**Rationale**

The Robot Vacuums are gaining popularity as prices are coming down. There needs to be a method to determine the effectiveness of the algorithm used by the robot so adjustments can be made to the software and/or hardware to improve performance.

This is a four (4) group project.

**Features**

There are three major objects of this project:

1. House object
2. Vacuum object
3. Simulation runner and data reviewer

House object

Allows for a default house with rooms, doors and obstructions. The user, via a GUI, can modify the house, rooms, doors and obstructions.

House size has to be at least 200 square feet with a maximum of 8,000 square feet.

Rooms size – Smallest room (i.e., a closet) is 4 square feet. Largest room is the maximum of the current house size. Each room has to have to at least one door (entry).

Floor covering – Listed in order of the easiest to hardest to clean: Hard (wood, laminate, tile, etc.), Loop Pile (Berber pile), Cut Pile, Frieze-cut pile (California shag). The floor covering is to be applied to the whole house. The simulation shall provide a visual display of the floor covering being used.

Obstructions can be selected and placed by the user via a GUI. There are two types:

Chairs/tables – four legs with fixed diameter with variable space between. Vacuum can go under

Chests – cannot go under.

Calculate total square feet, space that is covered that cannot be cleaned. This space should not be included in the cleanness calculations.

Vacuum object

Allows for a default vacuum which has a minimum set of attributes which should include:

Size: Fixed at 12.8 inches in diameter

Vacuum width: Fixed at 5.8 inches

Vacuum efficiency: Dependent on the floor covering. Must be selectable by the user based on the house’s floor covering.

Whiskers (vacuum extension) width: Fixed at 13.5 inches

Whiskers efficiency: Default at 30% effective

Speed in inches/second: Default at 3 inches/second

Battery life: Default at 150 minutes.

Path: Random, Spiral, Snaking, Wall follow. Default is Random

The vacuum will run until the battery life has been depleted or stopped by the user.

Simulation Set-up:

At the start of the program the user shall be allowed to perform the following:

1. Floor Plan:
   1. Recall a saved floor plan to be used for the simulation.
   2. Create a new floor plan (default floor plan is 2000 ft2 with project team defined number of rooms and layout of obstructions).
   3. Edit either a recalled or new floor plan. Can change size, rooms, floor covering and obstructions. Note: previously saved floor plans cannot be overwritten unless the user is notified and selects to overwrite an existing file.
2. Robot Setup:
   1. Battery life: User selectable. The minimum time is 90 minutes and the maximum is 200. Default is 150 minutes
   2. Vacuum efficiency: User selectable based on the floor covering type. The range can run for the vacuum from 10% to 90% and the Whiskers from 10% to 50%. Defaults for the floor covering type are:
      1. Hard 90%
      2. Loop Pile 75%
      3. Cut Pile 70%
      4. Frieze-cut pile 65%
   3. Robot Speed: Default is 12 inches per second. The range can run from 6 inches per second to 18 inches per second.
   4. Pathing algorithm. Can select specific algorithm or all. Default is all.

Simulation

Run in real-time x1, x5, x50 so that user can observe simulation and coverage with multiple runs and different pathing algorithms. The simulation will stop when all the selected pathing algorithms are run.

Automatically save the simulation results when the simulation run is completed.

Allow the user to stop the simulation before the time limit is reached and save results.

Simulation results are displayed using color coding to show coverage (dirtiest to cleanest)

Save results into database to include:

* Time/Date stamp of start of run.
* Pathing algorithm type and current version of the algorithm
* Floor plan unique identification
* Coverage percentage.
* Vacuum running time.

Pull up results from previous runs.

With the same floor plan allow summary of multiple runs in one display.

Note: Data does not have to be stored in a “database”. This is an implementation or design decision that needs to be made by the team.

**Constraints**

The choice of programming language and database is left up to the design team.

Users must be able to run the program on a PC under Windows.

The system will be considered to be open source and in the public domain, therefore all code must be original and may not include any copyrighted material.

**Student Feedback from previous semesters.**

The following information was provided by students at the end of the term and should be considered as lessons learned. Some of this information may or may not be useful because the project requirements may have changed from previous terms.

**Fall 2019**

Start work early - stay slightly ahead of the lectures.  Take the risk management planning very seriously.

Try to get the graphics working as soon as possible.  I knew a lot about the graphics side of things and had a working prototype in a week, but for those who have never touched graphics, it is not always simple.

Make sure to cement communications early, and ensure everyone is always doing something.

While I understand that the requirements were intentionally vague, this occasionally left us confused and misguided.

**Spring 2020**

Consider using a game engine or game engine library.

You don't have to maximize the algorithms' efficiencies, but make sure they function correctly

The vacuum should have a constant velocity (focus on spiral, don't let it get faster!)

Take into careful consideration whether to use Unity, Java, etc. Map out the pros and cons early on and make a decision.

As with any project START EARLY. You will run into so many bugs in the late stages so save time to deal with those.

Try having each team member take a pathing algorithm. Then whoever creates the best solution, use that same style for all the algorithms.

Make sure you constantly integrate your code with each other and try not to run into merge conflicts.

I encourage a "hack day" early on in the first month of the semester for planning. These hack days should be an 8-hour day of working together on the project. I would recommend having at least three throughout the semester.

Buckle up. This project is going to be harder than you think. It seems intimidating and GUI heavy, and, trust me, it is.

AVOID UNITY unless you're really confident in your skills with using the program.

Plan, plan, plan, plan from the moment you get this project.

Work all the time. Even when you don't think you have anything to work on, find something to do. You'll thank yourself for it later.

Continuous integration is key; put together a beta release as soon as you can so you can determine what bugs exist so that you can fix them.

Research. There's a lot of information on the requirements for the project. Break them all down farther and ask questions. You need to make sure everything is as clear as possible before you dive into things, and the algorithms are not as easy as you may think.

Pair programming will more than likely be beneficial to you, especially with the algorithms.

Test this simulation like you've never used a computer before. Go nuts. (You'll have to fix the issues, but it'll be helpful later).

Just don't use Unity. We had a student from a different project convince the other team members that it was going to "only take like 3 weeks" but it proved to have many challenges when it came to speeding up the simulation. Utilize JavaFX and it will be fine. It's easy to get scared about detecting collisions, but it's not as bad as you think when you get into detecting if a point lies within another shape.

Beyond that, start early and make sure to ask a lot about the spiral algorithm. It's definitely the hardest so don't waste time on it. If you get that one and Wall-follow the others are easy.

**Spring 2021**

When you starting doing the project, maybe you should make sure each of your teammates knows what is their responsibility throughout the entire implementation.

Make sure to balance work between team mates.

Java GUIs require a lot of work to tweak.

Java GUIs can be cumbersome. Research limitations before deciding to use them.

Make sure your GUI looks good, not just functional. Think of the software you use daily - aim for it to be more polished like that. Also please watch out for flickering in your display, that gets very annoying and looks bad in general.

It would be good to know how to use Netbeans' Github feature if your team decides to use Github (if your team also decides to work in Java)! It makes committing changes and fetching updates super efficient because you can do it right from your IDE!