Contents Page

Introduction 2

Research 2

Analysis 5

IPSO Diagram 11

Data Dictionary 12

Class Diagram 13

Requirements 15

Design 17

IPSO Diagram 17

Pseudo Code 18

Class Diagram 22

Interface Design 23

Data Definition Diagram 24

Testing 25

Boiler Plates 27

Forms 28

Menu 28

Simulation 28

Results 29

Erase Warning 29

Evaluation of Objectives 30

Description of improved algorithm 37

Description of Control algorithm 37

Finite state machine of improved algorithm 38

Possible improvements 37

Results 39

Conclusion 42

Appendix 43

Introduction:

My investigation is to see if there is a more efficient algorithm for a robotic vacuum cleaner to clean a floor in an unknown room. I will create a simulation to test a more efficient algorithm for covering the area in random rooms. Tests will be carried out to conclude if I have succeeded in creating a more efficient algorithm and I will evaluate the drawbacks and benefits of my system and how it could be implemented and possibly developed

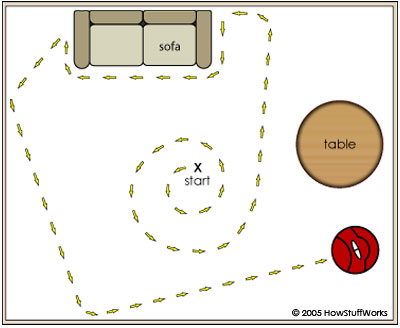
Research:

The research was conducted to discover how current automated robotic vacuum cleaners cover the floor of the room they are placed in and possible alternative algorithms for sweeping out an area.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of Source | Source | Date Accessed | Summary of Source | Used  Y/N | Reliability |
| Web | http://electronics.howstuffworks.com/gadgets/home/robotic-vacuum2.htm | 14/9/16 | A description of how Roomba robotic vacuums follow a route and how the algorithm appears to work. Starting in a spiral and moving to the outer perimeter of the room avoiding obstacles. Then moving seemingly without a planned path until stopped. This is the algorithm I will try to improve. | Y | This article is old, from November 2003, but it is from a reliable website and the algorithm will not have changed too drastically. |
| Book | AQA A-Level Computer Science By Bob Reeves Page 93 | 15/9/16 | An explanation of different ways to traverse graphs in discrete maths. | Y | This textbook has been approved by the AQA exam board so must be reliable |
| Web | http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html | 15/9/16 | How A\* path finding is a composite of Dijkstra’s algorithm and Greedy Best Fit First algorithm, and how the ratio of the 2 affects the outcome. | N | This is from the Stanford University website so will be reliable as it is from an educational source |
| Web | http://www.ucl.ac.uk/~ucahbtw/docs/d1lesson2/primswithtable\_notes.pdf | 17/9/16 | Notes with the definition of Prims algorithm and how it is used on a distance matrix | Y | From the university College London website which is highly regarded academic source which means it will be reliable |
| Web | http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/kruskalAlgor.htm | 17/9/16 | In depth explanation of Krushkal’s algorithm but it does not seem as useful or optimized as Prim’s algorithm due to edges having to be rejected if they create loops | N | I am unable to validate the reliability of the source as the website seems to be someone’s personal website |
| Web | http://mathworld.wolfram.com/AdjacencyMatrix.html | 17/9/16 | A description of how graphs in discrete maths can be represented in matrices which will make programming a solution for this problem easier. | Y | This information is from a well-regarded scientist Stephen Wolfram so will be regarded as reliable. |
| Forum | http://robotics.stackexchange.com/questions/628/what-algorithm-should-i-implement-to-program-a-room-cleaning-robot | 18/9/16 | A forum post discussing algorithms for robots to cover an area. It contains currently used algorithms for known and unknown areas, with and without obstacles. Also provides pictures of how current robots path. | Y | Due to the source being a forum the reliability cannot be determined as it is open to anyone but some posts have presented sources of their information so will most likely be reliable |
| Magazine | Which: September 2016  Page 60 | 19/9/16 | Article describing the algorithms of different robotic vacuums and how the more sophisticated ones have more efficient patching by minimising the area repeated. The more sophisticated algorithm took quarter of the time the less sophisticated robot took. | Y | The source is from a company trying to encourage consumers to buy products so it could be bias but the information is also recent so should be reliable. |
| Book | Edexcel Decision 1 Textbook | 19/9/16 | A chapter about the Chinese post man problem and whether a graph is traversable by only crossing each arc once, this makes it eulerian. A graph can be semi eulerian if some arcs have to be crossed twice. This information will help me optimise the path of a known room by factoring in this problem into the code | Y | This is a reliable source because it is a published textbook from a reputable educational exam board. |
| Web | http://www.explainthatstuff.com/how-roomba-works.html | 21/9/16 | Facts about how the Roomba robot vacuum works with each component | Y | This source should be reliable as it was created recently on 29/1/16 |

Analysis:

I am investigating how a robot vacuum can most efficiently sweep out a room with obstacles. This is the analysis of my research where describe what each piece of research has told me and decide which features are beneficial and which ones are not.

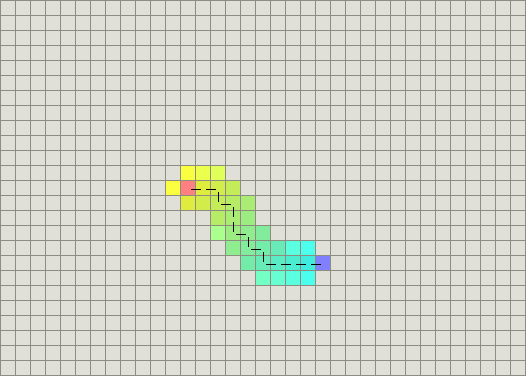


This is the algorithms for some robots, normally the cheaper ones. It involves spiralling outwards to cover as much area as possible and then randomly changing directions when an obstacle blocks its path. It is very inefficient due to the random reactions to obstacles. This will be what I will be using as a basis for my control algorithm.

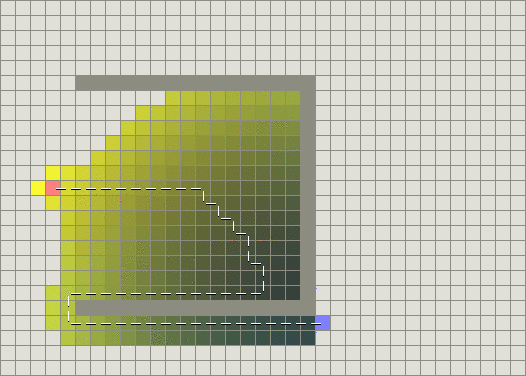
“Depth first is a method that explores as far in to the graph as possible before back tracking to the unvisited nodes. It is often implemented using a recursive algorithm”

“Breadth first is a method for traversing a graph that visits the nodes close to the starting point first. A queue is used to keep track of nodes to visit.

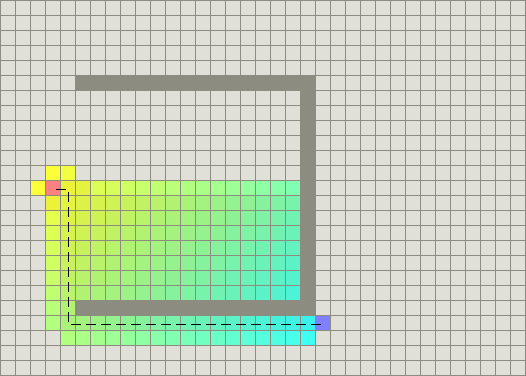
These are different logical methods of traversing a whole graph and I will have to choose which of these methods my algorithm approaches the problem if I use a graph based system to travel around the room.

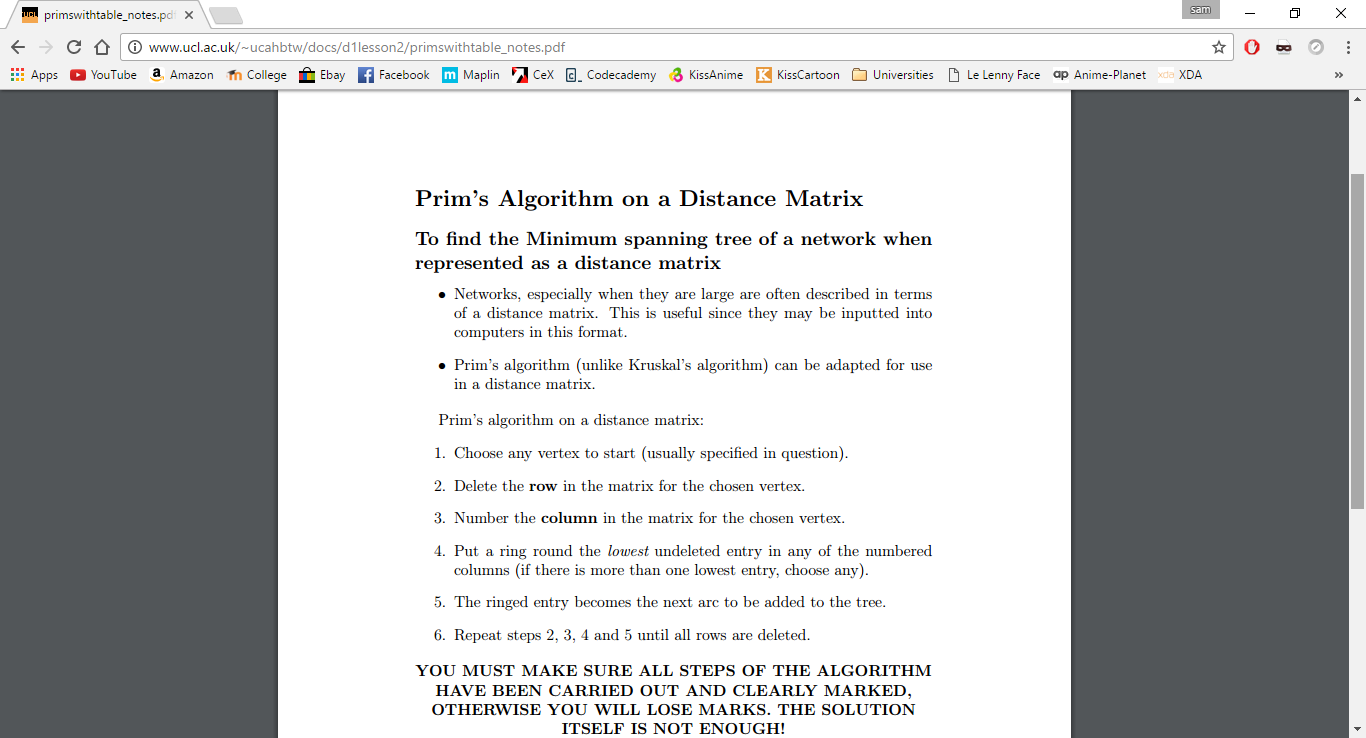
Greedy Best-First-Search: This algorithm is efficient in the fact it is simple and requires less processing but often produces longer paths if there are obstacles in the way

Dijkstra’s algorithm: This requires more processing to reach a destination but can deal with obstacles better leading to more efficient paths being found.



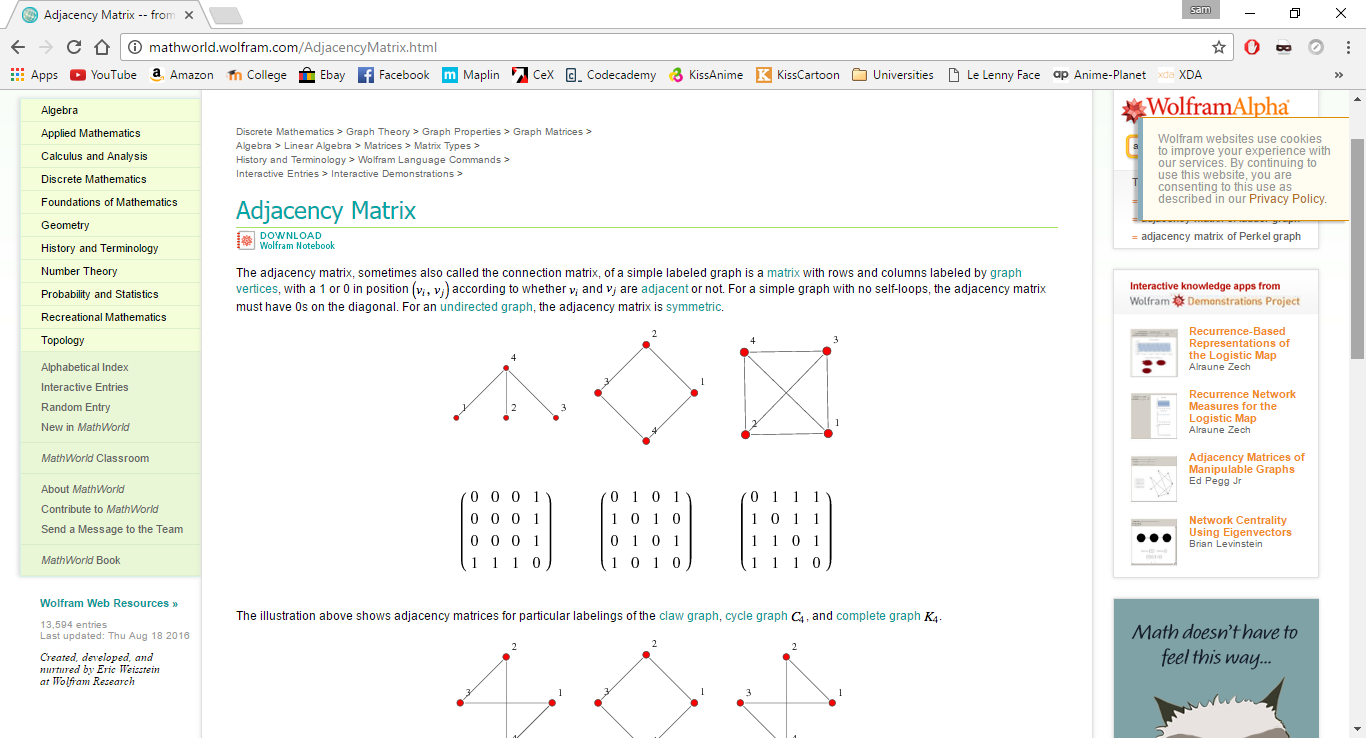
These two algorithms combine to form a\* path finding which will be useful for finding the shortest path and can be programmed to help me reduce the amount of unnecessarily repeated floor space and locate the charger but the problem is that the path is only able to be taken through known areas so will not be useful for the initial sweep



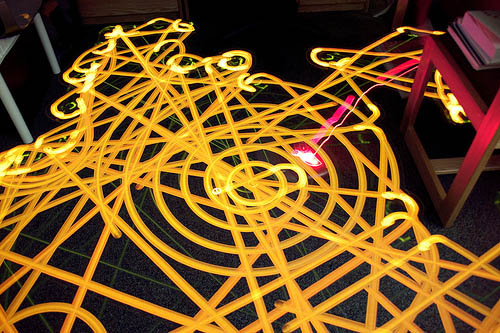


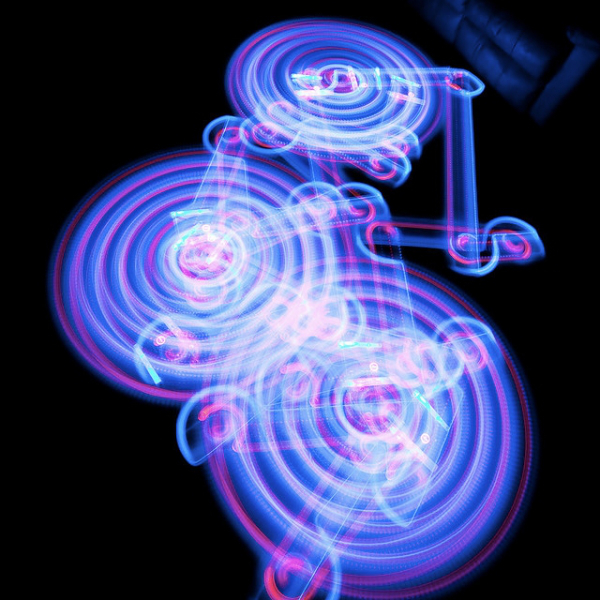
Knowing how prims algorithm works and how it can be applied to a distance matrix will be incredibly useful for applying to an algorithm for navigating a known room if the room is stored as a distance matrix.

Distance graphs on a matrix could allow me to map out an area in the form the computer can read. But this requires particular nodes to be chosen in the room so might not provide enough detail to represent an accurate room.



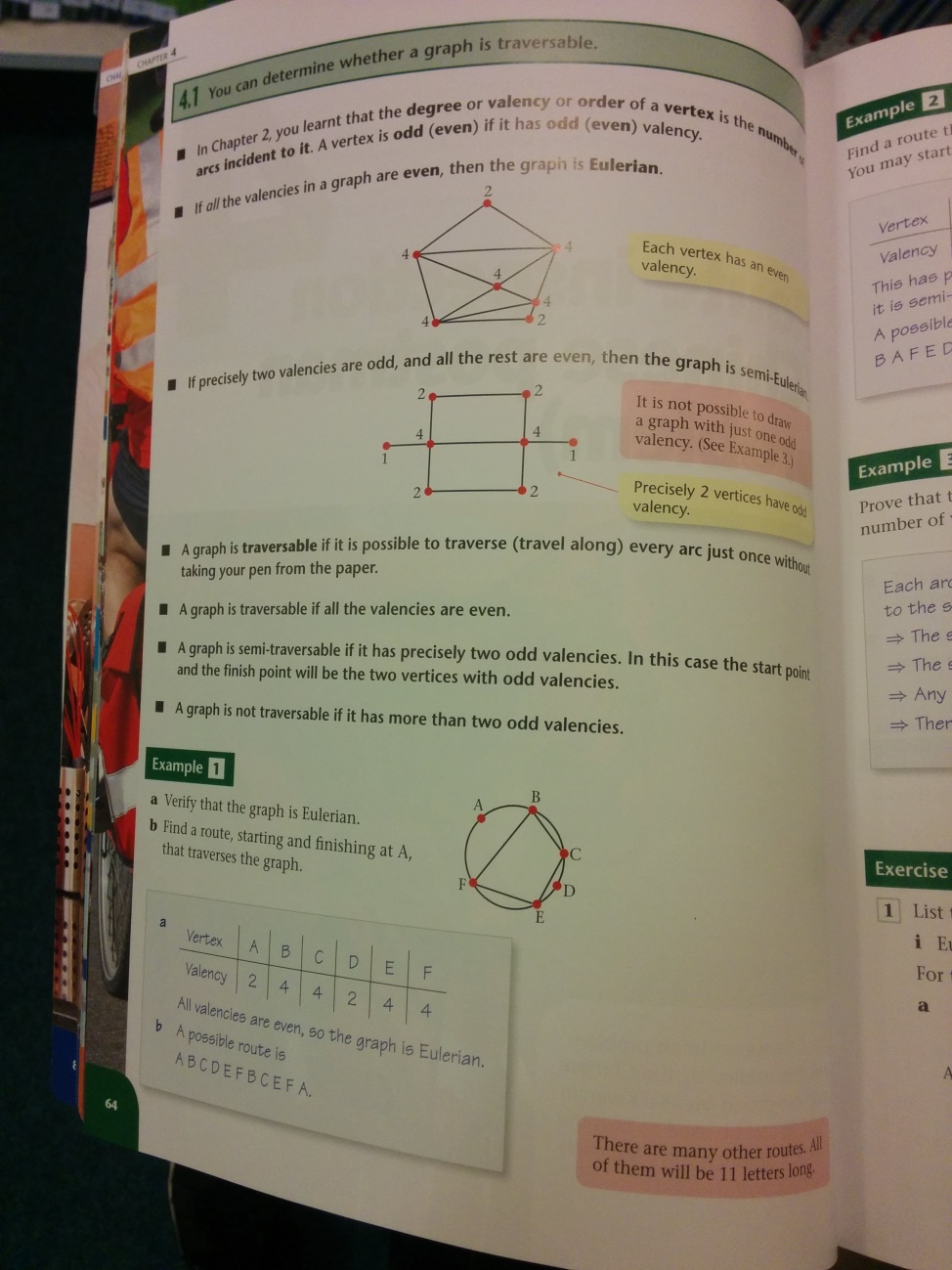
These images were created by using a long exposure on a camera and attaching led’s to a robot vacuum. As you can see they spiral allot and re-cover a lot of area that has already been cleaned.

A visualisation of the current algorithm shows how inefficient current algorithms are and gives me a better idea how to program the control along with a reference to the type of path the control algorithm will take



The picture below shows two extremes of algorithm based pathing where the left is very efficient and the right looks completely random.

The left image gives the impression that traveling in straight lines and turning right angles is the most efficient path for covering the most area efficiently. Whereas the right side has a chaotic path produced uneven coverage.

This page is about calculating the most efficient minimum spanning tree by only repeating the shortest arcs and only if necessary, demonstrating that shortest paths are achieved by repeating as little area as possible.

Input, Process, Stored, Output Diagram of the current system

This shows the different aspects of a robot vacuum system broken down in to different categories.

Processes:

Leave charging station when on button is pressed

Continue forward until obstacle is reached

If there is an obstacle in front: turn until there is not an obstacle.

If floor is particularly dirty turn on the spot over area until clean or a certain number of times.

When battery level becomes low return to charger

Inputs:

Proximity to a wall/ obstacle

Level of Dirt

Proximity of drops/ steps

Battery level

Charging station location

Outputs:

Turn

Forward

Repeat area

Return to charger

Stored:

Level of dirt

Presence of an object

Number of times area is repeated

‘Cliff’ detected

This shows me that the current algorithm relies heavily on processing and does not make use of storage as much which could be a point where it can be improved by storing more data and processing less information. The inputs and outputs should stay the same for my system to keep the simulation realistic

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Dictionary** | | | | | |
| Ref | Name | Data Type | Regex | Occurrence | Source of data / description |
|  | Level of dirt | Decimal |  | A maximum of area of the room divided by the area of the robot | Changed to true when over an area of dirt |
|  | Proximity to an object in front | Decimal |  | Once changing as the distance changes | The distance to an obstacle in front of it |
|  | Proximity to an object to the left | Decimal |  | Once changing as the distance changes | The distance to an obstacle to the left of it |
|  | Proximity to an object to the right | Decimal |  | Once changing as the distance changes | The distance to an obstacle to the right of it |
|  | Number of times area has be repeated | Integer |  | As many times as the dirt sensor it triggered | This is a counted number of times the vacuum has cleaned an area of dirt |
|  | Shortest path to charger | Vector |  | Once | The calculated shortest known path to the charger so the amount of battery required to reach the charger can be calculated |
|  | Cliff detected | Boolean |  | Once changing between true and false | A Boolean for if there is a drop in front of vacuum then it will be set to true and stop |

The data currently stored in the system is not stored for a long period of time due to the environment of the vacuum changing constantly and the area being unknown. If the vacuum could store the values for longer it could find a more efficient path to cover the area. Also if the robot knew what it had cleaned it would be able to avoid those areas minimising the repeated area.



The data flow diagram makes it easier to visualise how the data is passed through the current system and shows the relations between processes.

My analysis has shown me the current problems with robot vacuum path finding and the general lack of efficiency that is inherent of random wall bouncing but also the efficient aspects of using spirals. If the path previously travelled is recorded and the algorithm attempts to avoid repeating areas that have been cleaned I think the area can be swept out more efficiently despite not knowing the room.

My investigation to see if the current method of sweeping an area can be improved will be compared by creating a new algorithm for a simulated robot to follow and a control algorithm to compare it to. The area of the room covered by each robot in a time will be compared.

Requirements:

These are the requirements that my simulation will achieve when complete.

1. There will be a virtual room that will have two copies, in each copy a simulated robot programmed with different algorithms are placed.
   1. One robot will have my own algorithm and will have improved coverage of the floor compared to the control
   2. The other will have the control algorithm which will be what I am comparing to
      1. The two areas covered will be outputted on screen after each simulation completes
      2. The amount of repeated area will be outputted by the program
      3. The two algorithms will finish at the same time to keep the test fair
2. Each robot will have the same properties except the algorithm they run
   1. They will have a battery of limited capacity that will drain as time progresses
   2. They will be able to travel in any direction without having to turn
   3. they will be able to sense obstacles on any side of them within a certain
   4. Both will have a set travel speed.
3. Perfect efficiency is (100% area coverage subtract inaccessible areas) with no area crossed twice.
   1. My simulated robot will have as few repeated areas as possible.
   2. The program will calculate the area that is accessible to the robots and compare it to what my algorithm covers.
   3. The percentage difference of the areas covered will be calculated and shown on screen
   4. The results will be stored to a text document so they can be referred to later
   5. The room will not be known by either robot to keep the comparison fair
4. The simulated vacuum will always return to the charger when it reaches a certain amount of battery
   1. It will take the shortest path possible to the charger
   2. This path will not count towards the efficiency and number of repeated areas as it is necessary to get to the charger
   3. But the timer will not be stopped
   4. If the “robot” returns to the charger mid-way through a clean then it will continue to clean the rest of the room starting from where it stopped.
5. The room will be randomly generated for each simulation test
   1. The size of the room will be random but will always be a quadrilateral
   2. Using a general structure for an object it will place a random amount in the room
      1. Objects will not intersect each other
   3. Floor will have a value for dirt level
      1. The level of dirt will be set randomly
   4. The room will be given to both “robots” to traverse
      1. It will be two separate instances of the same room
   5. Drops can also be generated within the room to simulate steps
      1. Drops in the simulated room will be identified and the algorithm will not allow the robot to fall down them
6. The simulation will display the “robot” moving through the “room”
   1. The speed of the simulation will be able to be user controlled
      1. Using a slider
   2. The simulation will be able to be skipped
      1. Using a button
   3. Colours will be used to display different aspects of the simulation
7. My algorithm will store a map of what it has scanned in the room and use it to decide where to move next
   1. This will store the information of where objects and walls are and also where the robot has previously been. This will be to avoid repeating areas.
   2. The map will be updated with what the sensors on the bot find in the room

Design

IPSO For my simulation program with my algorithm

This diagram represents the aspects that my algorithm will require to fulfil my requirements

Input

Sensors

Battery level

Charging

Speed of simulation

Skip simulation

Process

Locate object

Change direction

Decide whether to go to charger

Find Charger

Recognise empty space

Return to location left off at

Calculate shortest path

Calculate shortest path with minimal repeated areas

Render image

Storage

Area previously covered

Objects discovered

Battery level

Location left off at

Output

Robot on screen

Average values

Percentage difference of values

Process design

My investigation focuses on improving the current, commonly used, area sweeping algorithms for robot vacuums. The algorithm I create will produce improved coverage in the same amount of time as the control algorithm. To achieve this it will avoid covering areas that it has already covered unless necessary.

Pseudo code for a possible algorithm

Below is some pseudo code for a possible control system for the robot, by following edges of objects There is not yet an overall method for deciding where to go in this code.

IfObjectInFront

Dim PossibleShortest as decimal

Dim NearestPoint(3) as decimal

If Sweep(Object) <> 0 then

Direction = mod(sweep(object),4) + 1

Else

For every Column in Columns //Finds Empty area

For every Row in Rows

If Floor(Column, Row) = 0 then

PossibleShortest = CalculateShortestDistance(Row, Column)

If PossibleShortest < Nearest then

NearestPoint(1) = Column

NearestPoint(2) = Row

NearestPoint(3) = PossibleShortest

Endif

Endif

Next

Next

MoveDirectlyTo (NearestPoint(1), NearestPoint(2)

CalculateShortest(x,y)

(x,y)=CurrentLocation(,)-(x,y)

Return x^2 + y^2

Coverarea

Dim lastobsticle as integer //Follows the edge of an object

Do Until Sweep(Unclean, 1) = 1

LastObsticle = Sweep(Obsticle)

If LastObsticle = 0 then

Move(Direction)

Else

Move(Mod(LastObsticle,4) + 1

Endif

If Sweep(Obsticle) = 0 then

Move(LastObsticle)

If Mod(Sweep(Obsticle),4) + 1 = LastObsticle Then //left turn made

Move(Direction)

ElseIf Mod(Sweep(Obsticle,Direction),4) – 1 = LastObsticle Then //Right turn made

Move(Direction)

EndIf

Endif

Loop

SavePerviousPath(GetLocation(Y,X), FloorInfo)

If Floor(x,0) Doesn’t Exist Then

Create Floor(x,0)

If Floor(x,y) = Column(NumberOfItemsInList) then

Add item to all lists

Floor(x,y) = FloorInfo

Move(Direction)

If Direction = 1 then

SetLocation(0,Speed) //Speed is the number the location is changed by

ElseIf Direction = 2 then

Setlocation(Speed,0)

Elseif Direction = 3 then

Setlocation(0,-Speed)

Elseif Direction = 4 then

Setlocation(-Speed,0)

NextLocation(Direction)

If Direction = 1 then

Return CurrentLocation + (0,1)

ElseIf Direction = 2 then

Return CurrentLocation + (1,0)

ElseIf Direction = 3 then

Return Currentlocation – (0,1)

ElseIf Direction = 4 then

Return CurrentLocation – (1,0)

Sensor(Direction)

If Direction = 1 then

Return Floor(Currentlocation + (0,1))

Elseif Direction = 2 then

Return Floor(Currentlocation + (1,0))

Elseif Direction = 3 then

Return Floor(Currentlocation - (0,1))

Elseif Direction = 4 then

Return Floor(Currentlocation - (1,0))

Endif

Sweep(Object to search for)

For x = 1 to 4

If Sensor(x) = Object to search for Then

Return x

Else Return 0

Endif

Floor(X,Y)

Retun Column(X(Y))

MoveDirectlyTo(X,Y)

Dim XStep, YStep as Integer = 1

If X > Y then

YStep = Y/X

Steps = X

ElseIf Y > X

XStep = X/y

Steps = Y

Else

Steps = X

Endif

For Count = 1 to Steps or Obsticle = True

SetLocation(XStep, YStep)

Next

Pseudo Code for Control Algorithm

Some pseudo code for a moving the robot with the control algorithm

Avoid(Direction)

If Sensor(Object) = Direction then

Turn Left

Endif

Forward

If Direction = 1 then

Location = CurrentLocation(,) + (0,1)

ElseIf Direction = 2 then

Location = CurrentLocation(,) + (1,0)

ElseIf Direction = 3 then

Location = CurrentLocation(,) - (0,1)

ElseIf Direction = 4 then

Location = CurrentLocation(,) - (1,0)

Spiral(MinimumDist)

Dim Obstacle as Boolean = False

For n = 1 to 2

For x = 1 to MinimumDist

If Sensor(Object) = Direction then

Obstacle = True

Break

Else

Forward

Next

If Obstacle = True then

Break

Else

Turn Left

Endif

Next

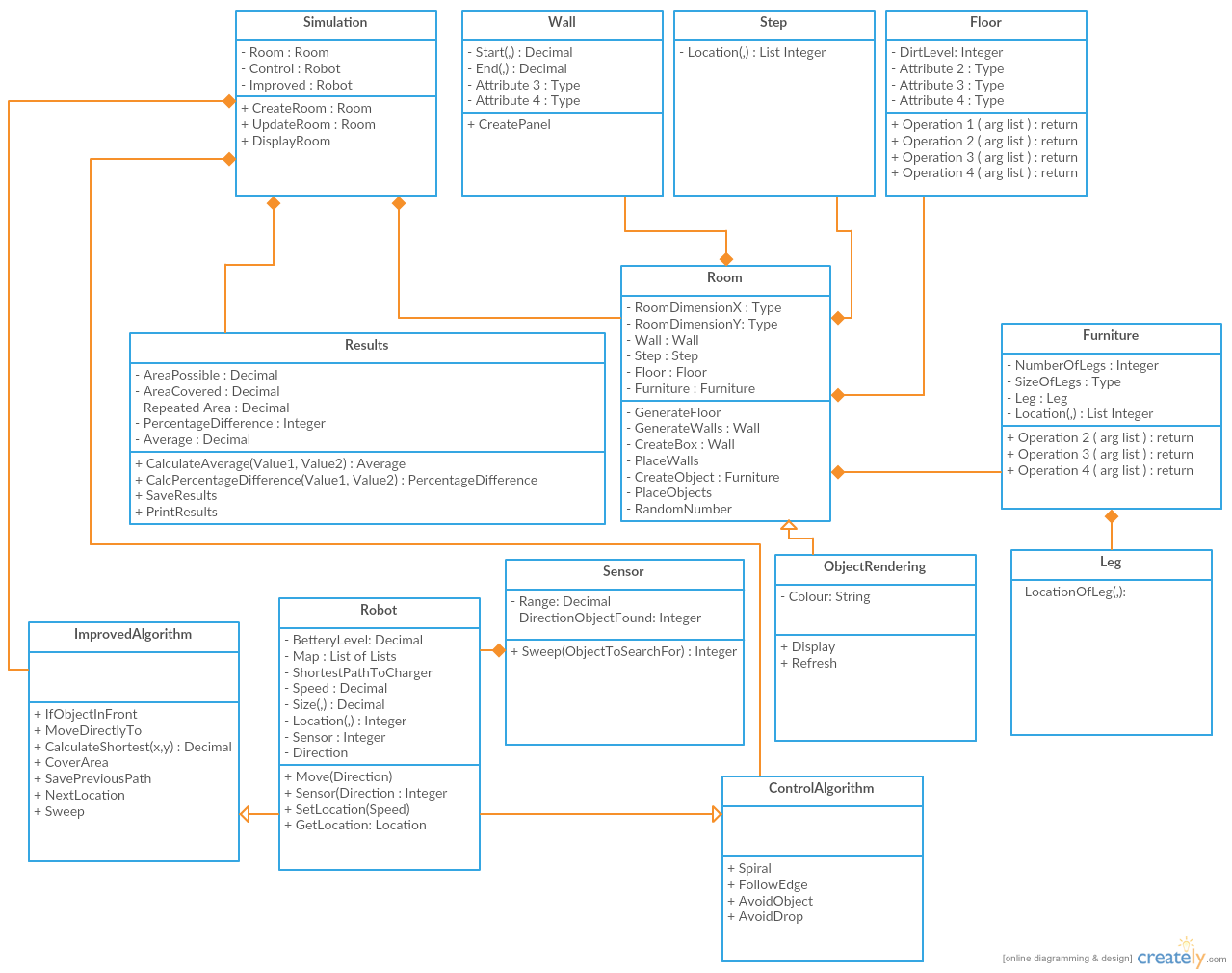
Spiral(MinimumDist + 1)

If Obstacle = True then

Break

Endif

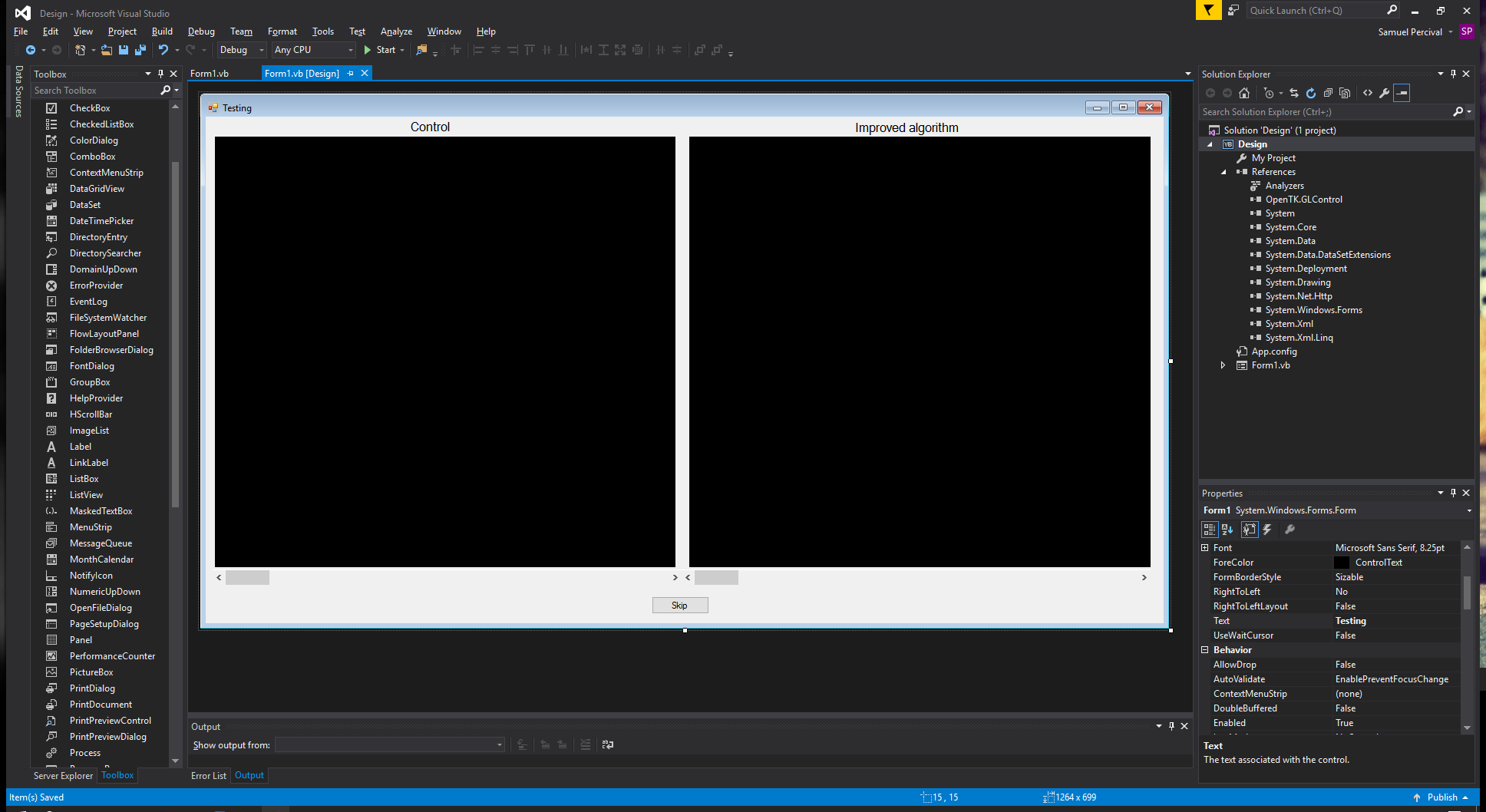
All my pseudo code uses a 2 dimensional array for locations but it might be more efficient to save them as a custom data type or as vectors. In the actual solution I will decide based on ease of use of the data and efficiency.

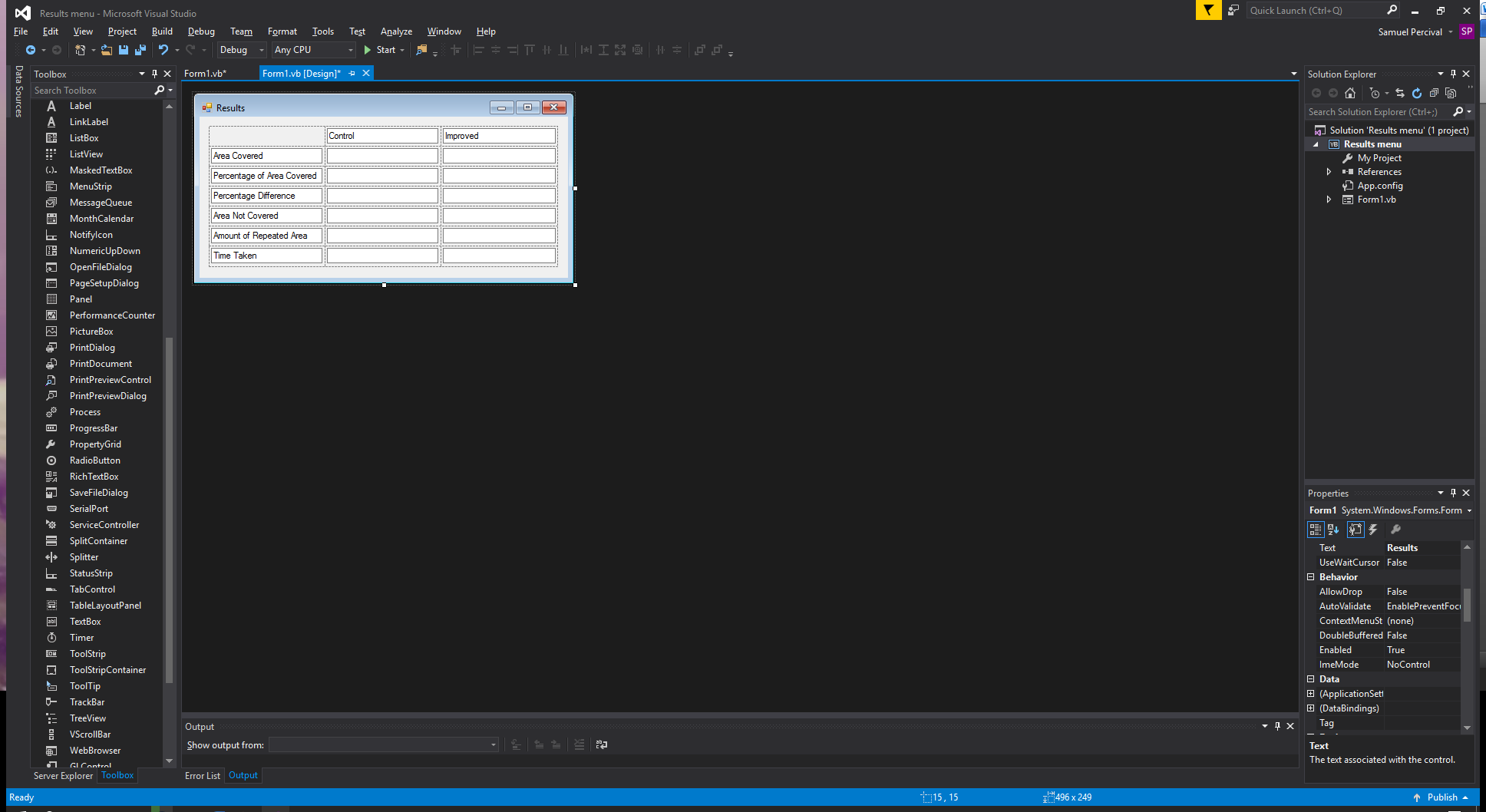
This is a class diagram of the system. Some small changes to the properties and procedures can be made.

Interface Design

Display of a simulation

Name of window





Which algorithm the value applies to

Type of value

Values will be outputted here

Name of window

Skip Button - To skip straight to results

Speed of simulation control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name of data | Type Of Data | Length (Characters) | Source | Occurrences |
| Direction | Interger Between 1 and 4 | 1 | Robot | Movement procedure |
| Location | Custom data type | 1 | Robot | Movement, graphics and starting location |
| FloorInfo | Integer | 1 | Robot | Each object in room |
| Speed | Integer | 1 | Robot | Robot, movement |
| Distance | Decimal | 4 | Robot | Sensors |
| SpeedOfSimulation | Integer | 2 | User Specified | Menu, Simulation |
| NumberOfSimulations | Integer | 2 | User Specified | Menu, Simulation |

Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Success/ Fail | Evidence | Actual result |
| Two Copies of the room are created | Success | 00:12 in evidence video |  |
| Each robot will be programmed with a different algorithm | Success | 00:12 - 29:42 in evidence video |  |
| My algorithm will have improved floor coverage | Success | Results from multiple runs below |  |
| The two algorithms will end at the same time | Success | 11:33 in evidence video |  |
| The two robots will have the same properties | Success | Robot class in main code is inherited by both bots |  |
| Limited battery capacity | Success | 11:26 in evidence video |  |
| Travel in any direction | Fail | 00:12 - 29:42 in evidence video | The robots travel in one of 4 directions |
| The robots can sense an objects with in a distance of them | Success | Map file generated and V |  |
| Set travel speed | Success | SpeedofBot property in Robot class and 00:12 - 29:42 in evidence video |  |
| Robot has as few repeated areas as possible | Success | 01:09-2:19, 04:50-5:46, 20:36- 28:28 in evidence video |  |
| Areas covered are compared and percentage difference displayed | Success | 29:45 in the evidence video |  |
| Results stored in a text document | Success | 30:54 in the evidence video |  |
| Robot will return to charger at low battery and other charger related objectives | Fail |  | It attempts to find its way back but never succeeds |
| The room is randomly generated | Success | 00:12, 04:27, 11:36 in the evidence video |  |
| The room is a random size each time | Fail |  | Room is always 200 by 200 as it caused too many performance issues |
| Objects are randomly placed in the room | Success | 00:12, 04:27, 11:36 in the evidence video |  |
| Objects do not intersect | Fail |  | The code to prevent intersecting did not work for object so they are able to generate on top of each other |
| Simulation is shown | Success | 00:12 - 29:42 in evidence video |  |
| Speed of simulation can be controlled | Success | 00:14 in evidence video |  |
| Simulation can be skipped | Success | 04:24 in evidence video |  |
| Colours are accurately representing obstacles | Success | 00:35 in evidence video |  |
| Graphics can be skipped | Success | 30:02 in evidence video |  |
| Results button opens results file | Success | 30:54 in evidence video |  |
| Multiple simulations | Success | 04:24, 11:34 in evidence video |  |
| Erase warning screen displays | Success | 29:50 in evidence video |  |

Evidence video located at: https://www.youtube.com/watch?v=sFKYkjgB6eo&feature=youtu.be

Map boiler plate: This is what the robot has stored as a map in its memory



Area bot can/ has seen in room

Bot location/ Charger

Unknown area

Room boiler plate: This is the room lain out using the map key numbers



Unclean floor

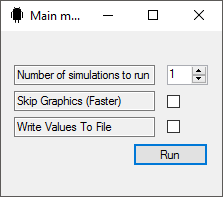
Objects

Wall

Cleaned floor

Menu

Number box to accept input

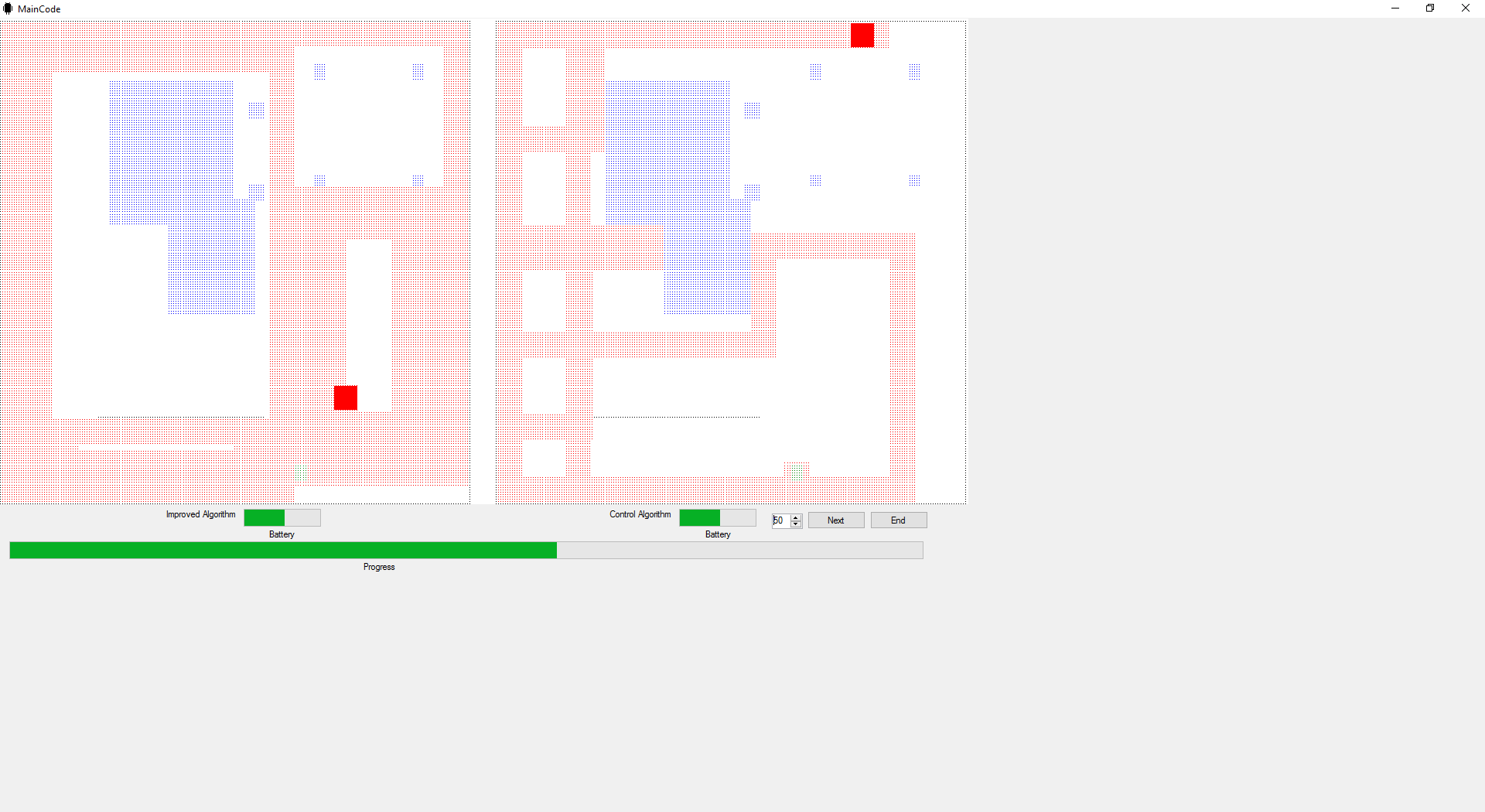


Start button

Tick boxes to accept inputs

Simulation

Uncleaned floor (blue)



Cleaned floor

Objects in blue

Robot

Wall

Simulation speed control

Next button

End button

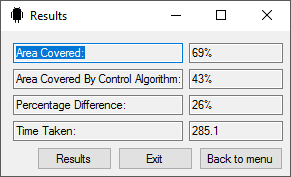
Improved algorithm on the left

Progress bar

Control algorithm on the right

Battery bars

Results

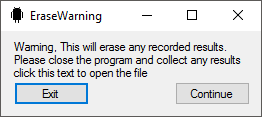


Navigationbuttons

Data from most recent run

Erase warning

Message text, can be clicked to open results



Exit the program

Continue to menu

Evaluation of objectives

In this section there will be a short paragraph about how I have met my objectives and an explanation for why others where not achieved

1. There will be a virtual room that will have two copies, in each copy a simulated robot programmed with different algorithms are placed.

This has been met very well as the robots do not affect each other’s rooms as ThisRoom2 is a copy of ThisRoom. (Line 34.11)

* 1. One robot will have my own algorithm and will have improved coverage of the floor compared to the control

The robot on the left of the simulation has my algorithm (described below). Generally covers more area than the control (results below). The class containing my algorithm can be found on line 105 in my code and the control can be found on line 145.

* 1. The other will have the control algorithm which will be what I am comparing to

The control algorithm is on the right of the simulation and the control can be found on line 145 of the code. The control algorithm simply changes to a random direction every time it collides with an object.

* + 1. The two areas covered will be outputted on screen after each simulation completes

The two areas are compared and calculated as a percentage difference. The results from each run are output to a text document and at the end of all simulations the results are displayed. The displaying of results is found from line 310 in my code to 319. And the writing to file is on line 52. To improve this, an average of all the runs could be output.

* + 1. The amount of repeated area will be outputted by the program

There was not enough time to implement a system to record the repeated area and it would compromise the performance of the program

* + 1. The two algorithms will finish at the same time to keep the test fair

Both algorithms complete the same number of cycles of each algorithm so the simulation time is the same, on line 40 there is there is the procedure to run both the improved algorithm and the control algorithm together so they complete the same number of cycles.

1. Each robot will have the same properties except the algorithm they run

Both robots inherit a robot class line 71 to keep the constant properties the same but the improved algorithm and the control algorithm required a few more properties to operate as desired. These have been implemented from 105 to 113 and 145 to 151 respectively

* 1. They will have a battery of limited capacity that will drain as time progresses.

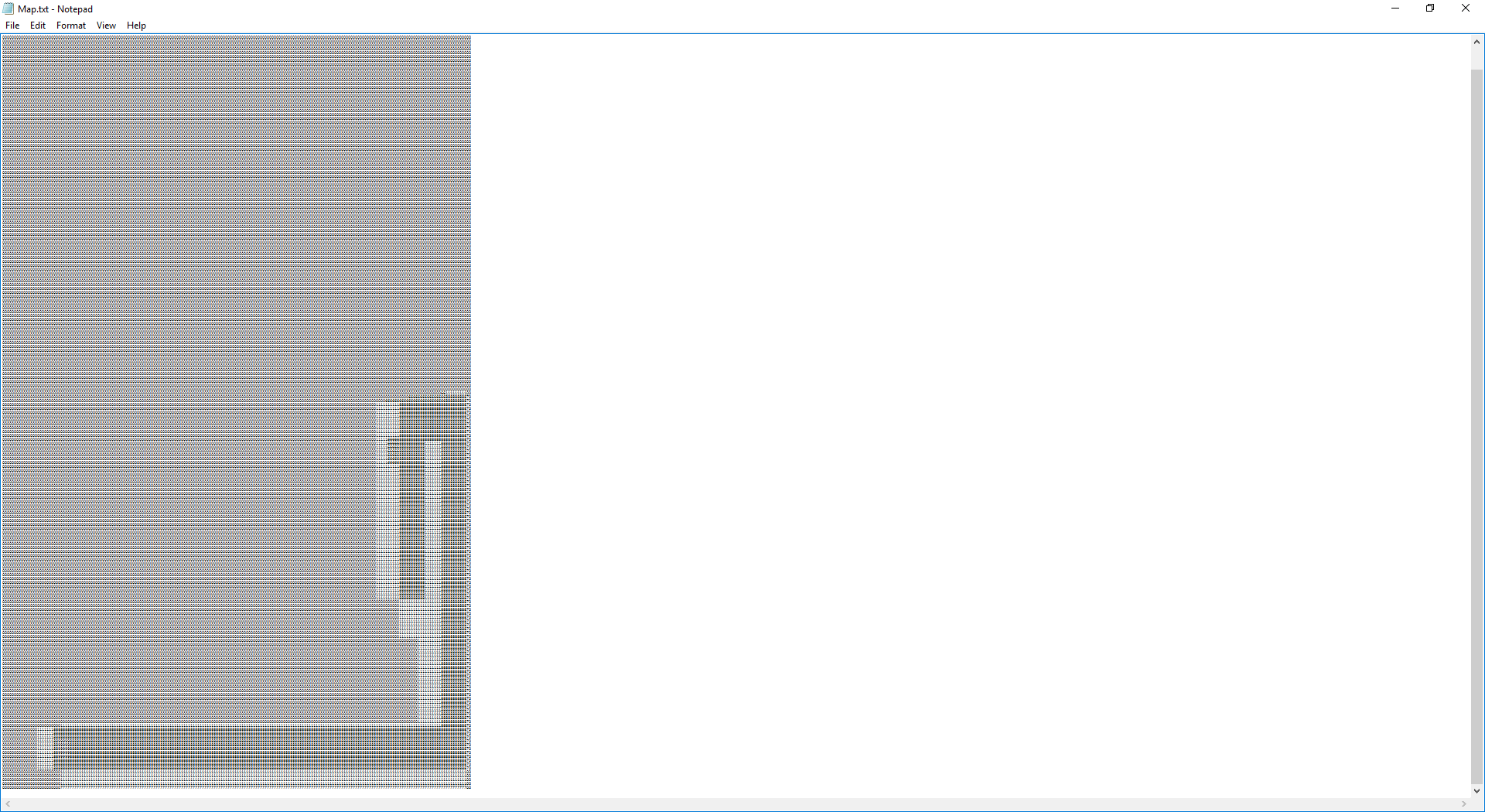
Both robots have a battery class which drains an amount every time a the related robot moves The class is located at line 170 and implemented in robot at line 77 and the battery is drained within the movement procedure at line 86.7

* 1. They will be able to travel in any direction without having to turn

Each robot can travel in any of 4 directions (North South East and West) and they do not turn to change direction. This was done to simplify the range of movement the robots have due to the room being built on a vector system and restricting the robot to one axis of movement at a time fixed many problems. The movement procedure can be found on line 86 and from this we can see the robot moves along the horizontal and vertical vectors x and y.

* 1. they will be able to sense obstacles on any side of them within a certain distance

The improved algorithm has sensors that allow what it can see to be mapped to an array, they have a scan range of 10 units, and the control algorithm can detect objects right next to it so it does not collide with them. The boiler plate for the sensors can be seen here



Empty uncleaned area

Cleaned area

Obstacle

Unknown area

As you can see from this the robot does not know the room before hand and only knows of what has been scanned. The scanners scan each side of the robot in turn moving from the robot outwards and stopping if it finds and obstacle as it cannot see through objects. To scan the room the room is passed to the sensors on line 159 where the room is then scanned 10 units away from each side of the bot

* 1. Both will have a set travel speed.

SpeedOfBot is a property in the robot class on line 74 used to dictate how far the bot will move on each movement command therefore giving them same speed.

1. Perfect efficiency is (100% area coverage subtract inaccessible areas) with no area crossed twice.
   1. My simulated robot will have as few repeated areas as possible.

My algorithm avoids repeating areas by attempting to travel in each direction without re-covering areas at first and if it unable it repeats an area meaning this objective has been met as best as possible for this algorithm, only improving the algorithm further can reduce repeated areas

* 1. The program will calculate the area that is accessible to the robots and compare it to what my algorithm covers.

The amount of clean and dirty floor there is tallied from 40.6 to 40.9 and the percentage of the accessible area the robot has covered is calculated on line 40.10 and the difference between the improved and control algorithm is calculated on line 311.5

* 1. The percentage difference of the areas covered will be calculated and shown on screen

On line 311.5 the value is displayed

* 1. The results will be stored to a text document so they can be referred to later

From lines 52 to 53 the result of the run are written to a file named Results located in the room folder where the program is run if write to file is selected by the user. These results are erased if the user returns to the main menu

* 1. The room will not be known by either robot to keep the comparison fair

The room and the robots are kept separated except for the sensors where they scan the room and the cleaner which cleans the room directly as accurate to real life. As seen in the boiler plate for the sensors, the room is unknown until scanned and saved in the improved algorithm and the control does not create a map. The only thing known to the improved algorithm is the size of the room so it knows how big to create its map.

1. The simulated vacuum will always return to the charger when it reaches a certain amount of battery

An attempt was made to implement returning to charger from line 142 but was not completed and therefore this objective has been left unfulfilled.

* 1. It will take the shortest path possible to the charger
  2. This path will not count towards the efficiency and number of repeated areas as it is necessary to get to the charger
  3. But the timer will not be stopped
  4. If the “robot” returns to the charger mid-way through a clean then it will continue to clean the rest of the room starting from where it stopped.

Instead of returning to the charger the simulation ends when the battery level reaches 0

1. The room will be randomly generated for each simulation test
   1. The size of the room will be random but will always be a quadrilateral

This objective presented too many issues with the robots going out of bound or not reaching the edges so the room is always 200 by 200 units

* 1. Using a general structure for an object it will place a random amount in the room

Both walls and objects are randomly generated (the number of them, their dimensions and their position.)

Walls are generated from lines 224 where a wall is added to the list of walls and in the constructor for wall from line 254 each wall is give 2 ends and they are set to random locations in the room. After that the gradient and intercept are calculated and assigned to the properties of the wall so it can then be set in the room.

Furniture is generated within 226 there the furniture constructor on 278 assigns the piece or furniture or object a size and randomly decides if it has 4 individual legs or is a solid “box” it then assigns the furniture a random location in the room and sets the 4 corners the correct distance away depending on the dimensions. If appropriate a leg is generate on each corner of the furniture and the leg constructor starting at line 291 creates a leg of dimensions 5 x 5 at the position passed. Back in 226 the area inside the solid “boxes” is filled in the room and the “legs” are also filled in.

* + 1. Objects will not intersect each other

Objects do intersect with each other, an attempt to stop this was made but would not work and there was not enough time to implement this. But overall it does not affect the overall purpose of the program

* 1. Floor will have a value for dirt level

The floor in the room is set a dirt level at line 232.8.2.1

* + 1. The level of dirt will be set randomly

Due to repeating areas not being included a random dirt level would not contribute to the simulation.

* 1. The room will be given to both “robots” to traverse

Both robots are placed in their own room on line 34.7 where it finds the bots a starting position in their respective rooms.

* + 1. It will be two separate instances of the same room

In the room constructor on line 214.1 a parameter called clone is passed which tells the code whether to create a copy of the room. In createroom on line 216.2.1 the size of the room is copied and on line 34.11 a procedure is called to clone the room. The procedure is found from lines 244 to 245. It copies each location in one room to the other.

* 1. Drops can also be generated within the room to simulate steps
     1. Drops in the simulated room will be identified and the algorithm will not allow the robot to fall down them.

The pursuit of drops was abandoned as they are almost identical to obstacles

1. The simulation will display the “robot” moving through the “room”

The simulation accurately depicts where each robot is in each room. This is verified by the boiler plates of the room matching the room the simulation draws and the robot map having the robot located in the same place as the simulation when the simulation is ended at any point. I use open gl to do this using vectors to decide what do draw.

* 1. The speed of the simulation will be able to be user controlled

The simulation speed can be user controlled but because the simulation does not run as smoothly as I initially thought it would the speed adjustment is not immediately responsive to changes

* 1. The simulation will be able to be skipped

A single simulation is able to be ended prematurely in the form of the next button, this is found on line 66 of the code where a procedure ends the simulation and starts the next one. Also the graphics of the robots can be turned off from the main menu which makes the program run a bit faster

* 1. Colours will be used to display different aspects of the simulation

The room is presented in different colours. This is achieved on lines 38.1.17 and 38.1.29 where the different aspects of the room are drawn in different distinct colours. To do this areas on the vector system used to map the room that have different “keys” are presented in different colours.

7. My algorithm will store a map of what it has scanned in the room and use it to decide where to move next

1. This will store the information of where objects and walls are and also where the robot has previously been. This will be to avoid repeating areas.

From line 159 the robot sweeps each side of the bot using the sensors on the room. Whatever is scanned is written to a map stored in a list of lists to represent a dynamically sizable grid system. On lines 159.2.3.2, 159.4.3.2, 159.6.3.4 and 159.8.3.2 the values at a location on the floor are mapped to the locations of a map in the robots “memory”.

1. The map will be updated with what the sensors on the bot find in the room

In the class sensor on line 157 the procedure called sweep record on line 159 scans outward from the robot 10 units and record what it finds at that location.

Evaluation

Description of algorithms

Improved

My algorithm involved systematically making right hand turns every time is collided with an object, avoiding repeating areas it had already covered at first and then ignoring them if it is unable to move. I know what it has and haven’t covered by recording a map to its memory and marking where it had gone and where obstacles had been found. This means the robot can be placed in any unknown room and still cover most the area assuming it doesn’t get stuck. To avoid getting stuck the robot count how long it is in an area that has already covered and randomly changes direction every 100 cycles of movement until free. The location of the charger it starts at is also stored so it has the possibility of finding its way back which could lead to the robot continuing forever.

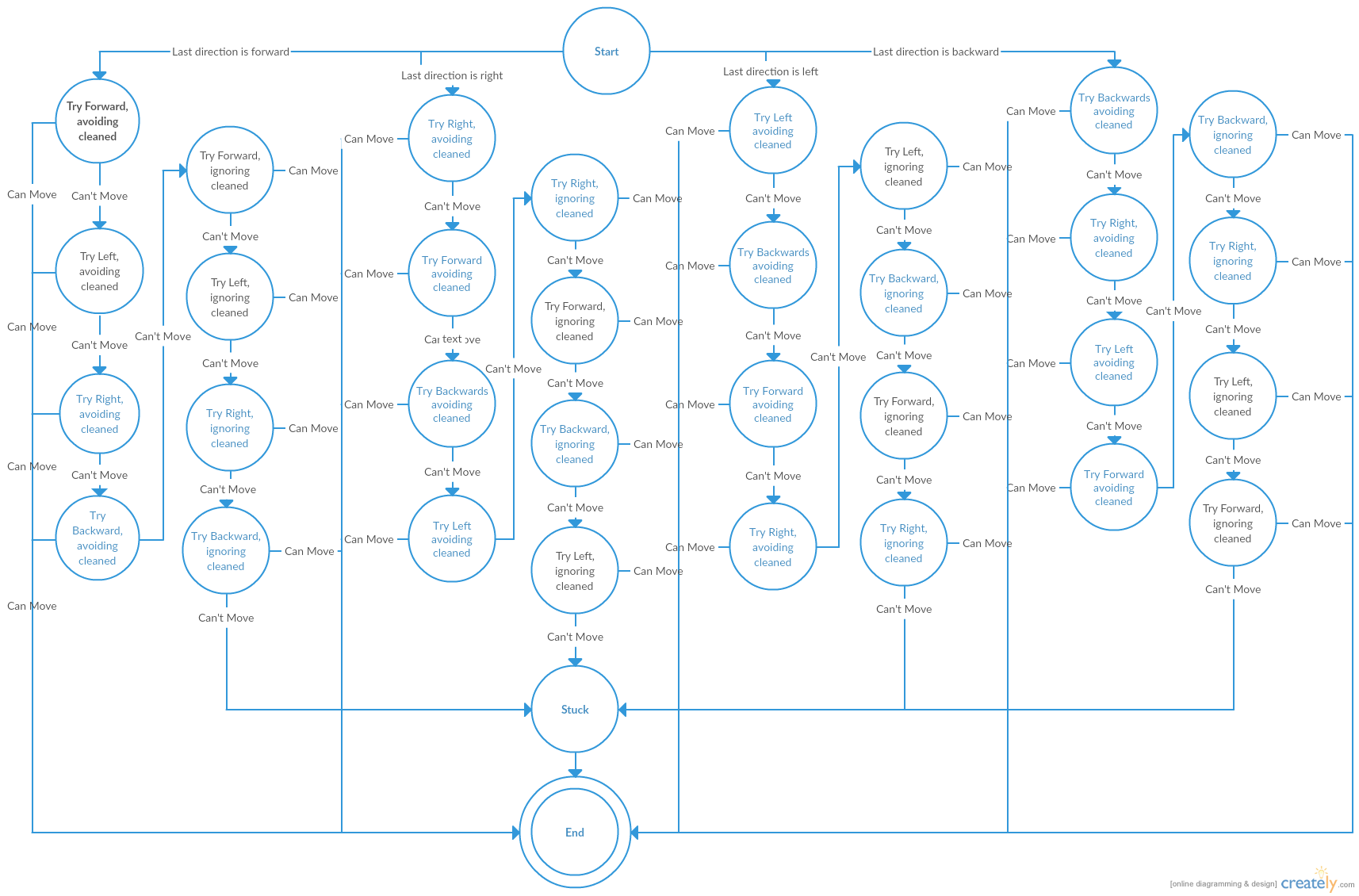
On the next page there is a finite state machine that describes my algorithm

The algorithm could have a less crude method of preventing getting stuck be searching the map for possible uncleaned areas and traveling to them. Also a more reliable charger locating algorithm could be implemented. Another improvement would be adapting the system to work in any direction rather than the limited 4 and also a system to locate itself in the room based on something other than its own movement in case it was knocked off course. Finally a feature could be added where a room is recognised whilst the robot travels around the room so it can take a more efficient path than before due to the whole room being known.

The fact the algorithms can be improved has implications such as having the batteries in physical units lasting longer due to the greater efficiency and therefore less battery drain.

Control

The algorithm involves moving in a random direction every time it cannot pass an object in the direction it is currently moving. Every 100 cycles it will change direction randomly to firstly cover more area and secondly prevent it getting stuck in loops. This bot does not differentiate between clean and unclean floor



These are the results from a few runs:

Simulation Number 5

Area Covered By Improved Algorithm: 87%

Area Covered By Control Algorithm: 49%

Percentage Difference between the two: 38%

Simulation Time: 441.6

Simulation Number 4

Area Covered By Improved Algorithm: 91%

Area Covered By Control Algorithm: 48%

Percentage Difference between the two: 43%

Simulation Time: 361.4

Simulation Number 3

Area Covered By Improved Algorithm: 83%

Area Covered By Control Algorithm: 47%

Percentage Difference between the two: 36%

Simulation Time: 668.1

Simulation Number 2

Area Covered By Improved Algorithm: 91%

Area Covered By Control Algorithm: 62%

Percentage Difference between the two: 29%

Simulation Time: 608.5

Simulation Number 1

Area Covered By Improved Algorithm: 71%

Area Covered By Control Algorithm: 58%

Percentage Difference between the two: 13%

Simulation Time: 739.7

Simulation Number 5

Area Covered By Improved Algorithm: 79%

Area Covered By Control Algorithm: 62%

Percentage Difference between the two: 17%

Simulation Time: 708.1

Simulation Number 4

Area Covered By Improved Algorithm: 77%

Area Covered By Control Algorithm: 45%

Percentage Difference between the two: 32%

Simulation Time: 716.8

Simulation Number 3

Area Covered By Improved Algorithm: 31%

Area Covered By Control Algorithm: 62%

Percentage Difference between the two: -31%

Simulation Time: 874.9

Simulation Number 2

Area Covered By Improved Algorithm: 91%

Area Covered By Control Algorithm: 60%

Percentage Difference between the two: 31%

Simulation Time: 665.4

Simulation Number 1

Area Covered By Improved Algorithm: 41%

Area Covered By Control Algorithm: 61%

Percentage Difference between the two: -20%

Simulation Time: 659.4

Conclusion

As you can see from the values above my algorithm almost always covered a significant amount more of the floor than the control algorithm 8/10.

Despite the size of the room being small this concept could be adapted to fit any size of room. If there were several robots that sent the areas that they had each discovered to a central system that combined it in to one map that was sent back to each robot, the system should still hold up.

My simulation could be improved with speed optimisation and unrestricted directions of movement.

In conclusion, yes, the current standard sweeping algorithm of changing direction randomly can be improved fairly easily but the drawback is that more processing power will be required to make the system run as quickly as the basic algorithm does.

Appendix

Main Menu

1. Public Class Form1
2. Private WriteToFile As Boolean 'options selected
3. Private SkipGraphics As Boolean
4. Private NumberOfSimulations As Integer
5. Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click 'loads main program
   1. Dim MainProg As New MainCode(NumberOfSimulations, SkipGraphics, WriteToFile)
   2. MainProg.Activate()
   3. Try
      1. MainProg.Show()
   4. Catch
   5. End Try
   6. Me.Close()
6. End Sub
7. Private Sub NumberOfSimulationsNumericUpDown\_ValueChanged(sender As Object, e As EventArgs) Handles NumberOfSimulationsNumericUpDown.ValueChanged 'number of simulation
   1. NumberOfSimulations = NumberOfSimulationsNumericUpDown.Value
8. End Sub
9. Private Sub SkipGraphicsBox\_CheckedChanged(sender As Object, e As EventArgs) Handles SkipGraphicsBox.CheckedChanged 'skip graphics option
   1. If SkipGraphicsBox.Checked = True Then
      1. SkipGraphics = True
   2. ElseIf SkipGraphicsBox.Checked = False Then
      1. SkipGraphics = False
   3. End If
10. End Sub
11. Private Sub WriteToFileBox\_CheckedChanged(sender As Object, e As EventArgs) Handles WriteToFileBox.CheckedChanged 'Write file
    1. If WriteToFileBox.Checked = True Then
       1. WriteToFile = True
    2. ElseIf WriteToFileBox.Checked = False Then
       1. WriteToFile = False
    3. End If
12. End Sub
13. Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load 'erases previous results in doc
    1. Dim FILE\_NAME As String = IO.Path.Combine(My.Application.Info.DirectoryPath, "Rooms\Results.txt")
    2. System.IO.File.WriteAllText(FILE\_NAME, "")
14. End Sub
15. End Class

MainCode

1. Imports OpenTK
2. Imports OpenTK.Graphics
3. Imports OpenTK.Graphics.OpenGL
4. Public Class MainCode
5. Private ThisRoom As New Room(False, 0, 0)
6. Private ThisRoom2 As New Room(True, ThisRoom.GetRoomSize(0) - 1, ThisRoom.GetRoomSize(1) - 1)
7. Private MyAlgo As New ImprovedAlgo(ThisRoom.GetRoomSize(0), ThisRoom.GetRoomSize(1))
8. Private ControlAlgo As New Control(ThisRoom2.GetRoomSize(0), ThisRoom2.GetRoomSize(1))
9. Private NumberOfSimulations As Integer
10. Private SkipGraphics As Boolean
11. Private WriteToFile As Boolean
12. Private FirstRun As Boolean = True
13. Private GroundTexture As Integer = 0
14. Private GroundData(ThisRoom.GetRoomSize(0), ThisRoom.GetRoomSize(1)) As Byte
15. Private GroundCovered, GroundCoveredControl As Integer
16. Private SimTime As ULong
17. Private Clean, Dirty As Double
18. Private TotalSimulations As Integer
19. Sub New(NumberOfSimulations, SkipGraphics, WriteToFile)
    1. InitializeComponent()
    2. ControlAlgo.GetBattery.SetBatteryLevel(MyAlgo.GetBattery.GetBatteryLevel) 'sets battery level
    3. SetNumberOfSimulations(NumberOfSimulations) 'number of simulations to run
    4. SetSkipGraphics(SkipGraphics) 'skip graphics
    5. SetWriteToFile(WriteToFile) 'write results to file
    6. Dim Start(1) As Integer
    7. Start = ThisRoom.RobotStartingPosition(MyAlgo.GetSizeOfBot) 'finds starting position for bots
    8. If Start(0) = 0 And Start(1) = 0 Then 'if no position found simulation is restarted
       1. Restart(NumberOfSimulations)
    9. Else
       1. MyAlgo.SetLocation(Start(0), Start(1)) 'sets starting locations
       2. ControlAlgo.SetLocation(Start(0), Start(1))
       3. ThisRoom.GetChargerObj.SetChargerLocation(Start(0), Start(1)) 'sets charger location
       4. ThisRoom.FillArea(ThisRoom.GetChargerObj.GetCorners.GetCorners(0, 0), ThisRoom.GetChargerObj.GetCorners.GetCorners(0, 1), ThisRoom.GetChargerObj.GetCorners.GetCorners(2, 0), ThisRoom.GetChargerObj.GetCorners.GetCorners(2, 1), ThisRoom.GetChargerObj.GetMapKey)
    10. End If
    11. ThisRoom.CopyRoom(ThisRoom, ThisRoom2) 'clones thisroom to thisroom2
20. End Sub
21. Private Sub GlControl1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles GlControl1.Load
    1. GL.ClearColor(Color.White) 'draws white background
22. End Sub
23. Private Sub GlControl1\_Paint(ByVal sender As Object, ByVal e As System.Windows.Forms.PaintEventArgs) Handles GlControl1.Paint 'paints stuff
    1. If SkipGraphics <> True Then
       1. GL.Clear(ClearBufferMask.ColorBufferBit)
       2. GL.Clear(ClearBufferMask.DepthBufferBit)
       3. 'Basic Setup for viewing
       4. Dim perspective As Matrix4 = Matrix4.CreateOrthographic(ThisRoom.GetRoomSize(0) + ThisRoom2.GetRoomSize(0) + 10, ThisRoom.GetRoomSize(1), 0, 1000) 'Setup Perspective
       5. Dim lookat As Matrix4 = Matrix4.LookAt(ThisRoom.GetRoomSize(0) + 5, ThisRoom.GetRoomSize(1) / 2, 150, ThisRoom.GetRoomSize(0) + 5, ThisRoom.GetRoomSize(1) / 2, 0, 0, 1, 0) 'Setup camera
       6. GL.MatrixMode(MatrixMode.Projection) 'Load Perspective
       7. GL.LoadIdentity()
       8. GL.LoadMatrix(perspective)
       9. GL.MatrixMode(MatrixMode.Modelview) 'Load Camera
       10. GL.LoadIdentity()
       11. GL.LoadMatrix(lookat)
       12. GL.Viewport(0, 0, GlControl1.Width, GlControl1.Height) 'Size of window
       13. GL.Enable(EnableCap.DepthTest) 'Enable correct Z Drawings
       14. GL.DepthFunc(DepthFunction.Less) 'Enable correct Z Drawings
       15. GL.Begin(BeginMode.Points)
       16. For Y = 0 To ThisRoom.GetRoomSize(1) - 1
       17. For X = 0 To ThisRoom.GetRoomSize(0) - 1
           1. If ThisRoom.FloorAt(X, Y) = 1 Then
           2. ElseIf ThisRoom.FloorAt(X, Y) = 8 Then
           3. GL.Color3(Color.Red)
           4. GL.Vertex2(X, Y)
           5. ElseIf ThisRoom.FloorAt(X, Y) = ThisRoom.GetFurniture(0).GetMapKey Then
           6. GL.Color3(Color.Blue)
           7. GL.Vertex2(X, Y)
           8. ElseIf ThisRoom.FloorAt(X, Y) = ThisRoom.GetWalls(0).GetMapKey Then
           9. GL.Color3(Color.Black)
           10. GL.Vertex2(X, Y)
           11. ElseIf ThisRoom.FloorAt(X, Y) = ThisRoom.GetChargerObj.GetMapKey Then
           12. GL.Color3(Color.Green)
           13. GL.Vertex2(X, Y)
           14. End If
       18. Next
       19. Next
       20. GL.End()
       21. GL.Begin(BeginMode.Quads)
       22. GL.Color3(Color.Red) 'draws myalgo
       23. For index = 0 To 3
       24. GL.Vertex2(MyAlgo.GetBotCorners(index, 0), MyAlgo.GetBotCorners(index, 1))
       25. Next
       26. GL.End()
       27. GL.Begin(BeginMode.Points) 'draws the elements in room
       28. For Y = 0 To ThisRoom2.GetRoomSize(1) - 1
       29. For X = 0 To ThisRoom2.GetRoomSize(0) - 1
           1. If ThisRoom2.FloorAt(X, Y) = 1 Then
           2. ElseIf ThisRoom2.FloorAt(X, Y) = 8 Then
           3. GL.Color3(Color.Red)
           4. GL.Vertex2(X + ThisRoom2.GetRoomSize(0) + 10, Y)
           5. ElseIf ThisRoom2.FloorAt(X, Y) = ThisRoom2.GetFurniture(0).GetMapKey Then
           6. GL.Color3(Color.Blue)
           7. GL.Vertex2(X + ThisRoom.GetRoomSize(0) + 10, Y)
           8. ElseIf ThisRoom2.FloorAt(X, Y) = ThisRoom2.GetWalls(0).GetMapKey Then
           9. GL.Color3(Color.Black)
           10. GL.Vertex2(X + ThisRoom2.GetRoomSize(0) + 10, Y)
           11. ElseIf ThisRoom2.FloorAt(X, Y) = ThisRoom2.GetChargerObj.GetMapKey Then
           12. GL.Color3(Color.Green)
           13. GL.Vertex2(X + ThisRoom2.GetRoomSize(0) + 10, Y)
           14. End If
       30. Next
       31. Next
       32. GL.End()
       33. GL.Begin(BeginMode.Quads)
       34. 'Draws controlalgo
       35. GL.Color3(Color.Red)
       36. For index = 0 To 3
       37. GL.Vertex2(ControlAlgo.GetBotCorners(index, 0) + ThisRoom.GetRoomSize(0) + 10, ControlAlgo.GetBotCorners(index, 1))
       38. Next
       39. 'Finish the begin mode with "end"
       40. GL.End()
       41. GraphicsContext.CurrentContext.VSync = False 'Caps frame rate
       42. GlControl1.SwapBuffers()
    2. End If
    3. RunImprovedAndControl()
24. End Sub
25. Public Sub RunImprovedAndControl()
    1. For index = 1 To SimulationSpeed.Value 'number of cycles of simulation to run before graphical refresh
       1. If MyAlgo.GetBattery.GetBatteryLevel > 0 Then 'if no battery left, end
       2. If MyAlgo.Move(ThisRoom, MyAlgo.GetLastDirection) = False Then
          1. EndOrRestart()
       3. End If
       4. RunControl()
       5. SimTime = SimTime + 1
       6. End If
    2. Next
    3. If FirstRun <> False Then 'fixes no graphics bug
       1. GlControl1.Invalidate() 'refresh graphics
    4. End If
    5. FirstRun = True
    6. Clean = 0
    7. Dirty = 0
    8. For Y = 0 To ThisRoom.GetRoomSize(1) - 1
       1. For X = 0 To ThisRoom.GetRoomSize(0) - 1
       2. If ThisRoom.FloorAt(X, Y) = 8 Then
          1. Clean = Clean + 1
       3. ElseIf ThisRoom.FloorAt(X, Y) = 1 Then
          1. Dirty = Dirty + 1
       4. End If
       5. Next
    9. Next
    10. GroundCovered = (Clean / (Clean + Dirty)) \* 100 'percentage of area cleaned
    11. MyAlgo.WriteMapToFIle() 'write map
    12. Progress.Value = GroundCovered 'progress bar
    13. BatteryBarImproved.Value = MyAlgo.GetBattery.GetBatteryLevel 'battery bars
    14. BatteryBarControl.Value = MyAlgo.GetBattery.GetbatteryLevel
    15. If GroundCovered > 90 Or MyAlgo.GetBattery.GetBatteryLevel <= 0 Then 'if most of ground covered end or restart
        1. EndOrRestart()
    16. ElseIf MyAlgo.GetBattery.GetBatteryLevel < 10 Then 'go to charger at 10% battery or less
        1. Do Until MyAlgo.GetBattery.GetBatteryLevel <= 0 Or MyAlgo.GetBattery.GetBatteryLevel = 100
        2. MyAlgo.GoToCharger(ThisRoom.GetChargerObj.GetChargerLocation(0), ThisRoom.GetChargerObj.GetChargerlocation(1))
        3. Loop
    17. End If
26. End Sub
27. Public Sub RunControl() 'runs control algorithm
    1. ControlAlgo.Move(ThisRoom2)
    2. 'ControlAlgo.GoToCharger(ThisRoom2.GetChargerObj.GetChargerLocation(0), ThisRoom2.GetChargerObj.getchargerlocation(1))
    3. Clean = 0
    4. Dirty = 0
    5. For Y = 0 To ThisRoom2.GetRoomSize(1) - 1
       1. For X = 0 To ThisRoom2.GetRoomSize(0) - 1
       2. If ThisRoom2.FloorAt(X, Y) = 8 Then
          1. Clean = Clean + 1
       3. ElseIf ThisRoom2.FloorAt(X, Y) = 1 Then
          1. Dirty = Dirty + 1
       4. End If
       5. Next
    6. Next
    7. GroundCoveredControl = (Clean / (Clean + Dirty)) \* 100 'percentage of ground covered
28. End Sub
29. Public Sub EndOrRestart() 'decides weather to end simulation or restart
    1. WriteResultsToFile(GroundCovered, GroundCoveredControl, SimTime, NumberOfSimulations)
    2. If NumberOfSimulations = 1 Then
       1. ShutDown()
    3. Else
       1. Restart(NumberOfSimulations - 1)
    4. End If
30. End Sub
31. Private Sub ExitButton(sender As Object, e As EventArgs) Handles Button1.Click
    1. ShutDown()
32. End Sub
33. Public Sub ShutDown() 'closes simulation
    1. Dim Results As New ResultsScreen(GroundCovered, SimTime, GroundCoveredControl)
    2. Results.Activate()
    3. Results.Show()
    4. Me.Close()
34. End Sub
35. Public Sub Restart(NumberOfSimulationsLeft As Integer) 'restarts the simulation
    1. Dim MainProg As New MainCode(NumberOfSimulationsLeft, SkipGraphics, WriteToFile)
    2. MainProg.Activate()
    3. MainProg.Show()
    4. Me.Close()
36. End Sub
37. Public Sub WriteResultsToFile(AreaCoveredImproved, AreaCoveredControl, TimeTaken, SimNumber) 'writes results to a text document
    1. If WriteToFile = True Then
       1. Dim PercentageDiff As Integer
       2. PercentageDiff = AreaCoveredImproved - AreaCoveredControl
       3. Dim FILE\_NAME As String = IO.Path.Combine(My.Application.Info.DirectoryPath, "Rooms\Results.txt")
       4. If System.IO.File.Exists(FILE\_NAME) = True Then
       5. Dim objWriter As New System.IO.StreamWriter(FILE\_NAME, True)
       6. objWriter.WriteLine("Simulation Number " & SimNumber)
       7. objWriter.WriteLine("Area Covered By Improved Algorithm: " & AreaCoveredImproved & "%")
       8. objWriter.WriteLine("Area Covered By Control Algorithm: " & AreaCoveredControl & "%")
       9. objWriter.WriteLine("Percentage Difference between the two: " & PercentageDiff & "%")
       10. objWriter.WriteLine("Simulation Time: " & SimTime / 10)
       11. objWriter.WriteLine("")
       12. objWriter.Close()
       13. Else
       14. 'MsgBox("File Not found")
       15. End If
    2. End If
38. End Sub
39. Public Sub SetNumberOfSimulations(Value As Integer)
    1. NumberOfSimulations = Value
40. End Sub
41. Public Sub SetSkipGraphics(Value As Boolean)
    1. SkipGraphics = Value
42. End Sub
43. Public Sub SetWriteToFile(Value As Boolean)
    1. WriteToFile = Value
44. End Sub
45. Public Sub SetWriteToFile(Value As Boolean)
    1. WriteToFile = Value
46. End Sub
47. Public Function GetNumberOfSimulations()
    1. Return NumberOfSimulations
48. End Function
49. Public Function GetSkipGraphics()
    1. Return SkipGraphics
50. End Function
51. Private Sub NextButton\_Click(sender As Object, e As EventArgs) Handles NextButton.Click
    1. EndOrRestart()
52. End Sub
53. Public Function GetWriteToFile()
    1. Return WriteToFile
54. End Function
55. End Class
56. Class Robot
57. Inherits GenRandomNum
58. Private SizeOfBot As Integer = 10 'Dimensions in cm
59. Private SpeedOfBot As Integer = 1 'cm's to move at once
60. Private CurrentLocation As New Location
61. Private BotCorners As New CornerData
62. Private Battery As New Battery
63. Public Structure Location
    1. Dim XAxis As Integer
    2. Dim YAxis As Integer
64. End Structure
65. Sub New(RoomsizeX As Integer, RoomSizeY As Integer)
    1. SetupCorners()
66. End Sub
67. Public Sub SetupCorners()
    1. For x = 0 To (3)
       1. BotCorners.AddCorners()
    2. Next
68. End Sub
69. Public Sub SetLocation(X As Integer, Y As Integer) 'directly sets location and corners to x and y
    1. CurrentLocation.XAxis = X
    2. CurrentLocation.YAxis = Y
    3. BotCorners.SetCorners((CurrentLocation.XAxis + SizeOfBot / 2), (CurrentLocation.YAxis + SizeOfBot / 2), 0) 'TopLeft
    4. BotCorners.SetCorners((CurrentLocation.XAxis + SizeOfBot / 2), (CurrentLocation.YAxis - SizeOfBot / 2), 1) 'Bottom Left
    5. BotCorners.SetCorners((CurrentLocation.XAxis - SizeOfBot / 2), (CurrentLocation.YAxis - SizeOfBot / 2), 2) 'Bottom right
    6. BotCorners.SetCorners((CurrentLocation.XAxis - SizeOfBot / 2), (CurrentLocation.YAxis + SizeOfBot / 2), 3) 'Top right
70. End Sub
71. Public Overridable Sub ChangeLocation(X As Integer, Y As Integer) 'changes location and corners by x and y
    1. CurrentLocation.XAxis = CurrentLocation.XAxis + X
    2. CurrentLocation.YAxis = CurrentLocation.YAxis + Y
    3. BotCorners.SetCorners((CurrentLocation.XAxis + SizeOfBot / 2), (CurrentLocation.YAxis + SizeOfBot / 2), 0) 'TopLeft
    4. BotCorners.SetCorners((CurrentLocation.XAxis + SizeOfBot / 2), (CurrentLocation.YAxis - SizeOfBot / 2), 1) 'Bottom Left
    5. BotCorners.SetCorners((CurrentLocation.XAxis - SizeOfBot / 2), (CurrentLocation.YAxis - SizeOfBot / 2), 2) 'Bottom right
    6. BotCorners.SetCorners((CurrentLocation.XAxis - SizeOfBot / 2), (CurrentLocation.YAxis + SizeOfBot / 2), 3) 'Top right
    7. Battery.DecreaseBattery()
72. End Sub
73. Public Function GetSizeOfBot()
    1. Return SizeOfBot
74. End Function
75. Public Function GetLocation(XY As Integer)
    1. If XY = 0 Then
       1. Return CurrentLocation.XAxis
    2. Else
       1. Return CurrentLocation.YAxis
    3. End If
76. End Function
77. Public Function GetBotCorners(WhichCorner As Integer, XY As Integer)
    1. Return BotCorners.GetCorners(WhichCorner, XY)
78. End Function
79. Public Function GetSpeedOfBot()
    1. Return SpeedOfBot
80. End Function
81. Public Function GetCurrentlocation()
    1. Return CurrentLocation
82. End Function
83. Public Function GetBattery()
    1. Return Battery
84. End Function
85. Public Overridable Sub GoToCharger(ChargerLocationX As Integer, ChargerLocactionY As Integer)
86. End Sub
87. Public Sub Charge(ThisRoom As Room) 'charges battery if robot is on the charger
    1. If ThisRoom.FloorAt(CurrentLocation.XAxis, CurrentLocation.YAxis) = ThisRoom.GetChargerObj.GetMapKey Then
       1. Battery.SetBatteryLevel(100)
    2. End If
88. End Sub
89. End Class
90. Class ImprovedAlgo 'Improved algorithm info
91. Inherits Robot
92. Private FloorMap As New List(Of Rows)
93. Private RobotSens As New Sensor
94. Private LastDirection As New Direction
95. Private RoboCleaner As New Cleaner
96. Private StartingOfEdge As Location
97. Private ScanRecordCount As Integer = 1
98. Private LastCleanedTimer As Integer = 0
99. Public Sub WriteMapToFIle() 'writes robot room map to text document for debugging
    1. Dim FILE\_NAME As String = IO.Path.Combine(My.Application.Info.DirectoryPath, "Rooms\Map.txt")
    2. System.IO.File.WriteAllText(FILE\_NAME, "")
    3. If System.IO.File.Exists(FILE\_NAME) = True Then
       1. Dim objWriter As New System.IO.StreamWriter(FILE\_NAME)
       2. For Y = GetFloorMapSize(1) - 1 To 0 Step -1
       3. For X = 0 To GetFloorMapSize(0) - 1
          1. objWriter.Write(FloorMap(Y).ItemAt(X))
       4. Next
       5. objWriter.WriteLine("")
       6. Next
       7. objWriter.Close()
    4. Else
       1. MessageBox.Show("File Does Not Exist")
    5. End If
100. End Sub
101. Class Direction
     1. Private PlusMinus As Integer
     2. Private YAxis As Boolean
     3. Sub New()
        1. PlusMinus = 1
        2. YAxis = False
     4. End Sub
     5. Public Sub SetPlusMinus(Value)
        1. PlusMinus = Value
     6. End Sub
     7. Public Sub SetYaxis(Value)
        1. YAxis = Value
     8. End Sub
     9. Public Function GetPlusMinus()
        1. Return PlusMinus
     10. End Function
     11. Public Function GetYaxis()
         1. Return YAxis
     12. End Function
102. End Class
103. Sub New(RoomSizeX As Integer, RoomSizeY As Integer)
     1. MyBase.New(RoomSizeX, RoomSizeY)
     2. AddToMapColumns(RoomSizeX)
     3. AddToMapRows(RoomSizeY)
104. End Sub
105. Public Overrides Sub ChangeLocation(X As Integer, Y As Integer)
     1. MyBase.ChangeLocation(X, Y)
     2. If X > 0 Or Y > 0 Then
        1. LastDirection.SetPlusMinus(1)
     3. Else
        1. LastDirection.SetPlusMinus(-1)
     4. End If
     5. If Math.Abs(X) > Math.Abs(Y) Then
        1. LastDirection.SetYaxis(False)
     6. Else
        1. LastDirection.SetYaxis(True)
     7. End If
106. End Sub
107. Public Sub AddToMapColumns(ToAdd As Integer)
     1. For Y = 1 To ToAdd
        1. FloorMap.Add(New Rows)
     2. Next
108. End Sub
109. Public Sub AddToMapRows(ToAdd As Integer)
     1. For index = 0 To FloorMap.Count - 1
        1. For X = 0 To ToAdd
        2. FloorMap(index).AddItems()
        3. Next
     2. Next
110. End Sub
111. Public Sub SetFloorMap(X As Integer, Y As Integer, WhatToSet As Integer)
     1. If FloorMap.Count - 1 < Y Then
        1. AddToMapColumns((Y + 1) - FloorMap.Count)
     2. End If
     3. If FloorMap(Y).CountItems - 1 < X Then
        1. AddToMapRows((X + 1) - FloorMap(Y).CountItems)
     4. End If
     5. FloorMap(Y).SetItems(X, WhatToSet)
112. End Sub
113. Public Function Move(ThisRoom As Room, LastDirection As Direction)
     1. If ScanRecordCount = RobotSens.GetScanRange / 2 Then
        1. RobotSens.SweepRecord(Me, ThisRoom)
        2. ScanRecordCount = 0
     2. End If
     3. If LastCleanedTimer > 100 Then
        1. Dim num As Integer = MyBase.CreateRandInt(0, 1)
        2. If num = 0 Then
        3. LastDirection.SetYaxis(False)
        4. Else
        5. LastDirection.SetYaxis(True)
        6. End If
        7. num = MyBase.CreateRandInt(0, 1)
        8. If num = 0 Then
        9. LastDirection.SetPlusMinus(-1)
        10. Else
        11. LastDirection.SetPlusMinus(1)
        12. End If
        13. LastCleanedTimer = 0
     4. Else
        1. If LastDirection.GetPlusMinus = 1 And LastDirection.GetYaxis = True Then 'if the robot last moved forward
        2. If TryForward(False) = False Then 'Trys to move forward
           1. If TryLeft(False) = False Then 'Left
           2. If TryRight(False) = False Then 'Right
              1. If TryBackWards(False) = False Then 'Backwards

If TryForward(True) = False Then 'Forward ignoring cleaned floor

If TryLeft(True) = False Then 'Left ignoring Cleaned floor

If TryRight(True) = False Then 'Right

If TryBackWards(True) = False Then 'Back

Return False

End If

End If

End If

End If

* + - * 1. End If
      1. End If
      2. End If
    1. End If
    2. ElseIf LastDirection.GetPlusMinus = 1 And LastDirection.GetYaxis = False Then
    3. If TryRight(False) = False Then
       1. If TryForward(False) = False Then
       2. If TryBackWards(False) = False Then
          1. If TryLeft(False) = False Then

If TryRight(True) = False Then

If TryForward(True) = False Then

If TryBackWards(True) = False Then

If TryLeft(True) = False Then

Return False

End If

End If

End If

End If

* + - * 1. End If
      1. End If
      2. End If
    1. End If
    2. ElseIf LastDirection.GetPlusMinus = -1 And LastDirection.GetYaxis = False Then
    3. If TryLeft(False) = False Then
       1. If TryBackWards(False) = False Then
       2. If TryForward(False) = False Then
          1. If TryRight(False) = False Then

If TryLeft(True) = False Then

If TryBackWards(True) = False Then

If TryForward(True) = False Then

If TryRight(True) = False Then

Return False

End If

End If

End If

End If

* + - * 1. End If
      1. End If
      2. End If
    1. End If
    2. ElseIf LastDirection.GetPlusMinus = -1 And LastDirection.GetYaxis = True Then
    3. If TryBackWards(False) = False Then
       1. If TryRight(False) = False Then
       2. If TryLeft(False) = False Then
          1. If TryForward(False) = False Then

If TryBackWards(True) = False Then

If TryRight(True) = False Then

If TryLeft(True) = False Then

If TryForward(True) = False Then

Return False

End If

End If

End If

End If

* + - * 1. End If
      1. End If
      2. End If
    1. End If
    2. End If
  1. End If
  2. ScanRecordCount = ScanRecordCount + 1
  3. Dim Cleaned As Boolean
  4. Cleaned = RoboCleaner.Clean(ThisRoom, GetCurrentlocation, GetSizeOfBot)
  5. If Cleaned = False Then
     1. LastCleanedTimer = LastCleanedTimer + 1
  6. Else
     1. LastCleanedTimer = True
  7. End If
  8. ThisRoom.WriteRoomToFIle()
  9. Return True

1. End Function
2. Public Function GetLastDirection()
   1. Return LastDirection
3. End Function
4. Public Function TryForward(IgnoreCleaned As Boolean) 'tries to move "up"
   1. Dim Passable As Boolean = True
   2. Dim Count As Integer = 0 'counter to check how much floor is clean along the edge
   3. Dim Limit As Integer 'how much clean floor has to be found before returns false
   4. Dim StoredSizeOfBot As Integer = GetSizeOfBot()
   5. Dim GetLocationX As Integer = GetLocation(0)
   6. Dim GetLocationY As Integer = GetLocation(1)
   7. If IgnoreCleaned = True Then
      1. Limit = StoredSizeOfBot \* 2 'limit set too large to be relevent in the scan
   8. Else
      1. Limit = StoredSizeOfBot - 2 'limit set so that if 2 units of area in front is not cleaned area is cleaned
   9. End If
   10. Dim X As Integer = -StoredSizeOfBot / 2
   11. Do Until X = StoredSizeOfBot / 2 Or Count >= Limit
       1. If GetFloorMapSize(1) - 1 >= GetLocationY + StoredSizeOfBot / 2 + 1 Then 'check for edge of room
       2. If FloorMap(GetLocationY + (StoredSizeOfBot / 2) + 1).ItemAt(GetLocationX + X) > 5 Then 'checks for obsticles
          1. If FloorMap(GetLocationY + (StoredSizeOfBot / 2) + 1).ItemAt(GetLocationX + X) = 8 Then 'checks for cleaned floor
          2. Count = Count + 1
          3. Else
          4. Return False
          5. End If
       3. End If
       4. Else
       5. Return False
       6. End If
       7. X = X + 1
   12. Loop
   13. If Count >= Limit Then
       1. Return False
   14. End If
   15. If Passable = True Then
       1. ChangeLocation(0, GetSpeedOfBot) 'move
   16. Else
       1. Passable = False
   17. End If
   18. Return Passable
5. End Function
6. Public Function TryLeft(IgnoreCleaned As Boolean) 'tries to move left
   1. Dim Passable As Boolean = True
   2. Dim Count As Integer = 0
   3. Dim Limit As Integer
   4. Dim StoredSizeOfBot As Integer = GetSizeOfBot()
   5. Dim GetLocationX As Integer = GetLocation(0)
   6. Dim GetLocationY As Integer = GetLocation(1)
   7. If IgnoreCleaned = True Then
      1. Limit = StoredSizeOfBot \* 2
   8. Else
      1. Limit = StoredSizeOfBot - 2
   9. End If
   10. Dim Y As Integer = -StoredSizeOfBot / 2
   11. Do Until Y = StoredSizeOfBot / 2 Or Count >= Limit Or Passable = False
       1. If 0 <= GetLocationX - StoredSizeOfBot / 2 - 1 Then
       2. If FloorMap(GetLocationY + Y).ItemAt(GetLocationX - (StoredSizeOfBot / 2) - 1) > 5 Then
          1. If FloorMap(GetLocationY + Y).ItemAt(GetLocationX - (StoredSizeOfBot / 2) - 1) = 8 Then
          2. Count = Count + 1
          3. Else
          4. Return False
          5. End If
       3. End If
       4. Else
       5. Return False
       6. End If
       7. Y = Y + 1
   12. Loop
   13. If Count >= Limit Then
       1. Return False
   14. End If
   15. If Passable = True Then
       1. ChangeLocation(-GetSpeedOfBot(), 0)
   16. Else
       1. Passable = False
   17. End If
   18. Return Passable
7. End Function
8. Public Function TryRight(IgnoreCleaned As Boolean) 'tries to move right
   1. Dim Passable As Boolean = True
   2. Dim Count As Integer = 0
   3. Dim Limit As Integer
   4. Dim StoredSizeOfBot As Integer = GetSizeOfBot()
   5. Dim GetLocationX As Integer = GetLocation(0)
   6. Dim GetLocationY As Integer = GetLocation(1)
   7. If IgnoreCleaned = True Then
      1. Limit = GetSizeOfBot() \* 2
   8. Else
      1. Limit = GetSizeOfBot() - 2
   9. End If
   10. Dim Y As Integer = -StoredSizeOfBot / 2
   11. Do Until Y = StoredSizeOfBot / 2 Or Count >= Limit Or Passable = False
       1. If GetFloorMapSize(0) - 1 >= GetLocationX + StoredSizeOfBot / 2 + 1 Then
       2. If FloorMap(GetLocationY + Y).ItemAt(GetLocationX + (StoredSizeOfBot / 2) + 1) > 5 Then
          1. If FloorMap(GetLocationY + Y).ItemAt(GetLocationX + (StoredSizeOfBot / 2) + 1) = 8 Then
          2. Count = Count + 1
          3. Else
          4. Return False
          5. End If
       3. End If
       4. Else
       5. Return False
       6. End If
       7. Y = Y + 1
   12. Loop
   13. If Count >= Limit Then
       1. Return False
   14. End If
   15. If Passable = True Then
       1. ChangeLocation(GetSpeedOfBot(), 0)
   16. Else
       1. Passable = False
   17. End If
   18. Return Passable
9. End Function
10. Public Function TryBackWards(IgnoreCleaned As Boolean) 'tries to move "down"
    1. Dim Passable As Boolean = True
    2. Dim Count As Integer = 0
    3. Dim Limit As Integer
    4. Dim StoredSizeOfBot As Integer = GetSizeOfBot()
    5. Dim GetLocationX As Integer = GetLocation(0)
    6. Dim GetLocationY As Integer = GetLocation(1)
    7. If IgnoreCleaned = True Then
       1. Limit = GetSizeOfBot() \* 2
    8. Else
       1. Limit = GetSizeOfBot() - 2
    9. End If
    10. Dim X As Integer = -StoredSizeOfBot / 2
    11. Do Until X = StoredSizeOfBot / 2 Or Count >= Limit Or Passable = False
        1. If 0 <= GetLocationY - (StoredSizeOfBot / 2) - 1 Then
        2. If FloorMap(GetLocationY - (StoredSizeOfBot / 2) - 1).ItemAt(GetLocationX + X) > 5 Then
           1. If FloorMap(GetLocationY - (StoredSizeOfBot / 2) - 1).ItemAt(GetLocationX + X) = 8 Then
           2. Count = Count + 1
           3. Else
           4. Return False
           5. End If
        3. End If
        4. Else
        5. Return False
        6. End If
        7. X = X + 1
    12. Loop
    13. If Count >= Limit Then
        1. Return False
    14. End If
    15. If Passable = True Then
        1. ChangeLocation(0, -GetSpeedOfBot())
    16. Else
        1. Passable = False
    17. End If
    18. Return Passable
11. End Function
12. Public Function GetFloorMapSize(XY As Integer)
    1. If XY = 0 Then
       1. Return FloorMap(0).CountItems()
    2. Else
       1. Return FloorMap.Count
    3. End If
13. End Function
14. Public Overrides Sub GoToCharger(ChargerLocationX As Integer, ChargerLocactionY As Integer) 'tries to return to charger
    1. If ChargerLocactionY < GetLocation(1) Then 'if charger is below go down
       1. If TryBackWards(True) = False Then
       2. TryForward(True)
       3. End If
    2. ElseIf ChargerLocactionY > GetLocation(1) Then 'if charger is above go up
       1. If TryForward(True) = False Then
       2. TryBackWards(True)
       3. End If
    3. End If
    4. If ChargerLocationX > GetLocation(0) Then 'if charger is to the right go right
       1. If TryRight(True) = False Then
       2. TryLeft(True)
       3. End If
    5. ElseIf ChargerLocationX < GetLocation(0) Then 'if charger is to the left go left
       1. If TryLeft(True) Then
       2. TryRight(True)
       3. End If
    6. End If
15. End Sub
16. End Class
17. Class Control 'control robot
18. Inherits Robot
19. Private DirectionFacing As Integer '1 to 4 represents N,E,S,W
20. Private RobotSens As New Sensor
21. Private RoboCleaner As New Cleaner
22. Private Count As Integer = 0 'if bot changes to random direction every 100 moves to reduce getting stuck
23. Private Count2 As Integer = 0 'bot only cleans when it needs to
24. Sub New(RoomsizeX As Integer, RoomSizeY As Integer)
    1. MyBase.New(RoomsizeX, RoomSizeY)
    2. DirectionFacing = CreateRandInt(1, 4)
25. End Sub
26. Public Sub Move(ThisRoom2 As Room) 'moves bot if it can
    1. Do While RobotSens.CheckInFront(DirectionFacing, Me, ThisRoom2) = False Or Count > 100
       1. DirectionFacing = CreateRandInt(1, 4)
       2. If Count <= 100 Then
       3. Count = Count + 1
       4. Else
       5. Count = 0
       6. End If
    2. Loop
    3. If DirectionFacing = 1 Then
       1. ChangeLocation(0, 1)
    4. ElseIf DirectionFacing = 2 Then
       1. ChangeLocation(1, 0)
    5. ElseIf DirectionFacing = 3 Then
       1. ChangeLocation(0, -1)
    6. ElseIf DirectionFacing = 4 Then
       1. ChangeLocation(-1, 0)
    7. End If
    8. If Count2 <= GetSizeOfBot() Then
       1. RoboCleaner.Clean(ThisRoom2, GetCurrentlocation, GetSizeOfBot)
       2. Count = Count + 1
    9. Else
       1. Count = 0
    10. End If
27. End Sub
28. End Class
29. Class Sensor 'scans room
30. Private ScanRange As Integer = 10
31. Public Sub SweepRecord(MyAlgo As ImprovedAlgo, ThisRoom As Room) 'Records the surrounding area on the map
    1. Dim EndScanOfStrip As Boolean = False 'if scanner finds an object in the room it stops scanning past thats for efficiency
    2. For Y = MyAlgo.GetSizeOfBot / 2 To -MyAlgo.GetSizeOfBot / 2 Step -1 'Scans left side of bot
       1. For X = -(MyAlgo.GetSizeOfBot / 2) - 1 To -MyAlgo.GetSizeOfBot / 2 - ScanRange Step -1 And EndScanOfStrip = False 'from the bot to the max range of the scanner
       2. If ThisRoom.GetRoomSize(0) - 1 < MyAlgo.GetLocation(0) + X Or ThisRoom.GetRoomSize(1) - 1 < MyAlgo.GetLocation(1) + Y Or MyAlgo.GetLocation(0) + X < 0 Or MyAlgo.GetLocation(1) + Y < 0 Then
          1. EndScanOfStrip = True
       3. Else
          1. If ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) <= 5 Then
          2. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y)) 'record what is found to the map
          3. ElseIf ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) > 5 Then
          4. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y))
          5. EndScanOfStrip = True
          6. End If
       4. End If
       5. Next
       6. EndScanOfStrip = False
    3. Next
    4. For Y = MyAlgo.GetSizeOfBot / 2 To -MyAlgo.GetSizeOfBot / 2 Step -1 'Scans right side of bot
       1. For X = MyAlgo.GetSizeOfBot / 2 + 1 To MyAlgo.GetSizeOfBot / 2 + ScanRange Step 1 And EndScanOfStrip = False
       2. If ThisRoom.GetRoomSize(0) - 1 < MyAlgo.GetLocation(0) + X Or ThisRoom.GetRoomSize(1) - 1 < MyAlgo.GetLocation(1) + Y Or MyAlgo.GetLocation(0) + X < 0 Or MyAlgo.GetLocation(1) + Y < 0 Then
          1. EndScanOfStrip = True
       3. Else
          1. If ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) <= 5 Then
          2. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y))
          3. ElseIf ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) > 5 Then
          4. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y))
          5. EndScanOfStrip = True
          6. End If
       4. End If
       5. Next
       6. EndScanOfStrip = False
    5. Next
    6. For X = -MyAlgo.GetSizeOfBot / 2 To MyAlgo.GetSizeOfBot / 2 'Scans forward
       1. For Y = (MyAlgo.GetSizeOfBot / 2) + 1 To MyAlgo.GetSizeOfBot / 2 - ScanRange Step -1 And EndScanOfStrip = False
       2. If ThisRoom.GetRoomSize(0) - 1 < MyAlgo.GetLocation(0) + X Or ThisRoom.GetRoomSize(1) - 1 < MyAlgo.GetLocation(1) + Y Or MyAlgo.GetLocation(0) + X < 0 Or MyAlgo.GetLocation(1) + Y < 0 Then
          1. EndScanOfStrip = True
       3. Else
          1. If ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) <= 5 Then
          2. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y))
          3. ElseIf ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) > 5 Then
          4. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y))
          5. EndScanOfStrip = True
          6. End If
       4. End If
       5. Next
       6. EndScanOfStrip = False
    7. Next
    8. For X = -MyAlgo.GetSizeOfBot / 2 To MyAlgo.GetSizeOfBot / 2 'scans Backward
       1. For Y = -(MyAlgo.GetSizeOfBot / 2) - 1 To -MyAlgo.GetSizeOfBot / 2 - ScanRange Step -1 And EndScanOfStrip = False
       2. If ThisRoom.GetRoomSize(0) - 1 < MyAlgo.GetLocation(0) + X Or ThisRoom.GetRoomSize(1) - 1 < MyAlgo.GetLocation(1) + Y Or MyAlgo.GetLocation(0) + X < 0 Or MyAlgo.GetLocation(1) + Y < 0 Then
          1. EndScanOfStrip = True
       3. Else
          1. If ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) <= 5 Then
          2. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y))
          3. ElseIf ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y) > 5 Then
          4. MyAlgo.SetFloorMap(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y, ThisRoom.FloorAt(MyAlgo.GetLocation(0) + X, MyAlgo.GetLocation(1) + Y))
          5. EndScanOfStrip = True
          6. End If
       4. End If
       5. Next
       6. EndScanOfStrip = False
    9. Next
32. End Sub
33. Public Function GetScanRange()
    1. Return ScanRange
34. End Function
35. Public Function CheckInFront(NESW As Integer, ControlAlgo As Control, ThisRoom2 As Room) 'Returns True if no obsticals, Checks if there is an obstical in a direction
    1. If NESW = 1 Then 'checks forward
       1. If ControlAlgo.GetLocation(1) + ControlAlgo.GetSizeOfBot / 2 + 1 > ThisRoom2.GetRoomSize(1) - 1 Then 'if at edge of room return false
       2. Return False
       3. Else 'Checks along edges of bot
       4. For X = -ControlAlgo.GetSizeOfBot() / 2 To ControlAlgo.GetSizeOfBot() / 2
          1. If ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) + X, ControlAlgo.GetLocation(1) + ControlAlgo.GetSizeOfBot / 2 + 1) > 5 And
          2. ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) + X, ControlAlgo.GetLocation(1) + ControlAlgo.GetSizeOfBot / 2 + 1) < 8 Then 'if it finds obstical return false
          3. Return False
          4. End If
       5. Next
       6. End If
    2. ElseIf NESW = 2 Then
       1. If ControlAlgo.GetLocation(0) + ControlAlgo.GetSizeOfBot() / 2 + 1 > ThisRoom2.GetRoomSize(0) - 1 Then
       2. Return False
       3. Else
       4. For Y = -ControlAlgo.GetSizeOfBot() / 2 To ControlAlgo.GetSizeOfBot() / 2
          1. If ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) + ControlAlgo.GetSizeOfBot / 2 + 1, ControlAlgo.GetLocation(1) + Y) > 5 And
          2. ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) + ControlAlgo.GetSizeOfBot / 2 + 1, ControlAlgo.GetLocation(1) + Y) < 8 Then
          3. Return False
          4. End If
       5. Next
       6. End If
    3. ElseIf NESW = 3 Then
       1. If ControlAlgo.GetLocation(1) - ControlAlgo.GetSizeOfBot / 2 - 1 < 0 Then
       2. Return False
       3. Else
       4. For X = -ControlAlgo.GetSizeOfBot() / 2 To ControlAlgo.GetSizeOfBot() / 2
          1. If ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) + X, ControlAlgo.GetLocation(1) - ControlAlgo.GetSizeOfBot / 2 - 1) > 5 And
          2. ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) + X, ControlAlgo.GetLocation(1) - ControlAlgo.GetSizeOfBot / 2 - 1) < 8 Then
          3. Return False
          4. End If
       5. Next
       6. End If
    4. ElseIf NESW = 4 Then
       1. If ControlAlgo.GetLocation(0) - ControlAlgo.GetSizeOfBot / 2 - 1 < 0 Then
       2. Return False
       3. Else
       4. For Y = -ControlAlgo.GetSizeOfBot() / 2 To ControlAlgo.GetSizeOfBot() / 2
          1. If ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) - ControlAlgo.GetSizeOfBot / 2 - 1, ControlAlgo.GetLocation(1) + Y) > 5 And
          2. ThisRoom2.FloorAt(ControlAlgo.GetLocation(0) - ControlAlgo.GetSizeOfBot / 2 - 1, ControlAlgo.GetLocation(1) + Y) < 8 Then
          3. Return False
          4. End If
       5. Next
       6. End If
    5. End If
    6. Return True
36. End Function
37. End Class
38. Class Cleaner 'Cleans the room
39. Public Function Clean(ThisRoom As Room, BotLocation As Robot.Location, BotSize As Integer)
    1. Dim Cleaned As Boolean = False
    2. For Y = BotLocation.YAxis - BotSize / 2 To BotLocation.YAxis + BotSize / 2
       1. For X = BotLocation.XAxis - BotSize / 2 To BotLocation.XAxis + BotSize / 2
       2. If ThisRoom.FloorAt(X, Y) < 5 Then
          1. ThisRoom.SetFloor(X, Y, 8)
          2. Cleaned = True
       3. End If
       4. Next
    3. Next
    4. Return Cleaned
40. End Function
41. End Class
42. Class Battery 'battery info
43. Inherits GenRandomNum
44. Dim BatteryLevel As Decimal
45. Sub New()
    1. SetRandomBetteryLevel()
46. End Sub
47. Public Sub SetBatteryLevel(Value As Integer)
    1. BatteryLevel = Value
48. End Sub
49. Public Sub DecreaseBattery() 'drains battery
    1. BatteryLevel = BatteryLevel - 0.01
    2. Debug.WriteLine(BatteryLevel)
50. End Sub
51. Public Sub SetRandomBetteryLevel()
    1. SetBatteryLevel(CreateRandInt(50, 100))
52. End Sub
53. Public Function GetBatteryLevel()
    1. Return BatteryLevel
54. End Function
55. End Class
56. Class Rows 'Rows for the room and bot maps
57. Private Items As New List(Of Integer)
58. Public Sub AddItems()
    1. Items.Add(New Integer)
59. End Sub
60. Public Sub SetItems(X As Integer, Value As Integer)
    1. Items(X) = Value
61. End Sub
62. Public Function CountItems()
    1. Return Items.Count
63. End Function
64. Public Function ItemAt(X As Integer)
    1. Return Items(X)
65. End Function
66. Public Sub RemoveItem()
    1. Items.RemoveAt(Items.Count - 1)
67. End Sub
68. End Class
69. Class CornerData 'Groups of corners corners
70. Private Corners As New List(Of Corner)
71. Public Sub SetCorners(X As Integer, Y As Integer, WhichCorner As Integer)
    1. Corners(WhichCorner).SetCorner(0, X)
    2. Corners(WhichCorner).SetCorner(1, Y)
72. End Sub
73. Public Function GetCorners(WhichCorner As Integer, XY As Integer)
    1. Return Corners(WhichCorner).GetCorner(XY)
74. End Function
75. Public Sub AddCorners()
    1. Corners.Add(New Corner)
76. End Sub
77. Class Corner 'specific corners
    1. Private X As Integer
    2. Private Y As Integer
    3. Public Sub SetCorner(XY As Integer, Value As Integer)
       1. If XY = 0 Then
       2. X = Value
       3. Else
       4. Y = Value
       5. End If
    4. End Sub
    5. Public Function GetCorner(XY As Integer)
       1. If XY = 0 Then
       2. Return X
       3. Else
       4. Return Y
       5. End If
    6. End Function
78. End Class
79. End Class
80. Class Room 'room info
81. Inherits GenRandomNum
82. Private EntireRoom As New List(Of Rows)
83. Private Walls As New List(Of Wall)
84. Private Furnitures As New List(Of Furniture)
85. Private ChargerObj As New Charger
86. Sub New(Clone As Boolean, XSize As Integer, YSize As Integer)
    1. CreateRoom(Clone, XSize, YSize)
    2. RoomWalls(GetRoomSize(0), GetRoomSize(1)) 'Bounding Walls for the room
    3. GenerateWalls()
    4. GenerateFurniture()
    5. FillArea(0, 0, EntireRoom(0).CountItems - 1, EntireRoom.Count - 1, 1)
    6. WriteRoomToFIle()
87. End Sub
88. Public Sub CreateRoom(Clone As Boolean, XSize As Integer, YSize As Integer)
    1. If Clone <> True Then
       1. SetRoomSize(200, 200)
    2. Else
       1. SetRoomSize(XSize, YSize)
    3. End If
89. End Sub
90. Public Sub WriteRoomToFIle() 'Writes room to a txt document for debugging
    1. Dim FILE\_NAME As String = IO.Path.Combine(My.Application.Info.DirectoryPath, "Rooms\Room.txt")
    2. System.IO.File.WriteAllText(FILE\_NAME, "")
    3. If System.IO.File.Exists(FILE\_NAME) = True Then
       1. Dim objWriter As New System.IO.StreamWriter(FILE\_NAME)
       2. For Y = GetRoomSize(1) - 1 To 0 Step -1
       3. For X = 0 To GetRoomSize(0) - 1
          1. objWriter.Write(FloorAt(X, Y))
       4. Next
       5. objWriter.WriteLine("")
       6. Next
       7. objWriter.Close()
    4. Else
       1. MessageBox.Show("File Does Not Exist")
    5. End If
91. End Sub
92. Class Charger 'Charger info
    1. Private ChargerLocation As Location
    2. Private Corners As New CornerData
    3. Private ChargerSize As Integer = 5
    4. Private MapKey As Integer = 9
    5. Structure Location 'location data type
       1. Dim XAxis As Integer
       2. Dim YAxis As Integer
    6. End Structure
    7. Public Sub SetChargerLocation(ValueX As Integer, ValueY As Integer)
       1. ChargerLocation.XAxis = ValueX
       2. ChargerLocation.YAxis = ValueY
       3. For index = 0 To 3
       4. Corners.AddCorners()
       5. Next
       6. Corners.SetCorners(ValueX - (ChargerSize / 2), ValueY - (ChargerSize / 2), 0)
       7. Corners.SetCorners(ValueX + (ChargerSize / 2), ValueY - (ChargerSize / 2), 1)
       8. Corners.SetCorners(ValueX + (ChargerSize / 2), ValueY + (ChargerSize / 2), 2)
       9. Corners.SetCorners(ValueX - (ChargerSize / 2), ValueY + (ChargerSize / 2), 3)
    8. End Sub
    9. Public Function GetMapKey()
       1. Return MapKey
    10. End Function
    11. Public Function GetChargerLocation(XY As Integer)
        1. If XY = 0 Then
        2. Return ChargerLocation.XAxis
        3. Else
        4. Return ChargerLocation.YAxis
        5. End If
    12. End Function
    13. Public Function GetCorners()
        1. Return Corners
    14. End Function
93. End Class
94. Public Sub SetRoomSize(X As Integer, Y As Integer)
    1. For YAxis = 0 To Y
       1. EntireRoom.Add(New Rows)
       2. For XAxis = 0 To X
       3. EntireRoom(YAxis).AddItems()
       4. Next
    2. Next
95. End Sub
96. Public Sub GenerateWalls() 'Sets walls in the EntireRoom
    1. Dim Numberofwalls As Integer = CreateRandInt(0, 4)
    2. Dim GradientofWall As Long
    3. Dim WallIntercept As Decimal
    4. For index = 4 To Numberofwalls + 4
       1. Dim StepY As Integer = 1
       2. Dim StepX As Integer = 1
       3. Walls.Add(New Wall(GetRoomSize(0), GetRoomSize(1), False))
       4. If Walls(index).GetWallEnds(0, 1) > Walls(index).GetWallEnds(1, 1) Then
       5. StepY = -1
       6. End If
       7. If Walls(index).GetWallEnds(0, 0) > Walls(index).GetWallEnds(1, 0) Then
       8. StepX = -1
       9. End If
       10. If Walls(index).GetWallEnds(0, 0) - Walls(index).GetWallEnds(1, 0) <> 0 Then
       11. GradientofWall = Walls(index).WallGradient(Walls(index).GetWallEnds(0, 0), Walls(index).GetWallEnds(0, 1), Walls(index).GetWallEnds(1, 0), Walls(index).GetWallEnds(1, 1))
       12. WallIntercept = Walls(index).CalculateC(GradientofWall, Walls(index).GetWallEnds(0, 0), Walls(index).GetWallEnds(0, 1))
       13. If GradientofWall <> 0 Then 'if it isnt vertical
           1. For Y = Walls(index).GetWallEnds(0, 1) To Walls(index).GetWallEnds(1, 1) Step StepY
           2. EntireRoom(Y).SetItems(Walls(index).WallEquation(GradientofWall, Y, WallIntercept, EntireRoom(0).CountItems), Walls(index).GetMapKey)
           3. Next
       14. Else 'if it is horizontal
           1. For X = Walls(index).GetWallEnds(0, 0) To Walls(index).GetWallEnds(1, 0) Step StepX
           2. EntireRoom(Walls(index).GetWallEnds(0, 1)).SetItems(X, Walls(index).GetMapKey)
           3. Next
       15. End If
       16. Else 'if vertical
       17. For Y = Walls(index).GetWallEnds(0, 1) To Walls(index).GetWallEnds(1, 1) Step StepY
           1. EntireRoom(Y).SetItems(Walls(index).GetWallEnds(0, 0), Walls(index).GetMapKey)
       18. Next
       19. End If
    5. Next
97. End Sub
98. Public Sub GenerateFurniture() 'creates the furniture in the room
    1. Dim NumberofPeicesofFurniture As Integer = CreateRandInt(1, 5)
    2. For index = 0 To NumberofPeicesofFurniture
       1. Furnitures.Add(New Furniture(GetRoomSize(0), GetRoomSize(1), EntireRoom))
       2. If Furnitures(index).GetGenerateLegs = 0 Then
       3. For index2 = 0 To 3
          1. FillArea(Furnitures(index).GetLegsCorners(index2, 0, 0), Furnitures(index).GetLegsCorners(index2, 0, 1), Furnitures(index).GetLegsCorners(index2, 2, 0), Furnitures(index).GetLegsCorners(index2, 2, 1), Furnitures(index).GetMapKey)
       4. Next
       5. Else
       6. FillArea(Furnitures(index).GetFurnituresCorners(0, 0), Furnitures(index).GetFurnituresCorners(0, 1), Furnitures(index).GetFurnituresCorners(2, 0), Furnitures(index).GetFurnituresCorners(2, 1), Furnitures(index).GetMapKey)
       7. End If
    3. Next
99. End Sub
100. Public Function FloorAt(X As Integer, Y As Integer) 'returns the floor at a location
     1. Return EntireRoom(Y).ItemAt(X)
101. End Function
102. Public Function GetRoomSize(xy As Integer) 'returns the size of the room
     1. If xy = 0 Then
        1. Return EntireRoom(0).CountItems()
     2. Else
        1. Return EntireRoom.Count
     3. End If
103. End Function
104. Public Sub FillArea(X1 As Integer, Y1 As Integer, X2 As Integer, Y2 As Integer, MapKey As Integer) 'fills in areas of the room
     1. Dim StepCountY, StepCountX As Integer
     2. If Y1 > Y2 Then
        1. StepCountY = -1
     3. Else
        1. StepCountY = 1
     4. End If
     5. If X1 > X2 Then
        1. StepCountX = -1
     6. Else
        1. StepCountX = 1
     7. End If
     8. For Y = Y1 To Y2 Step StepCountY
        1. For X = X1 To X2 Step StepCountX
        2. If MapKey > FloorAt(X, Y) Or FloorAt(X, Y) = 0 Then
           1. EntireRoom(Y).SetItems(X, MapKey)
        3. End If
        4. Next
     9. Next
105. End Sub
106. Public Sub RoomWalls(RoomSizeX As Integer, RoomSizeY As Integer) 'create bounding walls for the room
     1. For index = 0 To 3
        1. Walls.Add(New Wall(RoomSizeX, RoomSizeY, True))
     2. Next
     3. Walls(0).SetWallEnds(0, 0, 0)
     4. Walls(0).SetWallEnds(RoomSizeX - 1, 0, 1)
     5. Walls(1).SetWallEnds(RoomSizeX - 1, 0, 0)
     6. Walls(1).SetWallEnds(RoomSizeX - 1, RoomSizeY - 1, 1)
     7. Walls(2).SetWallEnds(RoomSizeX - 1, RoomSizeY - 1, 0)
     8. Walls(2).SetWallEnds(0, RoomSizeY - 1, 1)
     9. Walls(3).SetWallEnds(0, RoomSizeY - 1, 0)
     10. Walls(3).SetWallEnds(0, 0, 1)
     11. For index = 0 To 3
         1. Dim XStep As Integer = 1
         2. Dim YStep As Integer = 1
         3. If Walls(index).GetWallEnds(0, 1) > Walls(index).GetWallEnds(1, 1) Then
         4. YStep = -1
         5. End If
         6. If Walls(index).GetWallEnds(0, 0) > Walls(index).GetWallEnds(1, 0) Then
         7. XStep = -1
         8. End If
         9. For Y = Walls(index).GetWallEnds(0, 1) To Walls(index).GetWallEnds(1, 1) Step YStep
         10. For X = Walls(index).GetWallEnds(0, 0) To Walls(index).GetWallEnds(1, 0) Step XStep
             1. EntireRoom(Y).SetItems(X, Walls(0).GetMapKey)
         11. Next
         12. Next
     12. Next
107. End Sub
108. Public Function RobotStartingPosition(SizeofBot As Integer) 'Returns a location the robot can start
     1. Dim Start(1), Count As Integer
     2. Dim Valid As Boolean = True
     3. Count = 0
     4. Do
        1. Valid = True
        2. Start(0) = CreateRandInt(SizeofBot / 2 + 1, GetRoomSize(0) - (SizeofBot / 2) - 1) 'tries placing in a random location
        3. Start(1) = CreateRandInt(SizeofBot / 2 + 1, GetRoomSize(1) - (SizeofBot / 2) - 1)
        4. For Y = Start(1) - (SizeofBot / 2) To Start(1) + (SizeofBot / 2)
        5. For X = Start(0) - (SizeofBot / 2) To Start(0) + (SizeofBot / 2) 'checks to make sure its not inside an object
           1. If FloorAt(X, Y) > 5 Then
           2. Valid = False
           3. End If
        6. Next
        7. Next
        8. Count = Count + 1
     5. Loop Until Valid = True Or Count > 1000 'if it cant find a place after 1000 tries returns (0,0)
     6. If Valid = False Then
        1. MsgBox("Bot could not be placed in the room")
        2. Start(0) = 0
        3. Start(1) = 0
     7. End If
     8. Return Start
109. End Function
110. Public Function GetFurniture()
     1. Return Furnitures
111. End Function
112. Public Function GetWalls()
     1. Return Walls
113. End Function
114. Public Sub SetFloor(X As Integer, Y As Integer, Value As Integer) 'sets floor at a location with a value
     1. EntireRoom(Y).SetItems(X, Value)
115. End Sub
116. Public Sub CopyRoom(ByVal RoomToCopy As Room, ByRef RoomToReplace As Room) 'copys one room to another
     1. If RoomToCopy.GetRoomSize(1) < RoomToReplace.GetRoomSize(1) Then
        1. For RemoveFromY = 0 To RoomToReplace.GetRoomSize(1) - RoomToCopy.GetRoomSize(1)
        2. RoomToReplace.EntireRoom.RemoveAt(RoomToReplace.GetRoomSize(1) - 1)
        3. Next
     2. End If
     3. For Y = 0 To RoomToCopy.GetRoomSize(1) - 1
        1. For X = 0 To RoomToCopy.GetRoomSize(0) - 1
        2. RoomToReplace.SetFloor(X, Y, RoomToCopy.FloorAt(X, Y))
        3. Next
     4. Next
117. End Sub
118. Public Function GetChargerObj()
     1. Return ChargerObj
119. End Function
120. End Class
121. Class Wall 'wall info
122. Inherits GenRandomNum
123. Private WallEnds As New CornerData
124. Private MapKey As Integer = 7
125. Private BoundingWall As Boolean
126. Sub New(RoomSizeX As Integer, RoomSizeY As Integer, RoomWall As Integer) 'when a wall is created checks what type is is and then generates them differently
     1. BoundingWall = RoomWall
     2. If RoomWall = False Then
        1. GenerateWall(RoomSizeX, RoomSizeY)
     3. Else
        1. WallEnds.AddCorners()
        2. WallEnds.AddCorners()
     4. End If
127. End Sub
128. Public Sub GenerateWall(RoomSizeX As Integer, RoomSizeY As Integer) 'generates a random wall
     1. WallEnds.AddCorners()
     2. WallEnds.AddCorners()
     3. SetWallEnds(CreateRandInt(0, RoomSizeX - 1), CreateRandInt(0, RoomSizeY - 1), 0)
     4. SetWallEnds(CreateRandInt(0, RoomSizeX - 1), CreateRandInt(0, RoomSizeY - 1), 1)
129. End Sub
130. Public Sub SetWallEnds(X As Integer, Y As Integer, WhichEnd As Integer)
     1. WallEnds.SetCorners(X, Y, WhichEnd)
131. End Sub
132. Public Function WallGradient(End1X As Integer, End1Y As Integer, End2X As Integer, End2Y As Integer) 'calculates wall gradient
     1. Return (End1Y - End2Y) / (End1X - End2X)
133. End Function
134. Public Function CalculateC(Gradient As Decimal, X As Integer, Y As Integer) 'calculates wall's y intercept
     1. Return Y - (Gradient \* X)
135. End Function
136. Public Function WallEquation(Gradient As Decimal, Y As Integer, C As Integer, RoomSizeX As Integer) 'input y outputs x
     1. If ((Y - C) / Gradient) <= 0 Then
        1. Return 0
     2. ElseIf ((Y - C) / Gradient) >= RoomSizeX - 1 Then
        1. Return RoomSizeX - 1
     3. Else
        1. Return (Y - C) / Gradient
     4. End If
137. End Function
138. Public Function GetWallEnds(Index As Integer, XY As Integer) 'returns the location of a wall end
     1. Return WallEnds.GetCorners(Index, XY)
139. End Function
140. Public Function GetMapKey()
     1. Return MapKey
141. End Function
142. End Class
143. Class Furniture 'furniture info
144. Inherits GenRandomNum
145. Private FurnitureCorners As New CornerData
146. Private Legs As New List(Of Leg)
147. Private MapKey As Integer = 6
148. Private GenerateLegs As Integer
149. Private MaximumLegsize As Integer = 10
150. Sub New(RoomSizeX As Integer, RoomsizeY As Integer, EntireRoom As List(Of Rows)) 'creates a random piece of furniture
     1. Dim Size(1) As Integer
     2. GenerateLegs = CreateRandInt(0, 1) 'decides if a piece of furniture has legs
     3. For index = 1 To 4
        1. FurnitureCorners.AddCorners() 'adds corners
     4. Next
     5. Dim Valid As Boolean = False
     6. Dim Count As Integer = 0
     7. 'Do
     8. Size(0) = CreateRandInt(30, 60) 'Generates sizes between 30 to 60 cm
     9. Size(1) = CreateRandInt(30, 60)
     10. FurnitureCorners.SetCorners(CreateRandInt(MaximumLegsize \ 2, RoomSizeX - (Size(0) + (MaximumLegsize \ 2))), CreateRandInt(MaximumLegsize \ 2, RoomsizeY - (Size(1) + (MaximumLegsize \ 2))), 0) 'sets the 0ths corners to a random location in the room
     11. FurnitureCorners.SetCorners(FurnitureCorners.GetCorners(0, 0) + Size(0), FurnitureCorners.GetCorners(0, 1) + Size(1), 2) 'sets corner 2, the opposing corner, the generated size away
     12. FurnitureCorners.SetCorners(FurnitureCorners.GetCorners(0, 0), FurnitureCorners.GetCorners(2, 1), 1) 'sets other 2 corners
     13. FurnitureCorners.SetCorners(FurnitureCorners.GetCorners(2, 0), FurnitureCorners.GetCorners(0, 1), 3)
     14. Count = Count + 1
     15. If GenerateLegs = 0 Then 'generates the legs if needed
         1. AddLegs()
         2. For index = 0 To 3
         3. Legs(index).SetLegCorners(FurnitureCorners.GetCorners(index, 0), FurnitureCorners.GetCorners(index, 1))
         4. Next
     16. End If
151. End Sub
152. Public Sub AddLegs()
     1. For index = 1 To 4
        1. Legs.Add(New Leg)
     2. Next
153. End Sub
154. Public Function GetGenerateLegs()
     1. Return GenerateLegs
155. End Function
156. Public Function GetFurnituresCorners(WhichCorner As Integer, XY As Integer)
     1. Return FurnitureCorners.GetCorners(WhichCorner, XY)
157. End Function
158. Public Function GetLegsCorners(WhichLeg As Integer, WhichCornerofLeg As Integer, XY As Integer)
     1. Return Legs(WhichLeg).GetLegCorners(WhichCornerofLeg, XY)
159. End Function
160. Public Function GetMapKey()
     1. Return MapKey
161. End Function
162. End Class
163. Class Leg 'leg info
164. Inherits GenRandomNum
165. Private LegSize As Integer
166. Private LegCorners As New CornerData
167. Sub New()
     1. For index = 1 To 4
        1. LegCorners.AddCorners()
     2. Next
     3. LegSize = 5 'CreateRandInt(2, 10)
168. End Sub
169. Public Sub SetLegCorners(X As Integer, Y As Integer) 'Adds Corners around the point x,y
     1. LegCorners.SetCorners(X + (LegSize / 2), Y + (LegSize / 2), 0)
     2. LegCorners.SetCorners(X + (LegSize / 2), Y - (LegSize / 2), 1)
     3. LegCorners.SetCorners(X - (LegSize / 2), Y - (LegSize / 2), 2)
     4. LegCorners.SetCorners(X - (LegSize / 2), Y + (LegSize / 2), 3)
170. End Sub
171. Public Function GetLegSize()
     1. Return LegSize
172. End Function
173. Public Function GetLegCorners(WhichCornerOfleg As Integer, XY As Integer)
     1. Return LegCorners.GetCorners(WhichCornerOfleg, XY)
174. End Function
175. End Class
176. Class GenRandomNum 'generates random numbers
177. Sub New()
     1. Randomize()
178. End Sub
179. Function CreateRandInt(Min As Integer, Max As Integer)
     1. Return CInt(Math.Floor((Max - Min + 1) \* Rnd())) + Min
180. End Function
181. End Class

ResultsScreen

1. Public Class ResultsScreen
2. Sub New(GroundCovered, Timer, GroundCoveredControl)
   1. InitializeComponent()
   2. AreaCovered.Text = GroundCovered & "%" 'displays the area covered by my algo
   3. TimeTaken.Text = Timer / 10 'displays the simulation time taken
   4. AreaCoveredByControl.Text = GroundCoveredControl & "%" 'displays area covered by control algo
   5. PercentageDifference.Text = GroundCovered - GroundCoveredControl & "%"
3. End Sub
4. Private Sub MenuButton\_Click(sender As Object, e As EventArgs) Handles MenuButton.Click
   1. Dim EraseWarningScreen As New EraseWarning(Me)
   2. EraseWarningScreen.Activate() 'displays warning
   3. EraseWarningScreen.Show()
5. End Sub
6. Private Sub ExitButton\_Click(sender As Object, e As EventArgs) Handles ExitButton.Click
   1. Me.Close() 'exit
7. End Sub
8. Private Sub ResultsButton\_Click(sender As Object, e As EventArgs) Handles ResultsButton.Click
   1. Process.Start("Explorer.exe", IO.Path.Combine(My.Application.Info.DirectoryPath, "Rooms\Results.txt"))
9. End Sub
10. End Class

EraseWarning

1. Public Class EraseWarning
2. Private Results As ResultsScreen
3. Sub New(PassedResults As ResultsScreen)
   1. InitializeComponent()
   2. Results = PassedResults
4. End Sub
5. Private Sub Label1\_Click(sender As Object, e As EventArgs) Handles Label1.Click 'opens file location if lable is clicked
   1. Process.Start("Explorer.exe", IO.Path.Combine(My.Application.Info.DirectoryPath, "Rooms\Results.txt"))
6. End Sub
7. Private Sub ContinueButton\_Click(sender As Object, e As EventArgs) Handles ContinueButton.Click 'displays menu
   1. Dim FrontPage As New Form1
   2. FrontPage.Activate()
   3. FrontPage.Show()
   4. Results.Close()
   5. Me.Close()
8. End Sub
9. Private Sub ExitButton\_Click(sender As Object, e As EventArgs) Handles ExitButton.Click 'exit
   1. Results.Close()
   2. Me.Close()
10. End Sub
11. End Class