor
current_total_dirt: size_t
cleanHouse()
getHouse() const: const House&
getPath() const: const Path&
0..*
House
mat: std::vector<std::vector<House::Tile>>
rows: size_t
cols: size_t
0..*
getDirt(Location loc) const: size_t
getDirt(size_t row, size_t col) const: size_t
setDirt(Location loc, size_t value);
setDirt(size_t row, size_t col, size_t value);
getTile(Location loc): Tile&
getTile(size_t row, size_t col): Tile&
getTile(Location loc) const: const Tile&
getTile(size_t row, size_t col) const: const Tile&
getRowsCount() const: size_t
getColsCount() const: size_t
calc_total_dirt() const: size_t
FileReader
file_path: std::string
readFile() const: file_reader_output
FileWriter
file_path: std::string
writeHouse(const House& house)
writePath(const Path& path)
StepHouse
mat: std::vector<std::vector<char>>
rows: size_t
max_col: size_t
getFirstTileIndecies() const: std::pair<size_t, size_t>
print() const;
Direction
direction: Value
getValue() const: Value
getX() const: int
getY() const: int
enum: Value
Stay
North
South
East
West
Location
row: size_t
col: size_t
getRow() const: size_t
getCol() const: size_t
setRow(size_t row);
setCol(size_t col);
setBoth(size_t row, size_t col);
operator==(const Location& other) const: bool
operator!=(const Location& other) const: bool
print() const;
Algorithm
wallSensor: WallSensor
batterySensor: BatterySensor
dirtSensor: DirtSensor
battery_size: size_t
Path current_path;
rd: std::random_device
gen: std::mt19937
nextStep(): Direction


Input File
Read File
FileReader
Main
Clean
FileReader
Write File
Output File
Vacuum Cleaner
Return Next Step
Next Step
Algorithm
Dirt Query
Wall Query
Battery Query
Dirt Sensor
Wall Sensor
Battery Sensor