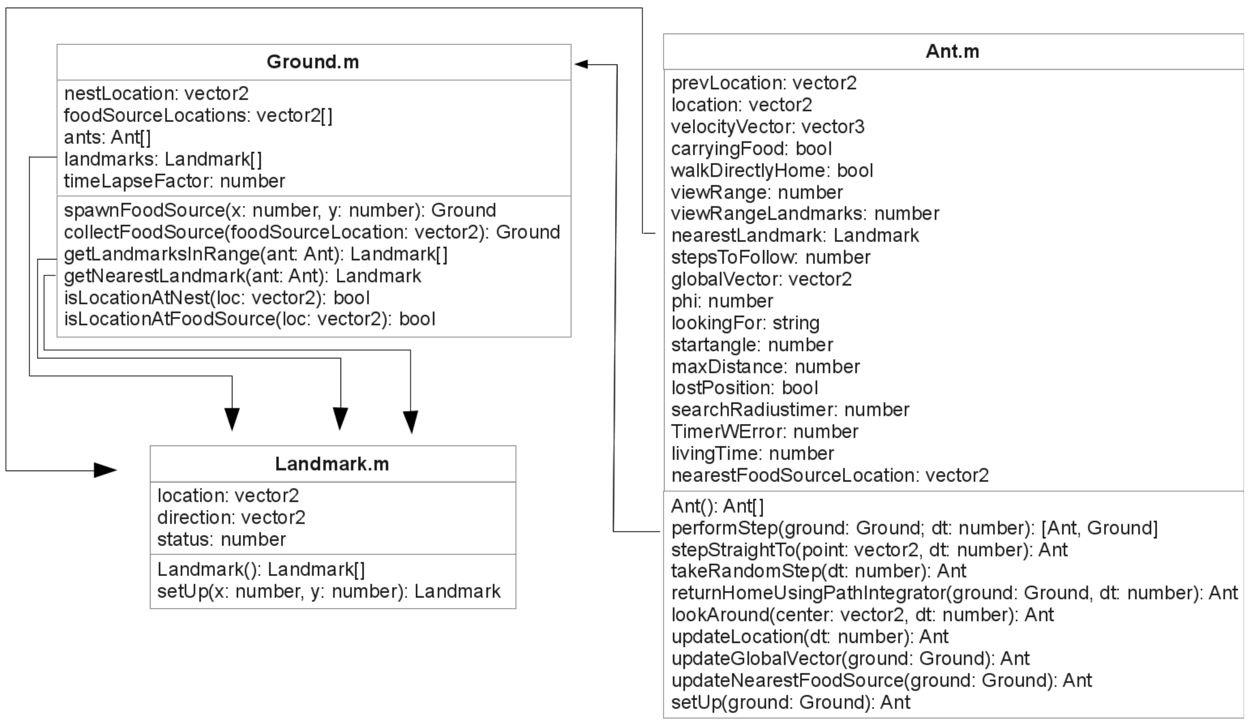
# Implementation

To implement our agent based model we decided to use object orientated programming because it seemed more intuitive than a procedural approach in this case.

## *UML-Diagram*

The following graphic shows the UML-Class diagram that describes the data model of our project. The files run.m, updateGround.m, vector2angle.m and distanceBetweenTwoPoints.m are not part of this diagram since they are procedural functions and not classes.


Ground.m
nestLocation: vector2
foodSourceLocations: vector2[]
ants: Ant[]
landmarks: Landmark[]
timeLapseFactor: number
spawnFoodSource(x: number, y: number): Ground
collectFoodSource(foodSourceLocation: vector2): Ground
getLandmarksInRange(ant: Ant): Landmark[]
getNearestLandmark(ant: Ant): Landmark
isLocationAtNest(loc: vector2): bool
isLocationAtFoodSource(loc: vector2): bool
Landmark.m
location: vector2
direction: vector2
status: number
Landmark(): Landmark[]
setUp(x: number, y: number): Landmark
Ant.m
prevLocation: vector2
location: vector2
velocityVector: vector3
carryingFood: bool
walkDirectlyHome: bool
viewRange: number
viewRangeLandmarks: number
nearestLandmark: Landmark
stepsToFollow: number
globalVector: vector2
phi: number
lookingFor: string
startangle: number
maxDistance: number
lostPosition: bool
searchRadiustimer: number
TimerWError: number
livingTime: number
nearestFoodSourceLocation: vector2
Ant(): Ant[]
performStep(ground: Ground; dt: number): [Ant, Ground]
stepStraightTo(point: vector2, dt: number): Ant
takeRandomStep(dt: number): Ant
returnHomeUsingPathIntegrator(ground: Ground, dt: number): Ant
lookAround(center: vector2, dt: number): Ant
updateLocation(dt: number): Ant
updateGlobalVector(ground: Ground): Ant
updateNearestFoodSource(ground: Ground): Ant
setUp(ground: Ground): Ant

This diagram shows the classes of the project and how they are related. Since Matlab does not support information hiding the public/private information is missing in the diagram. Because Matlab uses matrices in many cases all data-types with the exceptions of bool, string, Ant, Ground and Landmark are matrices of type double[1] as entries:

- vector2: double(2,1)
- vector3: double(3,1)
- vector2[]: double(2, n)
- number: double(1,1)

Something else to be mentioned is that Matlab does not make use of pointers but instead copies instances of objects and does not alter the original. This means that in order for a method that is supposed to alter an object the original object has to be overwritten. This can be done by letting methods return the object that is containing them (this).

```
ground.spawnFoodSource(xCoord(k),yCoord(k));     % WRONG: Has no effect
ground = ground.spawnFoodSource(xCoord(k),yCoord(k));   % CORRECT: Update gets written
```

1: http://ch.mathworks.com/help/matlab/numeric-types.html
2: merci Andreas ;-)
3: das staat sicher am gliiche Ort xD

For this reason nearly every method does return the object it belongs to as a result.

## *Source Code*

To shed some light on how we implemented the model we will quickly describe the source code. The object orientated approach made it possible to implement new ideas relatively fast at the correct location.

### run.m

This file does all the parametrization and executes the setup methods of the various classes utilized in this project. It is supposed to be used as an easy way to alter the setup if the need arises.

The file contains the before mentioned  setup section, followed by the execution section where the main-loop does update the ground and the ants objects. Also inside of the main-loop one can find the time handling. At the beginning of each iteration a timestamp is saved and then used at the end of the iteration to pause the execution in order to have a uniform speed throughout the running time of the program.

The last section of the file does handle callback-functions from the user interface. The user interface in our case is the plot that opens and displays the various agents moving in the world. The user interface provides the following functions:

- **Spawn new food source:** New sources of food for the ants to collect can be spawned by simply clicking at the location in the world were the source should appear.
- **Close:** In order for the program to terminate without errors we used a callback-function from closing event of our plot-figure to terminate the program.

### distanceBetweenTwoPoints.m

A simple helper function which calculates the distance between two points.

### vector2angle.m

This helper function calculates the angle of a given vector in radians.  The value of the result ranges from 0 to 2 pi. This is important in order for the path integrator to work correctly. Additional corrections (to use the acute and not obtuse angle) are done in Ant.m.

### updateGround.m

This function does not change anything on the Ground object as its name might suggest. This method does update the plot and redraws all ants, food sources, the nest and the landmarks.

### Landmark.m

This class represents a visual landmark. It is defined as having a location, a direction vector and a status which is represented by a number. The direction vector is used by incoming ants to leave the landmark in the direction provided by the landmark. This is supposed to mimic the ants procedural memory.

1: http://ch.mathworks.com/help/matlab/numeric-types.html
2: merci Andreas ;-)
3: das staat sicher am gliiche Ort xD

## Ground.m

This class serves as container for all agents and the objects they can interact with. Besides its function as a container it also provides methods to spawn new food sources and landmarks in the world as well as various methods which can determine the proximity of a certain target.

## Ant.m

This class represents our agent and implements the mathematical model proposed by R. Wehner in his 1988 paper[2]. The model is implemented pretty straight forward with a few exceptions. Because our ants aren't real and do not have a limited field of view but can see 360 degrees we had to create a property which tells the ant if it is leaving the nest so it doesn't  immediately go back to the nest after it left it. This is due to the assumption that foraging ants that come across their nest during their searches, reset their global vector to correct errors or in other words, they will use the local vector rather than the global vector[3]. Furthermore we have a much more mathematical problem when we are near the nest. Because we divide by the length of our global vector we cause a division by zero in our update method once the global vectors length becomes zero. To avoid this we actually wait until the nest is outside of the ants view distance and set the length of the global vector to the view distance. It is reasonable to assume that the real ants do something similar because of the before mentioned local vector.

Also important is the method that takes a random step. The math behind this method is discussed in detail in another chapter here we just describe the implementation. The implementation is basically the math plus the calling of the update method to set the ants position after the calculation is done.

As described above the ants vision was simplified for the implementation and a field of view is not implemented. Once the ant is within view distance of an object (food source, landmark, nest) it steps straight towards it (local vector). The decision if the ant has reached its target is made by checking the length of the difference vector between the ant's location and the target. Once the length is smaller than a certain threshold value the ant has reached its target. Since it is possible to have many different food sources as well as landmarks we implemented methods which update the properties nearestFoodSourceLocation and nearestLandmark. Those two properties are then used for the comparison. It is also important to note that the landmarks are visible from further away than food sources or the nest.

Real ants cannot stay outside for too long because they would die in the heat of the desert. We implemented a timer that every ant updates (with a small, random error) which tells the ant when it has to return to the nest during its search for food. If an ant does not reach the nest in time it will stop moving thus being dead.

1: http://ch.mathworks.com/help/matlab/numeric-types.html
2: merci Andreas ;-)
3: das staat sicher am gliiche Ort xD