G53DIA – Designing Intelligent Agents

Coursework

Intelligent Bee

Bc. Jiří Kremser *jxk19u*

The University of Nottingham

Academic Year 2009 – 2010

Background

Nature is very inspiring for computer scientists. Many general algorithms for searching solutions in a given state space are inspired by behaviour in nature. Genetic algorithms are based on evolution and on the fitness function, simulated annealing based on the idea of controlled cooling of a metal material. In the field of multi agent systems and emergent behaviour the ants, bees, fish, birds are often given as examples of the swarm intelligence [1].

The topic of this coursework is to simulate the foraging of bees. The simulation will take place in the artificial environment provided as a Java package by Neil Madden and Julian Zappala. The goal of this paper is to give a brief outline of specification and high level design of the whole artificial system i.e. including environment and agent(s). However, the final implementation may differ from this specification and design.

Specification

Environment Properties [2]

- discrete
  - infinite grid of cells (cell types are further described below)
  - time is represented by a sequence of time steps
  - duration of a time step can be modified
  - 10,000 time steps makes one day
  - the simulation takes ten days (can be changed)
- partially observable – a bee can see only 5 cells in each direction (Manhattan Metrics)
- dynamic – yield of flowers depends on the day time
- contains eight agents – bees
  - hence **multi-agent architecture**
  - there can be many agents on one cell (no collisions)
- in each time step each agent have to perform an action (actions are further described below)
- execution of the actions is synchronous – agents are sorted and the result of an action performed by *nth* agent will be visible for the *mth* agents, where $m > n$

Agent's Actions (in general)

- affect the environment
- infallible – action cannot fail in the environment (providing the bee has enough energy)
- may have costs

Other

- simulation can be slowed down, speeded up, paused, stored and replayed (for testing and debugging purposes)
- GUI and CLI interfaces will be provided
- Java SE platform

List of Actions

Name	Cost	Precondition	Postcondition
Move	*1*	Energy >= 1	Position of bee is at an adjacent cell
Rest	*-1*	true	Energy of bee is increased by 1
Create a nest	*100*	Actual cell has to be an empty cell and energy >= 1	New nest
Lay an egg	*1*	Actual cell has to be a nest Food in nest is grater or equal to 1000.	New egg Food in nest is decreased by 1000
Deposit nectar	*x*	Actual cell has to be a nest. Bee has at least x energy.	Food in nest is increased by x. Energy of the bee is decreased by x.
Harvest	*-x*	Actual cell has to be a flower and there is more flower in the clump than harvesting bees.	Energy of the bee is increased by x. x depends on daytime and type of flower.
Communicate	*x*	Actual cell has to be occupied by at least another bee.	May change the inner representations of bees.

Communication Protocol

Communication protocol contains an abstraction of bee's waggle dance [3] (showing the direction and distance of cells with high yield). It is a trade off between physical plausibility and the omnipotent “pseudo-multi-agent” system. There are only two allowed communication actions. Both represent the sequence of waggle-dances, but the difference is that the first type is exchange of flower paces and numbers in clump whilst the second represents the exchanging of flower samples (measured in the past). The cost of a particular communication action depends on the amount of information learned (not transmited). When many bees on the same cell are watching the communicating bee, the cost is the maximum value of values representing how much each bee has learnt. The cost is bounded from below by 2 and from above by 30 or 10 depending on communication type (30 for communicating positions).

Types of Cells

On any cell there can be many bees => no collision of bees is allowed.

Type	Description
Empty	GUI visualises it as a green/white[1] place. It is possible to build a nest on empty cells.
Nest	Represents bee's hive. Eggs can be laid only here.
Flower	Visualised as a flower. Bee can land on this cell and perform the harvest action. Each cell of this type represents a clump of flowers and therefore it has a different number of flowers in the clump. The yield of a particular flower is determined by its type and time of day.

Types of Flowers

There are five types of flowers. Each type determines a function of yield. Each type has a different peak time and a peak duration. In general all these function follows a Gaussian curve and the task of the agent is not to assume these function (by hard coding them into their program), but learn them. In short the flower types with high peek of yield has low duration of its peak and vice versa.

Design

The task is to design an intelligent agents which can cooperate to achieve the common goal. The common goal is to lay as many eggs as possible in ten days. The agents will not be aware of the fact that their life is limited by ten days. In other words they will not know when the simulation ends but they will know, how long the simulation runs and what time of day is. Each agent will have its own goals which may depend on its role and the knowledge it has got.

Possible Roles (number of dedicated agents)

- **Queen (1)** – The only think which make this agent different from other is that it is the one who finds the best place for the nest. It is because the character of this decision is more centralized then distributed. After finding the best place, the queen is the bee who creates the nest, after creating the nest, there is no difference between queen and non-queen.
- **Non-queen (7)** – Common bee. If there is no nest, this agent has to provide as much information of the landscape as possible and then follow the queen to the new nest.

1 Depending on the type of viewer (*"BeeViewer" x "Meadow"*)

Exploration

Although in the later phase the agent behave autonomous, at the beginning they are fully preprogrammed to behave according the exploration plan. Each of eight bees has its own exploration plan consisting of 693 actions. The way they explore the environment can be thought as a generalization of spiral pattern for eight agents as shown at figures at appendix. There are two different exploration plans and each of them has three another symmetric variants. The density of flowers was decreased from `0.05` to `0.025`. The number represent a probability, that arbitrary cell was generated as a flower. Since the environment contains only a half amount of flowers in comparison to the default environment, the well positioned nest has significant impact on the final result[1], therefore so much exploration is done. I assumed, that the simulation starts at night (time step 0) and all flowers have zero yields by that time. If the condition on zero yields does not hold the bees may waste their time by exploration since they can not deliberate from their static preprogrammed exploration phase and find out what is better option. On the other hand they explore the area of 47,937 cells only in 693 time steps during night.

Bee's Representation

A bee will store the map of cells it has visited and “measurements”, where the measurement is the value of yield of a given type of flower in a given time. On the basis of these measurements the bee will try to approximate the real Yield Function. The bee's representations can be communicated according to communication protocol. Another data structure the bee will store is an array with time periods representing how long the bee have not been communicated to each other. If a bee has not communicated to another one for a long time, there is a higher chance it will communicate.

Bee's Selective Function

In each time step the environment requires from a bee to perform the *senseAndAct* function so that the bee have to select its action. The action should be selected in correspondence with its intentions. Beside the main goal (to lay eggs) there are another maintenance goals such as not to exhaust its energy or return to nest if energy is full. The maintenance goals have higher priority than common goals so they are hard coded in bee's selective action. If one of their precondition is satisfied, the proper action or sequence of actions (i.e. plan [4]) is chosen.

To improve performance and to be closer to human thinking the bee can plan sequences of its actions in order to achieve a goal. Every bee has its planning component which is responsible for selecting suitable actions. The plans are short sequences of actions, in terms of Java language the plan is stored as a collection of type *Queue<Action>*. In each iteration the first element of the plan is removed and used as a current action. Once the plan is empty, the planning component is asked for a new plan. An average size of a plan is only about 3 actions. Since it is impossible to deliberate from performing the plan the plans are kept small and the plans of size greater than one are used for only movement (variable size) or harvesting (size ~5).

1 The number of eggs.

Implementation

This part will describe main structure of the attached project and the key algorithms for foraging of artificial swarm of bees. The technical details about running the application or tests will no be given here since they are described in attached readme file.

Finding a place for the nest

After the exploration phase, as described below, and after communicating the results the queen has to find the best place for the future nest. Bee's method `Point bestLocationForNest(int maxWindow)` is invoked with parameter `8` which seems to be enough for flower density `0.025`. This method starts with window of size `3` and if it does not find a "good enough" solution then it will try it with incremented window until maximum window size is reached. The method evaluates all known cells by `float[] evaluateCell(Cell cell, int windowSize, int bees)` function and then returns the best point in the environment. Evaluation of cells depends on its neighbourhood which is square of cells with side of size `windowSize * 2 + 1` (17 is maximum) and on number of bees in environment. The cell gain a bonus in case there is enough flowers for all bees. The method for finding the nest takes into consideration the weights of flower types, however at the beginning these weights are set to one. There should be improvement in the future by allowing the bees to migrate to another place, when they find out that one flower type is much better in general than another.

Approximation of the Yield Function

Every time a bee perform the harvest action, it stores the yield of energy together with the day time and flower type to its samples. Each bee has a component called approximator for guessing (method `int guessYield(long timestep, int flowerType)`) the yields of given flower type in given time. It has been done by using two adjacent known samples to the given time (one after and one before) and then calculating the result as the average of the two samples respecting the time scale. (Two points create a linear function.) The detail is shown at *figure 1*. If there is only one known sample, then the result is same as the known sample. If there is not any known sample before or after given time, then two known samples after or before given time are taken, as shown at *figure 2*. Finally the result is rounded to an integer.

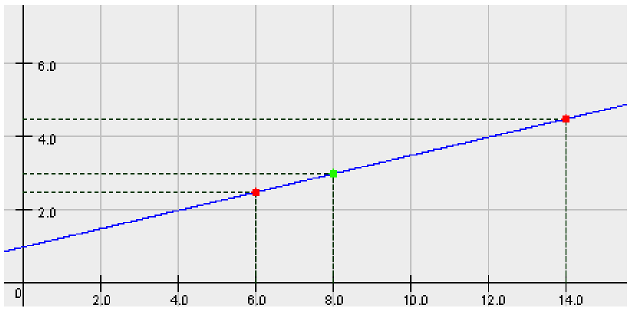


Figure 1: x axis is time step, y axis is yield. The guessed value of yield is green dot and known values are red dots.

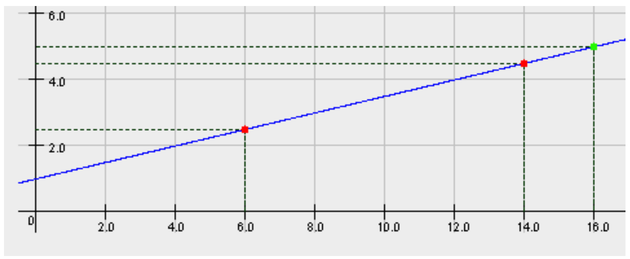


Figure 2: x axis is time step, y axis is yield. Two known samples before the given time step are taken, because no sample after given time step is known.

Finding a flower

Finding a flower is part of bee's planning component. Firstly the type of flower is determined by calling a method of class BeeYieldApproximator with signature `List<Float> guessYields(long timestep)` this method returns the list with preference weights to each flower type for given time of day (using approximator). Then the one type is chosen by using abstraction of roulette wheel for squared weights, i.e. weights are squared and the greater value a squared weight has (relatively to other), the higher degree of probability for choosing the flower type related to the weight. Thereafter method `List<Point> findClosestFlowers(Point actPos, int type, int range)` finds the closes flowers of given type in given range around the given position, flowers are evaluated by float numbers depending on their relative position to the `actPos` and the free slots[1] and then finally the flower is chosen by using "powered roulette wheel" on this weight vector again.

Communication

As described in specification there are two types of communication actions. Communicating the already seen positions and communicating the flower samples. The first type is used only on the end of exploration phase and the second type is used during the exploitation phase[2]. When on the same cell there are other bees with which the bee has not been communicated for a certain period of time, there is a chance for executing the communicating action. When harvesting a zero amount of nectar a bee can go to another bee in view to communicate. Again it is probabilistic rule..For the first three days of simulation the bees are encouraged to communicating more than for the rest of simulation.

Tests

I provided my project with a set of tests which compare my bee (MyBee) with a default bee (DemoBee). In directory *tests* there is a bash script for running a set of tests. These tests can be run also on the other platforms than Linux by running `java -jar $JAR_NAME [DemoBee/MyBee] test$i.properties`, where `$JAR_NAME` denotes the name of the jar file and `$i` an integer from one to ten.

Beside the DemoBee, another strategy can serve as a strategy for comparing the implementations of bee, lets call it a RestingBee. The RestingBee is such a bee, that it only create a nest at time step 0 (only if there is not any nest) and then performing the rest action until it reaches the maximum allowed energy, then it deposit the food to the nest. If there is enough food for laying egg, it lays an egg, this bee do not harvest nectar from flowers and it is obvious that it is an optimal strategy for meadows with no flowers. Number of eggs laid by *numberOfBees* bees during *n* days is equal to *x* in these equations[3]:

$$\lfloor x \rfloor = (numberOfBees * initialEnergy - nestCost + n * dayDuration * numberOfBees - y - z) / eggCost$$
$$y = eggCost * x / (maxEnergy - 1)$$
$$z = 2\mathrm{x}$$

For the default values and $n = 10$ $\lfloor x \rfloor \approx \lfloor 801500/1007 \rfloor = 795$

1 Number of flowers in clump minus number of bees on the flower.
2 Exploitation phase starts when nest is created.
3 *y* stands for energy "lost" by performing DepositNectar actions and *z* stands for energy "lost" by performing LayEgg actions.

I have run ten tests. The first three tests are set only to two days, the tests four up to six has duration of five days and the last four has duration set to ten days.

Results

t#	D	RS	**DemoBee eggs**				**DemoBee energy**				**MyBee eggs**				**MyBee energy**				**RB**	**Eggs ratio**
			min	max	avg	sum	min	max	avg	end	min	max	avg	sum	min	max	avg	end	egg	
1	2	1	8	11	10	**19**	2248	2405	2327	**2405**	174	194	184	**368**	427	947	687	**427**	**160**	5:100:43
2	2	2	57	59	58	**116**	2789	3150	2970	**3150**	177	207	192	**384**	212	847	530	**212**	**160**	30:100:42
3	2	3	37	37	37	**74**	2042	4441	3242	**4441**	189	211	200	**400**	123	275	199	**275**	**160**	19:100:40
4	5	4	42	45	44	**220**	1442	2896	2091	**1442**	173	191	186	**931**	181	767	498	**181**	**398**	24:100:43
5	5	5	21	24	22	**112**	2583	3913	3154	**3913**	179	199	192	**958**	551	953	695	**625**	**398**	12:100:42
6	5	6	33	40	38	**189**	1739	4155	2876	**2880**	160	195	187	**933**	377	982	638	**552**	**398**	20:100:43
7	10	7	45	60	57	**574**	1512	4605	3370	**1512**	167	199	195	**1946**	147	1382	492	**147**	**795**	30:100:41
8	10	8	8	13	10	**104**	1316	3383	2594	**3136**	99	259	192	**1917**	151	71435	7529	**326**	**795**	5:100:41
9	10	9	27	31	29	**291**	1227	3813	2756	**3813**	26	389	217	**2167**	65	173598	17742	**762**	**795**	13:100:37
10	10	-1	27	31	29	**288**	1029	3696	2311	**3391**	144	219	208	**1975**	60	23600	2759	**876**	**795**	15:100:40
Average			31	35	**33**		1793	3645	2769	3008	149	226	**195**		229	27479	3177	438		**17:100:41**

Legend:

t# – test number
D – duration, number of days
RS – random seed (-1 is randomly chosen random seed)
RB – RestingBee
eggs ratio – DemoBee sum of eggs : MyBee sum of eggs : : RB sum of eggs ratio

eggs:
min – minimum number of eggs laid per day
max – maximum number of eggs laid per day
avg – average number of eggs laid par dey
sum – number of all eggs laid during the whole simulation

energy (unused energy in the nest):
min – minimum amount of energy which is in the nest at the last time step of a day
max – maximum amount of energy which is in the nest at the last time step of a day
avg – average amount of energy which is in the nest at the last time step of a day
end – amount of energy which is in the nest at the last time step of simulation

Conclusion

My implementation is 100/17 = 5.88 times more efficient than DemoBee and 100/41 = 2.44 more efficient than RestingBee, however on the environments with flower density close to one (almost each cell is a flower) can DemoBee behave similarly or better than MyBee[1]. On the environments with flower density close to zero MyBee will be worse than RestingBee, but in general my implementation is flexible and robust to any slight changes.

1 Because of initial preprogrammed exploration.

Sources

[1] http://en.wikipedia.org/wiki/Swarm_intelligence

[2] some parameters from http://www.cs.nott.ac.uk/~bsl/G53DIA/slides/Project-Description.pdf

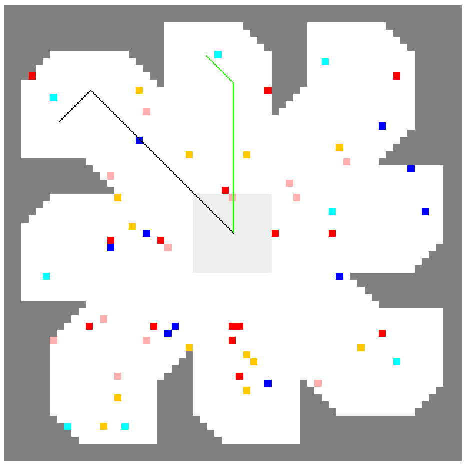
[3] http://en.wikipedia.org/wiki/Waggle_dance

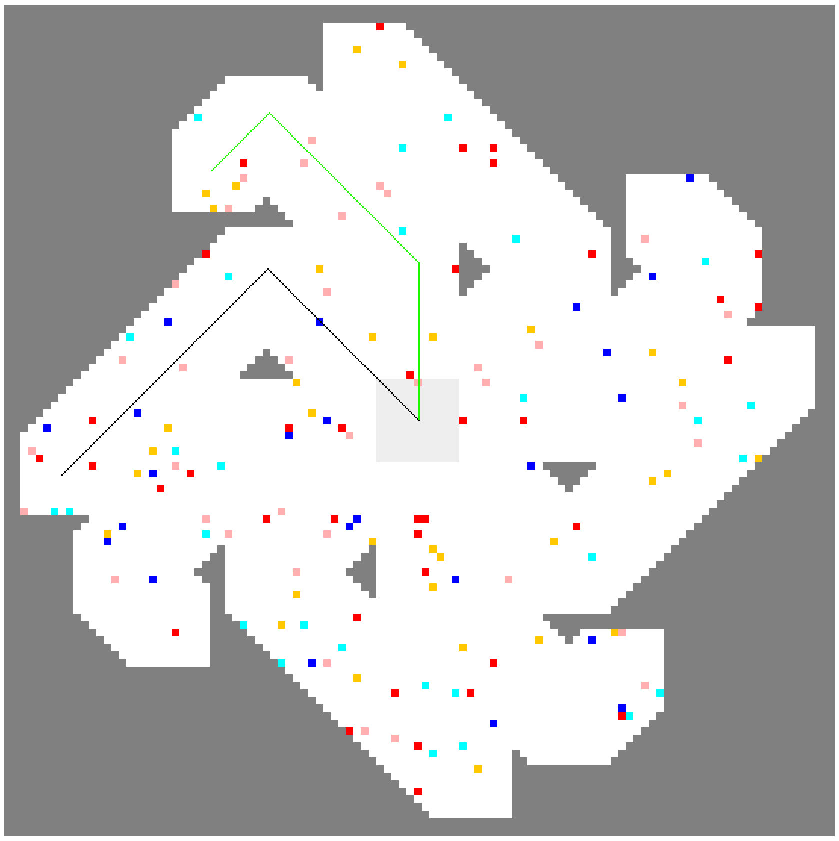
[4] idea based on the IRMA architecture http://www.cs.nott.ac.uk/~bsl/G53DIA/slides/Deliberative-Architectures-II.pdf

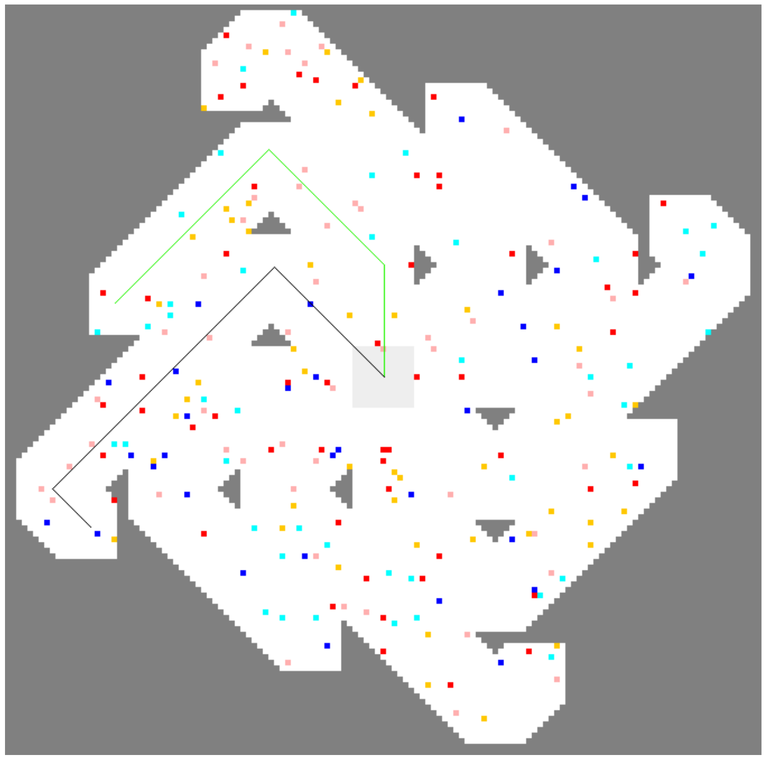
Appendix

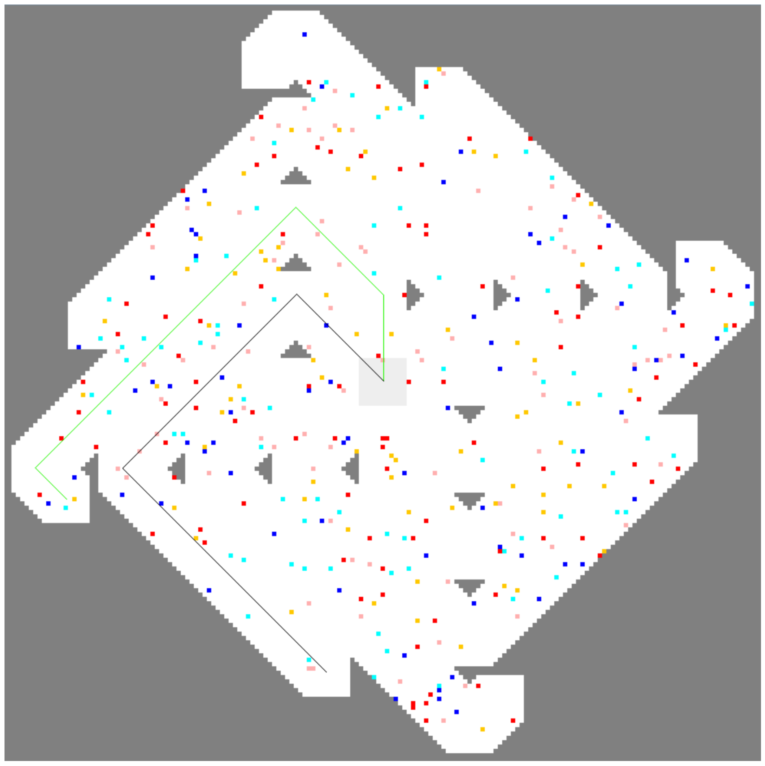
Progress of exploration of environment

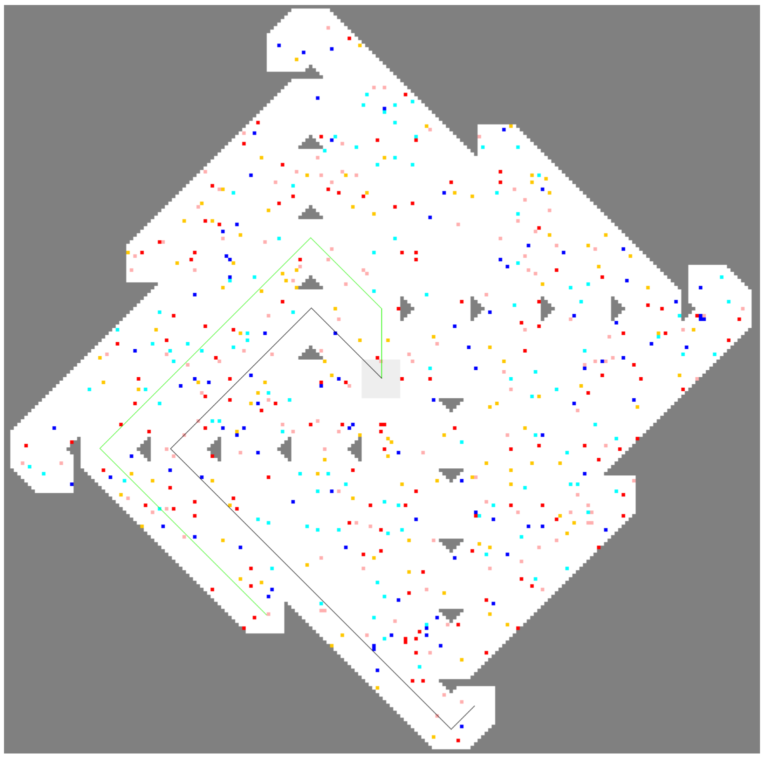
Pictures in higher resolution are attached to project documentation

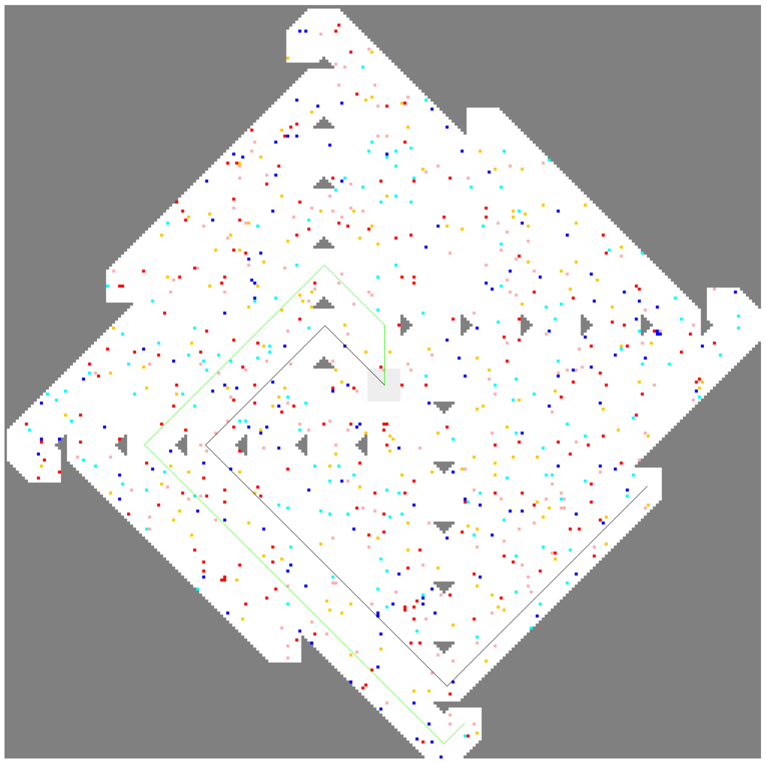

Exploration 2

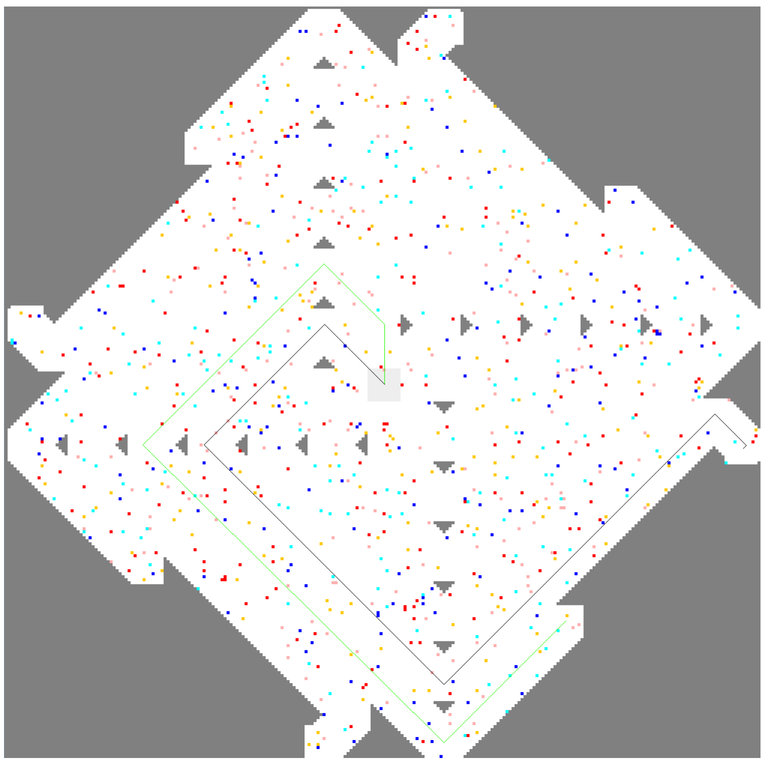

Exploration 1

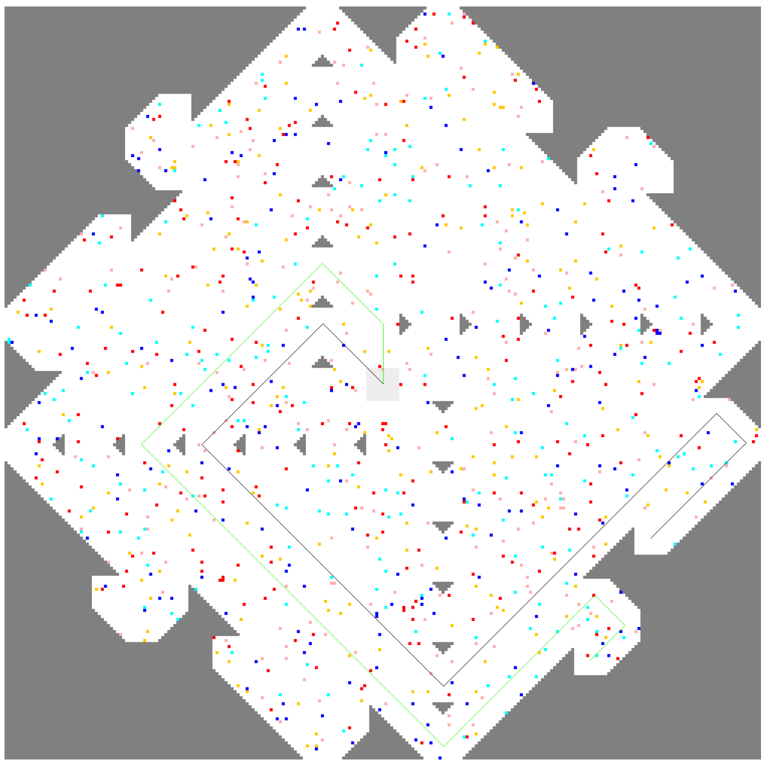

Exploration 3

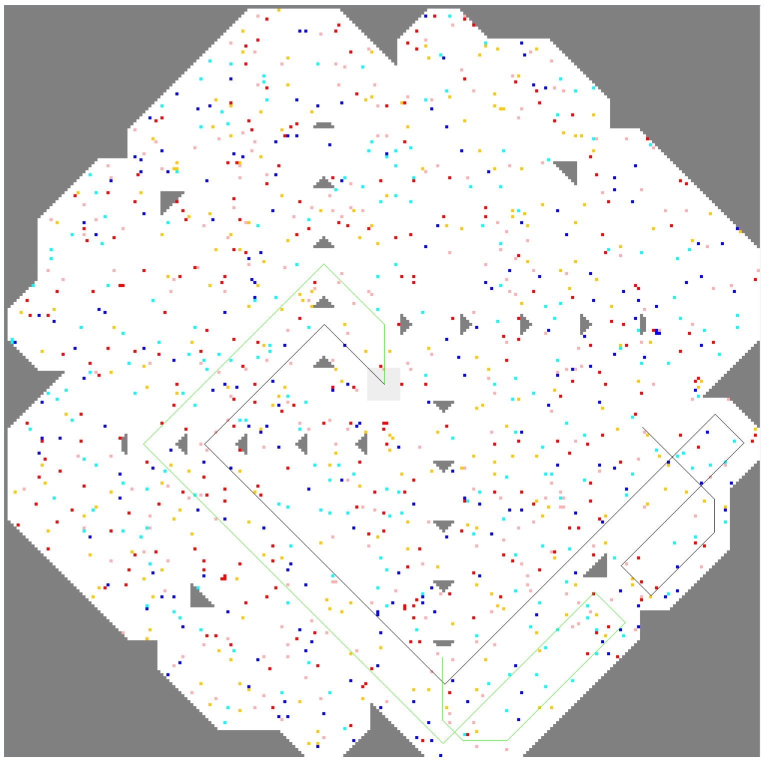

Exploration 4

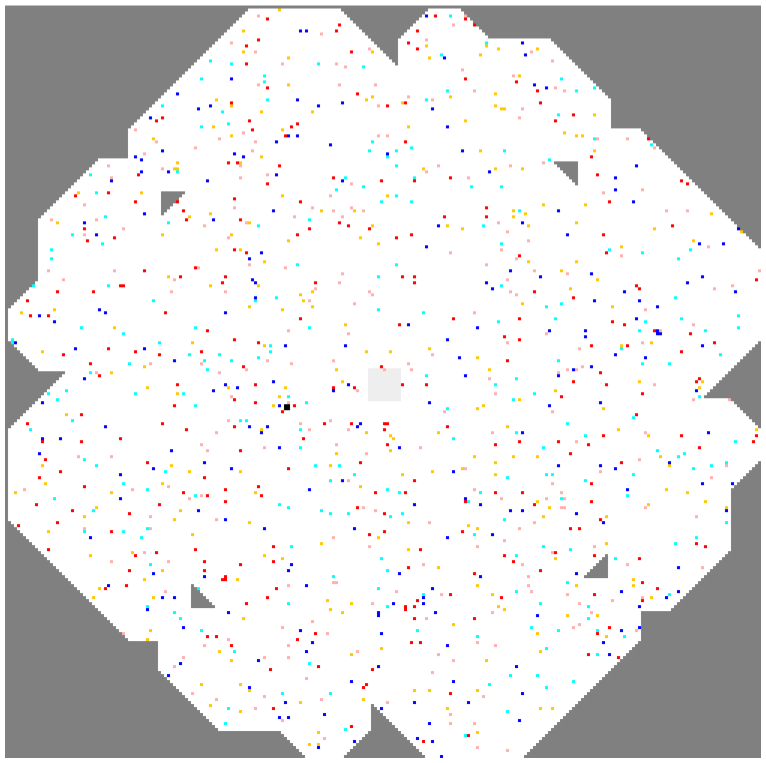

Exploration 5


Exploration 6


Exploration 7


Exploration 8


Exploration 9


Exploration 10