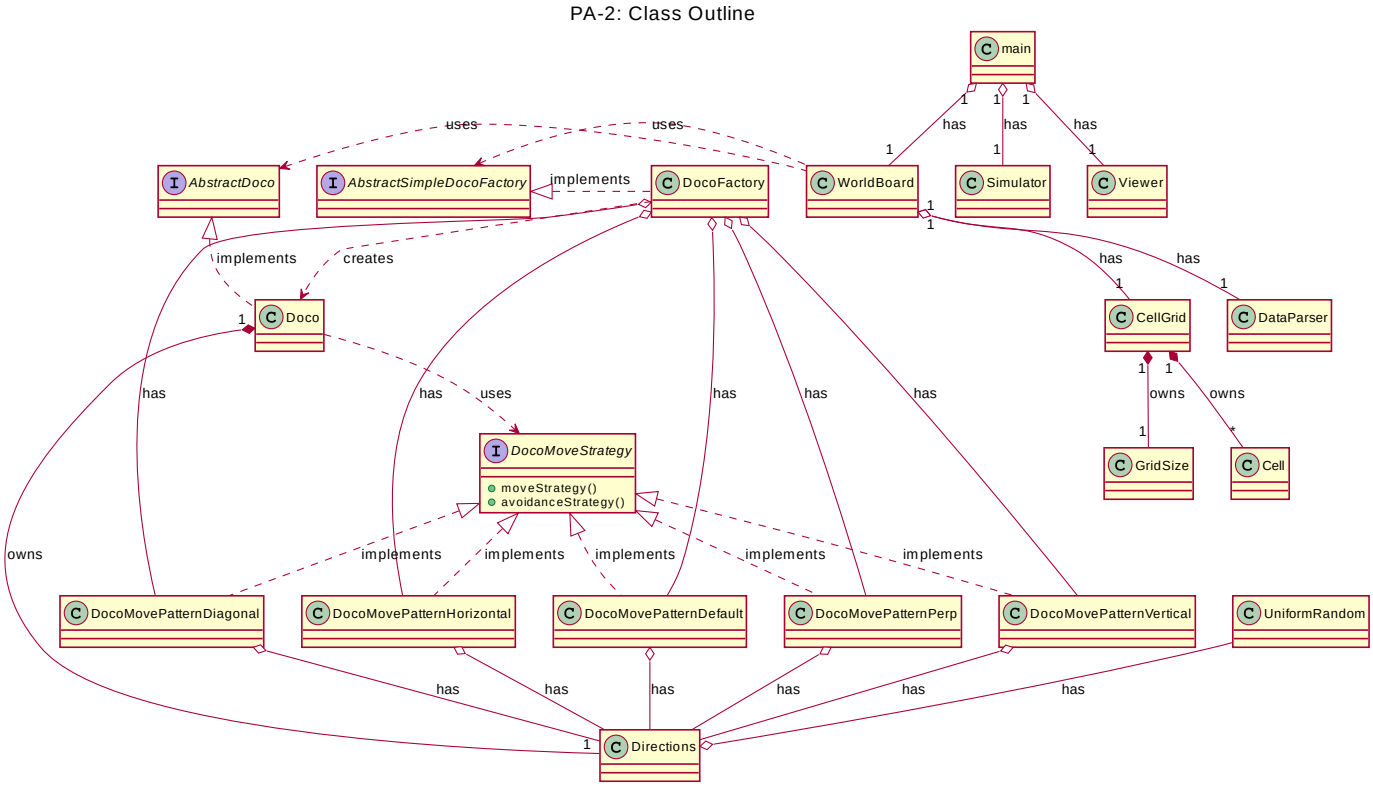
Digital One-Celled Organism (DOCO) Simulation

Programming Assignment 2 (PA-2)
Fall 2020, CS-307

Table of Contents

Preliminary Class Diagram



Last Updated: 10/23/2020

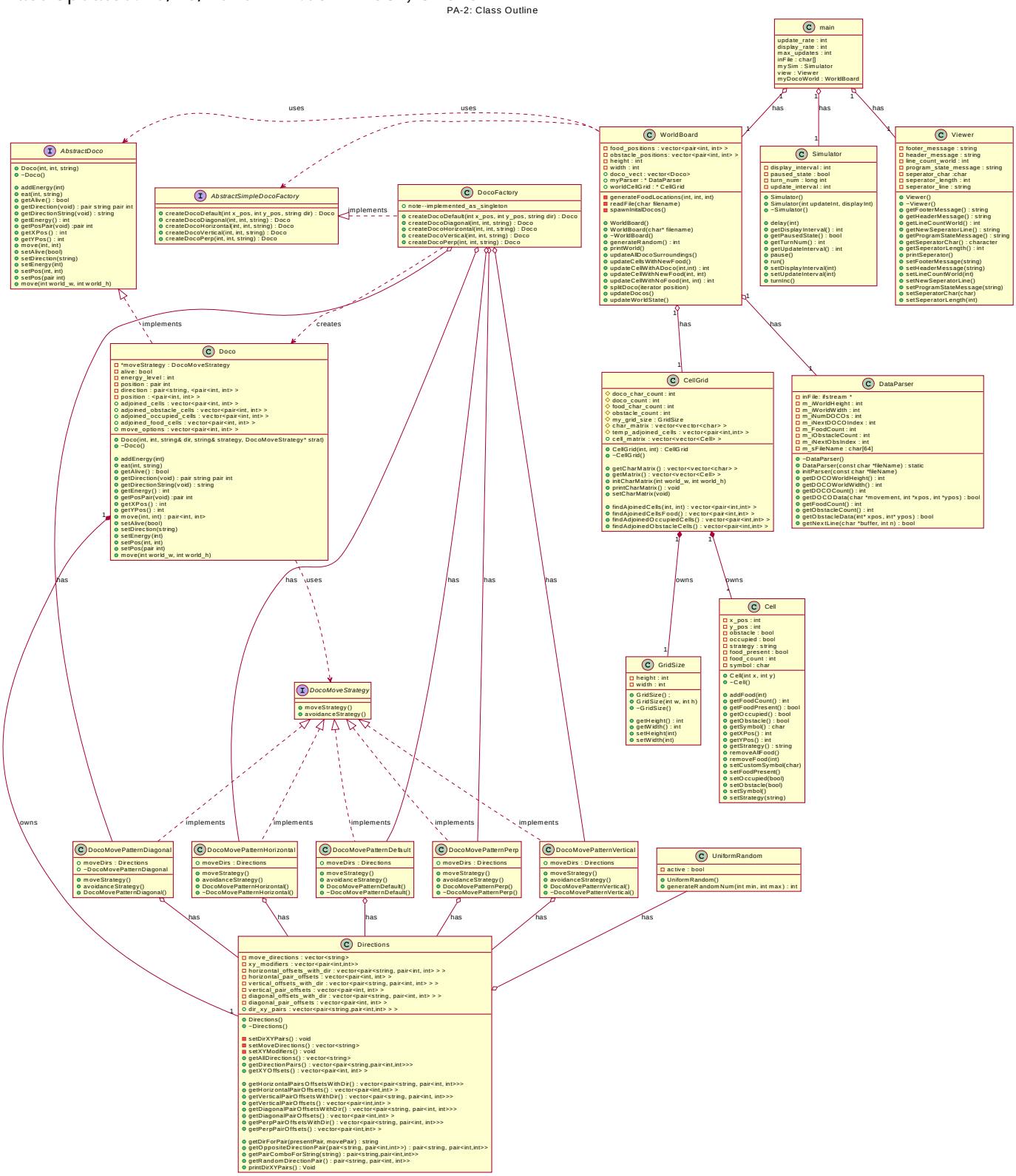
System Overview

Last Updated: 10/23/2020 – 11:58 AM CST, UTC-6

This is a console program that will run a simulation of a world where single celled organisms are spawned in. They will run around eating the food nearby and avoiding walking past the edge of the world. They will avoid walking into each other too. They now also avoid running into obstacles. Additionally strategy pattern and factory pattern have been implemented to allow for different types of DOCOs. The initial world specifications (height, width, DOCO's and positions, and food locations) will be read in from a provided file in XML format. As part of version two, the DOCO strategy is also read in. There are a lot of details involved and they are specified below.

Class Outline UML Diagram

Last Updated: 10/23/2020 – 12:03 PM CST, UTC-6



Functionality Outline

Last Updated: 10/31/2020 – 5:43 PM CST, UTC-6

Important Note: *Return value specified in brackets. Arguments shown inside function. Actions performed explained.*

1 - Main

1.1 - Summary

1.1.1 - The main class is to instantiate the class objects and run the program.

1.2 - Properties

1.2.1 - private:

1.2.1.1 - [int] update_rate – rate of simuluation updating stored here.
1.2.1.2 - [int] display_rate – rate of simulation displaying stored here.
1.2.1.3 - [int] max_updates – maximum number of updates stored here.
1.2.1.4 - [string] inFile –initialization file for objects

1.2.2 - public:

1.2.2.1 - [Viewer] view
1.2.2.2 - [WoarldBoard] myDocoWorld

```
int update_rate = 1;
int display_rate = 1;
Simulator* mySim = new Simulator(update_rate, display_rate); // containers the DocoSim Object
to adjust the sim.
long int max_updates = 1000;
char inFile[] = "DOCOData02.xml";
Viewer* view = new Viewer();
WorldBoard* myDocoWorld = new WorldBoard(inFile);
while (!mySim->getPausedState()) // while not false
{
mySim->delay(1);
system("cls");
std::cout << view->getHeaderMessage() << "\n";
std::cout << view->getNewSeparatorLine() << "\n";
myDocoWorld->updateWorldState();
myDocoWorld->printWorld();
std::cout << "Turn Number:             " << mySim->getTurnNum() << " of " << max_updates << "\
n";
std::cout << view->getNewSeparatorLine() << "\n";
std::cout << view->getFooterMessage() << "\n";
mySim->turnInc();
if (myDocoWorld->doco_vect.size() <= 0) mySim->pause();
if (mySim->getTurnNum() == max_updates) mySim->pause();
}
return 0;
```

1 - Viewer

1.1 - Summary

1.1.1 - the viewer class is just to hold some data for what will be output to the console.

1.2 - Properties

1.2.1 - private:

1.2.1.1 - [string] footer_message – text at bottom of console.
1.2.1.2 - [string] header_message – holds the start message at the top of the console.
1.2.1.3 - [int] line_count_world – keeps the total line count, basically number of matrix rows.
1.2.1.4 - [string] program_state_message – information related to the simulation
1.2.1.5 - [char] seperator_char – char used for the printing breaks
1.2.1.6 - [int] seperator_length – length of the separator lines
1.2.1.7 - [string] seperator_line – a separator line

1.3 - Methods

1.3.1 - public:

1.3.1.1 - [constructor] viewer() - create a viewer object
1.3.1.1.1 - initializes header_message value
1.3.1.1.2 - initializes footer_message value
1.3.1.1.3 - initializes separator_line value
1.3.1.1.4 - initializes program_state_message value
1.3.1.2 - [destructor] ~viewer – destroy the the viewer object
1.3.1.3 - [string] getFooterMessage()
1.3.1.3.1 - return end_message private class variable
1.3.1.4 - [string] getHeaderMessage()
1.3.1.4.1 - return header_message private class variable
1.3.1.5 - [int] getLineCountWorld()
1.3.1.5.1 - return line_count_world
1.3.1.6 - [string] getNewSeperatorLine() - for making line breaks in the console.
1.3.1.6.1 - get separator_line string
1.3.1.7 - [string] getProgramStateMessage()
1.3.1.7.1 - return program_state_message private class variable
1.3.1.8 - [char] getSeperatorChar
1.3.1.8.1 - get the char used in the separator
1.3.1.9 - [int] getSeperatorLength()
1.3.1.9.1 - get separator length
1.3.1.10 - printSeperator()
1.3.1.10.1 - print separator string to console with new line
1.3.1.11 - setFooterMessage(string) – update bottom_text
1.3.1.11.1 - set class private variable footer_message to the string passed in if it's less than 100 characters, otherwise reject it and make the new footer a message about how the passed in one is too long.
1.3.1.12 - setHeaderMessage(string) – update header_message
1.3.1.12.1 - if class private variable header_message < 100 characters, set header message to the string passed in. Otherwise set a message as too how it's too long.
1.3.1.13 - setLineCountWorld(int)
1.3.1.13.1 - update line_count_range to the int passed in
1.3.1.14 - setNewSeperatorLine() - create new separator based off new settings
1.3.1.14.1 - string separator_line = separator_length * separator_char
1.3.1.15 - setProgramStateMessage(string) – update program_state_message
1.3.1.15.1 - update the string for program_state_message to value passed in
1.3.1.16 - setSeperatorChar(char)
1.3.1.16.1 - update separator_char to the value passed in
1.3.1.17 - setSeperatorLength(int)
1.3.1.17.1 - update the separator_length to the value passed in

2 - Simulator

2.1 - Summary

2.1.1 - Responsible for maintaining the changing game state and status.

2.2 - Properties

2.2.1 - private:

2.2.1.1 - [int] display_interval - the update interval for displaying the world
2.2.1.2 - [bool] paused_state – whether the world is supposed to be running or not
2.2.1.3 - [int] turn_num – the current update of the world
2.2.1.4 - [int] update_interval - the update interval for the world

2.3 - Methods

2.3.1 - public:

2.3.1.1 - [constructor] Simulator() - creates simulator object with default settings
2.3.1.2 - [constructor] Simulator(int updateInterval, int displayInterval) – creates the simulator with specific intervals for game updates and display updates.

```
2.3.1.2.1 -  Simulator::Simulator(int updateInterval, int displayInterval)
2.3.1.2.2 -  {
2.3.1.2.3 -         this->update_interval = updateInterval;
2.3.1.2.4 -         this->display_interval = displayInterval;
2.3.1.2.5 - }
```

2.3.1.3 - [destructor] ~Simulator() - de-allocates space for the simulator object
2.3.1.4 - delay(int)

```
2.3.1.4.1 -  void Simulator::delay(int seconds_delay) {
2.3.1.4.2 -         using namespace std::this_thread; // sleep_for, sleep_until
2.3.1.4.3 -         using namespace std::chrono;       // nanoseconds, system_clock,
                    seconds
2.3.1.4.4 -         sleep_for(nanoseconds(10));
2.3.1.4.5 -         sleep_until(system_clock::now() + seconds(seconds_delay));
2.3.1.4.6 -  }
```

2.3.1.5 - [int] getGisplayInterval() - return display_update_interval
2.3.1.6 - bool] getPausedState() - return paused
2.3.1.7 - [int] getTurnNum() - return turn_num
2.3.1.8 - [int] getUpdateInterval() - return update_interval
2.3.1.9 - Pause() - set paused to True

```
2.3.1.9.1 -  void Simulator::pause()
2.3.1.9.2 -  {
2.3.1.9.3 -         this->paused_state = true;
2.3.1.9.4 -  }
```

2.3.1.10 - Run() - set paused to False

```
2.3.1.10.1 -  void Simulator::run()
2.3.1.10.2 -  {
2.3.1.10.3 -        this->paused_state = false;
2.3.1.10.4 -  }
```

2.3.1.11 - setDisplayInterval(int) – change the rate at which the World is updating on the screen. Adjust display_interval to the value provided.

```
2.3.1.11.1 -  void Simulator::setDisplayInterval(int newInterval)
2.3.1.11.2 -  {
2.3.1.11.3 -        this->display_interval = newInterval;
2.3.1.11.4 -  }
```

2.3.1.12 - setUpdateInterval(int) – change the rate at which the World is updating. Adjust update_interval to the value provided.

```
2.3.1.12.1 -  void Simulator::setDisplayInterval(int newInterval)
2.3.1.12.2 -  {
2.3.1.12.3 -        this->display_interval = newInterval;
```

2.3.1.13 - Turninc() - increments the turn count.

```
2.3.1.13.1 -  void Simulator::turnInc() {
2.3.1.13.2 -        this->turn_num += 1;
2.3.1.13.3 -  }
```

3 - WorldBoard

3.1 - Summary

3.1.1 - The world board is responsible for holding all the entities of the DOCO simulation. Creating it will create the other objects.

3.2 - Properties

3.2.1 - private:

3.2.1.1 - [vector] food_positions – store food positions
3.2.1.2 - [vector] obstacle_positions – store obstacle positions
3.2.1.3 - [int] height - store board height
3.2.1.4 - [int] width – store board width
3.2.2 - public:
3.2.2.1 - [vector<DOCO>] doco_vect – will hold a vector of all the current DOCO's on the board. These will be able to be iterated over and removed as part of the vector class functionality.
3.2.2.2 - [DataParser] myParser – the DataParser obejct for the class. mytParser = DataParser(char *filename). This filename will be DOCOData02.xml
3.2.2.3 - [Simulator] mySim – containers the DOCOSim Object
3.2.2.4 - [CellGrid] worldCellGrid – will hold the CellGrid Object which contains all the Cells and GridSize

3.3 - Methods

3.3.1 - private:

3.3.1.1 - generateFoodLocations(int height, int width, int count) – creates the food spawn locations and adds to a vector to be placed on the board.

```
3.3.1.1.1 - void WorldBoard::generateFoodLocations(int w, int h, int foodCount)
3.3.1.1.2 - {
3.3.1.1.3 -         // generate spawn locations
3.3.1.1.4 -         int x_pos = 0;
3.3.1.1.5 -         int y_pos = 0;
3.3.1.1.6 -         for (int i = 0; i < foodCount; ++i)
3.3.1.1.7 -         {
3.3.1.1.8 -               x_pos = uniRand->generateRandomNum(0, w-1);
3.3.1.1.9 -               y_pos = uniRand->generateRandomNum(0, h-1);
3.3.1.1.10 -              while ((this->worldCellGrid->cell_matrix[y_pos]
                  [x_pos].getFoodCount() > 3) // Food count > 3, generate new x and y
                  position.
3.3.1.1.11 -                        || (this->worldCellGrid->cell_matrix[y_pos]
                  [x_pos].getOccupied()) ) // Don't spawn food in occupied cells
3.3.1.1.12 -              {
3.3.1.1.13 -                    x_pos = uniRand->generateRandomNum(0, w-1);
3.3.1.1.14 -                    y_pos = uniRand->generateRandomNum(0, h-1);
3.3.1.1.15 -              }
3.3.1.1.16 -              auto location = std::make_pair(x_pos, y_pos);
3.3.1.1.17 -              this->food_positions.push_back(location);
3.3.1.1.18 -        }
3.3.1.1.19 -        this->food_positions.shrink_to_fit();
3.3.1.1.20 - }
```

3.3.1.2 - readFile() - reads the file in
3.3.1.3 - spawninitialDocos()
3.3.2 - public:
3.3.2.1 - [contructor] WorldBoard() - build WorldBoard object
3.3.2.2 - [constructor] WorldBoard(filename) – build WolrdBoard object given filename

```
3.3.2.3 -  [deconstructor] ~WorldBoard()
3.3.2.3.1 -  PrintWorld() - this print the Cell Board and use the CellGrid built in
                  function to do it.
3.3.2.3.1.1 -  void WorldBoard::printWorld()
3.3.2.3.1.2 -  {
3.3.2.3.1.3 -          this->worldCellGrid->printCharMatrix();
3.3.2.3.1.4 -          std::cout << "DOCOs on Board:          " << this->worldCellGrid-
                  >getDocoCharCount() << "\n";
3.3.2.3.1.5 -          std::cout << "DOCOs Actually Present:  " << this->doco_vect.size()
                  << "\n";
3.3.2.3.1.6 -          std::cout << "Unique Food Spots:       " << this->worldCellGrid-
                  >getFoodCharCount() << "\n";
3.3.2.3.1.7 -          std::cout << "Obstacles on Board:      " << this->worldCellGrid-
                  >getObstacleCount() << "\n";
3.3.2.3.1.8 -  }
3.3.2.3.2 -  setCellWithNewFood()
3.3.2.3.2.1 -  void WorldBoard::setCellWithNewFood(int x, int y)
3.3.2.3.2.2 -  {
3.3.2.3.2.3 -          this->worldCellGrid->cell_matrix[y][x].addFood(1);
3.3.2.3.2.4 -          this->worldCellGrid->cell_matrix[y][x].setFoodPresent();
3.3.2.3.2.5 -          this->worldCellGrid->cell_matrix[y][x].setSymbol();
3.3.2.3.2.6 -  }
3.3.2.3.3 -  updateCellsWithNewFood()  - updates cell to have no food inside it
3.3.2.3.3.1 -  void WorldBoard::updateCellsWithNewFood()
3.3.2.3.3.2 -  {
3.3.2.3.3.3 -          for (auto food : this->food_positions) {
3.3.2.3.3.4 -             this->setCellWithNewFood(food.first, food.second);
3.3.2.3.3.5 -          }
3.3.2.3.3.6 -          food_positions.clear(); // all new food positions have been
                  processed.
3.3.2.3.3.7 -  }
3.3.2.3.4 -  updateCellWithADoco()  - adds occupied = true to the cell
3.3.2.3.4.1 -  int WorldBoard::updateCellWithADoco(int x, int y)
3.3.2.3.4.2 -  {
3.3.2.3.4.3 -          this->worldCellGrid->cell_matrix[y][x].setOccupied(true);
3.3.2.3.4.4 -          int food_count = this->setCellWithNoFood(x, y);
3.3.2.3.4.5 -          this->worldCellGrid->cell_matrix[y][x].setFoodPresent();
3.3.2.3.4.6 -          this->worldCellGrid->cell_matrix[y][x].setSymbol();
3.3.2.3.4.7 -          return food_count;
3.3.2.3.4.8 -  }
3.3.2.3.5 -  updateCellWithNewFood() - adds food to that cell
3.3.2.3.6 -  setCellWithNoFood() - remove food from a cell
3.3.2.3.6.1 -  int WorldBoard::setCellWithNoFood(int x, int y)
3.3.2.3.6.2 -  {
3.3.2.3.6.3 -          int count = this->worldCellGrid->cell_matrix[y][x].getFoodCount();
3.3.2.3.6.4 -          this->worldCellGrid->cell_matrix[y][x].removeAllFood();
3.3.2.3.6.5 -          return count;
3.3.2.3.6.6 -  }
3.3.2.3.7 -  [Doco] splitDoco() - modify a doco to half energy, then copy, set copy to
                  opposite direction
3.3.2.3.8 -  UpdateDocos() - update all the doco's next move and stats one at a time.
3.3.2.3.8.1 -  void WorldBoard::updateDocos(void)
3.3.2.3.8.2 -  {
3.3.2.3.8.3 -          // --- Remove dead DOCOs from the list
3.3.2.3.8.4 -          auto size = this->doco_vect.size();
```

```
3.3.2.3.8.5 -         // TODO: 1 doco was killed at start???
3.3.2.3.8.6 -         while (size > 0)  // Go through doco_vect, delete item if it's dead
3.3.2.3.8.7 -         {
3.3.2.3.8.8 -            if (!this->doco_vect[size-1].getAlive()) {
3.3.2.3.8.9 -                  this->worldCellGrid->cell_matrix[this->doco_vect[size-
                      1].getYPos()][this->doco_vect[size-
                      1].getXPos()].setOccupied(false);
3.3.2.3.8.10 -                 this->worldCellGrid->cell_matrix[this->doco_vect[size-
                      1].getYPos()][this->doco_vect[size-1].getXPos()].setSymbol();
3.3.2.3.8.11 -                 auto pos = this->doco_vect.begin() + size - 1;
3.3.2.3.8.12 -                 this->doco_vect.erase(pos);
3.3.2.3.8.13 -                 size -= 1; // remove an extra item as the doco has been
                      erased.
3.3.2.3.8.14 -            }
3.3.2.3.8.15 -            size -= 1;
3.3.2.3.8.16 -        }
3.3.2.3.8.17 -
3.3.2.3.8.18 -        // --- Split Docos if (energy_level > 750)
3.3.2.3.8.19 -        int i = 0;
3.3.2.3.8.20 -        int currentEnergy = 0;
3.3.2.3.8.21 -        int newEnergy = 0;
3.3.2.3.8.22 -        // TODO: A doco COPY with inccorect ptr_moveStra, alive, and energy
                      level is created.
3.3.2.3.8.23 -        for (i = 0; i < int(this->doco_vect.size()); ++i)  // Go through
                      doco_vect, split them if high energy
3.3.2.3.8.24 -        {
3.3.2.3.8.25 -            // --- When engergy too high, split doco into 2 on cell and head
                      in opposite dirs.
3.3.2.3.8.26 -            if (currentEnergy = this->doco_vect[i].getEnergy() > 750) {
3.3.2.3.8.27 -                  // --- Split original's energy
3.3.2.3.8.28 -                  newEnergy = int(currentEnergy / 2);
3.3.2.3.8.29 -                  this->doco_vect[i].setEnergy(newEnergy);
3.3.2.3.8.30 -
3.3.2.3.8.31 -                  // --- Save direction of original.
3.3.2.3.8.32 -                  auto oldDir = this->doco_vect[i].getDirection();
3.3.2.3.8.33 -
3.3.2.3.8.34 -                  // --- Copy Constructor
3.3.2.3.8.35 -                  // Doco docoCopy = this->doco_vect[i];
3.3.2.3.8.36 -                  auto* docoCopy = new Doco(this->doco_vect[i]);
3.3.2.3.8.37 -
3.3.2.3.8.38 -                  // --- Reverse direction of the new Doco
3.3.2.3.8.39 -                  auto oppositeDir = dirs.getOppositeDirectionPair(oldDir);
3.3.2.3.8.40 -                  docoCopy->setDirection(oppositeDir.first);
3.3.2.3.8.41 -
3.3.2.3.8.42 -                  // --- Push back the copy
3.3.2.3.8.43 -                  doco_vect.push_back(*docoCopy);
3.3.2.3.8.44 -             }
3.3.2.3.8.45 -        }
3.3.2.3.8.46 -
3.3.2.3.8.47 -        // --- Vars for the upcoming DOCO update loop
3.3.2.3.8.48 -        int food_eaten;
3.3.2.3.8.49 -        int x = 0;
3.3.2.3.8.50 -        int y = 0;
3.3.2.3.8.51 -
```

```
3.3.2.3.8.52 -         // --- Update the current cells with DOCO actions that were decided
                       the previous turn / round.
3.3.2.3.8.53 -         for (i = 0; i < int(this->doco_vect.size()); i++)
3.3.2.3.8.54 -         {
3.3.2.3.8.55 -            // --- Current X any Y Pos
3.3.2.3.8.56 -            x = this->doco_vect[i].getXPos();
3.3.2.3.8.57 -            y = this->doco_vect[i].getYPos();
3.3.2.3.8.58 -
3.3.2.3.8.59 -            // ------------------
3.3.2.3.8.60 -            // --- Tell the DOCO what it's surrounding are, so it knows its
                       options.
3.3.2.3.8.61 -            this->doco_vect[i].adjoined_cells = this->worldCellGrid-
                       >findAdjoinedCells(x, y);  //Non Border Cells
3.3.2.3.8.62 -            this->doco_vect[i].adjoined_occupied_cells = this-
                       >worldCellGrid->findAdjoinedOccupiedCells();
3.3.2.3.8.63 -            this->doco_vect[i].adjoined_obstacle_cells = this-
                       >worldCellGrid->findAdjoinedObstacleCells();
3.3.2.3.8.64 -            this->doco_vect[i].adjoined_open_cells = this->worldCellGrid-
                       >findAdjoinedOpenCells();
3.3.2.3.8.65 -            this->doco_vect[i].adjoined_food_cells = this->worldCellGrid-
                       >findAdjoinedCellsFood();
3.3.2.3.8.66 -            this->doco_vect[i].adjoined_open_cells_with_food = this-
                       >worldCellGrid->findAdjoinedOpenCellsWithFood();
3.3.2.3.8.67 -
3.3.2.3.8.68 -            // --- Find NEW Cell to move to from available options. Chooses
                       desirable X_Y position and assingns the DOCO with that new X_Y
                       position
3.3.2.3.8.69 -            auto moved_to = this->doco_vect[i].move(this->width, this-
                       >height); // all doco's in list make new move decision one at a
                       time
3.3.2.3.8.70 -            // ------------------
3.3.2.3.8.71 -
3.3.2.3.8.72 -            // --- Update it's previous cell with data on being non-occupied
                       now.
3.3.2.3.8.73 -            this->worldCellGrid->cell_matrix[y][x].setOccupied(false);
3.3.2.3.8.74 -            this->worldCellGrid->cell_matrix[y][x].setFoodPresent();
3.3.2.3.8.75 -            this->worldCellGrid->cell_matrix[y][x].setSymbol();
3.3.2.3.8.76 -
3.3.2.3.8.77 -            // --- Update cell properties of new cell DOCO is at
3.3.2.3.8.78 -            // --- Eat Food for Current Cell. Gain Energy.
3.3.2.3.8.79 -            food_eaten = this->updateCellWithADoco(this-
                       >doco_vect[i].getXPos(), this->doco_vect[i].getYPos());
3.3.2.3.8.80 -            this->doco_vect[i].eat(food_eaten, "default");
3.3.2.3.8.81 -
3.3.2.3.8.82 -            this->worldCellGrid->cell_matrix[this->doco_vect[i].getYPos()]
                       [this->doco_vect[i].getXPos()].setOccupied(true); // set the cell
                       as populated
3.3.2.3.8.83 -            this->worldCellGrid->cell_matrix[this->doco_vect[i].getYPos()]
                       [this->doco_vect[i].getXPos()].setFoodPresent();
3.3.2.3.8.84 -            this->worldCellGrid->cell_matrix[this->doco_vect[i].getYPos()]
                       [this->doco_vect[i].getXPos()].setSymbol();
3.3.2.3.8.85 -         }
3.3.2.3.8.86 -  }
3.3.2.3.9 -  UpdateWorldState() - updates the entire worldBoard
3.3.2.3.9.1 -  // --- Perform a SINGLE update of the WorldBoard
```

```
3.3.2.3.9.2 -  void WorldBoard::updateWorldState()
3.3.2.3.9.3 -  {
3.3.2.3.9.4 -          // --- Reinitialize the Char Matrix each turn.
3.3.2.3.9.5 -          this->worldCellGrid->initCharMatrix(this->width, this->height);
3.3.2.3.9.6 -          // --- Find new positions to place food on the board that are not
                       already at max capacity
3.3.2.3.9.7 -          this->generateFoodLocations(this->width, this->height, uniRand-
                       >generateRandomNum(1, 10));
3.3.2.3.9.8 -          // --- Update the choosen new food cells with the food and change
                       symbol for that cell position.
3.3.2.3.9.9 -          this->updateCellsWithNewFood();
3.3.2.3.9.10 -         // --- Update every DOCO on the board.
3.3.2.3.9.11 -         this->updateDocos();
3.3.2.3.9.12 -         // --- Run through the character status of each cell and set the
                       matrix for it.
3.3.2.3.9.13 -         this->worldCellGrid->setCharMatrix();
3.3.2.3.9.14 -  }
```

4 - DOCO

4.1 - Summary

4.1.1 - A DOCO is an organism object that moves around on the CellGrid based off it's own desires and what is immediately around it.

4.2 - Properties

4.2.1 - private:

```
4.2.1.1 -  [docoMoveStragey] moveStrategy
4.2.1.2 -  [bool] alive – whether or not the DOCO is alive or dead, if it's dead it should
           be removed or become invisible on the screen.
4.2.1.3 -  [int] energy_level – the amount of energy the DOCO has. It will be initialized
           to 500 by default.
4.2.1.4 -  [pair int] position
4.2.1.5 -  [string] direction – A direction that the DOCO is currently heading. It will be
           be one of the following strings "N", "NE", "E", "SE", "S", "SW", "W",
           "NW".
4.2.1.6 -  [pair int] position
4.2.2 -  public:
4.2.2.1 -  [matrix] adjoined_cells – this will contain the matrix of adjoining cells to a
           DOCO. Adjoining means only the cells are touching, diagonal included.
4.2.2.2 -  [matrix] adjoined_obstacle_cells – this will contain the matrix of adjoining
           cells that are obstacles.
4.2.2.3 -  [matrix] adjoined_occupied_cells – this will contain the matrix of adjoining
           cells that are occupied.
4.2.2.4 -  [matrix] adjoined_food_cells – this will contain the matrix of adjoining cells
           that contain food. This is why the Cell object has a food_present boolean
           property.
4.2.2.5 -  [matrix] move_options – this will contain the matrix of movement options that
           are available to the DOCO based on it's movement preferences and
           requirements.
```

4.3 - Methods

4.3.1 - public:

```
4.3.1.1 -  [constructors] Doco(int x, int y, std::string& start_dir, std::string&
           strategy, DocoMoveStrategy* strat); // starting position, x, y, direction
```

```
4.3.1.1.1 -  Doco::Doco(int x, int y, std::string start_dir, std::string strategy) :
                AbstractDoco()
4.3.1.1.2 -  {
4.3.1.1.3 -         this->position.first = x;
4.3.1.1.4 -         this->position.second = y;
4.3.1.1.5 -         this->setDirection(start_dir);
4.3.1.1.6 -         this->setStrategy(strategy);
4.3.1.1.7 -  }
4.3.1.1.8 -  [copy constructor] Doco(const &originalObj)
4.3.1.1.9 -  Doco::Doco(const Doco &origObj) {
4.3.1.1.10 -        // current doco position = passed in doco position
4.3.1.1.11 -        this->position.first = origObj.position.first;              // copy x
4.3.1.1.12 -        this->position.second = origObj.position.second;     // copy y
4.3.1.1.13 -        this->setDirection(origObj.direction.first);         // copy direction
4.3.1.1.14 -        this->setStrategy(origObj.strategy);                       // copy
                strategy string
4.3.1.1.15 -        auto strat = origObj.ptr_moveStrategy;                     // copy old
                move strat
4.3.1.1.16 -        this->setPtrMoveStrategy(strat);                           // set new
                doco to same move strat
4.3.1.1.17 -  }
4.3.1.2 -  [destructor] ~DOCO() - deletes the DOCO
4.3.1.3 -  addEnergy(int) – add the specified amount of energy to the DOCO's energy_level
4.3.1.3.1 -  void Doco::addEnergy(int added_energy) {
4.3.1.3.2 -         this->energy_level += added_energy;
4.3.1.3.3 -  }
4.3.1.4 -  eat(x_pos, y_pos) – the DOCO regenerates 50 energy for each pellet eaten, and
              it eats all the pellets at this location. This call the
              CellGrid.Matrix.SpecificCell.setFoodPresent(bool) and setSymbol(char),
              removeAllFood() commands for the cell being eaten off of.
4.3.1.4.1 -  void Doco::eat(int amount_eaten, const std::string& type="default") { // type
                is selected without needing to specify
4.3.1.4.2 -         if (type == "default")
4.3.1.4.3 -         {
4.3.1.4.4 -               int food_e_value = 50;                         // food energy
                value -> could make this a class if needed later.
4.3.1.4.5 -               int total_replished_e = food_e_value * amount_eaten;
4.3.1.4.6 -               this->addEnergy(total_replished_e);
4.3.1.4.7 -         }
4.3.1.4.8 -  }
4.3.1.5 -  [bool] getAlive() - returns whether the DOCO is alive or dead.
4.3.1.6 -  [string pair int] getDirection() - returns the current direction of the DOCO
4.3.1.7 -  [string] getDirectionString() - returns the current direction of the DOCO
4.3.1.8 -  [int] getEnergy() - returns the energy_level of the DOCO
4.3.1.9 -  [pair int] getPosPair
4.3.1.10 -  [int] getXPos()
4.3.1.11 -  [int] getYPos()
4.3.1.12 -  [pair<int, int>] move(int, int) – doco chooses new move position and goes to
              it.
4.3.1.12.1 -  std::pair<int, int> Doco::move(int world_w, int world_h) // returns the pair
                that moved too.
4.3.1.12.2 -  {
4.3.1.12.3 -        // --- Setup move pair that is choosen
4.3.1.12.4 -        int x = this->getXPos();
4.3.1.12.5 -        int y = this->getYPos();
```

```
4.3.1.12.6 -        std::pair<int, int> moving_here;       // new (x, y) position for
                    DOCO.
4.3.1.12.7 -        std::pair<int, int> temp_next_pos;
4.3.1.12.8 -        std::pair<int, int> temp_next_valid_pos;
4.3.1.12.9 -        std::pair<int, int> option;
4.3.1.12.10 -       std::vector<std::pair<int, int> >::iterator it;
4.3.1.12.11 -       std::vector<std::pair<int, int> >::iterator identifying_value;
4.3.1.12.12 -
4.3.1.12.13 -       bool verified = false;
4.3.1.12.14 -       this->move_options.clear(); // Clear move options from previous turn to
                    generate new ones now that new data is in.
4.3.1.12.15 -       for (auto pair : this->adjoined_cells) {
4.3.1.12.16 -             this->move_options.push_back(pair);
4.3.1.12.17 -       }
4.3.1.12.18 -
4.3.1.12.19 -       //
                    ------------------------------------------------------------------------
                    -----
4.3.1.12.20 -       // | 1 | CONTINUE IN SAME DIRECTION IF POSSIBLE
4.3.1.12.21 -       auto temp_next_dir = this->direction;
4.3.1.12.22 -       temp_next_pos = std::make_pair(x + temp_next_dir.second.first, y +
                    temp_next_dir.second.second);
4.3.1.12.23 -       identifying_value = std::find(this->adjoined_open_cells.begin(),
                    adjoined_open_cells.end(), temp_next_pos); // returns (bool, pos in
                    searched vect)
4.3.1.12.24 -       std::pair<bool, int> searchResult = findItemInVect<std::pair<int, int>
                    >(this->adjoined_open_cells, temp_next_pos);
4.3.1.12.25 -       bool verified_bounds = false;
4.3.1.12.26 -       while (!verified_bounds) {
4.3.1.12.27 -             /*
4.3.1.12.28 -             - Behavior Patterns
4.3.1.12.29 -
4.3.1.12.30 -                   o     Behavior pattern 1 will cause the DOCO to move only
                    in a horizontal direction.If an edge of the
4.3.1.12.31 -                   world is encountered the DOCO will randomly elect to move
                    up or down a row and reverse its direction
4.3.1.12.32 -                   of movement.
4.3.1.12.33 -
4.3.1.12.34 -                   o     Behavior pattern 2 will cause the DOCO to move only
                    in a vertical direction.If an edge of the world
4.3.1.12.35 -                   is encountered the DOCO will randomly elect to move left
                    or right a column and reverse its direction
4.3.1.12.36 -                   of movement.
4.3.1.12.37 -
4.3.1.12.38 -                   o     Behavior pattern 3 will cause the DOCO to move only
                    in a diagonal direction.If an edge of the world
4.3.1.12.39 -                   is encountered the DOCO will randomly elect to move left,
                    right, up, or down and either reverse its
4.3.1.12.40 -                   direction of movement or move in the other diagonal
                    direction.
4.3.1.12.41 -
4.3.1.12.42 -             */
4.3.1.12.43 -
4.3.1.12.44 -             if (searchResult.first) // if there is an open position in that
                    direction
```

```
4.3.1.12.45 -            {
4.3.1.12.46 -                  temp_next_valid_pos = this->adjoined_open_cells.at(searchResult.second);       // --- Choose open position
4.3.1.12.47 -                  // | 5 | GO FOR FOOD IF AVAILABLE
4.3.1.12.48 -                  //  The cell containing food does not have to be along its' path defined by its movement behavior.
4.3.1.12.49 -                  if (this->adjoined_open_cells_with_food.size() > 0)
4.3.1.12.50 -                  {
4.3.1.12.51 -                        // Create move pair for jumping once in the direction of food and keeping heading
4.3.1.12.52 -                        temp_next_valid_pos = this->adjoined_open_cells_with_food.at(randDocoObj->generateRandomNum(0, int(this->adjoined_open_cells_with_food.size()) - 1)); // choose random food poition
4.3.1.12.53 -                  }
4.3.1.12.54 -            }
4.3.1.12.55 -            else // if can't continue in straight direction or jump to food, check if it's an obstacle in the way, check if it's a wall
4.3.1.12.56 -            {
4.3.1.12.57 -                  // | 2 | WHEN A WALL IS HIT, USE AVOIDANCE STRATEGY
4.3.1.12.58 -                  if (this->adjoined_cells.size() < 8)
4.3.1.12.59 -                  {
4.3.1.12.60 -                        // --- Check if wall in current direction path, if it is, pick a direction from movement stategy
4.3.1.12.61 -                        // Check for direction in adjoined_cells, if not in there then it hit a wall..
4.3.1.12.62 -                        //searchResult = std::find(this->adjoined_cells.begin(), adjoined_cells.end(), temp_next_pos); // returns (bool, pos in searched vect)
4.3.1.12.63 -                        searchResult = findItemInVect<std::pair<int, int> >(this->adjoined_cells, temp_next_pos); // searchResult = wall pair
4.3.1.12.64 -                        if (!searchResult.first) // If there is not an position for this, it's a wall
4.3.1.12.65 -                        {
4.3.1.12.66 -                              // --- Move to a random avoidance strategy that is open
4.3.1.12.67 -                              auto avoidance_pair_vect = this->ptr_moveStrategy->avoidanceStrategy();
4.3.1.12.68 -
4.3.1.12.69 -                              auto rand_avoid_pair = avoidance_pair_vect.at(randDocoObj->generateRandomNum(0, int(avoidance_pair_vect.size()) - 1));
4.3.1.12.70 -                              temp_next_pos = std::make_pair(x + rand_avoid_pair.first, y + rand_avoid_pair.second);
4.3.1.12.71 -                              auto avoid_pair_available = findItemInVect<std::pair<int, int> >(this->adjoined_open_cells, temp_next_pos); // returns (bool, pos in searched vect)
4.3.1.12.72 -                              int attempts = 12;
4.3.1.12.73 -                              while (!avoid_pair_available.first && attempts < 15) {
4.3.1.12.74 -                                    auto rand_avoid_pair = avoidance_pair_vect.at(randDocoObj->generateRandomNum(0, int(avoidance_pair_vect.size()) - 1));
```

```
4.3.1.12.75 -                                       temp_next_pos = std::make_pair(x +
            rand_avoid_pair.first, y + rand_avoid_pair.second);
4.3.1.12.76 -                                       auto avoid_pair_available =
            findItemInVect<std::pair<int, int> >(this->adjoined_open_cells,
            temp_next_pos); // returns (bool, pos in searched vect)
4.3.1.12.77 -                                       attempts++;
4.3.1.12.78 -                                 }
4.3.1.12.79 -
4.3.1.12.80 -                                 if (avoid_pair_available.first) // if avoid
            pair found in open cells
4.3.1.12.81 -                                 {
4.3.1.12.82 -                                       temp_next_valid_pos = temp_next_pos;
            // valid avoidence pair found.
4.3.1.12.83 -
4.3.1.12.84 -                                       // --- Set reverse direction and allow
            the special case for diagonal.
4.3.1.12.85 -                                       auto temp_pair_vect = this-
            >ptr_moveStrategy->moveStrategy(); // Check if next position for
            direction is in the path
4.3.1.12.86 -                                       auto viablePos =
            findItemInVect<std::pair<int, int> >(temp_pair_vect, temp_next_pos); //
            returns (bool, pos in searched vect)
4.3.1.12.87 -
4.3.1.12.88 -                                       if (viablePos.first) // If next
            position is available for it's current path...
4.3.1.12.89 -                                       {
4.3.1.12.90 -                                             auto pos =
            temp_pair_vect.begin() + identifying_value->second;
4.3.1.12.91 -                                             temp_pair_vect.erase(pos);
4.3.1.12.92 -                                             auto otherPairPos =
            temp_pair_vect.at(randDocoObj->generateRandomNum(0,
            int(temp_pair_vect.size()) - 1));
4.3.1.12.93 -                                             // get direction from point
4.3.1.12.94 -                                             std::string dir = directions-
            >getDirForPair(this->direction.second, otherPairPos);
4.3.1.12.95 -                                             this->setDirection(dir);
4.3.1.12.96 -                                       }
4.3.1.12.97 -                                 }
4.3.1.12.98 -                           }
4.3.1.12.99 -                     }
4.3.1.12.100 -                    // | 3 | WHEN OBSTACLE IS IN PATH, PICK RANDOM DIRECTION
            in an OPEN PATH.
4.3.1.12.101 -                    else if (this->adjoined_obstacle_cells.size() > 0)
4.3.1.12.102 -                    {
4.3.1.12.103 -                          // --- Check if obstacle in current direction path,
            if it is, pick a random open direction
4.3.1.12.104 -                          auto temp_next_dir = this->direction;
4.3.1.12.105 -                          temp_next_pos = std::make_pair(x +
            temp_next_dir.second.first, y + temp_next_dir.second.second);
4.3.1.12.106 -                          std::pair<bool, int> searchResult =
            findItemInVect<std::pair<int, int> >(this->adjoined_obstacle_cells,
            temp_next_pos); // returns (bool, pos in searched vect)
4.3.1.12.107 -                          if (searchResult.first) // if there is an obstacle
4.3.1.12.108 -                          {
4.3.1.12.109 -                                // --- Choose random open position
```

```
4.3.1.12.110 -                                 temp_next_valid_pos = this-
>adjoined_open_cells.at(randDocoObj->generateRandomNum(0, int(this-
>adjoined_open_cells.size()) - 1));
4.3.1.12.111 -                         }
4.3.1.12.112 -                 }
4.3.1.12.113 -                 // | 4 | WHEN DIRECTION HITS ANOTHER DOCO, REVERSE, CHECK
IF THAT'S A DOCO, IF IT IS, USE AVOIDANCE STRATEGY W/ RANDOM CHOICE
4.3.1.12.114 -                 else if (this->adjoined_occupied_cells.size() > 0) {
4.3.1.12.115 -                         this->ptr_moveStrategy->moveStrategy();
4.3.1.12.116 -
4.3.1.12.117 -                         std::pair<bool, int> other_doco_pair =
findItemInVect<std::pair<int, int> >(this->adjoined_occupied_cells,
temp_next_pos); // returns (bool, pos in searched vect)
4.3.1.12.118 -                         if (other_doco_pair.first) // if there is doco in
that direction
4.3.1.12.119 -                         {
4.3.1.12.120 -                                 this->direction = directions-
>getOppositeDirectionPair(this->direction); // Reverse doco direction,
but...for the move
4.3.1.12.121 -
4.3.1.12.122 -                                 // --- Check if doco in reverse direction
4.3.1.12.123 -                                 auto temp_next_dir = this->direction;
4.3.1.12.124 -                                 temp_next_pos = std::make_pair(x +
temp_next_dir.second.first, y + temp_next_dir.second.second);
4.3.1.12.125 -                                 std::pair<bool, int> other_doco_pair2 =
findItemInVect<std::pair<int, int> >(this->adjoined_occupied_cells,
temp_next_pos); // returns (bool, pos in searched vect)
4.3.1.12.126 -                                 if (other_doco_pair2.first) // if there is
doco in that direction
4.3.1.12.127 -                                 {
4.3.1.12.128 -                                         // --- doco in reverse direction, so
choose random avoidance stragey
4.3.1.12.129 -                                         auto temp_pair_vect = this-
>ptr_moveStrategy->avoidanceStrategy();
4.3.1.12.130 -                                         temp_next_pos =
temp_pair_vect.at(randDocoObj->generateRandomNum(0,
int(temp_pair_vect.size()) - 1)); // choose random position from
avoidance strategy.
4.3.1.12.131 -                                         std::pair<bool, int> validCheck =
findItemInVect<std::pair<int, int> >(this->adjoined_open_cells,
temp_next_pos); // returns (bool, pos in searched vect)


                                                        // --- Choose open position
4.3.1.12.132 -                                         if (validCheck.first) // if found in
open cells
4.3.1.12.133 -                                         {
4.3.1.12.134 -                                                 temp_next_valid_pos = this-
>adjoined_open_cells.at(identifying_value->second);
4.3.1.12.135 -                                         }
4.3.1.12.136 -                                 }
4.3.1.12.137 -                                 else {
4.3.1.12.138 -                                         // --- doco not in revese direction
keep it
```

```
4.3.1.12.139 -                                      // --- make sure it is in
            ajoined_open,don't want to turn into a wall or obstable
4.3.1.12.140 -                                      std::pair<bool, int> validCheck =
            findItemInVect<std::pair<int, int> >(this->adjoined_open_cells,
            temp_next_pos); // returns (bool, pos in searched vect)


                                                     // --- Choose open position
4.3.1.12.141 -                                     if (validCheck.first) // if found in
            open cells
4.3.1.12.142 -                                     {
4.3.1.12.143 -                                           temp_next_valid_pos = this-
            >adjoined_open_cells.at(identifying_value->second);
4.3.1.12.144 -                                     }
4.3.1.12.145 -                                     else temp_next_valid_pos = this-
            >position;  // move to same spot?
4.3.1.12.146 -                                }
4.3.1.12.147 -                           }
4.3.1.12.148 -                     }
4.3.1.12.149 -               }
4.3.1.12.150 -
4.3.1.12.151 -               // continue looping until item within the bounds of the map
4.3.1.12.152 -               // --- Find if that pair is in, and it's position in the
            adjoined_cells vector of pairs
4.3.1.12.153 -               std::pair<bool, int> result = findItemInVect<std::pair<int, int>
            >(this->adjoined_cells, temp_next_valid_pos);
4.3.1.12.154 -               if (result.first && (this-
            >adjoined_cells.at(result.second).first >= 0) && (this-
            >adjoined_cells.at(result.second).second >= 0) &&
4.3.1.12.155 -                     (this->adjoined_cells.at(result.second).first < world_w)
            && (this->adjoined_cells.at(result.second).second < world_h))
4.3.1.12.156 -               {
4.3.1.12.157 -                     // --- The next position for the same direction is valid.
4.3.1.12.158 -                     temp_next_valid_pos = temp_next_pos;
4.3.1.12.159 -                     verified_bounds = true;
4.3.1.12.160 -               }
4.3.1.12.161 -          }
4.3.1.12.162 -
4.3.1.12.163 -          moving_here = temp_next_valid_pos;
4.3.1.12.164 -
4.3.1.12.165 -          // --- Update doco position and energy if it's still alive.
4.3.1.12.166 -          if (this->getAlive()) // if alive, update position
4.3.1.12.167 -          {
4.3.1.12.168 -               docoMoveToPos(moving_here);
4.3.1.12.169 -          }
4.3.1.12.170 -          return moving_here;
4.3.1.12.171 -     }
4.3.1.13 -     setAlive(bool) - updates the alive status of the DOCO.
4.3.1.13.1 -     void Doco::setAlive(void)
4.3.1.13.2 -     {
4.3.1.13.3 -          if (this->energy_level <= 0) this->alive = false;
4.3.1.13.4 -          else this->alive = true;
4.3.1.13.5 -     }
```

4.3.1.14 - setDirection(string) – sets the direction of the DOCO, this will be one of the following strings “N”, “NE”, “E”, “SE”, “S”, “SW”, “W”, “NW”. Upon initialization this will be random or taken from the read in file.

```
4.3.1.14.1 -  void Doco::setDirection(std::string new_direction)
4.3.1.14.2 -  {
4.3.1.14.3 -        this->direction = directions->getPairComboForString(new_direction);
4.3.1.14.4 -  }
```

4.3.1.15 - setEnergy(int) – set the energy_level of the DOCO to a specified amount.

```
4.3.1.15.1 -  void Doco::setEnergy(int new_e_level)
4.3.1.15.2 -  {
4.3.1.15.3 -        if (new_e_level >= 0) {
4.3.1.15.4 -              this->energy_level = new_e_level;
4.3.1.15.5 -        }
4.3.1.15.6 -        else new_e_level = 0;
4.3.1.15.7 -        this->alive = false;
4.3.1.15.8 -  }
```

4.3.1.16 - setPos(x_pos, y_pos) – updates the x and y position of the DOCO.

```
4.3.1.16.1 -  void Doco::setPos(int x, int y)
4.3.1.16.2 -  {
4.3.1.16.3 -        this->position.first = x;
4.3.1.16.4 -        this->position.second = y;
4.3.1.16.5 -  }
```

4.3.1.17 - setPos(pair int) – sets Doco position

```
4.3.1.17.1 -  void Doco::setPos(std::pair<int, int> cordinates)
4.3.1.17.2 -  {
4.3.1.17.3 -        this->position = cordinates;
4.3.1.17.4 -  }
```

5 - Cell

5.1 - Summary

5.1.1 - Cells are rigid objects on the board, they don’t move or change positions. There will be many of these mapped onto a Cell Grid.

5.2 - Properties

5.2.1 - private:

5.2.1.1 - [int] x_pos – this will store the x position of a cell
5.2.1.2 - [int] y_pos – this will store the y position of a cell
5.2.1.3 - [bool] obstacle – this will store whether a cell is currently occupied by an obstacle
5.2.1.4 - [bool] occupied – this will store whether a cell is currently occupied by a DOCO or other organism.
5.2.1.5 - [string] strategy -show the DOCO movement strategy
5.2.1.6 - [bool] food_present – this will store whether there is food present in a cell so that the DOCO can smell it and go to it when nearby
5.2.1.7 - [int] food_count - this will store the food count in the cell
5.2.1.8 - [int] obstacle_count
5.2.1.9 - [char] symbol – will store the symbol to print to the board for the location

5.3 - Methods

5.3.1 - public:

5.3.1.1 - [constructor] Cell(x_pos, y_pos) – create the cell object, it can not be created without a position on the board.
5.3.1.1.1 - Return cell x position (this->x_pos)
5.3.1.1.2 - return cell y position (this->y_pos)

```
5.3.1.2 -  [destructor] ~Cell – de-allocate memory for the Cell when program ends.
5.3.1.3 -  addFood(int) – add food to the Cell with the amount specified
5.3.1.3.1 -  this->food_count += foodAdded
5.3.1.4 -  [bool] getFoodCount() - returns the number of food pellets in the cell
5.3.1.4.1 -  return this->food_count
5.3.1.5 -  [bool] getFoodPresent() - returns whether food is present or not in the cell
5.3.1.5.1 -  return this->food_present
5.3.1.6 -  [bool] getOccupied() - returns whether or not the cell is occupied by a DOCO
5.3.1.6.1 -  return this→occupied
5.3.1.6.2 -  [bool] getObstacle() - returns whether or not the cell is occupied by an
                  Obstacle
5.3.1.7 -  [char] getSymbol() - gets the character symbol for this cell
5.3.1.7.1 -  return this->symbol
5.3.1.8 -  [int] getXPos() - return the x_pos of the Cell
5.3.1.8.1 -  return cell x position (this->x_pos)
5.3.1.9 -  [int] getYPos() - returns the y_pos of the Cell
5.3.1.9.1 -  return cell y position (this->y_pos)
5.3.1.10 -  [void] removeAllFood() - set the food count to zero
5.3.1.10.1 -  Set food_count equal to zero.
5.3.1.11 -  [void] removeFood(int) – remove food pellets from cell with amount specified
5.3.1.11.1 -  subtract the amount of food passed in.
5.3.1.12 -  [void] setCustomSymbol(char)
5.3.1.12.1 -  set character symbol for the cell to the char provided.
5.3.1.13 -  [void] setFoodPresent(bool) – set whether there is any food in the Cell
5.3.1.13.1 -  if food_count = 0, then food_present = false, else food_present = true
5.3.1.13.1.1 -  void Cell::setFoodPresent()
5.3.1.13.1.2 -  {
5.3.1.13.1.3 -         if (this->food_count == 0) {
5.3.1.13.1.4 -            this->food_present = false;
5.3.1.13.1.5 -         }
5.3.1.13.1.6 -         else {
5.3.1.13.1.7 -            this->food_present = true;
5.3.1.13.1.8 -         }
5.3.1.13.1.9 -  }
5.3.1.14 -  [void] setOccupied(bool) – set whether the cell is occupied or not by a DOCO
5.3.1.14.1 -  set occupied to the bool passed in.
5.3.1.15 -  [void] setSymbol(char) – set character symbol for the cell to the char
                provided.
5.3.1.15.1 -  void Cell::setSymbol()
5.3.1.15.2 -  {
5.3.1.15.3 -        // The symbols are as follows : '*' = location of a DOCO,
5.3.1.15.4 -        // '.' = one or more food pellets, and '-' = an empty cell.
5.3.1.15.5 -        // prioritizes the occupied symbol.
5.3.1.15.6 -        if (this->occupied) {
5.3.1.15.7 -              if (this->strategy == "horizontal") {
5.3.1.15.8 -                    this->symbol = '=';
5.3.1.15.9 -              }
5.3.1.15.10 -             else if (this->strategy == "vertical") {
5.3.1.15.11 -                   this->symbol = '|';
5.3.1.15.12 -             }
5.3.1.15.13 -             else if (this->strategy == "diagonal") {
5.3.1.15.14 -                   this->symbol = 'X';
5.3.1.15.15 -             }
5.3.1.15.16 -             else { // default DOCO movement
5.3.1.15.17 -                   this->symbol = '*';
```

```
5.3.1.15.18 -             }
5.3.1.15.19 -       }
5.3.1.15.20 -       else if (this->obstacle) {
5.3.1.15.21 -             this->symbol = unsigned int(0xB2); // obstacle
5.3.1.15.22 -       }
5.3.1.15.23 -       else if (this->food_present) { // cell with one or more food
5.3.1.15.24 -             this->symbol = '.';
5.3.1.15.25 -       }
5.3.1.15.26 -       else { // empty cell
5.3.1.15.27 -             this->symbol = '-';
5.3.1.15.28 -       }
5.3.1.15.29 -  }
5.3.1.15.30 -  If the cell is occupied set symbol to “*’, if the cell has food set it to
                    ‘.’, if it has neither set it to ‘-’.
5.3.1.16 -  [void] setXPos(int) – set the x_pos of the cell
5.3.1.16.1 -  x_pos = int passed in
5.3.1.17 -  [void] setYPos(int) – set the y_pos of the cell
5.3.1.17.1 -  y_pos = int passed in
5.3.1.18 -  [string] getStrategy() - return a movement Stategy for a doco in the cell
5.3.1.19 -  setOccupied(bool) – set whether the cell is occupied or not by a DOCO
5.3.1.20 -  setObstacle(bool) – set whether the cell is occupied or not by a Obstacle
5.3.1.21 -  setStragey(string) – set DOCO movement strategy If DOCO present
5.3.1.22 -  [bool] getObstacle() - returns whether or not the cell is occupied by an
                Obstacle
```

6 - GridSize

6.1 - Summary

6.1.1 - GridSize contains your grid shape info.

6.2 - Properties

6.2.1 - private:

```
6.2.2 -  [int] height – this is for the height of the grid
6.2.3 -  [int] width – this for the width of the grid
```

6.3 - Methods

6.3.1 - public:

```
6.3.1.1 -  [constructor] GridSize()
6.3.1.1.1 -  create grid size object with uninitialized height and width.
6.3.1.1.1.1 -  GridSize::GridSize() {
6.3.1.1.1.2 -          this->height = 0;
6.3.1.1.1.3 -          this->width = 0;
6.3.1.1.1.4 -  }
6.3.1.2 -  [constructor] GridSize(int width, int height) – in order to make a grid object
                it should be required that the width and height are there.
6.3.1.2.1 -  GridSize::GridSize(int w, int h)
6.3.1.2.2 -  {
6.3.1.2.3 -         this->height = h;
6.3.1.2.4 -         this->width = w;
6.3.1.2.5 -  }
6.3.1.2.6 -  Set object’s private height information
6.3.1.2.7 -  sets object private width information
6.3.1.3 -  [destructor] ~GridSize() - want to de-allocate memory when this is destroyed
6.3.1.4 -  [int] getHeight() - returns height
6.3.1.4.1 -  returns the objects private height variable
6.3.1.5 -  [int] getWidth() - returns width
```

6.3.1.5.1 - returns the objects private width variable
6.3.1.6 - setHeight(int) – sets object height
6.3.1.6.1 - sets the objects private height variable
6.3.1.7 - setWidth(int) – sets object width
6.3.1.7.1 - sets the objects private width variable

7 - CellGrid

7.1 - Summary

7.1.1 - Is a grid of cell object in the form of a matrix

7.2 - Properties

7.2.1 - private
7.2.1.1 - [int[doco_char_count - # of doco's on the board
7.2.1.2 - [int] doco_count - # of docos from vector
7.2.1.3 - [int] food_char_count - # number of unique food positions on the board.
7.2.1.4 - [int] obstacle_count - # of obstacles
7.2.1.5 - [GridSize] my_grid_size – size of the grid
7.2.1.6 - [matrix] char_matrix – holds the matrix of cells in character format.
7.2.1.7 - [matrix] temp_adjoined_cells – holds temporary adjoined cells for each time findAdjoinedCells(x_pos, y_pos) is called.
7.2.2 - Public
7.2.2.1 - [Cell] cell_matrix – holds the matrix of cell objects

7.3 - Methods

7.3.1 - public:
7.3.1.1 - [constructor] CellGrid(height, width) – creates the gridShape to the specified height and width, then populates the cell_matrix with cell_objects initialized to each position.

```
7.3.1.1.1 - CellGrid::CellGrid(int h, int w)
7.3.1.1.2 - {
7.3.1.1.3 -         this->my_grid_size.setHeight(h);
7.3.1.1.4 -         this->my_grid_size.setWidth(w);
7.3.1.1.5 -         int height = my_grid_size.getHeight();
7.3.1.1.6 -         int width = my_grid_size.getWidth();
7.3.1.1.7 -         this->cell_matrix = std::vector<std::vector<Cell> > (height);
7.3.1.1.8 -         // build cell matrix
7.3.1.1.9 -         for (auto y = 0; y < height; y++)
7.3.1.1.10 -        {
7.3.1.1.11 -              for (auto x = 0; x < width; x++)
7.3.1.1.12 -              {
7.3.1.1.13 -                    this->cell_matrix[y].push_back(Cell(x, y));
7.3.1.1.14 -              }
7.3.1.1.15 -        }
7.3.1.1.16 -        this->cell_matrix.shrink_to_fit();          // removes excess
                    allocations
7.3.1.1.17 - }
```

7.3.1.2 - [destructor] ~CellGrid() - de-allocates memory for CellGrid object
7.3.1.3 - [matrix] getCharMatrix() - returns the char_matrix private variable
7.3.1.4 - [matrix] getMatrix() - returns the cell_matrix private variable
7.3.1.5 - [matrix] initCharMatrix()

```
7.3.1.5.1 - void CellGrid::initCharMatrix(int world_w, int world_h) {
7.3.1.5.2 -         // Set the char_matrix to their appropriate characters
7.3.1.5.3 -         // based on the status of the cells.
7.3.1.5.4 -         int x = 0;
7.3.1.5.5 -         int y = 0;
7.3.1.5.6 -         this->char_matrix = std::vector<std::vector<char> >(world_h);
```

```
7.3.1.5.7 -         for (y = 0; y < world_h; y++)
7.3.1.5.8 -         {
7.3.1.5.9 -               for (x = 0; x < world_w; x++)
7.3.1.5.10 -              {
7.3.1.5.11 -                    this->char_matrix[y].push_back(' ');
7.3.1.5.12 -              }
7.3.1.5.13 -        }
7.3.1.5.14 -  }
7.3.1.6 -  [void] printCharMatrix() - returns the character matrix in printed form based
               off of the char_matrix class property.
7.3.1.6.1 -  void CellGrid::printCharMatrix(void)
7.3.1.6.2 -  {
7.3.1.6.3 -         for (auto y = 0; y < my_grid_size.getHeight(); y++)
7.3.1.6.4 -         {
7.3.1.6.5 -               for (auto x = 0; x < my_grid_size.getWidth(); x++)
7.3.1.6.6 -               {
7.3.1.6.7 -                     std::cout << this->char_matrix[y][x] << ' ';
7.3.1.6.8 -               }
7.3.1.6.9 -               std::cout << "\n";
7.3.1.6.10 -        }
7.3.1.6.11 -  }
7.3.1.7 -  [matrix] setCharMatrix() - set the char_matrix to their appropriate characters
               based on the status of the cells.
7.3.1.7.1 -  void CellGrid::setCharMatrix(void)
7.3.1.7.2 -  {
7.3.1.7.3 -         // Set the char_matrix to their appropriate characters
7.3.1.7.4 -         // based on the status of the cells.
7.3.1.7.5 -         int x = 0;
7.3.1.7.6 -         int y = 0;
7.3.1.7.7 -         this->food_char_count = 0;
7.3.1.7.8 -         this->doco_char_count = 0;
7.3.1.7.9 -         for (y = 0; y < my_grid_size.getHeight(); y++)
7.3.1.7.10 -        {
7.3.1.7.11 -              for (x = 0; x < my_grid_size.getWidth(); x++)
7.3.1.7.12 -              {
7.3.1.7.13 -                    this->char_matrix[y][x] = this->cell_matrix[y]
              [x].getSymbol();
7.3.1.7.14 -                    if (this->cell_matrix[y][x].getFoodPresent()) this-
              >food_char_count += 1;
7.3.1.7.15 -                    if (this->cell_matrix[y][x].getOccupied()) this-
              >doco_char_count += 1;
7.3.1.7.16 -                    if (this->cell_matrix[y][x].getObstacle()) this-
              >doco_char_count += 1;
7.3.1.7.17 -              }
7.3.1.7.18 -        }
7.3.1.7.19 -  }
7.3.1.8 -  [matrix] findAdjoinedCells(x_pos, y_pos) – using the x and y position provided
               in conjunction with its data on the cell matrix, finds the cells within
               one space of it (N, E, S, W, NE, SE, SW, NW).
7.3.1.8.1 -  // Goal: tell a DOCO what it's adjoined occupied cells are. Update the DOCOs
                   private adjoined_food_cells matrix with this information
7.3.1.8.2 -  std::vector<std::pair<int, int> > CellGrid::findAdjoinedCells(int x, int y)
7.3.1.8.3 -  {
7.3.1.8.4 -         // using the xand y position provided in conjunction with its data
7.3.1.8.5 -         // on the cell matrix, finds the cells within one space of it
```

```
7.3.1.8.6 -         // (N, E, S, W, NE, SE, SW, NW)
7.3.1.8.7 -         // DO logic off of borders
7.3.1.8.8 -         this->temp_adjoined_cells.clear();
7.3.1.8.9 -         int x_start = 0;
7.3.1.8.10 -        int y_start = 0;
7.3.1.8.11 -        int x_border = my_grid_size.getWidth();
7.3.1.8.12 -        int y_border = my_grid_size.getHeight();
7.3.1.8.13 -        std::pair<int, int> viable_pair;
7.3.1.8.14 -        int i = 0;
7.3.1.8.15 -        int j = 0;
7.3.1.8.16 -        for (i = -1; i <= 1; i++) {
7.3.1.8.17 -              for (j = -1; j <= 1; j++) {
7.3.1.8.18 -                    if (i == 0 && j == 0) continue;
7.3.1.8.19 -                    if ( ((x + i) < x_border) && ((x + i) >= x_start) && ((y
                    + j) < y_border) && ((y + j) >= y_start))
7.3.1.8.20 -                    {
7.3.1.8.21 -                          viable_pair = std::make_pair(x + i, y + j);
7.3.1.8.22 -                          this->temp_adjoined_cells.push_back(viable_pair);
7.3.1.8.23 -                    }
7.3.1.8.24 -              }
7.3.1.8.25 -        }
7.3.1.8.26 -        this->temp_adjoined_cells.shrink_to_fit();
7.3.1.8.27 -        return this->temp_adjoined_cells;
7.3.1.8.28 -  }
7.3.1.9 -  [matrix] findAdjoinedccupiedCells() – using the temporary adjoining cell matrix
           part of CellGrid, it returns a new matrix of just the cells that are
           occupied around it. Using this temp variable allows removing some error
           checking here.
7.3.1.9.1 -  // tell a DOCO what it's adjoined occupied cells are.Update the DOCO's
7.3.1.9.2 -  // private adjoined_occupied_cells matrix with this informatio
7.3.1.9.3 -  std::vector<std::pair<int, int> > CellGrid::findAdjoinedOccupiedCells()
7.3.1.9.4 -  {
7.3.1.9.5 -         std::vector<std::pair<int, int> > tempOccupiedCells;
7.3.1.9.6 -         int* x = new int;
7.3.1.9.7 -         int* y = new int;
7.3.1.9.8 -         *x = 0;
7.3.1.9.9 -         *y = 0;
7.3.1.9.10 -        for (auto pair : this->temp_adjoined_cells) {
7.3.1.9.11 -              *x = pair.first;          // store x at allocated address
7.3.1.9.12 -              *y = pair.second;         // store y at allocated address
7.3.1.9.13 -              bool occupied = this->cell_matrix[*y][*x].getOccupied();
7.3.1.9.14 -              if (occupied)// if occupied
7.3.1.9.15 -              {
7.3.1.9.16 -                    tempOccupiedCells.push_back(std::make_pair(*x, *y));
7.3.1.9.17 -              }
7.3.1.9.18 -        }
7.3.1.9.19 -        delete x;
7.3.1.9.20 -        delete y;
7.3.1.9.21 -        return tempOccupiedCells;
7.3.1.9.22 -  }
7.3.1.10 -  [matrix] findAdjoinedCellsFood() – checks each of the temp_adjoining_cells and
           returns the matrix of cells that contain food. Using the temp variable
           eliminates some error checking.
7.3.1.10.1 -  // tell a DOCO what it's adjoined cells are. Update the DOCO's
7.3.1.10.2 -  // private adjoined_cells matrix with this information.
```

```
7.3.1.10.3 -  std::vector<std::pair<int, int> > CellGrid::findAdjoinedCellsFood()
7.3.1.10.4 -  {
7.3.1.10.5 -        // checks each of the temp_adjoining_cells and returns the
7.3.1.10.6 -        // matrix of cells that contain food.Using the temp variable
7.3.1.10.7 -        // eliminates some error checking.
7.3.1.10.8 -        std::vector<std::pair<int, int> > tempFoodCells;
7.3.1.10.9 -        int* x = new int;
7.3.1.10.10 -       int* y = new int;
7.3.1.10.11 -       *x = 0;
7.3.1.10.12 -       *y = 0;
7.3.1.10.13 -       for (auto pair : this->temp_adjoined_cells) {
7.3.1.10.14 -             *x = pair.first;          // store x at allocated address
7.3.1.10.15 -             *y = pair.second;         // store y at allocated address
7.3.1.10.16 -             if (this->cell_matrix[*y][*x].getFoodPresent()) // if food is
                    present
7.3.1.10.17 -             {
7.3.1.10.18 -                   tempFoodCells.push_back(std::make_pair(*x, *y));
7.3.1.10.19 -             }
7.3.1.10.20 -       }
7.3.1.10.21 -       delete x;
7.3.1.10.22 -       delete y;
7.3.1.10.23 -       return tempFoodCells;
7.3.1.10.24 -  }
7.3.1.11 -  [vector pair] findAdjoinedOpenCellsWithFood()
7.3.1.11.1 -  std::vector<std::pair<int, int> > CellGrid::findAdjoinedOpenCellsWithFood()
                    {
7.3.1.11.2 -        std::vector<std::pair<int, int> > tempOpenCellsWithFood;
7.3.1.11.3 -        int* x = new int;
7.3.1.11.4 -        int* y = new int;
7.3.1.11.5 -        *x = 0;
7.3.1.11.6 -        *y = 0;
7.3.1.11.7 -        for (auto pair : this->temp_adjoined_cells) {
7.3.1.11.8 -              *x = pair.first;          // store x at allocated address
7.3.1.11.9 -              *y = pair.second;         // store y at allocated address
7.3.1.11.10 -             bool obstacle = this->cell_matrix[*y][*x].getObstacle();
7.3.1.11.11 -             bool occupied = this->cell_matrix[*y][*x].getOccupied();
7.3.1.11.12 -             bool food = this->cell_matrix[*y][*x].getFoodPresent();
7.3.1.11.13 -             if (!obstacle && !occupied && food)    // if occupied or
                    obstacle
7.3.1.11.14 -             {
7.3.1.11.15 -                   tempOpenCellsWithFood.push_back(std::make_pair(*x, *y));
7.3.1.11.16 -             }
7.3.1.11.17 -       }
7.3.1.11.18 -       delete x;
7.3.1.11.19 -       delete y;
7.3.1.11.20 -       return tempOpenCellsWithFood;
7.3.1.11.21 -  }
7.3.1.12 -  [matrix] findAdjoinedObstacleCells() – using the temporary adjoining cell
                matrix part of CellGrid, it returns a new matrix of just the cells that
                are occupied around it. Using this temp variable allows removing some
                error checking here.
7.3.1.12.1 -  // tell a DOCO what it's adjoined occupied cells are.Update the DOCO's
7.3.1.12.2 -  // private adjoined_obstacle_cells matrix with this informatio
7.3.1.12.3 -  std::vector<std::pair<int, int> > CellGrid::findAdjoinedObstacleCells()
7.3.1.12.4 -  {
```

```
7.3.1.12.5 -        std::vector<std::pair<int, int> > tempObstacleCells;
7.3.1.12.6 -        int* x = new int;
7.3.1.12.7 -        int* y = new int;
7.3.1.12.8 -        *x = 0;
7.3.1.12.9 -        *y = 0;
7.3.1.12.10 -       for (auto pair : this->temp_adjoined_cells) {
7.3.1.12.11 -             *x = pair.first;          // store x at allocated address
7.3.1.12.12 -             *y = pair.second;         // store y at allocated address
7.3.1.12.13 -             bool obstacle = this->cell_matrix[*y][*x].getObstacle();
7.3.1.12.14 -             if (obstacle)// if occupied
7.3.1.12.15 -             {
7.3.1.12.16 -                   tempObstacleCells.push_back(std::make_pair(*x, *y));
7.3.1.12.17 -             }
7.3.1.12.18 -       }
7.3.1.12.19 -       delete x;
7.3.1.12.20 -       delete y;
7.3.1.12.21 -       return tempObstacleCells;
7.3.1.12.22 -   }
```

8 - DataParser

8.1 - Summary

8.1.1 - The data parser will read in the provided file and it will specify the number of DOCOs to spawn and the number of food pellets to spawn. Sample file to read in is in an XML format. This class is fully written and supplied already, so no code / pseudo code is necessary or required. It is in here for simple reference.

8.2 - Properties

8.2.1 - private:

8.2.1.1 - [ifstream] *inFile – DOCO world definition file
8.2.1.2 - [int] m_iWorldWidth – number of cells wide for DOCO grid read in
8.2.1.3 - [int] m_iWorldHeight – number of cells high for the DOCO grid read in
8.2.1.4 - [int] m_iNumDOCOs – Number of DOCOs in the world
8.2.1.5 - [int] m_iNextDOCOIndex – Index of next DOCO to read
8.2.1.6 - [int] m_FoodCount – Number of initial food pellets
8.2.1.7 - [int] m_ObstacleCount – Number of initial obstacles
8.2.1.8 - [int] m_iNextObsIndex – Index of next Obstacle to read
8.2.1.9 - [char[64]] m_sFileName – Data File name string

8.3 - Methods

8.3.1 - public:

8.3.1.1 - [constructor] DataParser(char *fileName) – creates the object an initializes from a file provided
8.3.1.2 - [destructor] ~DataParser() - destroys the object with delete
8.3.1.3 - initParser(char *filename) -
8.3.1.4 - [int] getDOCOWorldWidth() - returns the width of the world
8.3.1.5 - [int] getDOCOWorldHeight() - returns the height of the world
8.3.1.6 - [int] getDOCOCount() - returns how many DOCOs are to be spawned in
8.3.1.7 - [bool] getDOCOData(char *movement, int *xpos, int *ypos) – Reads to the current DOCO count. Returns true or false based on whether data for another DOCO is present.
8.3.1.8 - [int] getFoodCount() - returns amount of food to spawn in.
8.3.1.9 - [int] getObstacleCount() -
8.3.1.10 - [bool]getObstacleData(int x, int y) -

8.3.1.11 - [bool] getNextLine(char *buffer, int n) – Reads lines from a file and places them in buffer, removing any leading white space. Skips blank lines. Ignores comments starting with <!-- and ending with →. Returns true for a successful read, false if the end of file was encountered.

9 - Directions

9.1 - Summary

9.1.1 - Responsible for creating direction pairs associated with cardinal directions. Maps to a grid.

9.2 - Properties

9.2.1 - private:

9.2.1.1 - [vector string] move_directions
9.2.1.2 - [vector pair int] xy_modifiers
9.2.1.3 - [vector pair int] horizontal_offsets_with_dir
9.2.1.4 - [vector pair str int] horizontal_pair_offsets
9.2.1.5 - [vector pair int] vertical_offsets_with_dir
9.2.1.6 - [vector pair str int] vertical_pair_offsets
9.2.1.7 - [vector pair int] diagonal_offsets_with_dir
9.2.1.8 - [vector pair str int] diagonal_pair_offsets
9.2.2 - public:
9.2.2.1 - [vector pair str int] dir_xy_pairs

9.3 - Methods

9.3.1 - private:

9.3.1.1 - setDirXYPairs() - initializes the direction “N” and offset pair for each direction.

```
9.3.1.1.1 - void Directions::setDirXYPairs(void)
9.3.1.1.2 - {
9.3.1.1.3 -         for (int i = 0; i < 8; ++i)
9.3.1.1.4 -         {
9.3.1.1.5 -               this->dir_xy_pairs.push_back(std::make_pair(this-
                  >move_directions[i], this->xy_modifiers[i]));
9.3.1.1.6 -         }
9.3.1.1.7 -         this->dir_xy_pairs.shrink_to_fit();
9.3.1.1.8 - }
```

9.3.1.2 - SetMoveDirections() - sets the cardinal move options

```
9.3.1.2.1 - std::vector<std::string> Directions::setMoveDirections(void) {
9.3.1.2.2 -         static std::string move_directions[] = {
                    "NW","W","SW","N","S","NE","E","SE" };
9.3.1.2.3 -         return std::vector<std::string>(move_directions, (move_directions +
                    (sizeof(move_directions) / sizeof(std::string))));
9.3.1.2.4 - }
```

9.3.1.3 - setXYModifiers() - sets the XY offset modifiers

```
9.3.1.3.1 - void Directions::setXYModifiers(void)
9.3.1.3.2 - {
9.3.1.3.3 -         std::pair<int, int> viable_pair;
9.3.1.3.4 -         int i, j;
9.3.1.3.5 -         for (i = -1; i <= 1; i++) {
9.3.1.3.6 -               for (j = -1; j <= 1; j++) {
9.3.1.3.7 -                     if (i == 0 && j == 0) continue;
9.3.1.3.8 -                     viable_pair = std::make_pair(i, j);
9.3.1.3.9 -                     xy_modifiers.push_back(viable_pair);
9.3.1.3.10 -              }
9.3.1.3.11 -        }
```

```
9.3.1.3.12 -        xy_modifiers.shrink_to_fit();
9.3.1.3.13 -   }
9.3.2 -   public:
9.3.2.1 -   [void] setPairPatterns(void)
9.3.2.1.1 -   void Directions::setPairPatterns(void)
9.3.2.1.2 -   {
9.3.2.1.3 -         this->horizontal_offsets_with_dir.push_back(std::make_pair("W",
                    std::make_pair(-1, 0)));
9.3.2.1.4 -         this->horizontal_offsets_with_dir.push_back(std::make_pair("E",
                    std::make_pair(1, 0)));
9.3.2.1.5 -         this->horizontal_pair_offsets.push_back(std::make_pair(-1, 0));
9.3.2.1.6 -         this->horizontal_pair_offsets.push_back(std::make_pair(1, 0));
9.3.2.1.7 -         this->horizontal_offsets_with_dir.shrink_to_fit();
9.3.2.1.8 -
9.3.2.1.9 -         this->vertical_offsets_with_dir.push_back(std::make_pair("N",
                    std::make_pair(0, -1)));
9.3.2.1.10 -        this->vertical_offsets_with_dir.push_back(std::make_pair("S",
                    std::make_pair(0, 1)));
9.3.2.1.11 -        this->vertical_pair_offsets.push_back(std::make_pair(0, -1));
9.3.2.1.12 -        this->vertical_pair_offsets.push_back(std::make_pair(0, 1));
9.3.2.1.13 -        this->vertical_offsets_with_dir.shrink_to_fit();
9.3.2.1.14 -
9.3.2.1.15 -        this->diagonal_offsets_with_dir.push_back(std::make_pair("NW",
                    std::make_pair(-1, -1)));
9.3.2.1.16 -        this->diagonal_offsets_with_dir.push_back(std::make_pair("NE",
                    std::make_pair(1, -1)));
9.3.2.1.17 -        this->diagonal_offsets_with_dir.push_back(std::make_pair("SW",
                    std::make_pair(-1, 1)));
9.3.2.1.18 -        this->diagonal_offsets_with_dir.push_back(std::make_pair("SE",
                    std::make_pair(1, 1)));
9.3.2.1.19 -        this->diagonal_pair_offsets.push_back(std::make_pair(-1, -1));
9.3.2.1.20 -        this->diagonal_pair_offsets.push_back(std::make_pair(1, -1));
9.3.2.1.21 -        this->diagonal_pair_offsets.push_back(std::make_pair(-1, 1));
9.3.2.1.22 -        this->diagonal_pair_offsets.push_back(std::make_pair(1, 1));
9.3.2.1.23 -        this->diagonal_offsets_with_dir.shrink_to_fit();
9.3.2.1.24 -
9.3.2.1.25 -        this->perp_offsets_with_dir.push_back(std::make_pair("W",
                    std::make_pair(-1, 0)));
9.3.2.1.26 -        this->perp_offsets_with_dir.push_back(std::make_pair("E",
                    std::make_pair(1, 0)));
9.3.2.1.27 -        this->perp_offsets_with_dir.push_back(std::make_pair("N",
                    std::make_pair(0, -1)));
9.3.2.1.28 -        this->perp_offsets_with_dir.push_back(std::make_pair("S",
                    std::make_pair(0, 1)));
9.3.2.1.29 -        this->perp_pair_offsets.push_back(std::make_pair(-1, 0));
9.3.2.1.30 -        this->perp_pair_offsets.push_back(std::make_pair(1, 0));
9.3.2.1.31 -        this->perp_pair_offsets.push_back(std::make_pair(0, -1));
9.3.2.1.32 -        this->perp_pair_offsets.push_back(std::make_pair(0, 1));
9.3.2.1.33 -        this->perp_offsets_with_dir.shrink_to_fit();
9.3.2.1.34 -   }
9.3.2.2 -   [constructor] Directions()
9.3.2.2.1 -   Directions::Directions()
9.3.2.2.2 -   {
9.3.2.2.3 -         this->move_directions = setMoveDirections();
9.3.2.2.4 -         this->move_directions.shrink_to_fit();
```

```
9.3.2.2.5 -         this->setXYModifiers();
9.3.2.2.6 -         this->setDirXYPairs();
9.3.2.2.7 -         this->setPairPatterns();
9.3.2.2.8 -   }
9.3.2.3 -   [deconstructor] ~Directions()
9.3.2.4 -   getAllDirections() - returns all possible directions
9.3.2.5 -   getDirectionPairs() - returns all direction and pair combos
9.3.2.6 -   getXYOffsets() - returns the offsets for x and y
9.3.2.7 -   getHorizontalPairsOffsetsWithDir()
9.3.2.8 -   getHorizontalPairsOffsets()
9.3.2.9 -   getVerticalPairsOffsetsWithDir()
9.3.2.10 -   getVerticalPairsOffsets()
9.3.2.11 -   getDiagonalPairsOffsetsWithDir()
9.3.2.12 -   getDiagonalPairsOffsets()
9.3.2.13 -   getPerpPairsOffsetsWithDir()
9.3.2.14 -   getPerpPairsOffsets()
9.3.2.15 -   getDirForPair(string) - returns the direction given a pair offset.
9.3.2.15.1 -   std::string Directions::getDirForPair(std::pair<int, int> presentPair,
                    std::pair<int, int> movePair)
9.3.2.15.2 -   {
9.3.2.15.3 -         int d_y = -movePair.second + presentPair.second;
9.3.2.15.4 -         int d_x = movePair.first - presentPair.first;
9.3.2.15.5 -         if (this->dir_xy_pairs.at(4).second.first == d_x && this-
                    >dir_xy_pairs.at(4).second.second == d_y) return "N";
9.3.2.15.6 -         if (this->dir_xy_pairs.at(7).second.first == d_x && this-
                    >dir_xy_pairs.at(7).second.second == d_y) return "NE";
9.3.2.15.7 -         if (this->dir_xy_pairs.at(6).second.first == d_x && this-
                    >dir_xy_pairs.at(6).second.second == d_y) return "E";
9.3.2.15.8 -         if (this->dir_xy_pairs.at(5).second.first == d_x && this-
                    >dir_xy_pairs.at(5).second.second == d_y) return "SE";
9.3.2.15.9 -         if (this->dir_xy_pairs.at(3).second.first == d_x && this-
                    >dir_xy_pairs.at(3).second.second == d_y) return "S";
9.3.2.15.10 -         if (this->dir_xy_pairs.at(0).second.first == d_x && this-
                    >dir_xy_pairs.at(0).second.second == d_y) return "SW";
9.3.2.15.11 -         if (this->dir_xy_pairs.at(1).second.first == d_x && this-
                    >dir_xy_pairs.at(1).second.second == d_y) return "W";
9.3.2.15.12 -         if (this->dir_xy_pairs.at(2).second.first == d_x && this-
                    >dir_xy_pairs.at(2).second.second == d_y) return "NW";
9.3.2.15.13 -         else return "N";
9.3.2.15.14 -   }
9.3.2.16 -   getOppositeDirectionPair(string) - returns the opposite direction of what's
                  passed in.
9.3.2.16.1 -   std::pair<std::string, std::pair<int, int> >
                    Directions::getOppositeDirectionPair(std::pair<std::string,
                    std::pair<int, int> > dir_xy_offset) {
9.3.2.16.2 -         std::string direction = dir_xy_offset.first;
9.3.2.16.3 -         std::pair<std::string, std::pair<int, int> > opposite;
9.3.2.16.4 -         if (direction == "N") {
9.3.2.16.5 -               opposite = this->getPairComboForString("S");
9.3.2.16.6 -         }
9.3.2.16.7 -         else if (direction == "NE") {
9.3.2.16.8 -               opposite = this->getPairComboForString("SW");
9.3.2.16.9 -         }
9.3.2.16.10 -         else if (direction == "E") {
9.3.2.16.11 -               opposite = this->getPairComboForString("W");
```

```
9.3.2.16.12 -       }
9.3.2.16.13 -       else if (direction == "SE") {
9.3.2.16.14 -             opposite = this->getPairComboForString("NW");
9.3.2.16.15 -       }
9.3.2.16.16 -       else if (direction == "S") {
9.3.2.16.17 -             opposite = this->getPairComboForString("N");
9.3.2.16.18 -       }
9.3.2.16.19 -       else if (direction == "SW") {
9.3.2.16.20 -             opposite = this->getPairComboForString("NE");
9.3.2.16.21 -       }
9.3.2.16.22 -       else if (direction == "W") {
9.3.2.16.23 -             opposite = this->getPairComboForString("E");
9.3.2.16.24 -       }
9.3.2.16.25 -       else if (direction == "NW") {
9.3.2.16.26 -             opposite = this->getPairComboForString("SE");
9.3.2.16.27 -       }
9.3.2.16.28 -       return opposite;
9.3.2.16.29 -  }
9.3.2.17 -  getPairComboForString()
9.3.2.17.1 -  std::pair<std::string, std::pair<int, int> >
                  Directions::getPairComboForString(std::string dir)
9.3.2.17.2 -  {
9.3.2.17.3 -        if (dir == "N") return this->dir_xy_pairs.at(4);
9.3.2.17.4 -        else if (dir == "NE") return this->dir_xy_pairs.at(7);
9.3.2.17.5 -        else if (dir == "E") return this->dir_xy_pairs.at(6);
9.3.2.17.6 -        else if (dir == "SE") return this->dir_xy_pairs.at(5);
9.3.2.17.7 -        else if (dir == "S") return this->dir_xy_pairs.at(3);
9.3.2.17.8 -        else if (dir == "SW") return this->dir_xy_pairs.at(0);
9.3.2.17.9 -        else if (dir == "W") return this->dir_xy_pairs.at(1);
9.3.2.17.10 -       else if (dir == "NW") return this->dir_xy_pairs.at(2);
9.3.2.17.11 -       else return this->dir_xy_pairs.at(4);
9.3.2.17.12 -  }
9.3.2.18 -  getRandomDirectionPair()
9.3.2.19 -  printDirXYPairs() - prints all the pair offsets with the associated direction
              to cout
9.3.2.19.1 -  void Directions::printDirXYPairs()
9.3.2.19.2 -  {
9.3.2.19.3 -        for (auto i : this->dir_xy_pairs)
9.3.2.19.4 -        {
9.3.2.19.5 -              std::cout << i.first << "  (" << i.second.first << ", " <<
                  i.second.second << ")\n";
9.3.2.19.6 -        }
9.3.2.19.7 -  }
```

10 - DocoFactory

10.1 - Summary

10.1.1 - Responsible for creating docos of varying types

10.2 - Properties

10.2.1 - private:

```
10.2.1.1 -  [int] instance number
```

10.3 - Methods

10.3.1 - public:

```
10.3.1.1 -  int getInstanceNumber() => return number of DocoFactories
10.3.1.1.1 -  int DocoFactory::getInstanceNumber() {
```

```
10.3.1.1.2 -        return this->instanceNumber;
10.3.1.1.3 -  }
10.3.1.2 -  [Doco] createDocoDefault(int x_pos, int y_pos, std::string direction)
10.3.1.2.1 -  Doco* DocoFactory::createDocoDefault(int x_pos, int y_pos, std::string dir)
10.3.1.2.2 -  {
10.3.1.2.3 -        // --- Return a doco of random movement type
10.3.1.2.4 -        DocoMovePatternDefault* strategy = new DocoMovePatternDefault();
10.3.1.2.5 -        std::string patternName = std::string("Random");
10.3.1.2.6 -        Doco* newDoco = new Doco(x_pos, y_pos, dir, patternName);
10.3.1.2.7 -        newDoco->setPtrMoveStrategy(strategy);
10.3.1.2.8 -        return newDoco;
10.3.1.2.9 -  }
10.3.1.3 -  [Doco] createDocoDiagonal(int x_pos, int y_pos, std::string direction)
10.3.1.3.1 -  Doco* DocoFactory::createDocoDiagonal(int x_pos, int y_pos, std::string dir)
10.3.1.3.2 -  {
10.3.1.3.3 -        // --- Return a doco of diagonal movement type
10.3.1.3.4 -        DocoMovePatternDiagonal* strategy = new DocoMovePatternDiagonal();
10.3.1.3.5 -        std::string patternName = std::string("Diagonal");
10.3.1.3.6 -        Doco* newDoco = new Doco(x_pos, y_pos, dir, patternName);
10.3.1.3.7 -        newDoco->setPtrMoveStrategy(strategy);
10.3.1.3.8 -        return newDoco;
10.3.1.3.9 -  }
10.3.1.4 -  [Doco] createaeDocoHorizontal(int x_pos, int y_pos, std::string direction)
10.3.1.4.1 -  Doco* DocoFactory::createDocoHorizontal(int x_pos, int y_pos, std::string
              dir)
10.3.1.4.2 -  {
10.3.1.4.3 -        // --- Return a doco of horizontal movement type
10.3.1.4.4 -        DocoMovePatternHorizontal* strategy = new DocoMovePatternHorizontal();
10.3.1.4.5 -        std::string patternName = std::string("Horizontal");
10.3.1.4.6 -        Doco* newDoco = new Doco(x_pos, y_pos, dir, patternName);
10.3.1.4.7 -        newDoco->setPtrMoveStrategy(strategy);
10.3.1.4.8 -        return newDoco;
10.3.1.4.9 -  }
10.3.1.5 -  [Doco] createDocoVertical(int x_pos, int y_pos, std::string direction)
10.3.1.5.1 -  Doco* DocoFactory::createDocoVertical(int x_pos, int y_pos, std::string dir)
10.3.1.5.2 -  {
10.3.1.5.3 -        // --- Return a doco of vertical movement type
10.3.1.5.4 -        DocoMovePatternVertical* strategy = new DocoMovePatternVertical();
10.3.1.5.5 -        std::string patternName = std::string("Vertical");
10.3.1.5.6 -        Doco* newDoco = new Doco(x_pos, y_pos, dir, patternName);
10.3.1.5.7 -        newDoco->setPtrMoveStrategy(strategy);
10.3.1.5.8 -        return newDoco;
10.3.1.5.9 -  }
10.3.1.6 -  [Doco] createDocoPerp(int x_pos, int y_pos, std::string direction)
10.3.1.6.1 -  Doco* DocoFactory::createDocoPerp(int x_pos, int y_pos, std::string dir)
10.3.1.6.2 -  {
10.3.1.6.3 -        // --- Return a doco of perpendicular movement type
10.3.1.6.4 -        DocoMovePatternPerp* strategy = new DocoMovePatternPerp();
10.3.1.6.5 -        std::string patternName = std::string("Perp");
10.3.1.6.6 -        Doco* newDoco = new Doco(x_pos, y_pos, dir, patternName);
10.3.1.6.7 -        newDoco->setPtrMoveStrategy(strategy);
10.3.1.6.8 -        return newDoco;
10.3.1.6.9 -  }
```

11 - DocoMovePatternDiagonal

11.1 - Summary

11.1.1 - Responsible for the diagonal move pattern preference and the move avoidance preference. Goal is to return pairs that it can go to given it's pattern.

11.2 - Properties

11.2.1 - public:

11.2.1.1 - [Directions] moveDirs

11.3 - Methods

11.3.1 - public:

11.3.1.1 - moveStrategy() - returns the pairs diagonal can move to

```
11.3.1.1.1 -        return moveDiagDirs->getDiagonalPairOffsets();
```

11.3.1.2 - avoidanceStrategy() - returns the pairs diagonal moves to when a wall is hit

```
11.3.1.2.1 -        return moveDiagDirs->getPerpPairOffsets();
```

11.3.1.3 - [constructor]DocoMovePatternDiagonal()

11.3.1.4 - [destructor]~docoMovePatternDiagonal

12 - DocoMovePatternHorizontal

12.1 - Summary

12.1.1 - Responsible for the Horizontal move pattern preference and the move avoidance preference. Goal is to return pairs that it can go to given it's pattern.

12.2 - Properties

12.2.1 - public:

12.2.1.1 - [directions] moveDirs

12.3 - Methods

12.3.1 - public:

12.3.1.1 - moveStrategy() - returns the pairs horizontal move pattern can move to.

```
12.3.1.1.1 -        return moveHorDirs->getHorizontalPairOffsets();
```

12.3.1.2 - avoidanceStrategy() - returns the pairs horizontal moves to when a wall is hit.

```
12.3.1.2.1 -  std::vector<std::pair<int, int> >
                    DocoMovePatternHorizontal::avoidanceStrategy() {
12.3.1.2.2 -        // Implement horizontal movement avoidance strategy
12.3.1.2.3 -        //          If an edge of the world is encountered the DOCO will
                    randomly elect to move up
12.3.1.2.4 -        //          or down a row and reverse its direction of movement.
12.3.1.2.5 -        std::vector<std::pair<int, int> > avoidanceStrategy;
12.3.1.2.6 -        avoidanceStrategy = moveHorDirs->getVerticalPairOffsets(); // Option to
                    move up / down a row.
12.3.1.2.7 -        return avoidanceStrategy;
12.3.1.2.8 -  }
```

12.3.1.3 - [constructor]DocoMovePatternHorizontal()

12.3.1.4 - [deconstructor] ~DocoMovePatternDiagonal()

13 - DocoMovePatternDefault

13.1 - Summary

13.1.1 - Responsible for the default move pattern preference and the move avoidance preference. Goal is to return pairs that it can go to given it's pattern.

13.2 - Properties

13.2.1 - public:

13.2.1.1 - [directions] moveDirs

13.3 - Methods

13.3.1 - public:
13.3.1.1 - moveStrategy() - returns the pairs a default pattern moves to
13.3.1.1.1 - return moveDefaultDirs->getXYOffsets();
13.3.1.2 - avoidanceStrategy() - returns the pairs a default pattern moves to when a wall is hit
13.3.1.2.1 - return moveDefaultDirs->getXYOffsets(); // TODO: decide what offsets to get
13.3.1.3 - [constructor]DocoMovePatternDefaultl()
13.3.1.4 - [deconstructor] ~DocoMovePatternDefault()

14 - DocoMovePatternPerp

14.1 - Summary

14.1.1 - Responsible for the perpendicular move pattern preference and the move avoidance preference. Goal is to return pairs that it can go to given it's pattern.

14.2 - Properties

14.2.1 - public:
14.2.1.1 - [directions] moveDirs

14.3 - Methods

14.3.1 - public:
14.3.1.1 - moveStrategy() - returns the pairs a perpendicular pattern can move to
14.3.1.2 - avoidanceStrategy() - returns the pairs a perpendicular pattern can move to when a wall is hit.
14.3.1.3 - [constructor] DocoMovePatternPerp()
14.3.1.4 - [deconstructor] ~DocoMovePatternPerp()

15 - DocoMovePatternVertical

15.1 - Summary

15.1.1 - Responsible for the vertical move pattern preference and the move avoidance preference. Goal is to return pairs that it can go to given it's pattern.

15.2 - Properties

15.2.1 - public:
15.2.1.1 - [directions] moveDirs

15.3 - Methods

15.3.1 - public:
15.3.1.1 - moveStrategy()
15.3.1.1.1 - return movePerpDirs->getPerpPairOffsets();
15.3.1.2 - avoidanceStrategy()
15.3.1.2.1 - return movePerpDirs->getDiagonalPairOffsets(); // TODO: decide what offsets to get
15.3.1.3 - [constructor]DocoMovePatternVertical()
15.3.1.4 - [deconstructor] ~DocoMovePatternVertical()

16 - UniformRandom

16.1 - Summary

16.1.1 - Responsible for generating random numbers in a uniform distribution.

16.2 - Properties

16.2.1 - private:
16.2.1.1 - [bool] active

16.3 - Methods

16.3.1 - public:

```
16.3.1.1 -  [contructor]UniformRandom()
16.3.1.1.1 -  UniformRandom::UniformRandom()
16.3.1.1.2 -  {
16.3.1.1.3 -        this->active = true;
16.3.1.1.4 -  }
16.3.1.2 -  [int] generateRandomNum(int, int) – generates a random number between a lower
              and upper bound inclusive.
16.3.1.2.1 -  int UniformRandom::generateRandomNum(int min, int max) {
16.3.1.2.2 -        std::uniform_int_distribution<int> uni(min, max);
16.3.1.2.3 -        auto rand_int = uni(rng);
16.3.1.2.4 -        return rand_int;
16.3.1.2.5 -  }
16.3.1.3 -  [Uniform Random Instance] getInstance()
16.3.1.3.1 -  UniformRandom* UniformRandom::getInstance()
16.3.1.3.2 -  {
16.3.1.3.3 -        static UniformRandom* theInstance = nullptr;
16.3.1.3.4 -        static int counter = 1;
16.3.1.3.5 -        if (theInstance == nullptr) {
16.3.1.3.6 -              theInstance = new UniformRandom();
16.3.1.3.7 -              theInstance->instanceNumber = counter;
16.3.1.3.8 -              counter++;   // If another instance is created ever, the counter
                    will show it.
16.3.1.3.9 -        }
16.3.1.3.10 -       return theInstance;
16.3.1.3.11 -  }
```

17 - AbstractDoco

17.1 - Summary

17.1.1 - Responsible for creating the contract for what Doco needs to implement.

17.2 - Properties

17.3 - Methods

17.3.1 - public:

```
17.3.1.1 -  Doco()
17.3.1.2 -  ~Doco
17.3.1.3 -  addEnergy()
17.3.1.4 -  eat()
17.3.1.5 -  getAlive()
17.3.1.6 -  getDirection()
17.3.1.7 -  getDirectionString()
17.3.1.8 -  getEnergy()
17.3.1.9 -  getPosPair()
17.3.1.10 -  getXPos()
17.3.1.11 -  getYPos()
17.3.1.12 -  move()
17.3.1.13 -  setAlive()
17.3.1.14 -  setDirection()
17.3.1.15 -  setEnergy()
17.3.1.16 -  setPos()
17.3.1.17 -  setPos()
17.3.1.18 -  move()
```

18 - AbstractSimpleDocoFactory

18.1 - Summary

18.1.1 - Responsible for creating the contract for a Doco factory and what it has to implement.

18.2 - Properties

18.3 - Methods

18.3.1 - public:

```
18.3.1.1 -  createDocoDefault()
18.3.1.2 -  createDocoDiagonal()
18.3.1.3 -  createDocoHorizontal()
18.3.1.4 -  createDocoVertical()
18.3.1.5 -  createDocoPerp()
```

19 - DocoMoveStrategy

19.1 - Summary

19.1.1 - Responsible for creating the move strategy of the doco, this is the interface Doco is programmed to.

19.2 - Properties

19.3 - Methods

19.3.1 - public:

```
19.3.1.1 -  moveStrategy() - subclass must implement this
19.3.1.2 -  avoidanceStrategy() - subclass must implement this
```

Class Outline PlantUML Text

Edit and Recreate Diagram with: https://www.planttext.com

Last Updated: 10/31/2020 – 5:42 PM CST, UTC-6

```
@startuml

title PA-2: Class Outline

interface DocoMoveStrategy
{
    +moveStrategy()
    +avoidanceStrategy()
}

class Doco
{
    -*moveStrategy : DocoMoveStrategy
    -alive: bool
    -energy_level : int
    -position : pair int
    -direction : pair<string, <pair<int, int> >
    -position : <pair<int, int> >
    +adjoined_cells : vector<pair<int, int> >
    +adjoined_obstacle_cells : vector<pair<int, int> >
    +adjoined_occupied_cells : vector<pair<int, int> >
    +adjoined_food_cells : vector<pair<int, int> >
    +move_options : vector<pair<int, int> >
    +Doco(int, int, string& dir, string& strategy, DocoMoveStrategy* strat)
    +~Doco()

    +addEnergy(int)
    +eat(int, string)
    +getAlive() : bool
    +getDirection(void) : pair string pair int
    +getDirectionString(void) : string
    +getEnergy() : int
    +getPosPair(void) :pair int
    +getXPos()  :  int
    +getYPos()  : int
    +move(int, int) : pair<int, int>
    +setAlive(bool)
    +setDirection(string)
    +setEnergy(int)
    +setPos(int, int)
    +setPos(pair int)
    +move(int world_w, int world_h)
}

interface AbstractSimpleDocoFactory
{
    +createDocoDefault(int x_pos, int y_pos, string dir) : Doco
    +createDocoDiagonal(int, int, string) : Doco
    +createDocoHorizontal(int, int, string) : Doco
    +createDocoVertical(int, int, string) : Doco
    +createDocoPerp(int, int, string) : Doco
}

class DocoFactory {
```

```
    +note--implemented_as_singleton
    +createDocoDefault(int x_pos, int y_pos, string dir) : Doco
    +createDocoDiagonal(int, int, string) : Doco
    +createDocoHorizontal(int, int, string) : Doco
    +createDocoVertical(int, int, string) : Doco
    +createDocoPerp(int, int, string) : Doco
}

class CellGrid
{
    #doco_char_count : int
    #doco_count : int
    #food_char_count : int
    #obstacle_count : int
    #my_grid_size : GridSize
    #char_matrix :  vector<vector<char> >
    #temp_adjoined_cells :  vector<pair<int,int> >
    +cell_matrix : vector<vector<Cell> >

    +CellGrid(int, int) : CellGrid
    +~CellGrid()

    +getCharMatrix() :  vector<vector<char> >
    +getMatrix() : vector<vector<Cell> >
    +initCharMatrix(int world_w, int world_h)
    +printCharMatrix() : void
    +setCharMatrix(void)

    +findAjoinedCells(int, int) : vector<pair<int,int> >
    +findAjoinedCellsFood() :  vector<pair<int,int> >
    +findAdjoinedOccupiedCells() : vector<pair<int,int> >
    +findAdjoinedObstacleCells() : vector<pair<int,int> >

}

class WorldBoard
{
    -food_positions : vector<pair<int, int> >
    -obstacle_positions: vector<pair<int, int> >
    -height : int
    -width : int
    +doco_vect : vector<Doco>
    +myParser : * DataParser
    +worldCellGrid : * CellGrid

    -generateFoodLocations(int, int, int)
    -readFile(char filename)
    -spawnInitalDocos()

    +WorldBoard()
    +WorldBoard(char* filename)
    +~WorldBoard()
    +generateRandom() : int
    +printWorld()
    +updateAllDocoSurroundings()
    +updateCellsWithNewFood()
    +updateCellWithADoco(int,int) : int
    +updateCellWithNewFood(int, int)
    +updateCellWithNoFood(int, int) : int
    +splitDoco(iterator position)
    +updateDocos()
```

```
    +updateWorldState()
}

class main {
    update_rate : int
    display_rate : int
    max_updates : int
    inFile : char[]
    mySim : Simulator
    view : Viewer
    myDocoWorld : WorldBoard
}

class DataParser {
    -inFile: ifstream *
    -m_iWorldHeight : int
    -m_iWorldWidth : int
    -m_iNumDOCOs : int
    -m_iNextDOCOIndex : int
    -m_FoodCount : int
    -m_iObstacleCount : int
    -m_iNextObsIndex : int
    -m_sFileName : char[64]

    +~DataParser()
    +DataParser(const char *fileName) : static
    +initParser(const char *fileName)
    +getDOCOWorldHeight() : int
    +getDOCOWorldWidth() : int
    +getDOCOCount() : int
    +getDOCOData(char *movement, int *xpos, int *ypos) : bool
    +getFoodCount() : int
       +getObstacleCount() : int
+getObstacleData(int* xpos, int* ypos) : bool
    +getNextLine(char *buffer, int n) : bool
}

class Directions {
    -move_directions : vector<string>
    -xy_modifiers : vector<pair<int,int>>
    -horizontal_offsets_with_dir : vector<pair<string, pair<int, int> > >
-horizontal_pair_offsets : vector<pair<int, int> >
-vertical_offsets_with_dir : vector<pair<string, pair<int, int> > >
-vertical_pair_offsets : vector<pair<int, int> >
-diagonal_offsets_with_dir : vector<pair<string, pair<int, int> > >
-diagonal_pair_offsets : vector<pair<int, int> >
    +dir_xy_pairs : vector<pair<string,pair<int,int> > >

    +Directions()
    +~Directions()

    -setDirXYPairs() : void
    -setMoveDirections() : vector<string>
    -setXYModifiers() : void
    +getAllDirections() : vector<string>
    +getDirectionPairs() : vector<pair<string,pair<int,int>>>
    +getXYOffsets() : vector<pair<int, int> >

    +getHorizontalPairsOffsetsWithDir() : vector<pair<string, pair<int, int>>>
    +getHorizontalPairOffsets() : vector<pair<int,int> >
    +getVerticalPairOffsetsWithDir() : vector<pair<string, pair<int, int>>>
```

```
    +getVerticalPairOffsets() : vector<pair<int,int> >
    +getDiagonalPairOffsetsWithDir() : vector<pair<string, pair<int, int>>>
    +getDiagonalPairOffsets() : vector<pair<int,int> >
    +getPerpPairOffsetsWithDir() : vector<pair<string, pair<int, int>>>
    +getPerpPairOffsets() : vector<pair<int,int> >

    +getDirForPair(presentPair, movePair) : string
    +getOppositeDirectionPair(pair<string, pair<int,int>>) : pair<string, pair<int,int>>
    +getPairComboForString(string) : pair<string,pair<int,int>>
    +getRandomDirectionPair() : pair<string, pair<int, int>>
    +printDirXYPairs() : Void
}

class Simulator {
    -display_interval : int
    -paused_state : bool
    -turn_num : long int
    -update_interval : int

    +Simulator()
    +Simulator(int updateInt, displayInt)
    +~Simulator()

    +delay(int)
    +getDisplayInterval() : int
    +getPausedState() : bool
    +getTurnNum() : int
    +getUpdateInterval() : int
    +pause()
    +run()
    +setDisplayInterval(int)
    +setUpdateInterval(int)
    +turnInc()
}

class GridSize {
    -height : int
    -width : int

    +GridSize() ;
    +GridSize(int w, int h)
    +~GridSize()

    +getHeight() : int
    +getWidth() : int
    +setHeight(int)
    +setWidth(int)
}

class Cell {
    -x_pos : int
    -y_pos : int
    -obstacle : bool
    -occupied : bool
    -strategy : string
    -food_present : bool
    -food_count : int
    -symbol : char

    +Cell(int x, int y)
    +~Cell()
```

```
    +addFood(int)
    +getFoodCount() : int
    +getFoodPresent() : bool
    +getOccupied() : bool
    +getObstacle() : bool
    +getSymbol() : char
    +getXPos() : int
    +getYPos() : int
    +getStrategy() : string
    +removeAllFood()
    +removeFood(int)
    +setCustomSymbol(char)
    +setFoodPresent()
    +setOccupied(bool)
    +setObstacle(bool)
    +setSymbol()
    +setStrategy(string)
}

class Viewer {
    -footer_message : string
    -header_message : string
    -line_count_world : int
    -program_state_message : string
    -seperator_char :char
    -seperator_length : int
    -seperator_line : string

    +Viewer()
    +~Viewer()
    +getFooterMessage() : string
    +getHeaderMessage() : string
    +getLineCountWorld() : int
    +getNewSeperatorLine()  : string
    +getProgramStateMessage() : string
    +getSeperatorChar() : character
    +getSeperatorLength() : int
    +printSeperator()
    +setFooterMessage(string)
    +setHeaderMessage(string)
    +setLineCountWorld(int)
    +setNewSeperatorLine()
    +setProgramStateMessage(string)
    +setSeperatorChar(char)
    +setSeperatorLength(int)
}


interface AbstractDoco {
    +Doco(int, int, string)
    +~Doco()

    +addEnergy(int)
    +eat(int, string)
    +getAlive() : bool
    +getDirection(void) : pair string pair int
    +getDirectionString(void) : string
    +getEnergy() : int
    +getPosPair(void) :pair int
    +getXPos()  :  int
```

```
    +getYPos()  : int
    +move(int, int)
    +setAlive(bool)
    +setDirection(string)
    +setEnergy(int)
    +setPos(int, int)
    +setPos(pair int)
    +move(int world_w, int world_h)
}

class DocoMovePatternDiagonal {
    +moveDirs : Directions
    +moveStrategy()
    +avoidanceStrategy()
    +DocoMovePatternDiagonal()
    +~DocoMovePatternDiagonal
}

class DocoMovePatternHorizontal {
    +moveDirs : Directions
    +moveStrategy()
    +avoidanceStrategy()
    +DocoMovePatternHorizontal()
    +~DocoMovePatternHorizontal()
}

class DocoMovePatternVertical {
    +moveDirs : Directions
    +moveStrategy()
    +avoidanceStrategy()
    +DocoMovePatternVertical()
    +~DocoMovePatternVertical()
}

class DocoMovePatternDefault {
    +moveDirs : Directions
    +moveStrategy()
    +avoidanceStrategy()
    +DocoMovePatternDefault()
    +~DocoMovePatternDefault()
}

class DocoMovePatternPerp {
    +moveDirs : Directions
    +moveStrategy()
    +avoidanceStrategy()
    +DocoMovePatternPerp()
    +~DocoMovePatternPerp()
}

class UniformRandom
{
    -active : bool
    +UniformRandom()
    +generateRandomNum(int min, int max) : int
}

CellGrid "1" *-down- "*" Cell : owns
CellGrid "1" *-down- "1" GridSize : owns
WorldBoard "1" o-down- "1" CellGrid : has
WorldBoard "1" o-down- "1" DataParser : has
```

```
DocoMovePatternDefault .up.⊳ DocoMoveStrategy : implements
DocoMovePatternDiagonal .up.⊳ DocoMoveStrategy : implements
DocoMovePatternHorizontal .up.⊳ DocoMoveStrategy : implements
DocoMovePatternVertical .up.⊳ DocoMoveStrategy : implements
DocoMovePatternPerp .up.⊳ DocoMoveStrategy : implements

Doco .down.> DocoMoveStrategy : uses

AbstractSimpleDocoFactory ⊲.right. DocoFactory : implements
DocoFactory ..> Doco : creates
Doco "1" *-left- "1" Directions : owns
Doco .left.⊳ AbstractDoco : implements
WorldBoard .left.> AbstractDoco : uses
WorldBoard .left.> AbstractSimpleDocoFactory : uses

DocoMovePatternDiagonal

 --o DocoFactory : has
DocoMovePatternHorizontal --o DocoFactory : has
DocoMovePatternVertical --o DocoFactory : has
DocoMovePatternPerp --o DocoFactory : has
DocoMovePatternDefault --o DocoFactory : has


DocoMovePatternDiagonal o-down- Directions : has
DocoMovePatternHorizontal o-down- Directions : has
DocoMovePatternVertical o-down- Directions : has
DocoMovePatternPerp o-down- Directions : has
DocoMovePatternDefault o-down- Directions : has

UniformRandom --o Directions : has


main "1" o-down- "1" Simulator : has
main "1" o-down- "1" Viewer : has
main "1" o-down- "1" WorldBoard : has

@enduml
```

Appendix

Naming:

- Class Specific Variables: member variables, properties, attributes
- Class Specific Functions: member functions, methods, behaviors

Tools:

- https://www.planttext.com

Guides:

- https://plantuml.com/class-diagram
- http://ogom.github.io/draw_uml/plantuml