Faculty of Information Technology

Háskólinn á Akureyri
Upplýsingatæknideild

Final Year Dissertation 2006

Max E.Y. Ólafsson

Faculty of Information Technology

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

____
Max E.Y. Ólafsson

Abstract

A design for a simulator to test autonomous mobile robots is discussed in this work. Only a partial implementation of this simulator was realized. Most of the work for the simulation world has been completed but the area concerning the running of robot programs for use in this world has not. The simulator's intent was to model a world of obstacle, one target location and one mobile robot. The target location is supposed to represent the location of an incapacitated person at risk. A successful run of the simulation is one where a mobile robot finds its way to this target location within, at least, a reasonable amount of time and the robot using some search algorithm to reach it. A way of creating a map of this world where a user can add obstacles and one target location by using a graphical user interface has been implemented. Currently, it is possible to store these maps of differing worlds as files and later to load them from files.

Table of Contents

.

Motivation

This work has been undertaken to help develop an effective search strategy for a autonomous mobile robot with limited processing capabilities and sensors. The intent of this work was to allow a more rapid development of search algorithms for robots involved in similar types of harsh environments.

Specifically, the intent was for the simulator to enable the development of effective robot controllers that could be used in rescue operations, where a robot is deployed and has to find a target or victim in an unknown environment. It has no knowledge of its victim's location except that he/she has some beacon device emitting a source of light or signal of some kind. The robot will have to use sensors to read the light intensity in order to pin point its target –or at least make the robot able to predict a location where the reading is stronger enabling it to navigate to that location or a nearby point.

This is intended to simulate conditions whereby a person in a critical situation has nothing more but some type of beacon device. The type of robot for which this simulator was intended has little processing power or storage capacity so the mapping of its environment through its tactile sensors is rather coarse and limited. The simulated world is equally 'coarse' and simple. This is reflected in the design of the simulator that is based on modeling the world as a grid based system consisting of a collection of rectangular cells. Another simplification used is the fact that the environment remains fixed- it does not change after the robot is introduced into it.

Description of the work

Intent of the simulator

The simulator once completed was to help to develop effective search algorithms or strategies for robots that have a limited memory capacity and sensors. The robot is to find some target location or goal by using light sensor readings and following approximately a path to where the light intensity is strongest. This has been termed the localization problem, i.e. finding the goal location relative to the current position.

There are a few problems for a robot navigating in a physical environment such as tracking. In tracking the goal the robot has to factor in its uncertainties in its current position which can for instance be caused by the effectors used to move about, sliding on the contact surface or the whole robot being pushed by some object. The simulation environment developed thus far does not take this into account. This is also because modeling external environment effects on tracking is highly dependent on the type of tracking used.

As the robot navigates through its simulator world it should be building a small internal representation of the world, i.e. a model. This will ensure it does not constantly, needlessly bump into obstacles. Wherever the intensity is strongest, the robot will go in that direction. Some problems with this strategy in the real world are:

- Obstacles between robot and correct light source
- Strong light emanating from another direction
- Environmental changes have implications on mapping
- Other mobile robots

The simulator environment only takes the first of these; obstacles between the robot and target or goal into account- this means there is no light reading for the robot to read where a wall is present. It only does take the first factor partially into account- it should in fact consider the effects of obstacles on light readings. As an example any wall present between the goal and robot should weaken the light intensity on the robot's side. This problem has not been addressed.

The mapping function of the robot will help it navigate around obstacles to the light source. It is assumed for the purposes of this scenario, that places in the internal map that are mapped as obstacles do not 'disappear'. Once a robot manages to find the target location its mission will be complete and no further action has been assumed for the robot at this stage. The robot controller developed could then be tested in the real world and level of success or failure measured.

In retrospect, to track its way to the goal the robot should thus keep trying to measure in what direction light is strongest. The simulator environment would allow the robot ease the process of navigating to the correct location by mapping its environment in memory. This can be done using the same data structure the simulator world uses to model the environment.

Implementing the mapping aspect in a robot can be proven easy but its implementation hinges on the fact that there must be enough memory in the robot to store the map. The memory necessary in mapping is in direct proportion to size and the resolution of the environment the robot is situated in.

The way the simulator environment deals with this is by using rather coarse resolution of the real world since it is mapped as grids of cells but it is questionable whether the adopted strategy should not instead have been to implement two types of resolutions, one for the environment modeled by the simulator and another for the robot which would have been coarser to mimics its limited storage capacity.

Currently, the software developed assumes the robot has only one sensor giving a perfect reading of intensity at any cell location. It is thus possible with some minor coding for a robot to use this sensor to find the cell location with the strongest intensity and use it to direct the robot towards it.

Related work

Deploying a mobile robot into an unknown environment is a multi-facetted problem. Among these problems are *navigation, searching, exploration* and *localization*. Navigation for a mobile robot means that given a specified location the robot must reach some other location. Simultaneously, the robot can be exploring its surroundings and create a map of it. The robot must also be able to deduce its position on this map. There is also the physical aspect, e.g. deciding what type of locomotion to use, sensors and actuators to use there is the software part. The software will be used to design a type of robot architecture that will then be used to develop a type of control for the robot. There are several of these to choose from.

Types of architectures used for robots

One characteristic of the more successful architectures that are in use now are those that seek to combine reactive with deliberative control. These are called hybrid architectures. Reactive control is useful in circumstances that require immediate action, e.g. a mobile robot sensing it is about to drive off a platform must act quickly and make a decision to prevent it from damage. A robot capable of doing only this is rather useless hence we have the other type of control paired with the reactive called deliberative. This is to ensure that the robot is also working towards some global goal, such as navigating to some goal location. Two very common types of architectures are the Subsumption architecture [5] and the three-layer architecture [1].

The three-layer architecture is a type of hybrid architecture. It consists of a reactive, executive and deliberate layer. The deliberate layer takes care of planning actions to achieve some global goal. It does this relatively slowly. It communicates with the reactive layer through the executive layer. The reactive layer is responsible for low-level control of the robot. It must act quickly. The separation between these layers need not be clear-cut. Most modern-day robot software systems have a variant of this architecture.

The Subsumption architecture was conceived by Rodney A. Brooks of MIT Artificial Intelligence Lab in the mid to late 1980s [9]. Brookes achieved his goal of connecting sensors to actuators directly in a parallel and distributed architecture. Brooks initially set out to develop an architecture where no central brain or representation would be used and traditional notions of planning would be discarded. This architecture is built in layers or behaviours.

Each behaviour can use (subsume) the underlying behaviours. The behaviours of the system as a whole is the result of many interacting simple behaviours. Each behaviour tries to learn when it should become active by measuring when it maximizes positive feedback, minimizes negative feedback and also by measuring its effect on the global task. The higher levels build upon the lower levels to create more complex behaviours.

By using this method it would be possible to program the behaviour of a robot by choosing a set of behaviours from a library of behaviours, defining when behaviours give positive or negative feedback and make each of the behaviours learn from experience when it should become active.

The Subsumption architecture has also demonstrated robust navigation for mobile robots in dynamically changing environments. Its layered structure is easily adaptable for hardware implementation.

Simulators

Once a type of architecture has been selected the designers can then proceed to implementation of the type of control chosen. At any stage of actual robot development the program could be tested in some simulator. The testing is done in such a manner that many 'runs' are performed with the virtual robot situated in different simulated worlds. Often the reason for doing this is that sometimes it is too expensive or impractical to execute hundreds of testing operations with a real robot. If the simulator is good enough it will reveal the effectiveness of the program as well as show its performance under special circumstances or unexpected situations. Still, even if testing might reveal some flaws in the program there could still be other faults in the program present and to this there is no solution.

Robot mapping

Robots situated in any unknown environment must deal with exploration and localization. The task of exploration is to create a complete map of an unknown environment. The problem of *robotic mapping* is for an autonomous robot to be able to construct a map and to localize itself in it. The popular term used for mobile robots involved in performing localization and mapping, has been SLAM problem-*simultaneous localization and mapping problem* [1]. This is in fact one of the fundamental problems in mobile robotics. Since the 1980s and early 1990s this field has been divided into metric and topological approaches [4].

There are essentially two kinds of maps used, metric maps and topological maps. Metric maps capture the geometric properties of an environment while topological maps describe the connectivity of different places. An early example of the former approach was Elfes' and Moravec's important *occupancy grid mapping algorithm* [6,7,8], which represents maps by fine-grained grids that model the occupied and free space of the environment.

This approach has been used in a great number of robotic systems, such as [8, 9, 10, 42, 83, 98, 106,107]. An alternative metric mapping algorithm can also be used by using sets of polyhedra to describe the geometry of environments.

Many techniques have been developed aimed at mapping an environment that is static, structured and of limited size. Mapping an unstructured, dynamic environment complicates the mapping procedure even further. In fact, this is an area where there has relatively been little research made.

Example of rover navigation in practice

Navigation for a mobile robot means that given a specified location the robot must reach some other location. On the surface of the planet Mars there are currently 2 robotic rovers which are part of NASA's (**National Aeronautics and Space Administration)** Mars Exploration Rover (MER) Mission [2]. A glimpse at their system of navigation can prove useful. Their system of navigation allows them to navigate safely through unknown and potentially hazardous terrain.

They are equipped with a set of instruments to collect data. Sequences of commands are sent once per Martian solar day specifying what data to collect and destinations. At the end of each Martian solar day the rovers send back collected data to help human operators on earth plan the next sequence for the following solar day.

The rovers are equipped with two cameras mounted at the front of the rovers, i.e. the rovers have stereo vision. Two images, one from the left camera and another from the right camera are processed by an algorithm to derive a three dimensional map of the area. The images are manipulated in various ways, e.g. resolution is reduced and filters applied. These operations in turn helps to reduce the computation required by the rover to map the world and thus allowing it to drive safely at faster speeds.

When the operators- which are located on earth- want to send a rover to a particular location or waypoint it must first be processed by set of software routines onboard the rover called *Grid-based Estimation of Surface Traversability Applied to Local Terrain,* or for short GESTALT. Given a goal location, rover's current position and state of the world, this system determines in what direction is best for the rover to move next. It achieves this with the help of a stored *local map* of the area surrounding the rover.

This map is necessary to navigate effectively since the rover's cameras have a restricted field of view and unexpected events such as stumbling upon an obstacle might force it into an unseen area and potentially dangerous situation. This map that is part of the GESTALT system models the world as a grid of regularly spaced cells, each being equal to the size of a rover's wheel. Each cell has related properties, e.g. whether it is part of an obstacle or unknown that the GESTALT system uses when plotting a course.

Once the rover has chosen a direction it will drive blindly forwards for distance of about 35 cm. It is interesting to note that during this distance a rover will not be scanning with its cameras for obstacles ahead. If however, any obstacle is present other instruments of the rover such as the tilt sensor will pick this up. At this point the stereo vision system can be applied autonomously to discover an obstacle(s). Accordingly then this obstacle will be marked in its internal map and another direction will be chosen to circumvent this obstacle. Even in the absence of any obstacle, the rover is not required to move exactly according to initial intent. The rover's new position and orientation is inferred from its instruments, wheel odometry and its prior position, thus using a form of *deduced reckoning*.

Status of project

In what has been created thus far nothing has been borrowed from other types of simulators. What is different with the intended simulators from others is the fact that this one was intended for robots with limited power, especially in terms of memory. The design decisions were taken with this in mind, hence a grid based world. This means the simulator environment consists of grids. This also had the added effect of simplifying the simulator's implementation.

Each grid in this world can correspond to three states. An empty grid means the area is free of any obstacles so the robot can travel through it. A grid filled with a particular colour corresponds to an obstacle so the robot can only collide with it but not move through it. One grid cell, and always only one, in any simulated world has a special value attached to it denoting it as a target-actually a value of zero. If the simulated robot reaches it, then the simulation should end.

The simulator environment will allow the user to create grid-based worlds of varying size, from a 10x10 grid of cells to about 100x100 grid of cells. These can then be saved to a file and loaded later. If the user loads a grid from a file of different size than that currently selected in the simulator, it will re-adjust automatically; this is to keep the user from any guesswork. The grids are stored as files of integers.

Currently the simulator can be used to simulate robots using an approach based on Brook's subsumption architecture or in the least, with that intent. After a map of the world has been created and loaded into the simulator we can run the simulator to see the effectiveness of using this behaviour based strategy. To judge performance we used the time factor, as this is the single most important factor in search and rescue. To measure this, the simulator also has a timer and stops once the robot reaches its target destination.

In order to achieve this a set of three behaviour classes were created. These are named 'drive behaviour', 'light seeking behaviour' and 'collision behaviour'. These behaviours are part of the robot. A simulated robot has these three behaviours and during each instant of time in a simulation run only one of them is active. There is an arbitrating object that is part of the simulator that determines what behaviour should be activated. Once this has been done the relevant behaviours' actions are run and shown in the animation part of the simulator.

In order to simulate this, an arbitrating object was designed that cycles and checks what behaviour should become active. The behaviours are arranged in a priority order. Behaviours also have some set of conditions that need to be satisfied in order for them to become active. The 'drive behaviour' is the topmost behaviour and this the arbitrating object will always try to run before the other two behaviours. The light-seeking behaviour in a similar manner has precedence over the collision behaviour and this, the arbitrating object will try to run before it.

Both the 'drive behaviour' and the 'collision behaviour' have a timer variable telling them for how many simulator time steps to drive forwards when they have been activated. This variable is simply an integer that is set to some multiple of simulator steps and hard coded in each behaviour. The arbitration process will wait for this amount of time to expire before considering another behaviour, unless the preconditions for the same behaviour should be satisfied again. In that case the behaviour's actions should be re-executed from the beginning. This is not what happens though in case of the collision behaviour. Currently it executes its actions entirely before re-execution. This is a flaw discovered later, it will be discussed in the design section.

The software framework strives to allow easy extensibility for creating further behaviours. The timer variable can be tuned in the behaviours without too much difficulty. These determine for instance for how long a robot should drive in a particular direction after a collision has occurred in the collision behaviour, or for how long to drive blindly forwards in the drive behaviour. Further variables could be created in each behaviour to remember the state of the robot's speed or heading prior to the behaviour's execution. This could be used should the user wish to reset the robot to these values once the robot finishes executing this behaviour's actions.

Another incomplete feature added to this simulator is some sort of noise generator. This was supposed to mimic the weakening of the signal due to obstacles in its path. A simplistic attempt at recreating this effect was made. The way this was done is that when a cell was selected to contain an obstacle instead of a light reading, any surrounding cells with a weaker light reading were weakened even more to simulate the obstacles' weakening effect of the signal. This does not give a correct approximation when obstacles have other obstacles between them and the light source. A way to improve upon this feature this would be to imagine a ray drawn from the obstacle to the target and determine if any other obstacle cells are in its path. This and probably some other heuristic or techniques could be used to improve this feature.

Another addition that was not completed in time is the implementation of a feature that changes the simulator world as time goes by, i.e. some obstacles could disappear or appear elsewhere with time. This feature was implemented by simply creating a thread that selects any cell at random, then gives it a 10 % chance of being changed either into an obstacle or free space. Currently the thread runs every 500 ms and the chance of selection is 10% but these can be changed easily. This was done with the intention of simulating search and rescue operations more closely as the state of the world is often dynamic in those cases. In the case of simulating fires in buildings this thread could probably be programmed to run more often and with a higher chance with the passage of time.

At the end or during a simulation run we can ask to the program to plot the path taken by the robot. This can sometimes help interpret the behaviour of the search strategies employed.

During a simulation run the robot typically either bumping or sliding on obstacles, sensing and turning towards a light source, or simply driving straight. Some things were done to add some level of realism to this simulator and others ignored. For instance modelling the turning of a robot. As a real world robot cannot turn instantaneously a function was devised to give the amount of turn the robot can achieve during the smallest unit of simulator time.

This modelling of the turn bear some resemblance to how the movement forward of the robot is modelled, given its velocity it is multiplied with the simulator's smallest measured unit of time-time step, and this gives the maximum possible magnitude of displacement during one simulation step.

Some improvements were discovered as implementation went on, e.g. that there is probably a way of adding more realism to the sensor readings. In the current case the robot supposed sensor is omni-directional and pretty much senses the reading at each point as if it were actually located there.

After completing this simulator a series of tests were conducted to judge the effectiveness of three search algorithms. An assumption made during the tests was that the robot's sensor used in the 'light-seeking behaviour' is only able to sense strength of the target's signal in the 8 cells centered around the robot. A side effect of this strategy is that the robot can become stuck on plateaus where there is a local maximum of signal strength. Later, other defects were discovered and are discussed under the following design section.

3 Design

This design of this software reflects how the object-oriented approach is used. Since the simulator in brief really consists of the simulator world, simulator, a robot and animation part. All these are separate classes that communicate in programming code. The following is a description of these classes.

3.1 Mapping

The world is modeled as a rectangular area of grid cells that have only one attribute. This attribute indicates whether occupied space is empty or contains an obstacle. This bears some similarities to Elfes' and Moravec's *occupancy grid mapping algorithm* [6,7,8], which represents maps by fine-grained grids that model the occupied and free space of the environment.

The size of these grid cells was chosen so that the robot covers four grid cells at any time. This decision was made to mimic the real world more closely, as it could imitate it more closely by increasing the number of cells. Each of these cells has a corresponding sensor reading. This does seem a reasonable thing to do in view of the GESTALT system which models the world as a grid of cells, being equal to the size of a rover's wheel.

The strongest value of the sensor reading is at the target and is denoted with a value of zero. Other sensor readings for cells are calculated by using the radial distance to the target's position and decaying the sensor readings for each grid cell in a radial manner using the inverse square law. This technique imitates a source quite well. Cells that contain obstacles are a special case and contain no light sensor reading and this is denoted by using a special marker value, i.e. '-2' for its reading. The architecture could easily allow robots with different sensor ranges by reading the values of all surrounding cells within reach of the sensor.

The user can create maps of this nature to test the effectiveness of search algorithm. This is done with a graphical interface where the user must select, the target's position and select cells where obstacles should be present. After this has been done it is then possible to store this map of a world by storing it digitally or loading it directly into the simulator. The user has several options when editing, he/she can erase many cells or reposition the target at any time. The size of a map can be adjusted from a size 10x10 to 100x100.

To model the real world more precisely it would have been better to factor in the effects of obstacles on the sensor readings. One way of doing this is to add noise around those cells that surround obstacles. This feature was not completed though.

Another factor not considered in the design is to allow for a dynamic state of the world. The state of a burning building is fairly dynamic. A function to alter state of cells, both containing obstacles or free space was designed and run as a thread and it showed that it worked but this feature was not completed either and therefore was not tested either.

This feature was worked on as in such a dynamic scenario, some things in the simulated world would have to become obstacles and vice versa. Burning furniture could either fall on the platform in places, which the robot had crossed previously, and form insurmountable obstacles for the robot or burn out of its way thus forming a new path for it. Judging the effectiveness of search algorithms would though have been more difficult if many different types of dynamic conditions were simulated in different world simulations.

3.2 Simulator

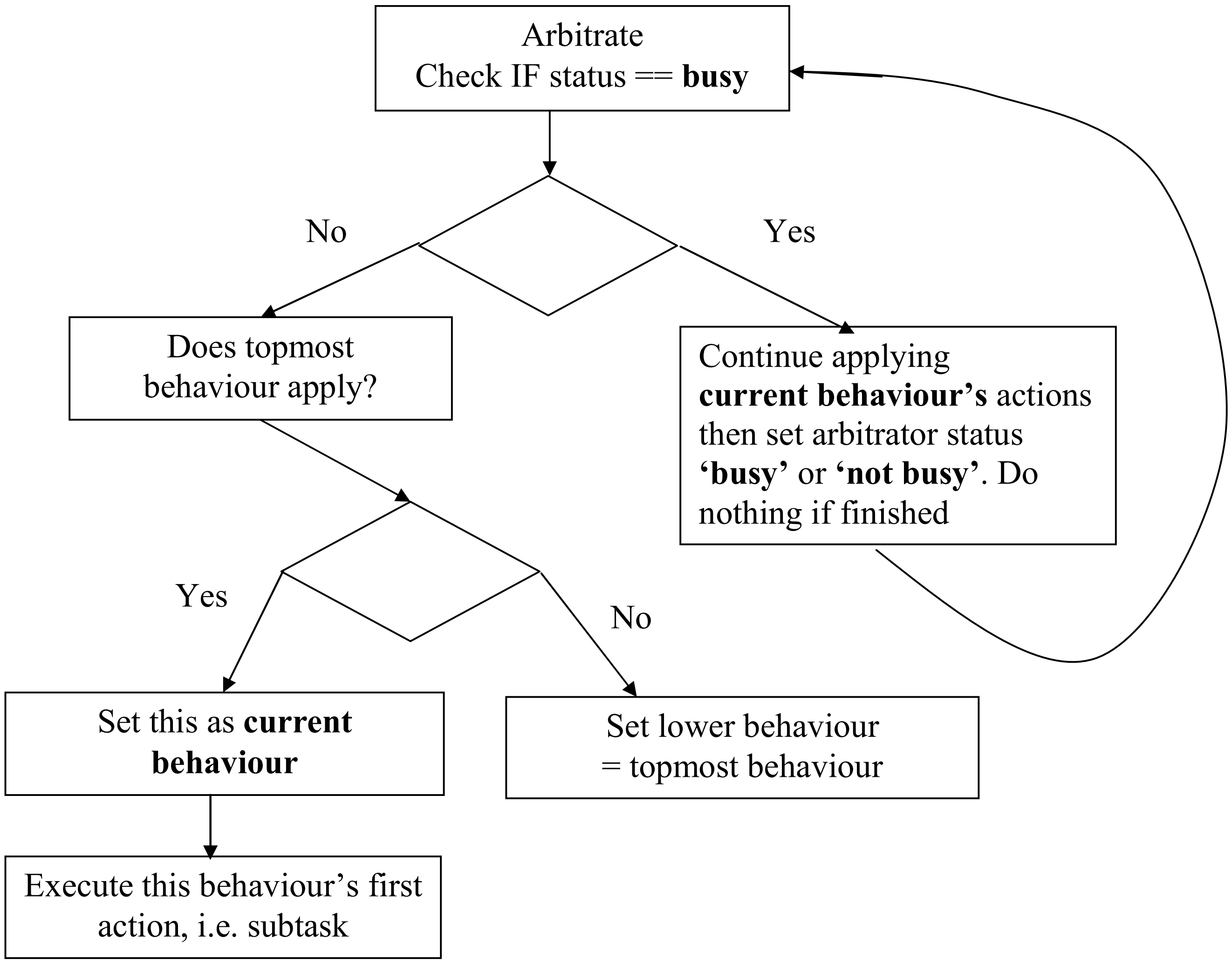
This part allows maps created from the mapping part of the software to be loaded in and subsequently a robot to be loaded and tested on the respective map. Currently the only way to configure the robot is through direct coding. The simulator has a timer to give some sort of measure on the performance of robots.

The intent of the simulator was to model the performance of robots using a behaviour based approach in a simple world of obstacles and a target for the robot to reach. To simulate such a thing did not prove easy. The simulator is passed a robot object through software. This robot has a set of behaviours coupled to it. The simulator also contains an arbitrating object that checks which behaviour of the robot it should activate. This is the most complicated and difficult part of the software to design.

From the literature studied it seems the arbitrator should select a behaviour by checking a Boolean function every behaviour contains. This function returns a value of true when it is appropriate to apply the behaviour but false otherwise. The function contains a set a preconditions that must be satisfied in order for the behaviour to become relevant. The preconditions vary for behaviours.

The behaviours are arranged in priority so if many behaviours apply at one time only the one with highest priority is executed. This mean that during simulation the arbitrating object continually checks this function for each behaviour starting from the behaviour with highest priority. A common approach is to put the behaviours into an array and implement a loop to do this type of checking in code.

Below is a flowchart that describes this sequence of events.


Arbitrate
Check IF status == **busy**
No
Yes
Does topmost behaviour apply?
Continue applying **current behaviour's** actions then set arbitrator status **'busy'** or **'not busy'**. Do nothing if finished
Yes
No
Set this as **current behaviour**
Set lower behaviour = topmost behaviour
Execute this behaviour's first action, i.e. subtask

When the arbitration process finds the first behaviour it can apply, it executes the behaviour's action. The main problem in implementing this idea in code was the action part. A behaviour can last for several simulation time steps but the arbitration process runs every simulation time step -but only one behaviour is applied at any one time. Since a behaviour's actions are not atomic in this case and must be divided between time steps this complicates things quite. The robot must somehow have a memory of what point it is in applying the behaviour.

Consider the collision routine. If we say the robot must back up after a collision and turn 90°. This means it has to back up for several time steps, and then turn also for some amount of time steps. To try to resolve this issue a few variables were set in each behaviour and robot class to keep some sort of a memory of the behaviour's state. There are timer variables associtated with each behaviour which are integers that signify a certain number of simulation time steps. Another variable is part of the robot's class and denotes the heading the robot is trying to reach, i.e. target heading which is measured in radians.

When a behaviour required backing up, or advancing, a timer was set that expired when the robot had driven the amount required forwards or backwards. Should the behaviour then also request that the robot turn some set amount a target heading was set for the robot and its current heading updated to bring it closer to the target heading during each simulation step. Using this combination of timer and setting a variable for the heading seemed to be the obvious approach.

Another problem with implementing this idea is that the arbitration process should allow the behaviour to execute its actions before applying another behaviour. To remedy this the behaviours were allowed to tell the arbitration process to stop while they were busy executing. When the behaviour's action had been fully applied they told the arbitration process to resume. In this way some control is handed over to the behaviours, which maybe goes against the idea of the arbitrator having total control from the literature in behaviour based robotics.

This problem stems from the fact that whenever a behaviour becomes eligible for execution during the next cycle or time step the arbitration process could find it valid for execution and restart this behaviour's actions from the beginning. Either that or select some other behaviour. Allowing this to happen would mean the simulator would never run a behaviour's actions fully but only a combination of starting actions from a set of behaviours. How to circumvent this issue differently was not discovered.

This way of tackling the problem was used but has a flaw. What if the robot collides with an obstacle and then again shortly afterwards? In the better scenario it would start its collision behaviour from the beginning after the second collision but this is not the way it works currently. As it goes now it completes the collision behaviour resulting from the first collision and then restarts after the second collision. Still, this is in order, in case of the light seeking behaviour, which is simple behaviour and does not have different states it can be in. In this case it is okay to re-start this behaviour because each activation seeks to bring the robots heading into the same direction.

The simulator provides collision detection with the following technique. Whenever the robot wishes to move in some direction the simulator first checks ahead the immediate cells surrounding the robot- if the area of the robot *would* intersect with the area of an obstacle. The only difference here with other techniques of collision detection is that usually colliding objects are approximated by imaginary rectangles. Should these areas intersect a collision has occurred and the simulator will not allow the robot to go ahead.

3.3 Robot

The robot class contains all the properties that are associated with a robot object in the simulator. These are size of the robot, speed, position and heading. During each simulation step the robot object asks for its position to be updated based on the location it would have after another simulation step. The simulator then updates the robot's new position based on these factors. If this next location coincides with an obstacle the simulator will bring it as close to it as it can get and also allowing the robot to slide along the surface.

To run the robot with the behaviour based approach a set a of behaviour are attached to a robot created from the robot class. These behaviours can then access any robot's variable and change them if need arises. In case of a collision the collision behaviour sets a robot speed to zero and then changes its heading over the course of some simulation steps. These behaviours must all implement a few function specified in the behaviours interface. Among these are an action function which specifies the action taken by the behaviour and the function specifying the preconditions that must be satisfied in order for the behaviour to be activated.

3.4 Animation

At the same time as updating the robot's position the simulator calls the animation class to reflect the robot's current position onscreen. Aside from showing the map of the world and location of the robot, the heading, simulation time and whether robot has collided is also shown. The animation part is a separate class that can be decoupled from the simulator run without any effect to it.

Implementation

To model this simulator the Java programming language was chosen. Because of the object-oriented nature the software was the Java programming language was used. The software was tested and implemented on a computer using Windows XP Pro.

A lot of time was devoted to reading about and getting to know the details of animation techniques in Java. Concepts such as image buffering were studied on the Java site; http://www.sun.com/java and examples thereof looked at. The simulator part of the software contains a timed task that updates and queries the robot object for its status and draws a visual representation of the robot according to this and the simulator world's constraints. Quite a lot of good functionality is provided by the Swing framework in Java.

One concern was that the processing speed might be too slow because of time taken by the animation part. There were no symptoms of overload on the system though, except for when using the trail option of the software. This is understandable since the trail function stores the number of points visited by the robot in list type of structure and typically when the simulation runs for a long time the number of points becomes too big and the list too. This could be avoided by storing only points that differ substantially in position.

Problems encountered were mostly relating to keeping track of objects using the co-ordinate system and trigonometry functions. Another point worth noting is that some thing in relation to this project such a collision detection could have been easier to implement if some reading on computer gaming concepts and web sites had been done.

Regarding the collision detection scheme, Java actually provides Boolean operators to tell if simple two-dimensional geometric shapes intersect such as ovals and rectangles and this was made use of in the partial simulation software. There seems however to be a problem with the precision of these when movement of the robots is less than about 2 pixels during each simulation step.

The arbitration aspect of the software was implemented by using a special arbitrating class the check cycles thorugh an array of behaviours in a loop. Every behaviour has Boolean function the arbitrator checks and when this returns a value of 'true' the actions associated with the behaviour are run. This is not implemented in a seperate thread, there is no need for that.

5 Evaluation

Following implementation a series of tests were conducted. These were to find out the effectiveness of three different simple search algorithms. During the tests the variables in each behaviour were kept constant except for the collision behaviour's turn angle telling the robot how much to turn after a collision had occurred with an obstacle.

The tests consisted of ten different configurations of simulated worlds. The three algorithms were run at least once on every world and the results recorded. The ten different configurations were created with the intent of testing the algorithms in worlds of varying symmetry and chaos. What they all have in common is the robot's start position. During each simulation the robot start's from the upper left corner of the simulated world situated in the lower right corner, or as in the last case, center of the world.

The performance of each algorithm is judged by the time needed for the robot to reach its target. Another parameter that might be useful in judging performance is the area covered by the robot in its search but this is less important than the time taken or whether the target is reached. Quite possibly this would matter in a multi robot environment where agent collaborate and communicate the areas that have been searched, to home in on the signal.

Specifically, the algorithms are:

1. Turn 90° left in case of collision and then advance.
2. Turn 90° right in case of collision and then advance.
3. Turn random amount of degrees in case of a collision and then advance.

The first algorithm tells that upon encountering a collision with an obstacle the robot should turn right 90 degrees and then advance for a fixed amount of time. If an obstacle is encountered again before this fixed amount of time, which is embedded in the collision behaviour has passed, the arbitrating object will re-initalize the collision behaviour from its beginning.

The second algorithm does the same except it makes a turn to the left. The third algorithm uses random amount of turn before advancing each time the robot encounters an obstacle.

Other kinds of tests were conducted such as collision detection tests, and some cases of failure had been detected. What this mean is that the simulated robot has succeeded in travelling through grid cells in the environment marked as obstacles but this is thought to be either part of the nature of the programming language or bug in the program, this is discussed further under the design section.

As mentioned before the robot´s modeled sensor was only ably to sense the strength of signal readings in the cells surrounding it. This appoach can lead the robot to a plateau if there is some noise interfering the signal such as a wall. In fact, it was discovered that especially if any obstacles were in the shape of walls with sides leading away from the target, like some concave shape, the robot could become stuck for a very long time if not ad infinitum.

If the sensor were modified to sense the signal strength at greater distance we would still face this problem. The testing was not done with any noise so this was not a factor that had an effect on the tests.

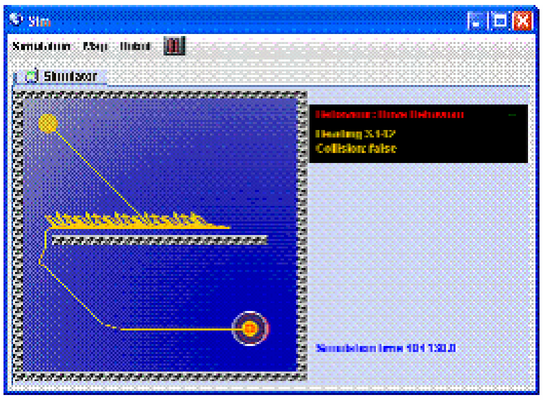
Below the results from the test are discussed. Next to the discussion of each algorithm is a picture of the simulation end state-when applicable- showing the robot at its target location along with a trail showing the path taken by the robot as simulation progressed. Following the description of these 30 tests is a conclusion section with a summary of the results. More accurate pictures of the relevant configurations are located in appendix C.

5.1 Results

Configuration 1- Results

Algorithm 1 - Simulation end time: 104.130

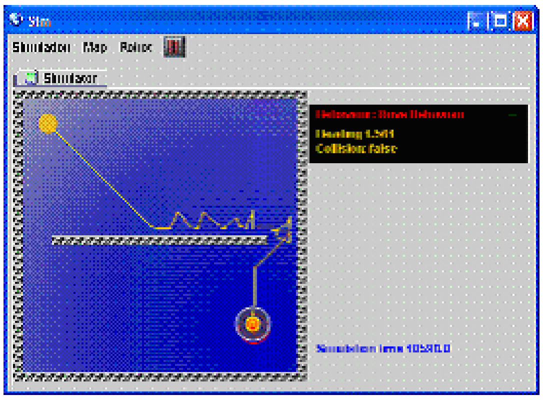
As can be discerned from the picture the simulated robot manages to find its way in the end to its target. In the beginning, it sets out toward the source but the reaches the wall when a series of collision-drive-seeking behaviours each take their turn being activated. It was presumed the robot should reach its target from the left side instead of the right side of the world. This is however not the case because the collision behaviour adds an amount of displacement to the robot that is not small when compared to the displacement added by the drive behaviour. It seems the collision behaviour and it preference for turning right determines what way around this obstacle the robot will go.



[illegible]
Collision: false
[illegible]

Algorithm 2 - Simulation end time: 10.590

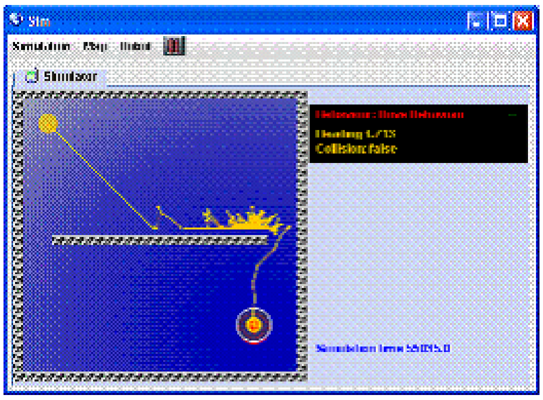
Although this case might seem similar to the above, there are some differences. Firstly, the number of collisions with this wall-like obstacle is reduced and the time taken is much smaller. Since the collision behaviour tends for the robot to turn right and its target is also to the right the seeking and collision behaviours work together instead of against each other as in the previous case. The robot stops just a little distance from the wall enough to avoid a collision to initiate its signal seeking behaviour. A direction almost parallel to the wall is given but then it collides with the wall and the collision behaviour drives the robot to the right around the obstacle and finally shortly thereafter the robot reaches its destination.



Picture of algorithm 2 run

Algorithm 3 - Simulation end time: 55.035

In this case the robot proceeds to the wall like obstacle as before. It collides several times with it as before, each time choosing a random direction to turn to after a collision. The small random turns do contribute little change to the state with time. The robot's progression to the right and not to the left can be attributed to its light-seeking behaviour. The simulation time is better than for algorithm1 but worse than for algorithm 2.

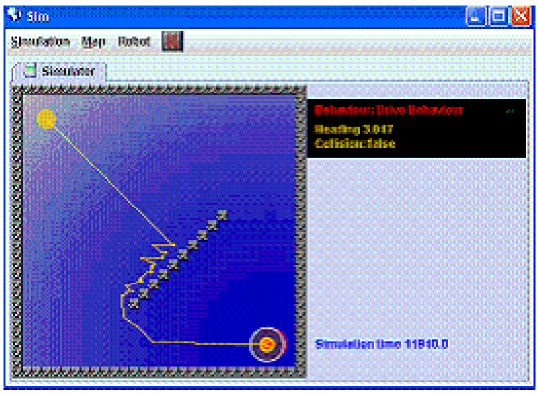


Picture of algorithm 3 run

Configuration 2- Results

Algorithm 1- Simulation end time: 11.940

This time the first algorithm enables the robot to find its target quicker than in the first configuration. Again it seems the collision routine's preference for the right turn and layout of the world are the decisive factors. This is easy to see by how this goes. The robot sets off then collides with the wall. A series of collision take place and robot reacts accordingly but as the robot progresses along the wall it travels gradually to cells with a weaker signal reading until finally the robot reaches a point beyond the wall. Thus the collision behaviour dictates its direction along the wall but then the seeking and drive behaviours take over once the robot is positioned beyond the obstacle.

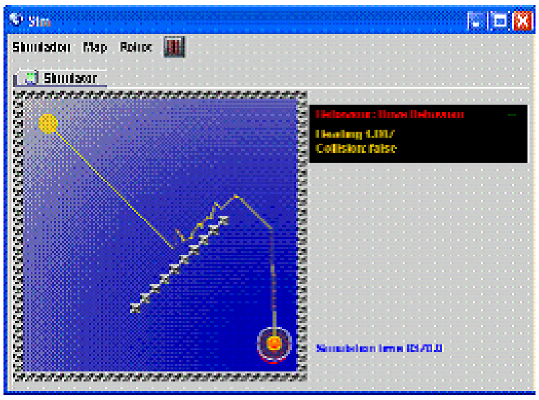


Behaviour: Drive Behaviour
Heading 3.047
Collision:false
Simulation time 11840.0

Picture of algorithm 1 run

Algorithm 2 - Simulation end time: 8.370

This case is very similar and symmetrical to the previous case and similar arguments apply.

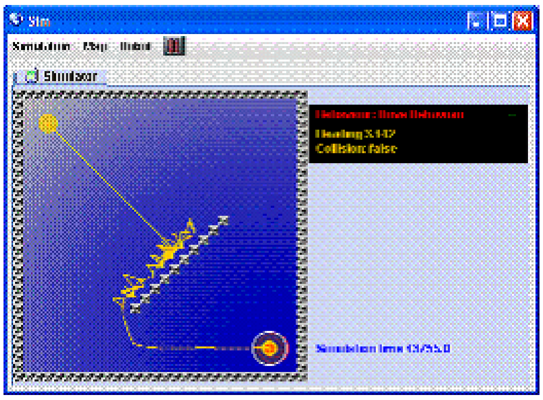


[illegible]
Heading [illegible]
Collision: false
Simulation time [illegible]

Picture of algorithm 2 run

Algorithm 3 - Simulation end time: 43.755

Still performing worse than the other two algorithms here. Seems the algorithm is performing poorly with a symmetrical shape.



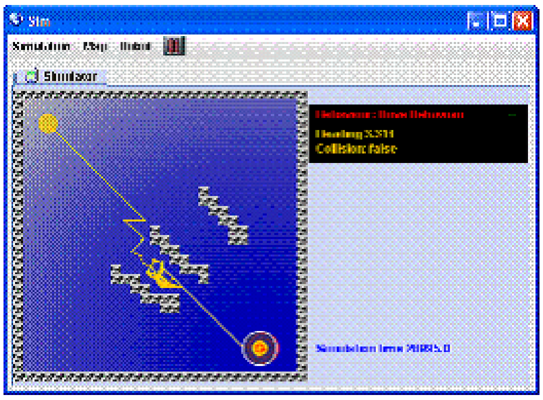
[illegible]
Heading [illegible]
Collision: false
Simulation time [illegible]

Picture of algorithm 3 run

Configuration 3- Results

Algorithm 1 - Simulation end time: 28.995

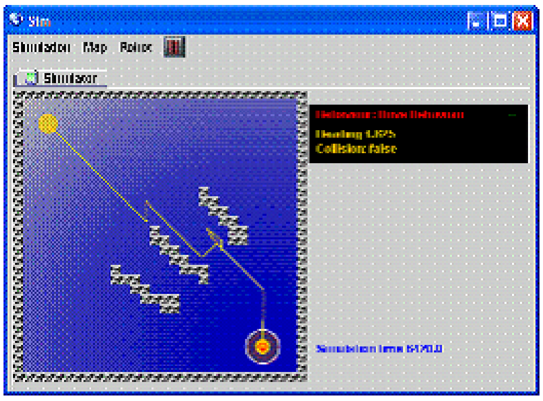
In this configuration the robots sets off, then collides with the obstacle in the middle, and gets 'trapped' for a while in between the lower obstacles. Bouncing back and forth, sliding along the obstacles but in the end the robot manages to trigger its signal-seeking behaviour when it is almost in the middle of the two obstacles and then a clear path is taken up to the target.



Picture of algorithm 1 run

Algorithm 2 - Simulation end time: 6.390

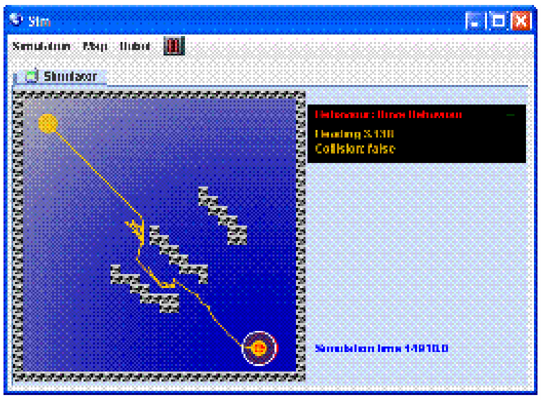
Unlike the previous test case the robot, this time goes to the right between the obstacles and since there is some more distance between these structures avoids wasting too much time in a bounce back and forth between structures.



Picture of algorithm 2 run

Algorithm 3 - Simulation end time: 14.910

During this run the algorithm did not perform badly but this is probably due to relatively few collisions before reaching a rather empty path to the target where the seeking behaviour takes charge.



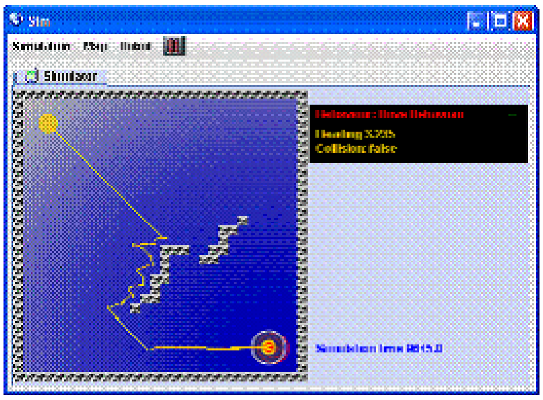
Behaviour: [illegible]
Heading 3.138
Collision: false
Simulation time 14910.0

Picture of algorithm 3 run

Configuration 4- Results

Algorithm 1 - Simulation end time: 9.645

The behaviour here is very similar to that of configuration 2, even the time is similar. Once again -as was the case in that scenario- the collision behaviour is a dominating behaviour once the robot reaches the obstacle to the left. Once it is no longer in the way the robot proceeds via its light signal seeking and driving behaviour.

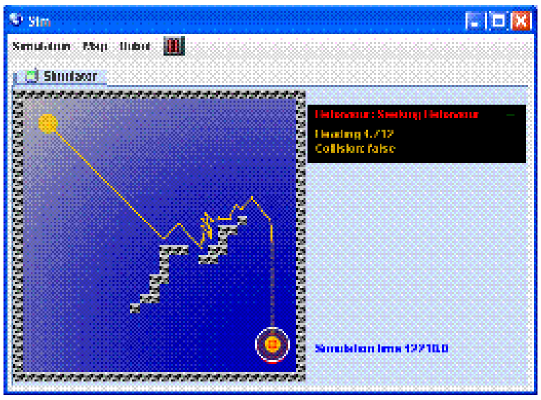


Behaviour: [illegible]
Heading 3.295
Collision: false
Simulation time 9645.0

Picture of algorithm 1 run

Algorithm 2 - Simulation end time: 12.210

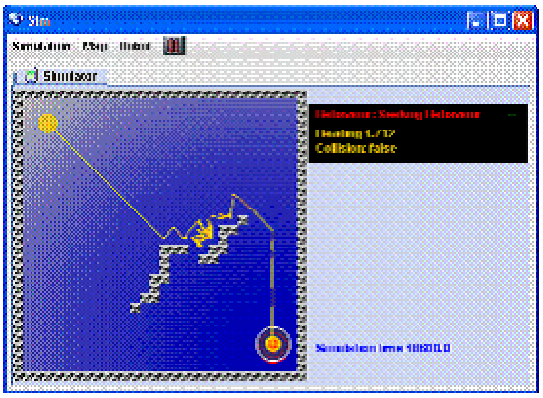
This performs as expected from the collision behaviour's preference for the left turn. Similar arguments apply as to algorithm 1.



Picture of algorithm 2 run

Algorithm 3 - Simulation end time: 18.600

The robot traces a path that resembles the path of the algorithm 2 but collides more often because the angle of deflection is small in many cases this in part explains why it takes longer.

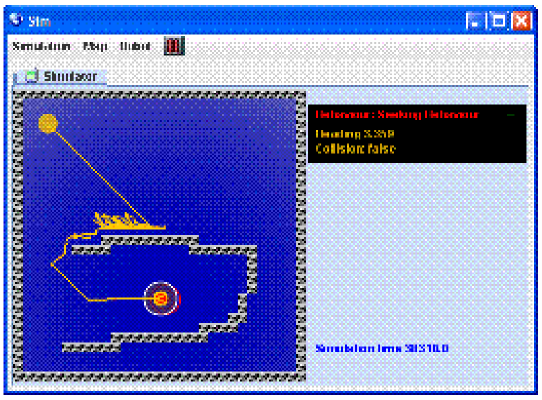


Picture of algorithm 3 run

Configuration 5- Results

Algorithm 1 - Simulation end time: 38.310

This is similar to scenario of configurations 2 and 4 -although time is worse here. Robots sets off to the cell of strongest signal on one side of the wall. But to reach its destination it must go through cells of a weaker signal. The collision behaviour manages to 'drag' the robot around to the left to a point where it can head to its destination.

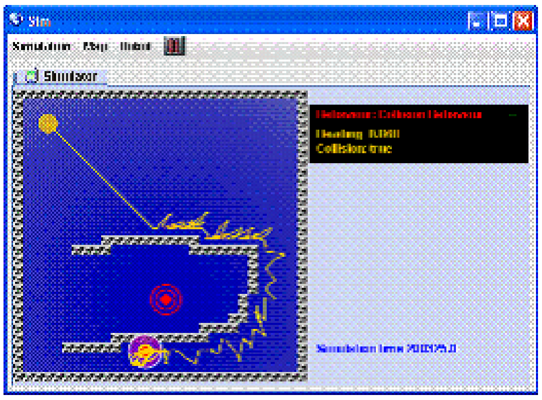


Behaviour: Seeking Behaviour
Heading [illegible]
Collision: false
Simulation time [illegible]

Picture of algorithm 1 run

Algorithm 2 - Simulation end time: Probably never!

The robot correctly goes to a point where strength is strongest, then collides with an obstacle. After this it is then drag around to the right where finally it gets stuck in a never ending sequence of following a rectangle like path, i.e. colliding with an obstacle turning, colliding, turning colliding and so on continuously.

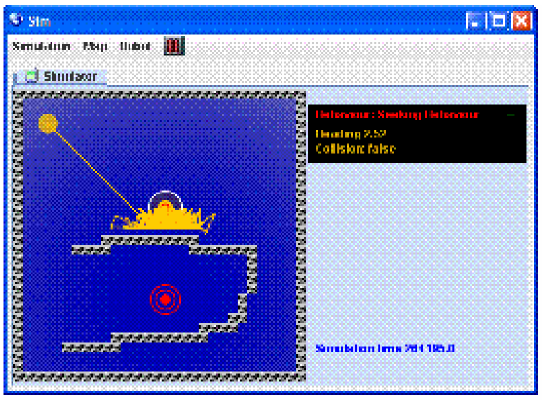


Behaviour: Collision Behaviour
Heading [illegible]
Collision: true
Simulation time [illegible]

Picture of algorithm 2 run

Algorithm 3 - Simulation end time: Probably never!

Here the random strategy plays a small part. The light seeking behaviour seem to be a dominating factor since whenever the randomness drags the robot to a path that would bring it to the target the light seeking behaviour drags it back to a location of strongest signal on the other side of the obstacle, where the target is not located. If the range of the sensor were greater the light seeking behaviour could sense more precisely where the target is located and these behaviours could act in a more co-operative manner.



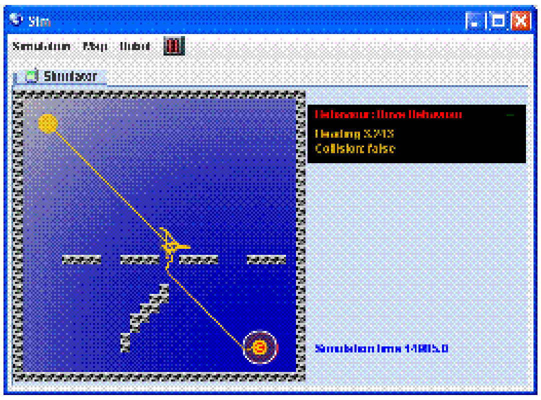
Behaviour : Seeking Behaviour
Heading 2.52
Collision: false
Simulation time 261185.0

Picture of algorithm 3 run

Configuration 6- Results

Algorithm 1 - Simulation end time: 14.985

Robot starts moving forwards then just about midway collides with two obstacles on either left or right side, eventually moving –sliding, to a point closer to the target where it finds a course straight to its target.

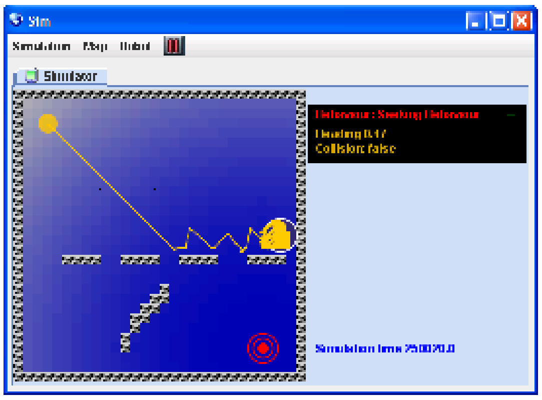


Behaviour : Home Behaviour
Heading 3.743
Collision: false
Simulation time 14985.0

Picture of algorithm 1 run

Algorithm 2 - Simulation end time: Probably never!

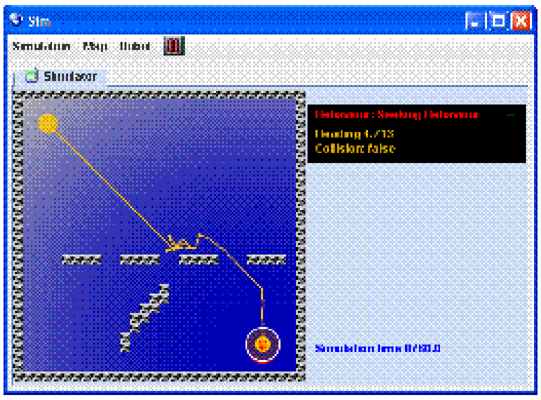
Here the robot sets a course, collides with an obstacle, which results in the collision behaviour moving the robot further to the right where another obstacle is present. The robot is then trapped in a dynamic equilibrium, doomed to repeat the same sequence of movements repeatedly. Had this first obstacle not been present the robot would probably have succeeded in reaching its target but this demonstrates once again the negative effect a collision routine can have on the global goal.



Behaviour : Seeking Behaviour
Heading 0.47
Collision: false
Simulation time 250020.0

Algorithm 3 - Simulation end time: 16.215

Here, unlike the case of algorithm 2 the robot does not get stuck as its light-seeking behaviour takes charge nearby an 'opening' after a few collisions with an obstacle bring it nearby.

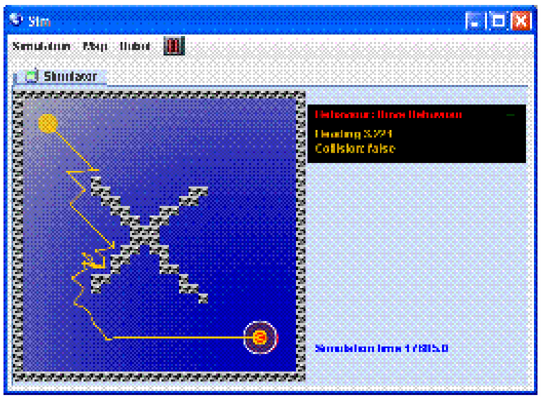


Picture of algorithm 3 run

Configuration 7- Results

Algorithm 1 - Simulation end time: 17.685

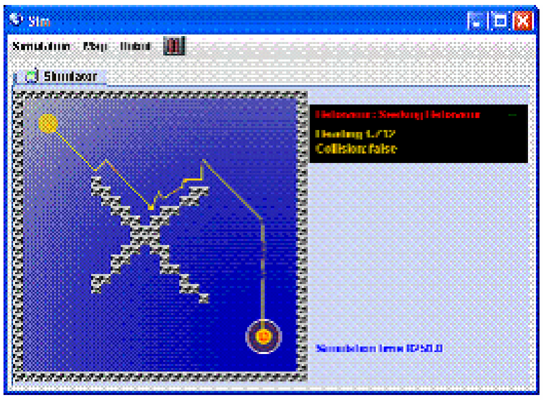
Robot sets off shortly thereafter colliding with the obstacle, then turns to its left side where it keeps colliding and reacting but always moving a little more to the left and finally it reaches a free region where no obstacles block its path to the target.



Picture of algorithm 1 run

Algorithm 2-Simulation end time: 8.250

By symmetry, similar arguments and description apply as to algorithm 1's result in this case although time is shorter.

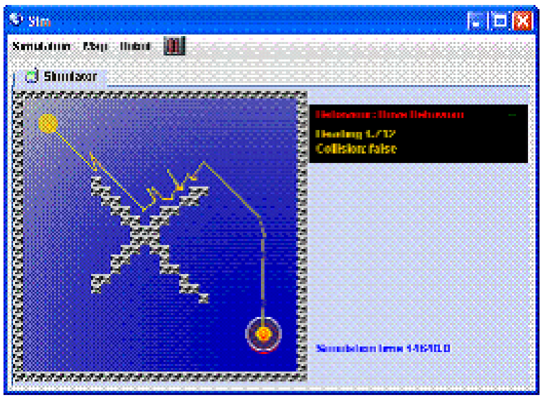


[illegible]
Heading [illegible]
Collision: false
Simulation time [illegible]

Picture of algorithm 2 run

Algorithm 3 - Simulation end time: 16.515

After we see what path the robot follows, e.g. upper part of this obstacle which is the same path taken by algorithm 2, it is not surprising to see it perform slightly worse than algorithm 2. Any strategy here that employs a preference for either right or left would perform better here to get around the obstacle after getting trapped in its corner by the light seeking behaviour.



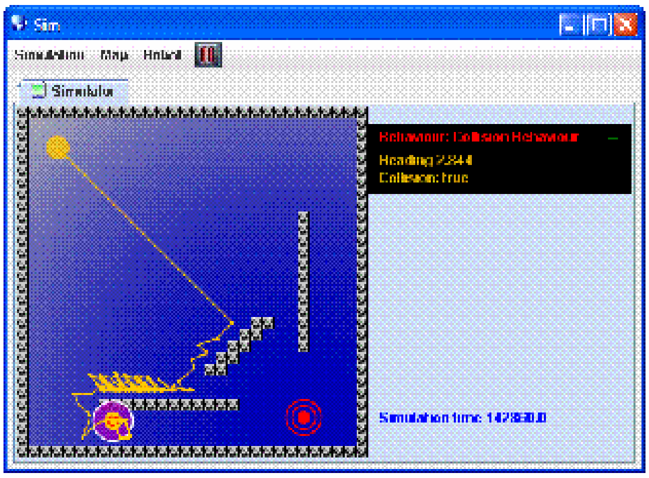
[illegible]
Heading [illegible]
Collision: false
Simulation time [illegible]

Picture of algorithm 3 run

Configuration 8- Results

Algorithm 1 - Simulation end time: Probably never!

In this case, the robot fails to take the shortest path to the target by moving in between obstacles. Because of its preference for the left direction and domination of collision behaviour the robot finds its way between narrow obstacles where it gets stuck in a never ending cycle of colliding with an obstacle, turning 90° slide and collide again, turn 90°, slide, and repeat this over and over again. As the picture shows the robot keeps tracing a path indefinitively in the shape of a rectangle.

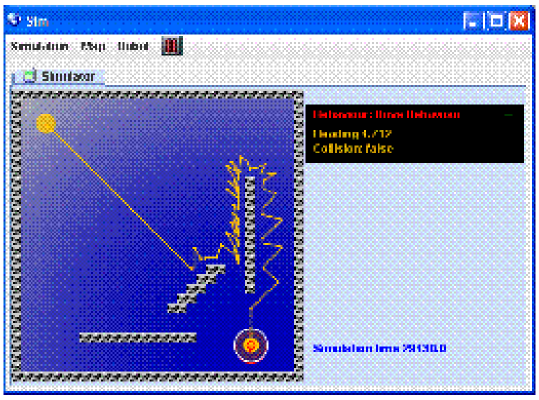


Behaviour: Collision Behaviour
Heading 234.4
Collision: true
Simulation time: 142830.0

Picture of algorithm 1 run

Algorithm 2 - Simulation end time: 29.130

Although this configuration does appear symmetrical around a diagonal axis it actually isn't. Similar to algorithm 1 this algorithm moves the robot in between a tunnel but this time the collision behaviour's actions finish just before the robot touches the other obstacle. This permit the robot to come a little closer to the obstacle each time and finally it reaches its destination.

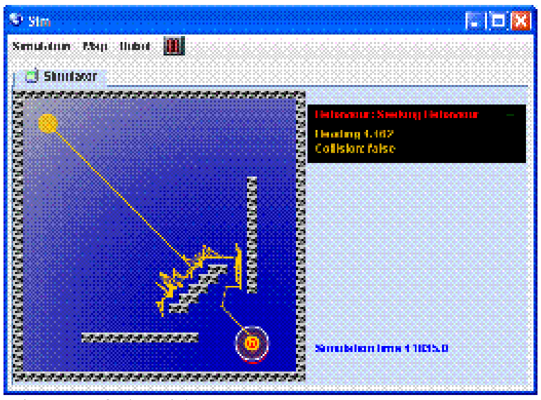


Behaviour: Home Behaviour
Heading 1.717
Collision: false
Simulation time 29130.0

Picture of algorithm 2 run

Algorithm 3-Simulation end time: 41.835

This algorithm takes a different path than either other two algorithms in this configuration. In fact a path along the shortest path is taken. The time to complete is still larger than that of algorithm one, and this seems normal as the distance travelled is also larger since the robot does spend some time colliding and turning.



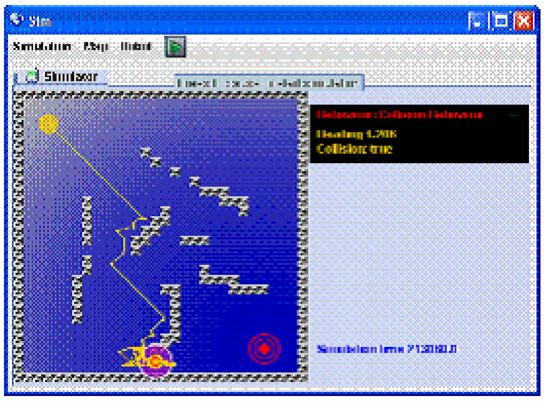
Behaviour: Seeking Behaviour
Heading 1.467
Collision: false
Simulation time 11005.0

Picture of algorithm 3 run

Configuration 9- Results

Algorithm 1 - Simulation end time: Probably never!

Here the robot collides with an obstacle turns and collides with another obstacle a few times and then settles into state where it keeps repeating the same sequence of movements, although the shape traced is not that of a rectangle. The end result is similar to the algorithm's result in the 8th configuration.

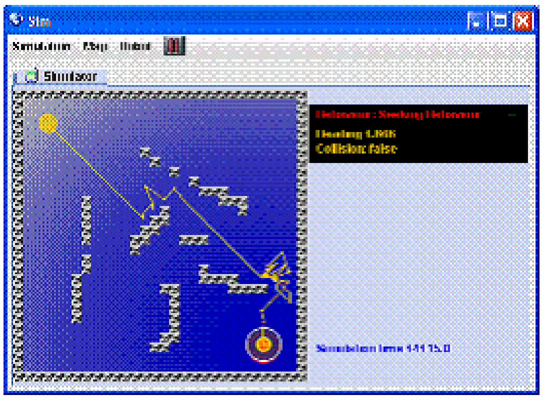


Behaviour: Collision Behaviour
Heading 1.706
Collision: true
Simulation time 712080.0

Picture of algorithm 1 run

Algorithm 2-Simulation end time: 14.115

Unlike the previous case the robot does not get stuck in an unproductive endless loop of regularities.

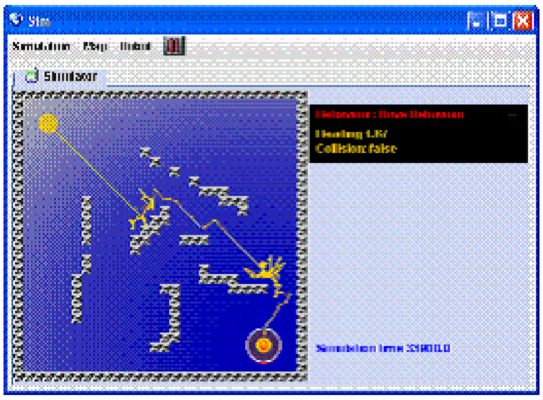


Behaviour: Seeking Behaviour
Heading 1.805
Collision: false
Simulation time 14115.0

Picture of algorithm 2 run

Algorithm 3-Simulation end time: 33.900

Again the random factor means more time is spent colliding with obstacles, one positioned about midway to the goal, the other closer to the target but unlike algorithm 1 it at least finishes.

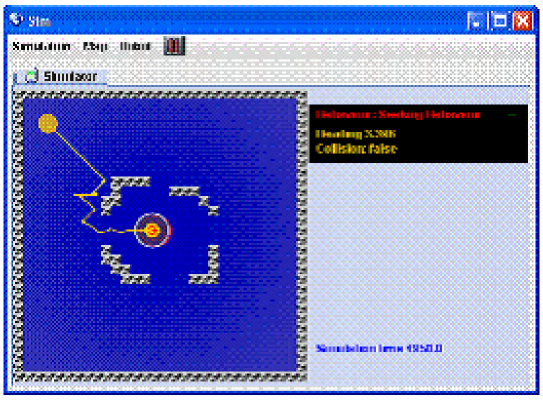


Picture of algorithm 3 run

Configuration 10- Results

Algorithm 1-Simulation end time: 4.950

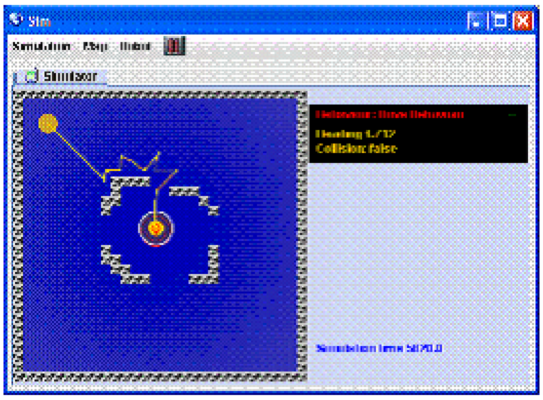
This configuration is different from others where the goal is located farther away. Here it is located around the center of the world. This partly explains the short time taken to complete this run. The robot only experiences around 4 collisions with an obstacle and shortly afterwards it reaches a region free of obstacle and then it goes straight to the source of the signal.



Picture of algorithm 1 run

Algorithm 2-Simulation end time: 5.820

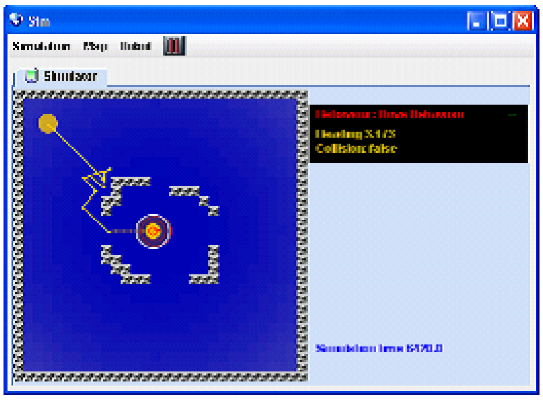
This case is has similar results to algorithm 1 by symmetry.



Picture of algorithm 2 run

Algorithm 3-Simulation end time: 6.420

Here it performs similarly as the other two algorithms in this configuration. Maybe this is due to a smaller sized obstacle than usual and concaveness of the obstacle.



Picture of algorithm 3 run

5.2 Conclusion

The results from the tests show that the random strategy never performed best in any test case. When the obstacles are regular the first two algorithms seem to perform well. The random algorithm seems to perform better when the robot encounters an irregular obstacle than when it collides with a regular shaped obstacle. The tests also showed that the collision routine could prove detrimental in cases where the period for the robot to advance after a collision brings it into contact with another obstacle. This result is especially apparent and happens when the robot is located in a narrow ‘corridor’ of obstacles.

The collision behaviour time to go forward could be reduced to handle this case but this could have an adverse effect on other cases where the robot collides more often with an obstacle. These latter are cases where the light seeking behaviour is trapping the robot into region of signal maxima on one side of an obstacle but the collision routine would then be unable to ‘overshoot’ it because of reduced distance to travel after a collision.

These test made it apparent that whatever strategy used in search and rescue, it would be very difficult if not impossible to devise one good strategy or controller for robots in different environments and differing types of robots. One problem is adaptability. The robot’s controller may not be adapted too much to one type of environment because it might fail in another type of environment even if very similar.

The problem with the first two algorithms is that they are fixed, i.e. their collision behaviours are. The random strategy’s weakness and strength is its variability, sometimes yielding good results, sometimes bad. In a successful approach, it is presumed there is a need for a balance between memory, adaptability and randomness. The robot will use its memory to apply a productive action due to similar conditions encountered, then when that fails, either change it reaction or apply a little randomness at that time.

Summary of results:

	Algorithm 1 time	Algorithm 2 time	Algorithm 3 time	Best performer	Worst performer
Configuration 1	104.130	8.355	55.035	2	1
Configuration 2	11.940	8.370	43.755	2	3
Configuration 3	28.995	6.390	14.910	2	1
Configuration 4	9.645	12.210	18.600	1	3
Configuration 5	38.310	Never	Never	1	2,3
Configuration 6	14.985	Never	16.215	1	2
Configuration 7	17.685	8.250	16.515	2	1
Configuration 8	Never	29.130	41.835	2	1
Configuration 9	Never	14.115	33.900	2	1
Configuration 10	4.950	5.820	6.420	1	3

Algorithm 1=turn 90° left in case of collision and then advance
Algorithm 2=turn 90° right in case of collision and then advance
Algorithm 3=turn random amount of degrees in case of collision and then advance

time=simulator ticks

6. Further work

The developed software so far needs improvement. On one side is the level of realism. The simulator fails to model the effects of forces. What this means is that the effects of acceleration and friction on the robot are not simulated. When the robot stops or sets off, it does so instantaneously. A robot in reality is typically affected by friction along it contact surfaces. In this case friction between it and the walls and floor.

The behaviour and arbitration system also needs a better solution. As is mentioned one flaw of the system is that when the collision behaviour becomes active it executes until it is finished even if the robot does collide a second time before it has finished acting on the first collision. There is a need to devise some sort of mechanism that will both keep track of this behaviour’s status during simulation steps and allow it to be interrupted before it has finished.

Sensor modelling could also be made more realistic. The addition of noise on readings and what the sensor would read were it not omni-directional could be added. Although not quite clear how this could be implemented, a way of configuring and creating more behaviours could also be added. If the set of inputs were allowed to grow, there would be more room to create different and creative behaviours. Since there are only two inputs now, for touch and light, this limits things quite.

The problem with this behaviour is that it has a similar set of actions as the drive behaviour. Behaviours that are incremental, i.e. can be divided into smaller sets of identical action, such as a turning behaviour are easier to process than for instance the collision behaviour which requires movement in the x,y plane as well as turning.Maybe the solution would be to tell a behaviour to restart from the beginning if it senses that its set of preconditions has been satisfied again whilst it is executing? Again this seem to go against Brook’s intention where control is left to an arbitrating object.

7. Conclusion of entire work

From this project is seems clear that whatever environment a robot deployed for search and rescue will be located in, it must be able to learn quickly, since time is often a critical factor. Work on the project seemed to indicate that whatever strategy used, it must have some type of 'memory' of its actions. Strategies that use behaviour and memory exist such as the Q-learning algorithm. This is a type of reinforcement learning that can for instance be used to teach a robot to walk or follow an object.

This project was very revealing in what capabilities a real simulator must have. Designing and coding one is hard work demanding knowledge of physics, mathematics, and programming skills. The animation aspect was presumed to be a difficult aspect from the beginning but this seems to be a fairly minimal factor when compared to the modeling of the physical laws that must apply on object.
It would have been interesting to see what would have happened with the tests and algorithms in a more realistic simulator and compare the results.

Overall, this project showed me that coding can with time become easier and more like using pattern matching. Several times while coding, the same routines were used from one class into another to perform similar things but slightly different still. Often this involved arrays. When on get used to the Java programming language, some routine things with it reminds one of assembling Lego bricks, they can become second nature with time. Indeed, almost fun!

There is still a lot to learn about but this project was of good value in honing programming skills. No longer am I afraid of doing more research into games and animation stuff in Java. That will be the next step, and hopefully something more will come out of it-who knows? -maybe a realistic, multi-sensor, behaviour based simulator. Another idea that came to mind was to develop something, maybe create a behaviour based character for a computer game that could evade falling objects for instance. I think this a real possibility and maybe this could demonstrate something unique. This involves AI and there is a lot of potential for anything that seems or acts intelligently in the computer gaming field. The only problem in this field is the lack of relationship between researchers of AI in universities and computer game programmers.

References

1. Russell, Stuart and Norvig, Peter. *Artificial Intelligence: A Modern Approach.* New Jersey: Upper Saddle River. Pearson Education, 2003

2. S. Goldberg， M. Maimone, L. Matthies. “Stereo Vision and Rover Navigation Software for Planetary Exploration”. *Proceedings of the 2002 IEEE Aerospace Conference*. Big Sky, MT, March 2002.

3. S. Koenig, C. Tovey, and W. Halliburton. *Greedy mapping of terrain.* In Proceedings of the International Conference on Robotics and Automation, pages 3594--3599. IEEE, 2001.

4. S. Thrun. *Robotic mapping: A survey.* In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium.* Morgan Kaufmann, 2002.

5. R. A. Brooks. "*A Robust Layered Control System for a Mobile Robot*", IEEE Journal of Robotics and Automation, Vol. 2, No. 1, March 1986, pp. 14–23; also MIT AI Memo 864, September 1985.

6. A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

7. A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

8. H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

9. R. A. Brooks, P. Maes. Learning to Coordinate Behaviors. A1-laboratory. MIT

Appendix A – Code listing

The simulator part of the architecture.

```
import javax.swing.Timer;
import java.awt.Toolkit;
import javax.swing.JComponent;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.awt.event.*;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.RenderingHints;
import java.awt.Shape;
import java.awt.geom.Ellipse2D;
import java.awt.Dimension;
import java.awt.image.*;
import javax.swing.ImageIcon;
/**
 * The server class. This is in charge of the simulator world
 *
 *
 * Created: Fri Mar 18 16:00:09 2005-2006
 *
 * @author <a href="mailto:Max@CERXX"></a>
 * @version 1.0
 */
//public class simGuiExp extends JComponent implements
ActionListener  {
public class simGuiExp implements ActionListener  {
   private Graphics2D g2;
   public  robot[] robot;
   protected static int cells=30, length=10,timeSize=15,
diamRobot=14;
   private static int
scanRange=(int)((double)(diamRobot)/(double)length+0.5);

   protected int[][] cellVal;
   protected static int[][] tmpCellVal;
   //  private int[][] lightVal;
   private Timer timer; int delay=20;
   private static int collides=0;
   private static int maxTravel=cells*length-diamRobot/2;
   protected static int selectedSource;
   protected static Animator animate;
   public static double timerf=0;
   protected static double simTime=0;
   private Behavior colide, seek, drive;
   private Arbitrator arbi=new Arbitrator();
   protected boolean running=false, randomizer=false;

   int starty,startx,lasty, lastx;
   /**
    * Creates a new <code>simGui</code> instance.
    *
    */

   public simGuiExp(int[][] grid) {

      this.cellVal=grid;
      maxTravel=cells*length-diamRobot/2;
      timer=new Timer(delay, this);
      cells=grid.length;
      //robot=new robot[2];
      robot=new robot[1];
      // robot[0]=new robot(25,25,29,29,this, Color.orange);
      robot[0]=new robot(35f,35f,0.2f,4.5,this, Color.orange);

      drive=new driveBehavior();
      colide=new collisionBehavior();
      seek=new seekingBehavior();
      colide.behavior(robot[0], arbi);
      seek.behavior(robot[0], arbi);
      drive.behavior(robot[0], arbi);
      //seek.behavior(robot[1], arbi);
      //Behavior[] behaviors={drive,seek,colide};
      Behavior[] behaviors={colide,seek,drive};
      arbi.setBehaviors(behaviors);

      // colide=new collisionBehavior();
      // colide.behavior(robot[0]);

      //  robot=new robot[1];
      //              robot[0]=new robot(20,20,21,21,this);

      animate=new Animator(cellVal, robot,arbi, this);
      //test.tpane.addTab("Simulator",animate);
test.tpane.insertTab("Simulator",new
ImageIcon("simSmall.gif"),animate,null,0);

      //start();

   }

   public void start(){animate.simstatus="running";running=true;
timer.start();}
   public void
stop(){animate.simstatus="stopped";running=false;timer.stop();}
   public void update(){}

   public Arbitrator getArbitrator(){
      return arbi;
   }
   public void worldShifter(int[][] cellVal){

      int ranNum=(int)(Math.random()*10);
      System.out.print(" number... "+ranNum);
      int x=1,y=1;
      if (ranNum>=9) {
         x=(int)(Math.random()*cellVal.length);
         y=(int)(Math.random()*cellVal.length);
         int tmp=cellVal[x][y];
         cellVal[x][y]=cellVal[y][x];
         cellVal[y][x]=tmp;
         animate.refreshWorld();
System.out.println(" world shifter..............");

     //  if (cellVal[x][y]==-2) {

//       }
      }}

public void worldShiftThread(){

 new Thread() {
       public void run() {
    randomizer=true;
   while(randomizer) { //

      worldShifter(cellVal);

         try { Thread.sleep(500);  // sleep a bit
         }
         catch(InterruptedException ex){}
        }
   System.out.print("End of Thread!");
         }
        }.start();
   }

   protected simGuiExp resetSim(){
      simTime=0;
      randomizer=false;
      return new simGuiExp(cellVal);
}

   public final void actionPerformed(final ActionEvent actionEvent)
{
      //timeScheduler=timeScheduler+timeSize+delay;
      simTime=simTime+timeSize;
      long start=System.currentTimeMillis();
      animate.setRobotImages();
      //   System.out.println("Time between actionperformed calls:
"+(start-timerf));
```

```
      //       //System.out.println("Elapsed time from last call:
"+timePassed);
      //        System.out.println("Robots 0 current location
"+robot[0].loc_x+" "+robot[0].loc_y);
      //       System.out.println("Robots 0 next location
"+robot[0].dest_x+" "+robot[0].dest_y);
      //          System.out.println("Robots 1 current location
"+robot[1].loc_x+" "+robot[1].loc_y);
      //       System.out.println("Robots 1 next location
"+robot[1].dest_x+" "+robot[1].dest_y);
      draw();
      //long time = System.currentTimeMillis() - start;
      //System.out.println("simu time "+simTime);
      start=System.currentTimeMillis();
      for (int i=0;i<robot.length;i++) {
      robot[i].update(timeSize);}
      arbi.arbitrate();

   }

   public void draw(){
      animate.draw();
   }

  public void runny( )
//       /* Repeatedly update, render, sleep */
   {
//       //running = true;
      long beforeTime, timeDiff, sleepTime, period=10;

      beforeTime = System.currentTimeMillis( );

       timeDiff = System.currentTimeMillis( ) - beforeTime;
       sleepTime = period - timeDiff;   // time left in this loop

          if (sleepTime <= 0)  // update/render took longer than
period
                   sleepTime = 5;    // sleep a bit anyway

         try {
            Thread.sleep(sleepTime);  // sleep a bit
         }
         catch(InterruptedException ex){}
         beforeTime = System.currentTimeMillis( );
      }

   } // end of run( )


   protected double calcHeading(double dirx, double diry){
      double heading=0;
      int quadrant=1;
```

```
 //      if (dirx>=0 &&  diry<=0) {quadrant=1;}else if (dirx<0 &&
diry<0){quadrant=2;}
//       else if (dirx<0 && diry>0) {quadrant=3;}else {quadrant=4;}
  if (dirx>=0 &&  diry<=0) {quadrant=1;}else if (dirx<=0 &&
diry<=0){quadrant=2;}
      else if (dirx<=0 && diry>=0) {quadrant=3;}else {quadrant=4;}

      if (dirx!=0) {
         heading=Math.atan(diry/dirx);
      }
      else if (dirx==0 && diry>0) {
         heading=1.5*Math.PI;
      }
      else if (dirx==0 && diry<=0) {
         heading=Math.PI/2;
      }

      if (heading<=0 && quadrant==1) {
         heading=Math.PI+heading;
      }
      else if (heading<=0 && quadrant==3) {
         heading=2*Math.PI+heading;
      }
      else if (heading>=0 && quadrant==4 ) {
         heading=Math.PI+heading;
      }

      return heading;
   }

   public double[] checkCollision(robot rob){
   //public float[] checkCollision(robot rob){

      double vector_X=rob.dest_x-rob.loc_x; double
vector_Y=rob.dest_y-rob.loc_y;
      double res=Math.sqrt(vector_X*vector_X+vector_Y*vector_Y);
      double directx=(1/res*vector_X);
      double directy=(1/res*vector_Y);
      double[] newLoc={rob.loc_x, rob.loc_y};
      double safe_advance_x=directx;//*0.5;
      double safe_advance_y=directy;//*0.5;


      boolean collision=false;
      //current grid coordinate of robot
      int gridx=(int)(rob.loc_x/length);
      int gridy=(int)(rob.loc_y/length);

      double test_x=rob.loc_x+safe_advance_x;
      double test_y=rob.loc_y+safe_advance_y;

      double tempVarX=rob.loc_x;
      double tempVarY=rob.loc_y;
      int d=0;

      while ((Math.abs(safe_advance_x)<Math.abs(vector_X) ||
Math.abs(safe_advance_y)<Math.abs(vector_Y)))//
      {
         //  && ((test_x-rob.dest_x)*(test_x-rob.dest_x)+(test_y-
rob.dest_y)*(test_y-rob.dest_y))>1)  {
         // System.out.println("inside while"); d++;
         //Pinging ahead--testing for collisions with a circle

         Shape body=pingShape(test_x, test_y);
         Shape bodyX=pingShape(test_x, tempVarY);
         Shape bodyY=pingShape(tempVarX, test_y);

         gridx=(int)(test_x/length);
         gridy=(int)(test_y/length);
         int[] scanx=findScanDirx();
         int[] scany=findScanDiry();

         //   for (int i=0;i<scany.length &&
(scany[i]+gridy)<Grid.grids.length && (scany[i]+gridy)>=0;i++ ) {

         //             for (int j=0;j<scanx.length &&
(scanx[j]+gridx)<Grid.grids.length && (scanx[j]+gridx)>=0 ;j++) {

         for (int i=0;i<scany.length && (scany[i]+gridy)<cells &&
(scany[i]+gridy)>=0;i++ ) {

            for (int j=0;j<scanx.length && (scanx[j]+gridx)<cells &&
(scanx[j]+gridx)>=0 ;j++) {

               boolean
wall=(cellVal[gridx+scanx[j]][gridy+scany[i]]==-2);

               // boolean
touch=body.intersects(Grid.grids[gridx+scanx[j]][gridy+scany[i]]);
               boolean
touch=body.intersects(Grid.rectangle(gridx+scanx[j],gridy+scany[i]))
;
               boolean
touchX=bodyX.intersects(Grid.rectangle(gridx+scanx[j],gridy+scany[i]
));
               boolean
touchY=bodyY.intersects(Grid.rectangle(gridx+scanx[j],gridy+scany[i]
));

               if ((touch && wall) || test_x>=maxTravel ||
test_y>=maxTravel) {//if touch wall or new location out of
bounds!!!--touching wall equal going out bounds
                  //  System.out.println("Collision! at "+tempVarX+"
"+tempVarY);
                  //                   System.out.println("X:
"+(int)(tempVarX/length)+"Y "+(int)(tempVarY/length));
                  // Toolkit.getDefaultToolkit().beep();
                  // System.out.println("bodyX: "+touchX+" bodyY
"+touchY);
                  //  if (bodyX!=null && animate!=null) {
                  //                      animate.drawShape(bodyX);
                  //                   }

                  newLoc[0]=tempVarX-directx;
                  newLoc[1]=tempVarY-directy;
                  //                   collides++;
                  rob.collision=true;
                  //                   return newLoc;
                  //if robot collides then set destination
robot=curr-location?????
                  if (!touchX && touchY) {
                     newLoc[0]=test_x;
                     rob.dest_x=newLoc[0];rob.dest_y=newLoc[1];
                     // System.out.println("increasing x ");
                     return newLoc;
                  }
                  else if (!touchY && touchX) {
                     newLoc[1]=test_y;
                     rob.dest_x=newLoc[0];rob.dest_y=newLoc[1];
                     return newLoc;
                  }
                  else if (touchY && touchX) {
                     //System.out.println("final exit");
                     rob.dest_x=newLoc[0];rob.dest_y=newLoc[1];
                     return newLoc;
                  }
                  else {
                     //just keep on looping!
                  }
               }
            }
         }
         safe_advance_x+=directx;
         safe_advance_y+=directy;
         tempVarX=test_x;
         tempVarY=test_y;
         test_x=rob.loc_x+safe_advance_x;
         test_y=rob.loc_y+safe_advance_y;
      }//end of while
      rob.collision=false;
      newLoc[0]=tempVarX;//-directx;
      newLoc[1]=tempVarY;//-directy;
      //System.out.println("value of d "+d);
      return newLoc;
   }

   public int[] findScanDirx(){
      int[] direct={-1,0,1};
      return direct;
   }
   public int[] findScanDiry(){
      int[] direct={1,0,-1};
      return direct;
   }

   public Shape pingShape(double locx,double locy){
      return new Ellipse2D.Double(locx-diamRobot/2,locy-
diamRobot/2,diamRobot,diamRobot);
   }

   public double signalDirection(robot rob){
      int distance=(rob.sensorRange-36)/length+1;
      return scanMultipleCells(rob, distance);
      // double direct;
//       int gridx=(int)(rob.loc_x/length);
//       int gridy=(int)(rob.loc_y/length);

//       int[] scanx=findScanDirx();
//       int[] scany=findScanDiry();

//       double lowestVal=cellVal[gridx][gridy];
//       int posx=gridx, posy=gridy;

//       //  for (int i=0;i<scany.length &&
(scany[i]+gridy)<Grid.grids.length && (scany[i]+gridy)>=0;i++ ) {
//       //          for (int j=0;j<scanx.length &&
(scanx[j]+gridx)<Grid.grids.length && (scanx[j]+gridx)>=0 ;j++)
//       for (int i=0;i<scany.length &&
(scany[i]+gridy)<cellVal.length && (scany[i]+gridy)>=0;i++ ) {
//          for (int j=0;j<scanx.length &&
(scanx[j]+gridx)<cellVal.length && (scanx[j]+gridx)>=0 ;j++) {
//             // if
(grfxSquare[gridx+scanx[j]][gridy+scany[i]].range_class<lowestVal &&
lowestVal>=0) {
//             if (0<=cellVal[gridx+scanx[j]][gridy+scany[i]] &&
cellVal[gridx+scanx[j]][gridy+scany[i]]<lowestVal) {

//                lowestVal=cellVal[gridx+scanx[j]][gridy+scany[i]];
//                posx=gridx+scanx[j];posy=gridy+scany[i];
//                //   System.out.println("Light seeking strength
loop ...Strongest cell at x,y:  "+posx+" ,"+posy);
//                //                 System.out.println("value there
is: "+cellVal[posx][posy]);
//                //                 System.out.println("Robot's
current position, x,y, "+gridx+" "+gridy);
//                //                  System.out.println(" value
here: "+cellVal[gridx][gridy]);
//                //System.out.println("value of non-existent
"+grfxSquare[gridx+1][gridy].range_class);
//             }
//          }
//       }
//       direct=calcHeading(((posx+0.5)*length-rob.loc_x),
((posy+0.5)*length-rob.loc_y));
//       // System.out.println("According to signalDirection;
heading is: "+direct);
//       // if (cellVal[gridx][gridy]==0) {
//       //           stop();//rob.speed=0;
//       //        }
//       return direct;
   }

 public double scanMultipleCells(robot rob, int scanCellDistance){
      int gridx=(int)(rob.loc_x/length);
      int gridy=(int)(rob.loc_y/length);

      double lowestVal=cellVal[gridx][gridy];
      int posx=gridx, posy=gridy;
      double direct;

    int m1=(((gridx-scanCellDistance)<1) ? -gridx+1 : -
scanCellDistance);
    int m2=(((gridx+scanCellDistance)<(cellVal.length-2)) ?
scanCellDistance : (cellVal.length-2-gridx));
    int n1=(((gridy-scanCellDistance)<1) ? -gridy+1 : -
scanCellDistance);
    int n2=(((gridy+scanCellDistance)<(cellVal.length-2))  ?
scanCellDistance : (cellVal.length-2-gridy));
     for (int i=n1;i<(n2+1);i++ ) {
       for (int j=m1;j<(m2+1);j++ ) {
 if (0<=cellVal[gridx+j][gridy+i] &&
cellVal[gridx+j][gridy+i]<lowestVal) {

                 lowestVal=cellVal[gridx+j][gridy+i];
                 posx=gridx+j;posy=gridy+i;
            }
          }
       }
direct=calcHeading(((posx+0.5)*length-rob.loc_x),
((posy+0.5)*length-rob.loc_y));
System.out.println("According to signalDirection; heading is:
"+direct+"cell is at "+posx+" "+posy);
return direct;
   }

   // Implementation of java.awt.event.ActionListener

   /**
    * Describe <code></code> method here.
    *
    * @param
    */

   public final void advanceRobot() {
      for (int i=0;i<robot.length;i++) {
         double dis_x, dis_y;

         dis_x=(robot[i].loc_x-robot[i].dest_x)*(robot[i].loc_x-
robot[i].dest_x);
         dis_y=(robot[i].loc_y-robot[i].dest_y)*(robot[i].loc_y-
robot[i].dest_y);

         //if (dis_x+dis_y<1) {
         //  robot[i].dest_x=robot[i].loc_x;
         // robot[i].dest_y=robot[i].loc_y;
            //  System.out.println("Robot"+i+" has stopped");
            //}

         //    else if ((dis_x+dis_y)>=1){

            double[] safeNoCollision_xy=new double[2];
            safeNoCollision_xy=(checkCollision(robot[i]));
            robot[i].loc_x=safeNoCollision_xy[0];
            robot[i].loc_y=safeNoCollision_xy[1];
            //    }
          if (robot[i].loc_x<(diamRobot/2+length)) {
robot[i].loc_x=diamRobot/2+length; }
          if (robot[i].loc_y<(diamRobot/2+length)) {
robot[i].loc_y=diamRobot/2+length; }
          if (robot[i].loc_x>(maxTravel-length)) {
robot[i].loc_x=maxTravel-length; }
          if (robot[i].loc_y>(maxTravel-length)) {
robot[i].loc_y=maxTravel-length; }
      }
   }
}
```

The map creating part of the architecture.

```
import javax.swing.JComponent;
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.RenderingHints;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseMotionAdapter;
import java.awt.event.*;
import javax.swing.JButton;
//import javax.swing.JPanel;
/**
 * The server class. This is in charge of the simulator world
 *
 *
 * Created: Fri Mar 18 16:00:09 2005
 *
 * @author <a href="mailto:Max@CERXX"></a>
 * @version 1.0
 */
public class simWorldCreator extends JComponent   {
   private Graphics2D g2;
   protected static int cells, length=10;
   private int diamRobot;
   public int[][] cellVal;
   protected static int[][] tmpCellVal;
   private int[][] lightVal;
   private static int maxTravel;//=cells*length-diamRobot/2;
   protected static int selectedSource=-1;
   protected static BufferedImage bimage;
   int starty,startx,lasty, lastx;
   private double start;
   private boolean trigger=false;
   protected boolean noise=false;
   private JButton playButton;

   /**
    * Creates a new <code>simGui</code> instance.
    *
    */
   public simWorldCreator() {
      start=System.currentTimeMillis();
      cells=simGuiExp.cells;
      diamRobot=simGuiExp.diamRobot;
      maxTravel=cells*length-diamRobot/2;
      cellVal=new int[cells][cells];
      lightVal=new int[cells][cells];
      cellVal=Grid.lightGridGenrtr(cells/2,cells/2);
      lightVal=Grid.lightGridGenrtr(cells/2,cells/2);
      // System.arraycopy(cellVal, 0, lightVal, 0, cellVal.length);
      // lightVal=Grid.lightGridGenrtr(cells/2,cells/2);
      tmpCellVal=cellVal;

      System.out.println("Simworld cells "+cells);
      addMouseListener(new MouseAdapter(){
            public void mousePressed(MouseEvent e){
               int gridx=(int)(e.getX()/length);
               int gridy=(int)(e.getY()/length);

               if (selectedSource==-3 && !trigger) {//set if erasing
mode selected and user has clicked once
                  startx=(e.getX());starty=(e.getY());
                  trigger=true;
                  //  System.out.println("mouse press");
                  //                    System.out.println("Val of
start co-ordinates "+startx+" "+starty);
                  repaint();
                  //set flag..already clicked once in erasing
mode...
               }
               else if (trigger) {
                  clearCells();
                  //System.out.println("Triggggre");
                  //  startx=(e.getX());starty=(e.getY());
                  //                lastx=startx;lasty=starty;
                  trigger=false;
                  repaint();
               }//then 'erase' cells from (int)(startx/length) to
(int)(lastx/length) using
cellVal[gridx][gridy]=lightVal[gridx][gridy];
               else markCell(gridx,gridy);
            }
         });
      addMouseMotionListener(new MouseMotionAdapter(){
            public void mouseMoved(MouseEvent e){
               lastx=(e.getX());
               lasty=(e.getY());
               if (selectedSource==-3 && trigger) {
                  //repaint();//
                  refreshWorld(20);
               }
            }});
      addMouseMotionListener(new MouseMotionAdapter(){
            public void mouseDragged(MouseEvent e){
               int gridx=(int)(e.getX()/length);
               int gridy=(int)(e.getY()/length);
               markCell(gridx,gridy);
               //tmpCellVal=cellVal;
            }});
   }

   public void markCell(int gridx, int gridy){
      //boundary check for drawn grid...
      if (gridx>0 && gridy>0 && gridx<(cells-1) && gridy<(cells-1))
{

         if (selectedSource==-2 && cellVal[gridx][gridy]>0 ) {
            //lastSelected=cellVal[gridx][gridy];

            if (noise) {
               noise(gridx,gridy);
               cellVal[gridx][gridy]=-2;
               repaint();
            }
            else {
               cellVal[gridx][gridy]=-2;

simWorldCreator.this.repaint(gridx*length,gridy*length,
length,length);
            }
         }
         else if (selectedSource==-2 && cellVal[gridx][gridy]==0)
         {;}
         else if (selectedSource==-4) {
            cellVal[gridx][gridy]=lightVal[gridx][gridy];
            simWorldCreator.this.repaint(gridx*length,gridy*length,
length,length);
         }
         else if (selectedSource==-1)
         {
            //cellVal[gridx][gridy].range_class=0;
            try {
               tmpCellVal=Grid.lightGridGenrtr(gridx,gridy);
               lightVal=Grid.lightGridGenrtr(gridx,gridy);
               repositionGoal();
               cellVal=tmpCellVal;
               //System.arraycopy(cellVal, 0, lightVal, 0,
cellVal.length);
               if (noise) {noiseMapGen();tmpCellVal=cellVal;}
               // cellVal=tmpCellVal;
               repaint();

            } catch (Exception rre) {System.out.println("Error in
grid creation-Grid class: "+rre); }
         }
      }//eof outer if
      else {
         ;
      }
   }

   private void repositionGoal(){
      for (int row=1;row<(cellVal.length-1);row++) {
         for (int column=1; column<(cellVal.length-1); column++) {
            if (cellVal[column][row]==-2) {
               tmpCellVal[column][row]=-2;
               // System.out.println("repositioning goal");
            }

         }
      }
      if (noise) {removeNoise();noiseMapGen();}
   }

   private void clearCells(){
      int stx=(int)(startx/length+0.5),
sty=(int)(starty/length+0.5),
         lstx=(int)(lastx/length+0.5),lsty=(int)(lasty/length+0.5);
      int tmpx=stx;
      int tmpy=sty;
      System.out.println("Value of stx, sty"+stx+","+sty);
      System.out.println("Value of lstx, lsty"+lstx+","+lsty);
      if (lstx<stx) {
         stx=lstx;lstx=tmpx;
      }
      if (lsty<sty) {
         sty=lsty;lsty=tmpy;
      }
      for (int row=sty;row<lsty;row++) {
         for (int column=stx; column<lstx; column++) {
            cellVal[column][row]=lightVal[column][row];
            //System.out.println("lllooooooooooooopppepped)");
            //System.out.println(lightVal[column][row]);
         }
         //System.out.println();
      }
      //remember to cancel noise effects of obstacles cells when
erased...
      //their effects can go beyond range of erased area so we opt
to reprocess whole map
       if (noise) {removeNoise();noiseMapGen();}
   }

   public void eraseRect(){
      //   int width=Math.abs(lastx-startx);
      //       int height=Math.abs(lasty-starty);

      int width=(lastx-startx);
      int height=(lasty-starty);
      width=((width<0) ? -width : width);
      height=((height<0) ? -height : height);
      int tmpx=startx, tmpy=starty;

      System.out.println("start x "+startx+" width is "+width);
      tmpx=((startx>lastx) ? (startx-width) : startx);
      tmpy=((starty>lasty) ? (starty-height) : starty);

      if (g2!=null) {
         g2.setColor(new Color(100,100,100,100));
         g2.fillRect(tmpx,tmpy,width,height);
         g2.setColor(Color.white);
         g2.drawRect(tmpx,tmpy,width,height);
      }
   }

   public void noise(int gridx, int gridy){
      multipleCellNoise(gridx, gridy, 2);
     //  for (int i=-1;i<2;i++ ) {
//          for (int j=-1;j<2;j++ ) {
//             if (0<cellVal[gridx+j][gridy+i]){

//                if
(cellVal[gridx+j][gridy+i]>lightVal[gridx][gridy]) {
//
cellVal[gridx+j][gridy+i]=(int)(1.5*lightVal[gridx+j][gridy+i]);
//                }
//                else {
//                   if ((int)(0.9*lightVal[gridx+j][gridy+i])!=0) {
//
cellVal[gridx+j][gridy+i]=(int)(0.9*lightVal[gridx+j][gridy+i]);
//                      // System.out.println(
//                   }
//                }
//             }
//          }
//       }
   }

 public void multipleCellNoise(int gridx, int gridy, int
cellDistance){
    int m1=(((gridx-cellDistance)<1) ? -gridx+1 : -cellDistance);
    int m2=(((gridx+cellDistance)<(cellVal.length-2)) ? cellDistance
: (cellVal.length-2-gridx));
    int n1=(((gridy-cellDistance)<1) ? -gridy+1 : -cellDistance);
    int n2=(((gridy+cellDistance)<(cellVal.length-2))  ?
cellDistance : (cellVal.length-2-gridy));
     for (int i=n1;i<(n2+1);i++ ) {
    for (int j=m1;j<(m2+1);j++ ) {

            if (0<cellVal[gridx+j][gridy+i]){

               if (cellVal[gridx+j][gridy+i]>lightVal[gridx][gridy])
{

cellVal[gridx+j][gridy+i]=(int)(1.5*lightVal[gridx+j][gridy+i]);
               }
               else {
                  if ((int)(0.9*lightVal[gridx+j][gridy+i])!=0) {

cellVal[gridx+j][gridy+i]=(int)(0.9*lightVal[gridx+j][gridy+i]);
                     // System.out.println(
                  }
               }
            }
         }
      }
   }

   public void noiseMapGen(){

      for (int row=1;row<(cellVal.length-1);row++) {
         for (int column=1; column<(cellVal.length-1); column++) {
            if (cellVal[column][row]==-2) {
               noise(column, row);
               // System.out.println("adjusting cell");
            }
         }
      }
      repaint();
   }

   public void removeNoise(){

      for (int row=1;row<(cellVal.length-1);row++) {
         for (int column=1; column<(cellVal.length-1); column++) {
            if (cellVal[column][row]!=-2) {
                cellVal[column][row]=lightVal[column][row];
                // System.out.println("resetting cell");
            }
        }
    }
    repaint();
  }
  public void refreshWorld(long refreshtime){
    if (System.currentTimeMillis()-start>=refreshtime) {
        start=System.currentTimeMillis();
        repaint();
    }
  }

  public void update(){}

  public void paintComponent(Graphics g){
    // System.out.println("painting component");
    g2=(Graphics2D)g;

g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.V
ALUE_ANTIALIAS_ON);
    //to set background colour of map
    g2.setColor(Color.black);
    // g2.fillRect(0, 0, getWidth(), getHeight());

    Grid.drawGrid(g2, cellVal);
    if (selectedSource==-3 && trigger) {eraseRect();
System.out.println("ffffff");}
    // g2.dispose();
    //g2.drawImage(bimage,0,0,null);
  }

  public Dimension getPreferredSize() {
    int size=(int)cellVal.length*length+1;
    return new Dimension(size, size);
  }
  public Dimension getMinimumSize() {
    int size=(int)cellVal.length*length+1;
    return new Dimension(size, size);
    // return new Dimension(100, 100);
  }
  public Dimension getMaximumSize() {
    int size=(int)cellVal.length*length+1;
    return new Dimension(size, size);
  }
}
```

```
/**
 * The robot class.
 *
 *
```

* Created: Tue Mar 22 01:21:51 2005-2006
*
* @author <a href="mailto:Max@CERXX"></a>
* @version 1.0
*/

```
import java.awt.Color;
import java.util.LinkedList;
import java.awt.*;

/**
 * The client-the robot class.
 *
 *
 * Created: Tue Mar 22 01:21:51 2005
 *
 * @author <a href="mailto:Max@CERXX"></a>
 * @version 1.0
 */
public class robot {
   protected double loc_x, loc_y, dest_x, dest_y, s_x,s_y,heading=0,
targetHead;
   // protected float loc_x, loc_y, dest_x, dest_y;,
   // protected float heading=0, targetHead, s_x,s_y;
   //protected double heading=0, targetHead, s_x,s_y;
   protected double speed=0.2d;
   protected boolean collision=false;
   protected simGuiExp sim;
   protected Color xcolor;
   private LinkedList trailList;
   protected static int sensorRange=36;
   protected boolean targetReached=false;
   protected double time2trgt=-1;
   /**
    * Creates a new <code>robot</code> instance.
    *
    */
   public robot() {

   }

   public robot(double loc_x, double loc_y, double speed, double
heading,simGuiExp sim, Color colour){

 this.loc_x=loc_x; this.loc_y=loc_y; this.dest_x=loc_x;
this.dest_y=loc_y;
      this.s_x=loc_x; this.s_y=loc_y;
      this.sim=sim; this.xcolor=colour;
      trailList = new LinkedList();
      this.heading=heading;
   }

   public void update(long elapsedTime){
      Point p = new Point((int)loc_x,(int)loc_y);
      //to draw robot trails uncomment this....
      trailList.addFirst(p);

      //     this.dest_x=Math.round(this.loc_x-
Math.cos(heading)*speed*elapsedTime);
      //this.dest_y=Math.round(this.loc_y-
Math.sin(heading)*speed*elapsedTime);

 this.dest_x=this.loc_x-Math.cos(heading)*speed*elapsedTime;
 this.dest_y=this.loc_y-Math.sin(heading)*speed*elapsedTime;

      // System.out.println("Robots next location before advance
location "+dest_x+" "+dest_y);
      sim.advanceRobot();
int gridx=(int)(loc_x/simGuiExp.length);
      int gridy=(int)(loc_y/simGuiExp.length);
      if (sim.cellVal[gridx][gridy]==0) {
         sim.stop();targetReached=true;time2trgt=sim.simTime;
      }
      //System.out.println("Robots next location after advance
location "+dest_x+" "+dest_y);
   }

   public LinkedList getRobotTrail(){return trailList;}

   public void stop(){
     //  this.dest_x=this.loc_x;
//       this.dest_y=this.loc_y;
      speed=0;
   }

   public void turn(){
      double change=(targetHead-heading)/10;
      // if (Math.abs(this.targetHead-this.heading)>0.1) {
      //
this.heading=(this.heading+change)%(2*Math.PI);
      if (Math.abs(targetHead-heading)>0.2) {
         change=(targetHead-heading)/3;
         if (change>0) {
            // heading=(float)((heading+0.08f)%(2*Math.PI));
            heading=(heading+0.08)%(2*Math.PI);
         } else if(change<0) {
            //heading=(float)((heading-0.08f)%(2*Math.PI));
             heading=(heading-0.08)%(2*Math.PI);
         }
         // this.heading=(this.heading+change)%(2*Math.PI);
         // System.out.println("the first if head...change:
"+change);
      }
      else {
         heading=targetHead;
      }
   }
   public void turnS(){

      if (Math.abs(targetHead-heading)>0.2) {

         if ((targetHead-heading)<0) {
            if ((targetHead+Math.PI)>heading)
               //{heading=(float)((heading-0.08f)%(2*Math.PI));}
             {heading=(heading-0.08)%(2*Math.PI);}
            else
               // {heading=(float)((heading+0.08f)%(2*Math.PI));}
             {heading=(heading+0.08)%(2*Math.PI);}

         } else if ((targetHead-heading)>0) {
            if ((heading+Math.PI)>targetHead)
               // {heading=(float)((heading+0.08f)%(2*Math.PI));}
             {heading=(heading+0.08)%(2*Math.PI);}
            else
               //{heading=(float)((2*Math.PI+heading-
0.08f)%(2*Math.PI));}
             {heading=(2*Math.PI+heading-0.08)%(2*Math.PI);}
         }
         // this.heading=(this.heading+change)%(2*Math.PI);
         // System.out.println("the first if head...change:
"+change);
      }

      else {heading=targetHead;}
   }

   public boolean hasStopped(){
      //return this.speed==0;
      return (this.loc_x==this.dest_x && this.loc_y==this.dest_y);

   }
   public boolean finishedTurn(){
      return (heading==targetHead);}
}
```

The ‘behaviour interface’

```
public interface Behavior {

   public void behavior(robot rob, Arbitrator arbi);

    /**
   * Returns a boolean to indicate if this behavior should take
control of the robot.
   */
   public boolean takeControl();

  /**
  * The code in action() represents the action the robot should
*take when this behavior becomes active.
```

```
 */
   public void action();


   public int getTime2Last();
   public void setTime2Last(int i);

}
```

The 'collision' behaviour class

```
public class collisionBehavior implements Behavior {

   Arbitrator arbi;
   private robot rob;
   protected double ticks=0;
   public int time2Last=5;
   private double oldSpeed;

   public void behavior(robot rob, Arbitrator arbi){
      this.rob=rob; this.arbi=arbi;
   }
 /**
    * Returns a boolean to indicate if this behavior should take
control of the robot.
    */
   public boolean takeControl() {

      if (rob.collision) {
         oldSpeed=rob.speed;
         rob.stop();

         // rob.targetHead=(rob.heading+Math.PI/2)%(2*Math.PI);
         // rob.targetHead=(rob.heading-Math.PI/2)%(2*Math.PI);
        rob.targetHead=Math.random()*(2*Math.PI);

         System.out.println("Robot collided......Robbies' heading
"+rob.heading);
         ticks=time2Last;
         return true;
      }
      else {return false;}
   }

   /**
    * The code in action() represents the action the robot should
take when the     * this behavior becomes active.
    */
   public void action()  {

      if (!rob.finishedTurn()) {
         System.out.println("in Collision routine");
         rob.turnS();
      }
      //if finished turning then go forwards
      else if (!(ticks<0)) {
         rob.speed=oldSpeed;
         ticks=ticks-1;
          System.out.println("in Collision routine driving");
      }
      else {
         arbi.busy=false;
      }
   }

   public int getTime2Last(){
      return time2Last;
   }

 public void setTime2Last(int ticks){
       time2Last=ticks;
   }

   public String toString(){return "Collision Behaviour";}

}
```

The ‘light seeking’ behaviour class

```
public class seekingBehavior implements Behavior {

   private robot rob;
   private Arbitrator arbi;
   private double oldSpeed;

   public int time2Last=-1;

   public void behavior(robot rob, Arbitrator arbi){
      this.rob=rob;
      this.arbi=arbi;
   }
   /**
    * Returns a boolean to indicate if this behavior should take
control of the robot.
    */
   public boolean takeControl() {
      if (!rob.collision) {
         oldSpeed=rob.speed;
         System.out.println("LightSeeking
engaged......................");
         return true;

      }
      else {
         return false;
      }
   }

  /**
    * The code in action() represents the action the robot should
take when the     * this behavior becomes active.
    */

   public void action(){

      rob.stop();
      rob.targetHead=rob.sim.signalDirection(rob);
      if (!rob.finishedTurn()) {

         //System.out.println("Light seeker turning..........Tick:
");//+ticks);
         System.out.println("heading "+rob.heading);
         System.out.println("target heading "+rob.targetHead);
         //rob.turn2Light();
         rob.turnS();
         // ticks=ticks-1;
      }
      else //if (ticks>1) {
      { System.out.println("Finished turning going forwards:
");//+ticks);
      arbi.busy=false;}
   }

   public int getTime2Last(){
      return time2Last;
   }

   public void setTime2Last(int ticks){
       time2Last=-1;
   }

   public String toString(){return "Seeking Behaviour";}


}
```

The ‘drive’ behaviour class.

```
public class driveBehavior implements Behavior {

   private robot rob;
   private Arbitrator arbi;
   private double oldSpeed;
   private double ticks=0;
   public int time2Last=2;
   //private double ticks=-1;

   public void behavior(robot rob, Arbitrator arbi){
      this.rob=rob;
      this.arbi=arbi;
   }
 /**
   * Returns a boolean to indicate if this behavior should take
control of the robot.
   */
   public boolean takeControl() {
      if (!rob.collision &&
rob.targetHead==rob.sim.signalDirection(rob)) {
         //oldSpeed=rob.speed;
         ticks=time2Last;
         System.out.println("Value of drive...true");
         return true;
      }
      else {
          System.out.println("Value of drive...false");
         return false;
      }
   }

    /**
    * The code in action() represents the action the robot should
take when the     * this behavior becomes active.
    */
   public void action(){
      System.out.println("driving...");//+ ticks);
     if (!(ticks<0) && !rob.collision) {
        //rob.speed=oldSpeed;
         rob.speed=0.2;
         System.out.println("Ticks so far driving.... "+ticks);
         // rob.speed=oldSpeed;
         ticks=ticks-1;
      }
     else {
        arbi.busy=false;//
     }
      //rob.speed=oldspeed;
      //rob.forward();
   }

   public int getTime2Last(){
      return time2Last;
   }

   public void setTime2Last(int ticks){
       time2Last=ticks;
   }

   public String toString(){return "Drive Behaviour";}

}
```

The ‘driver’ class.

```
import javax.swing.JTabbedPane;
import java.io.IOException;
import javax.swing.JFrame;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;
import javax.swing.BorderFactory;
import java.awt.BorderLayout;
import javax.swing.ImageIcon;
import java.io.File;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.Component.*;
import java.awt.Color;
import java.awt.Font;
import javax.swing.JLabel;
import javax.swing.*;
import java.awt.*;
//import java.awt.image.BufferedImage;
/**
 * Describe class test here.
 *
 *
 * Created: Wed Feb 16 15:44:43 2005
 *
 * @author <a href="mailto:Max@CERXX"></a>
 * @version 1.0
 */
public final class test {

   /**
    * Creates a new <code>test</code> instance.
    *
    */

   final static JTabbedPane tpane=new JTabbedPane();
   final static JFrame window= new JFrame("Sim");
   final static JPanel panel=new JPanel();
   /**
    * Describe <code>main</code> method here.
    *
    * @param args a <code>String[]</code> value
    */
   public static void main(final String[] args) throws IOException {

      new Grid();
      //File file = new File("jum.gex");
       simGuiExp msim=new simGuiExp(Grid.lightGridGenrtr(9,4));
      //simGuiExp msim=new simGuiExp(fileOps.loadFile(file));

      ImageIcon im=new ImageIcon("frameIco.gif");

      window.setIconImage(im.getImage());
      mainMenu item=new mainMenu(msim);
      window.setJMenuBar(item);
      panel.add(tpane, BorderLayout.EAST);
      window.getContentPane().add(panel);
      tpane.setOpaque(false);
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setLocation(300,300);
      window.pack();
      window.setVisible(true);
   }
}
```

The class for drawing the mapped environments onscreen

```
import java.io.IOException;
import java.text.DecimalFormat;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics2D;
import javax.swing.ImageIcon;
import java.awt.Image;
import java.awt.Color;
import java.awt.BasicStroke;
import java.awt.image.*;
import java.awt.*;

public class Grid {

   private static int cellLength=simGuiExp.length, cells;
   public static Color[] colorHolder=new Color[256];
   private static BufferedImage bimage;
   private static Graphics2D ig;
   private static ImageIcon icon;
   private static Image image=(icon=new
ImageIcon("wall.gif")).getImage();
   private static Image imageGoal=(icon=new
ImageIcon("goal.gif")).getImage();
   private static int ci=0;

   public Grid(){
      cells=simGuiExp.cells;
      //caching of colors for grid
         for (int row=0;row<simGuiExp.cells;row++) {
               for (int column=0; column<simGuiExp.cells; column++)
{
                     float
i=255*((float)(column*column+row*row)/(2*simGuiExp.cells*simGuiExp.c
ells));
```

```
                    //float
i=((float)(column*column+row*row)/(float)(2*simGuiExp.cells*simGuiEx
p.cells));
                      int j=(int)i;
                      //  System.out.println("VAL of 'i' "+j);
                       //  colorHolder[j]=new Color(j,j,240);
                       //int x=(int)(i+0.5);int k=(int)(255%20);
                       //                      colorHolder[j]=new
Color(j,j,200);

                       // colorHolder[j]=new Color(i,i,0.6f);

                       colorHolder[j]=new Color(j,j,180);
                 }
              }
         colorHolder[255]=Color.white;
   bimage=new
BufferedImage(simGuiExp.cells*cellLength,simGuiExp.cells*cellLength,
BufferedImage.TYPE_INT_RGB);

ig=bimage.createGraphics();ig.setRenderingHint(RenderingHints.KEY_AN
TIALIASING,RenderingHints.VALUE_ANTIALIAS_ON);
   cellLength=simGuiExp.length;
   }

   public static int[][] lightGridGenrtr(int x, int y) {
      DecimalFormat decform = new DecimalFormat("00");
      int[][] grid=new int[simGuiExp.cells][simGuiExp.cells];
      System.out.println("Value of simGuiExpCells
"+simGuiExp.cells);

      for (int row=0;row<grid[0].length;row++) {
         for (int column=0; column<grid.length; column++) {

            //   int
radialDistance=(int)Math.abs(Math.round(Math.sqrt((Math.pow((column-
x),2)+Math.pow((row-y),2)))));
            int radialDistance=(column-x)*(column-x)+(row-y)*(row-
y);

            if (column==0 || column==(grid[0].length-1) || row==0 ||
row==(grid[0].length-1) ) {
               grid[column][row]=-2;
            } else {
               grid[column][row]=radialDistance;
            }
            //            System.out.print(" " + grid[column][row]);
         }
         // System.out.println();
      }
      return grid;
   }

   public static Rectangle2D.Double rectangle(int x, int y){
      return new
Rectangle2D.Double(x*cellLength,y*cellLength,cellLength,cellLength);
   }

   public static void drawGrid(Graphics2D g2, int[][] cellVal){
      //  public static Graphics2D drawGrid(Graphics2D g2, int[][]
cellVal){
    //   ImageIcon icon = new ImageIcon("wall.gif");
//       Image image = icon.getImage();
long start = System.currentTimeMillis();

      int goalx=1, goaly=1;
      int j=0;
      for (int row=0;row<cellVal.length;row++) {
         for (int column=0; column<cellVal.length; column++) {

            if (cellVal[column][row]!=-2) {
            float
i=255*(float)cellVal[column][row]/(2*simGuiExp.cells*simGuiExp.cells
);
            j=(int)i;
              while (j<colorHolder.length && colorHolder[j]==null) {
j++;}
              if (j>(colorHolder.length-1)) {
                 j=255;
              }
             }

            if (cellVal[column][row]<-1) {
               //g2.drawImage(image, column*cellLength,
row*cellLength,cellLength,cellLength, null);
               ig.drawImage(image, column*cellLength,
row*cellLength,cellLength,cellLength, null);
            }

            else if (cellVal[column][row]!=0){

                  ig.setColor(colorHolder[j]);

ig.fillRect(column*cellLength,row*cellLength,cellLength,cellLength);

//ig.drawRect(column*cellLength,row*cellLength,cellLength,cellLength
);
            }else {

               ig.setColor(colorHolder[j]);

ig.fillRect(column*cellLength,row*cellLength,cellLength,cellLength);
               goalx=column; goaly=row;
            }
         }
      }
      //   ig.setColor(new Color(0,0,240,60));
      //ig.fillRect(0,0,cellLength*cells,cellLength*cells);
```

```
 ig.setStroke(new BasicStroke(1.5f));
      ig.setColor(Color.red);

      ig.drawOval((int)((goalx-1)*cellLength),(int)((goaly-
1)*cellLength),3*cellLength,3*cellLength);
      //ig.fillOval((int)((goalx-0.5)*cellLength+1),(int)((goaly-
0.5)*cellLength+1),2*cellLength,2*cellLength);
     if (ci==0) {ci=1;
      ig.setColor(Color.orange);
      }else {
         ci=0;
         ig.setColor(Color.white);
      }
      ig.drawOval((int)((goalx-0.5)*cellLength),(int)((goaly-
0.5)*cellLength),2*cellLength,2*cellLength);
      ig.setColor(Color.yellow);

ig.drawOval((int)(goalx*cellLength),(int)(goaly*cellLength),cellLeng
th,cellLength);


      // ig.drawOval((int)(goalx-0.5)*cellLength,(int)(goaly-
0.5)*cellLength,3*cellLength,3*cellLength);
      // ig.drawImage(imageGoal, (int)(goalx*cellLength-
imageGoal.getWidth(null)/2+0.5*cellLength),(int)(goaly*cellLength-
imageGoal.getHeight(null)/2+0.5*cellLength),Color.blue,null);

 g2.drawImage(bimage,0,0,null);

      //    g2.dispose();
long time = System.currentTimeMillis() - start;
System.out.println("elapsed time drawgrid "+time);
//return ig;
      }

public static BufferedImage gridDrawing(int[][] cellVal){
    //   ImageIcon icon = new ImageIcon("wall.gif");
//       Image image = icon.getImage();
BufferedImage bufmage=new
BufferedImage(simGuiExp.cells*cellLength,simGuiExp.cells*cellLength,
BufferedImage.TYPE_INT_RGB);Graphics2D
igs=bufmage.createGraphics();igs.setRenderingHint(RenderingHints.KEY
_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_ON);
long start = System.currentTimeMillis();


      int goalx=0, goaly=0;
      int j=0;
      for (int row=0;row<cellVal.length;row++) {
         for (int column=0; column<cellVal.length; column++) {
            //divide all values by maximum val of range and X 250
            // double
colsc=((double)(grfxSquare[column][row].range_class)/26)*250;
            if (cellVal[column][row]!=-2) {
```

```
            float
i=255*(float)cellVal[column][row]/(2*simGuiExp.cells*simGuiExp.cells
);
            j=(int)i;
              while (j<colorHolder.length && colorHolder[j]==null) {
j++;}
              if (j>(colorHolder.length-1)) {
                 j=255;
              }
             }
            if (cellVal[column][row]<-1) {

               //g2.drawImage(image, column*cellLength,
row*cellLength,cellLength,cellLength, null);
               igs.drawImage(image, column*cellLength,
row*cellLength,cellLength,cellLength, null);
            }
            else if (cellVal[column][row]!=0){

               //g2.setPaint(colorHolder[j]);
               //
g2.fillRect(column*cellLength,row*cellLength,cellLength,cellLength);
               igs.setColor(colorHolder[j]);

igs.fillRect(column*cellLength,row*cellLength,cellLength,cellLength)
;
            }else {
               igs.setColor(colorHolder[j]);

igs.fillRect(column*cellLength,row*cellLength,cellLength,cellLength)
;
               goalx=column; goaly=row;
            }
         }
      }
      //System.out.println("time for figuring colors..."+(
System.currentTimeMillis() - start));
     //  g2.setStroke(new BasicStroke(1.5f));
//       g2.setColor(Color.red);
//       g2.drawOval((int)(goalx-0.5)*cellLength,(int)(goaly-
0.5)*cellLength,3*cellLength,3*cellLength);
//       g2.drawImage(imageGoal, (int)(goalx*cellLength-
imageGoal.getWidth(null)/2+0.5*cellLength),(int)(goaly*cellLength-
imageGoal.getHeight(null)/2+0.5*cellLength),Color.blue,null);

      igs.setStroke(new BasicStroke(1.5f));
      igs.setColor(Color.red);
      //goalx+(0.5-6*length)
      igs.drawOval((int)((goalx-1)*cellLength),(int)((goaly-
1)*cellLength),3*cellLength,3*cellLength);

      igs.drawOval((int)((goalx-0.5)*cellLength),(int)((goaly-
0.5)*cellLength),2*cellLength,2*cellLength);
        ig.setColor(Color.yellow);
```

```
igs.fillOval((int)(goalx*cellLength),(int)(goaly*cellLength),cellLen
gth+1,cellLength+1);

      // igs.drawImage(imageGoal, (int)(goalx*cellLength-
imageGoal.getWidth(null)/2+0.5*cellLength),(int)(goaly*cellLength-
imageGoal.getHeight(null)/2+0.5*cellLength),Color.blue,null);

 //g2.drawImage(bimage,0,0,null);

      //    g2.dispose();
long time = System.currentTimeMillis() - start;
//System.out.println("elapsed time drawgrid "+time);
return bufmage;
      }
}
```

The class for store/load operations of simulator worlds or maps.

```
import java.io.File;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.*;
/**
 * Class for storing and retrieving map data
 *
 *
 * Created: Wed Feb 16 15:44:43 2005
 *
 * @author <a href="mailto:Max@CERXX"></a>
 * @version 1.0
 */
public final class fileOps {

   /**
    * Creates a new <code>test</code> instance.
    *
    */

   private static int gridSize=simGuiExp.cells;

   public static void setFileGridSize(int cells){
      gridSize=cells;
}
```

```
   public static boolean saveFile(int[][] grid, File file) {

      try {
      OutputStream file2 = new FileOutputStream(file.getName());
      OutputStream buffer = new BufferedOutputStream(file2);
      ObjectOutputStream out=new ObjectOutputStream(buffer);
      //ObjectOutputStream out=new ObjectOutputStream(new
FileOutputStream(file.getName()));

      for (int row=0;row<grid[0].length;row++) {
         for (int column=0; column<grid.length; column++) {
            out.writeInt(grid[column][row]);
         }
      }
      System.out.println("File Saved");

      out.close();
      return true;

      } catch (IOException e) {
         System.out.println(e);
         return false;
      }
   }

   public static boolean fileSizeSetter(File file){
       int countCheck=0;int[][] resgrid=new int[gridSize][gridSize];
      try {
      InputStream file2 = new FileInputStream(file.getName());
      InputStream buffer = new BufferedInputStream(file2);
      ObjectInputStream in=new ObjectInputStream(buffer);

      //ObjectInputStream in=new ObjectInputStream(new
FileInputStream(file.getName()));

      while (true) {

         int i=(int)in.readInt();
          countCheck++;
         if (i<-2) {return false;}
         }
      }
      catch (EOFException e) {
          System.out.println("cells in file: "+countCheck);}
      catch (Exception e) {
         System.out.println(e);}
     gridSize=(int)Math.sqrt(countCheck);
     //checking size of supposed map file to load...
     if (gridSize>=10 && gridSize%2==0) {
         simGuiExp.cells=gridSize;
         new Grid();
         return true;
     }
     else {
        return false;
     }
 }

   public static int[][] loadFile(File file){
      if (!fileSizeSetter(file)) {
         return null;
      }

      int countCheck=0;
      int[][] resgrid=new int[gridSize][gridSize];

      try {
      InputStream file2 = new FileInputStream(file.getName());
      InputStream buffer = new BufferedInputStream(file2);
      ObjectInputStream in=new ObjectInputStream(buffer);

      //ObjectInputStream in=new ObjectInputStream(new
FileInputStream(file.getName()));

      for (int row=0;row<resgrid.length;row++) {

         for (int column=0; column<resgrid.length; column++) {
            countCheck++;
            resgrid[column][row]=(int)in.readInt();

            // System.out.print(" " + resgrid[column][row]);
         }
         //System.out.println("");
      }

      //simGuiExp.cells=gridSize;
      in.close();
      System.out.println("File Loaded "+gridSize);

      return resgrid;}

      catch (Exception e) {
         System.out.println(e);return null;}

   }
}
```

The graphical user interface

```
import java.io.File;
import java.awt.BorderLayout;
import java.awt.Image;
import java.awt.Cursor;
import java.awt.Toolkit;
import java.awt.Point;
import java.awt.Dimension;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.KeyEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JToolBar;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JMenu;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.ButtonGroup;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.ImageIcon;
import javax.swing.KeyStroke;
import javax.swing.JPanel;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JCheckBox;
import javax.swing.JSlider;
import javax.swing.border.TitledBorder;
import javax.swing.BorderFactory;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;
import javax.swing.border.EtchedBorder;
import javax.swing.JSpinner;
import javax.swing.border.Border;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import javax.swing.JLabel;
import javax.swing.BoxLayout;
//import javax.swing.FlowLayout
import java.awt.FlowLayout;
import javax.swing.JSeparator;
import javax.swing.*;
// import java.awt.image.BufferedImage;
// import java.awt.image.BufferStrategy;

public class mainMenu extends JMenuBar {

   private final JMenuItem lgt,itemOb, saveMap, mapEditor,
clearMap,loadImMap;//
   private final JCheckBoxMenuItem timeStpChbox, trailChbox,
dynmWld;
   private JPanel timeStpPanel = new JPanel(new BorderLayout());
   private final JToolBar toolbar;
   private JButton playButton;
   private JSpinner spinner;
   private final JCheckBox filterBox, gridLock;
   private final JSlider gridSize=new JSlider(JSlider.VERTICAL,10,
100, simGuiExp.cells);
```

```
   private final Border loweredetched;

   private final simGuiExp msim;
   private simWorldCreator msim2=new simWorldCreator();
   //private final simWorldCreator msim2=new simWorldCreator();

   private ImageIcon goRoll, pause, go;
   private boolean mapping=false;
   // private JPanel mapPanel;
   //Image cursed=Animator.drawImage();

   Image hell=new ImageIcon("lightSourceCursor.gif").getImage();
   Cursor source =
Toolkit.getDefaultToolkit().createCustomCursor(hell,new Point(0,0),
"something");

   public mainMenu(simGuiExp msim) {
      this.msim=msim;
      //this.msim2=msim2;

      JMenu robotMenu = new JMenu("Robot");
      JMenu simMenu = new JMenu("Simulation");
      simMenu.setMnemonic(KeyEvent.VK_S);
      JMenu otherMenu = new JMenu("Map");
      otherMenu.setMnemonic(KeyEvent.VK_M);
      JMenu subMenu = new JMenu("Light type");
      final SpinnerModel model = new
SpinnerNumberModel(msim.timeSize, //initial value
                                                         15, //min
                                                         150, //max
                                                         1);
//step
      // JMenu subMenu2 = new JMenu("Obstacle unit erase/add");
      //mainMenu.this.spinner3;// = new JSpinner(model);
      gridSize.setMajorTickSpacing(10);
      gridSize.setMinorTickSpacing(1);
      gridSize.setPaintTicks(true);
      gridSize.setPaintLabels(true);
      Border raisedbevel = BorderFactory.createEmptyBorder(2,2,2,2);

      loweredetched =
BorderFactory.createEtchedBorder(EtchedBorder.LOWERED, Color.gray,
Color.white);

      TitledBorder
bord=BorderFactory.createTitledBorder(loweredetched,"World
size",TitledBorder.CENTER,TitledBorder.DEFAULT_POSITION,new
Font("SansSerif",Font.ITALIC+Font.BOLD,12),Color.blue);
      gridSize.setBorder(bord);

      gridSize.addChangeListener(new ChangeListener(){
            public void stateChanged(ChangeEvent e) {

               JSlider source = (JSlider)e.getSource();
               int size = (int)source.getValue();
               if (!source.getValueIsAdjusting()) { //done adjusting

                  simGuiExp.cells=size;
                  new Grid();
                  //    int[][]
tmpCells2=mainMenu.this.msim2.cellVal;
                  int tabCount=test.tpane.getTabCount();

                  while (tabCount>0) {
                     test.tpane.removeTabAt(0);
                     tabCount--;
                  }
                  //   test.tpane.removeTabAt(0);
                  //                   test.tpane.removeTabAt(0);
                  mainMenu.this.msim=new
simGuiExp(Grid.lightGridGenrtr(9,4));
                  startMapEditor();
                  revalidate();
                  repaint();
                  test.window.pack();
               }//adjusting
            }
         });
      ActionListener typeListener = new ActionListener(  ) {
            public void actionPerformed(ActionEvent event) {
               if (event.getActionCommand()=="Light source") {
                  mainMenu.this.msim2.selectedSource=-1;
                  //mainMenu.this.itemlight.setSelected(true);
                  lgt.setSelected(true);

mainMenu.this.msim2.setCursor(Cursor.getPredefinedCursor(Cursor.DEFA
ULT_CURSOR));
                  //mainMenu.this.msim2.setCursor(source);
                  // mainMenu.this.msim2.setCursor(invisibleCursor);
                  // System.out.println(simGuiExp.selectedSource);
               }
               else if (event.getActionCommand()=="Obstacle")
               {

mainMenu.this.msim2.setCursor(Cursor.getPredefinedCursor(Cursor.CROS
SHAIR_CURSOR));
                  mainMenu.this.msim2.selectedSource=-2;

System.out.println(mainMenu.this.msim2.selectedSource);
                  //mainMenu.this.itemObst.setSelected(true);
                  itemOb.setSelected(true);
               }
               else if (event.getActionCommand()=="Frame Eraser")  {
                  System.out.println("-3");
                  mainMenu.this.msim2.selectedSource=-3;
               }
               else if (event.getActionCommand()=="Cell Eraser")  {
                  System.out.println("-4");
                  mainMenu.this.msim2.selectedSource=-4;
               }
               //  else if (event.getActionCommand()=="Filter")  {
               //                  System.out.println("-5");
               //
mainMenu.this.msim2.selectedSource=-5;
               //                }
               else if (event.getActionCommand()=="Clear Map")  {
                  System.out.println("-6");
                  mainMenu.this.msim2.selectedSource=-6;
                  startMapEditor();
                  repaint();
               }
            }
         };

      ActionListener mapSaveListener = new ActionListener(  ) {
            public void actionPerformed(ActionEvent event) {
               mainMenu.this.msim.stop();
               playButton.setIcon(pause);
               //if (mainMenu.this.msim2!=null) {

               JFileChooser chooser = new JFileChooser();
               int returnVal =
chooser.showSaveDialog(mainMenu.this);
               if (returnVal == JFileChooser.APPROVE_OPTION) {
                  File file = chooser.getSelectedFile();
                  //   File directory=chooser.getCurrentDirectory();
            //       String
fileName=directory.getName()+File.pathSeparator+file.getName();
//                 File file3=new File(fileName);
//                 System.out.println("saving: "+file3.getName());
                  System.out.println("saving: "+file.getName());
                  //  System.out.println("current directory:
"+directory.getName());

//fileOps.saveFile(simWorldCreator.tmpCellVal,file3);
                  fileOps.saveFile(simWorldCreator.tmpCellVal,file);
               }else {
                  JOptionPane.showMessageDialog(mainMenu.this,
"Warning", "Nothing saved", JOptionPane.ERROR_MESSAGE);
               }

            }
         };

      ActionListener mapStartListener=new ActionListener( ){
            public void actionPerformed(ActionEvent event) {
               mainMenu.this.mapping=!mainMenu.this.mapping;
               // saveMap.setEnabled(true);
               if (mainMenu.this.mapping) {
                  mapEditor.setText("Quit Map editor");
                  saveMap.setEnabled(true);
                  loadImMap.setEnabled(true);
                  startMapEditor();
                  gridSize.setValue(mainMenu.this.msim2.cells);
                  revalidate();
                  test.window.pack();
                  //when loading new board apply garbage
collection...
                  System.gc();
               }else {
                  saveMap.setEnabled(false);
                  clearMap.setEnabled(false);
                  loadImMap.setEnabled(false);
                  mapEditor.setText("Map editor");
                  test.tpane.removeTabAt(1);
                  revalidate();
                  test.window.pack();
                  System.gc();
                  //mainMenu.this.msim2=null;
               }
            }
         };
      ActionListener mapLoadListener = new ActionListener(  ) {
            public void actionPerformed(ActionEvent event) {
               //trailChbox.setSelected(false);
               //dynmWld.setSelected(false);
               JFileChooser chooser = new JFileChooser();
               int returnVal =
chooser.showOpenDialog(mainMenu.this);
               mainMenu.this.msim.stop();
               playButton.setIcon(goRoll);
               File file = chooser.getSelectedFile();
               int[][] grid=fileOps.loadFile(file);

               if (returnVal == JFileChooser.APPROVE_OPTION &&
grid!=null){//fileOps.loadFile(file)!=null) {
                      gridSize.setValue(simGuiExp.cells);
                  int tabCount=test.tpane.getTabCount();

                  while (tabCount>0) {
                     test.tpane.removeTabAt(0);
                     tabCount--;
                  }
                  //test.tpane.removeTabAt(0);
                  //  setFileGridSize(mainMenu.this.msim.cells)

                  mainMenu.this.msim=new
simGuiExp(grid);//fileOps.loadFile(file));
                  mainMenu.this.msim.simTime=0;
                  playButton.setEnabled(true);
                  //when loading new board apply garbage
collection...
                  if (mainMenu.this.mapping) {
                     //System.out.println("map editor...");
                     startMapEditor();
                     }
                  test.tpane.setSelectedIndex(0);
                  revalidate();
                  repaint();
                  test.window.pack();
                  System.gc();
               }else {
                  JOptionPane.showMessageDialog(mainMenu.this,
"Warning!...Danger!", "Nothing loaded", JOptionPane.ERROR_MESSAGE);

               }
            }
         };

      JMenuItem item;
      robotMenu.add(item = new JMenuItem("Set robot sensor range"));
      item.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
               for (int i=0;i<mainMenu.this.msim.robot.length;i++) {
                  mainMenu.this.msim.robot[0].sensorRange=50;}
               mainMenu.this.msim.animate.repaint();
            }
         });

      robotMenu.add(item = new JMenuItem("Quit"));
      item.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){

               System.exit(0);
            }
         });



      simMenu.add(item = new JMenuItem("(Re)-Start simulation",new
ImageIcon("sim.gif")));

      item.setAccelerator(KeyStroke.getKeyStroke('T',

Toolkit.getDefaultToolkit( ).getMenuShortcutKeyMask( ), true));
      item.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
               test.tpane.removeTabAt(0);
               mainMenu.this.msim.stop();
               playButton.setEnabled(true);
               mainMenu.this.msim=mainMenu.this.msim.resetSim();
               playButton.setIcon(goRoll);
               gridSize.setEnabled(true);
               revalidate();
               repaint();
               test.tpane.setSelectedIndex(0);
               test.window.pack();
            }
         });
      final JMenuItem itemPlay;
      simMenu.add( itemPlay= new JMenuItem("Resume simulation"));

      itemPlay.setAccelerator(KeyStroke.getKeyStroke('P',

Toolkit.getDefaultToolkit( ).getMenuShortcutKeyMask( ), true));
      itemPlay.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
               if (mainMenu.this.msim.running) {
                  playButton.setIcon(goRoll);
                  gridSize.setEnabled(true);
                  itemPlay.setText("Resume simulation");
                  mainMenu.this.msim.stop();
                  revalidate();
               }
               else {
                  playButton.setIcon(pause);
                  itemPlay.setText("Pause simulation");
                  gridSize.setEnabled(false);
                  mainMenu.this.msim.start();
                  revalidate();
               }
            }
         });
      simMenu.add(item = new JMenuItem("Configure Behaviours"));

      item.setAccelerator(KeyStroke.getKeyStroke('B',

Toolkit.getDefaultToolkit( ).getMenuShortcutKeyMask( ), true));
      item.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
               JPanel pan=configBehaviours();
               mainMenu.this.msim.stop();
               playButton.setEnabled(false);
               test.tpane.removeTabAt(0);
               test.tpane.insertTab("Configuring Behaviours",new
ImageIcon("simSmall.gif"),pan,null,0);
               //test.panel.add(pan);
                revalidate();
               repaint();
               test.window.pack();
               test.tpane.setSelectedIndex(0);
            }
         });

      simMenu.addSeparator();
      simMenu.add(timeStpChbox = new JCheckBoxMenuItem("Time step
size"));
      timeStpChbox.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e) {

               if (timeStpChbox.isSelected()) {

                  Integer tmpValue = new
Integer(mainMenu.this.msim.timeSize);
```

```
                  JLabel label2 = new JLabel("Time step size");

                  if (spinner==null) {
                     //spinner.addChangeListener(listener);
                     timeStpPanel.add(label2, BorderLayout.NORTH);
                     spinner= new JSpinner(model);
                     spinner.setValue(tmpValue);
                     spinner.addChangeListener(new ChangeListener()
{
                           public void stateChanged(ChangeEvent e) {

                              Integer myValue =
(Integer)spinner.getValue();

                              int tSize = myValue.intValue();
                              System.out.println("Source: " +
tSize);
                              mainMenu.this.msim.timeSize=tSize;
                              System.out.println("After change
"+mainMenu.this.msim.timeSize);
                           }
                        });
                     timeStpPanel.add(spinner, BorderLayout.SOUTH);
                     test.panel.add(timeStpPanel);
                     revalidate();
                     repaint();
                     test.window.pack();}
                  //
               }else {
                  //timeStpPanel=null;
                  spinner=null;
                  test.panel.remove(timeStpPanel);
                  // test.panel.remove();
                  //timeStpPanel.setVisible(false);
                  timeStpPanel.invalidate();
                  test.panel.validate();//add(timeStpPanel);
                  //revalidate();
                  repaint();
                  test.window.pack();
               }
            }
         });

      simMenu.add(trailChbox = new JCheckBoxMenuItem("Draw trail"));
      trailChbox.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e) {
               if (trailChbox.isSelected())
{Animator.drawTrail=true;}
               else {
                  Animator.drawTrail=false;
mainMenu.this.msim.animate.clear();
               }
            }
         });
      simMenu.add(dynmWld=new JCheckBoxMenuItem("World twister"));
      dynmWld.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e) {
               if (dynmWld.isSelected())
{mainMenu.this.msim.worldShiftThread();}
               else {
                  mainMenu.this.msim.randomizer=false;
               }
            }
         });

      otherMenu.add(mapEditor = new JMenuItem("Map editor",new
ImageIcon("map.gif")));
      mapEditor.addActionListener(mapStartListener);
      mapEditor.setAccelerator(KeyStroke.getKeyStroke('M',

Toolkit.getDefaultToolkit(  ).getMenuShortcutKeyMask(  ), true));
      otherMenu.addSeparator( );

      otherMenu.add(clearMap = new JMenuItem("Clear Map", new
ImageIcon("")));
      clearMap.setEnabled(false);
      clearMap.setAccelerator(KeyStroke.getKeyStroke('C',

Toolkit.getDefaultToolkit(  ).getMenuShortcutKeyMask(  ), true));
      clearMap.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
               startMapEditor();
               //  repaint();
}
         });
otherMenu.add(loadImMap = new JMenuItem("Load current map", new
ImageIcon("")));
      loadImMap.setEnabled(false);
      loadImMap.setAccelerator(KeyStroke.getKeyStroke('I',

Toolkit.getDefaultToolkit(  ).getMenuShortcutKeyMask(  ), true));
      loadImMap.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                  mainMenu.this.msim.stop();

                     test.tpane.removeTabAt(0);
                  mainMenu.this.msim=new
simGuiExp(simWorldCreator.tmpCellVal);
                  mainMenu.this.msim.simTime=0;
                  playButton.setIcon(goRoll);
                  playButton.setEnabled(true);

                  // repaint();
            }
         });

      loadImMap.setEnabled(false);
      otherMenu.addSeparator( );
      otherMenu.add(item = new JMenuItem("Load map from file", new
ImageIcon("load.gif")));
      item.setAccelerator(KeyStroke.getKeyStroke('L',

Toolkit.getDefaultToolkit( ).getMenuShortcutKeyMask( ), true));
      item.addActionListener(mapLoadListener);

      otherMenu.add(saveMap = new JMenuItem("Save map", new
ImageIcon("save.gif")));
      saveMap.setEnabled(false);
      saveMap.setAccelerator(KeyStroke.getKeyStroke('S',

Toolkit.getDefaultToolkit( ).getMenuShortcutKeyMask( ), true));
      saveMap.addActionListener(mapSaveListener);

      toolbar=new JToolBar();
      toolbar.setOrientation(JToolBar.HORIZONTAL);

      //toolbar.setBorder(loweredetched);
      //toolbar.add(Box.createRigidArea(new Dimension(200, 15)));
      //toolbar.addSeparator();
      //toolbar.setFloatable(false);
      loadImages();
      playButton=createButton("play", "Press to pause or start
simulation");
      playButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){

               test.tpane.setSelectedIndex(0);
               if (mainMenu.this.msim.running) {
                  itemPlay.setText("Resume simulation");
                  playButton.setIcon(goRoll);
                  gridSize.setEnabled(true);
                  mainMenu.this.msim.stop();
                  revalidate();
               }
               else {
                  itemPlay.setText("Pause simulation");
                  playButton.setIcon(pause);
                  gridSize.setEnabled(false);
                  mainMenu.this.msim.start();
                  revalidate();
               }
            }
         });

      // Finally, add all the menus to the menu bar.
      //     add(toolbar);
      add(simMenu);
      add(otherMenu);
      add(robotMenu);
```

```
        add(Box.createRigidArea(new Dimension(5,25)));
        add(playButton);

        JPanel pan=new JPanel();
        //JPanel listPane = new JPanel();
        pan.setLayout(new BoxLayout(pan, BoxLayout.PAGE_AXIS));
        pan.add(Box.createRigidArea(new Dimension(0,15)));
        ButtonGroup buttonGroup = new ButtonGroup();
        toolbar.setLayout(new BoxLayout(toolbar, BoxLayout.X_AXIS));
        //    toolbar.add(item = new JRadioButtonMenuItem("Light
source", true));
        lgt = new JRadioButtonMenuItem("Light source", new
ImageIcon(""),true);
        lgt.addActionListener(typeListener);
        buttonGroup.add(lgt);
        pan.add(lgt);
        // pan.add(Box.createRigidArea(new Dimension(0,15)));

        itemOb = new JRadioButtonMenuItem("Obstacle", new
ImageIcon(""));
        itemOb.addActionListener(typeListener);
        buttonGroup.add(itemOb);
        pan.add(itemOb);
        //pan.add(Box.createRigidArea(new Dimension(0,13)));

        buttonGroup.add(item = new JRadioButtonMenuItem("Frame
Eraser",new ImageIcon("")));
        item.addActionListener(typeListener);
        // buttonGroup2.add(item);
        pan.add(item);

        buttonGroup.add(item = new JRadioButtonMenuItem("Cell
Eraser",new ImageIcon("")));
        item.addActionListener(typeListener);
        //buttonGroup2.add(item);
        pan.add(item);

        filterBox = new JCheckBox("Noise Filter");
        filterBox.addItemListener(new ItemListener(){
              public void itemStateChanged(ItemEvent e) {
                 if (filterBox.isSelected()) {
                    mainMenu.this.msim2.noise=true;
                    mainMenu.this.msim2.noiseMapGen();
                 }
                 else {mainMenu.this.msim2.removeNoise();
                 mainMenu.this.msim2.noise=false;
                 }
              }
           });
        pan.add(filterBox);

        // item.addActionListener(typeListener);
        //       otherMenu.add(item);
        //       //buttonGroup2.add(item);
```

```
      //       pan.add(item);


      gridLock = new JCheckBox("Grid Lock");
      gridLock.addItemListener(new ItemListener(){
            public void itemStateChanged(ItemEvent e) {
               if (gridLock.isSelected()) {
                  gridSize.setEnabled(false);
                  // Animator.drawTrail=true;
               }
               else {
gridSize.setEnabled(true);//Animator.drawTrail=false;
mainMenu.this.msim.animate.clear();
               }
            }
         });
      pan.add(gridLock);

      toolbar.add(pan);
      toolbar.setBorder(mainMenu.this.loweredetched);
      //toolbar.setPreferredSize(new Dimension(150,350));
      //toolbar.add(item2,BorderLayout.SOUTH);
      //toolbar.add(gridSize);
      // toolbar.add(Box.createHorizontalStrut(90));

toolbar.setRollover(true);//toolbar.add(Box.createHorizontalGlue()
);
   }

   private void loadImages() {

      goRoll = new ImageIcon("playbutton.gif");//("goRoll.gif");
      pause = new ImageIcon("pausebutton.gif");
      go = new ImageIcon("go.gif");
   }

   public JPanel configBehaviours(){
      JPanel pan=new JPanel();
      Border bevebord= BorderFactory.createEmptyBorder(10,10,10,10);
      JPanel subPan1=new JPanel();
      JPanel subPan2=new JPanel();
       subPan1.setLayout(new BoxLayout(subPan1, BoxLayout.Y_AXIS));
       subPan2.setLayout(new BoxLayout(subPan2, BoxLayout.Y_AXIS));

      pan.setLayout(new BoxLayout(pan, BoxLayout.Y_AXIS));

      Arbitrator arbit=msim.getArbitrator();
      pan.add(Box.createVerticalGlue());
      for (int i=0;i<arbit.getSize();i++) {
         //         JLabel label=new JLabel(
         SpinnerModel bmodel = new SpinnerNumberModel(msim.timeSize,
//initial value
                                                      0, //min
                                                      100, //max
```

```
                                                            1);
          final JSpinner bSpinner= new JSpinner(bmodel);
          int time=arbit.getBehavior(i).getTime2Last();
          if (time<0) {
             bSpinner.setEnabled(false);
          }

          Integer tmpValue = new Integer(time);
          bSpinner.setValue(tmpValue);
                     bSpinner.addChangeListener(new ChangeListener()
{
                           public void stateChanged(ChangeEvent e) {

                              Integer myValue =
(Integer)bSpinner.getValue();

                              int tSize = myValue.intValue();
                              System.out.println("Source: " +
tSize);
                              // mainMenu.this.msim.timeSize=tSize;
                              //System.out.println("After change
"+mainMenu.this.msim.timeSize);
                           }
                        });
         JPanel subPan=new JPanel();
         // subPan.setLayout(new BoxLayout(subPan,
BoxLayout.X_AXIS));
         //
         JLabel label=new
JLabel(arbit.getBehavior(i).toString(),JLabel.TRAILING);
         subPan.add(label);
         //subPan1.add(Box.createRigidArea(new Dimension(15,0)));
         // subPan1.add(label);

         subPan.add(bSpinner);
         // subPan.add(new
JSeparator(JSeparator.HORIZONTAL),BorderLayout.LINE_START);
         //
         subPan.setBorder(mainMenu.this.loweredetched);
         subPan.add(Box.createVerticalGlue());
         pan.add(subPan);
      }
      //subPan1.add(Box.createVerticalGlue());
      // subPan2.add(Box.createVerticalGlue());
     //  subPan1.setBorder(mainMenu.this.loweredetched);
//       subPan2.setBorder(mainMenu.this.loweredetched);
//       pan.add(subPan1);
//       pan.add(subPan2);
      //     pan.setMinimumSize(new Dimension(350, 325));
      pan.add(Box.createVerticalGlue());

bevebord=BorderFactory.createCompoundBorder(loweredetched,bevebord);
      //bevebord=BorderFactory.createCompoundBorder(bevebord,
loweredetched);
```

```
      pan.setBorder(bevebord);

      //pan.setBorder(mainMenu.this.loweredetched);
      return pan;
   }

   public JButton createButton(String name, String toolTip) {

      // create the button
      JButton button = new JButton();
      button.setIgnoreRepaint(true);
      button.setFocusable(false);
      button.setToolTipText(toolTip);
      button.setBorder(null);
      button.setContentAreaFilled(false);
      //  button.setCursor(cursor);
      button.setIcon(goRoll);
      //button.setRolloverIcon(goRoll);

      return button;
   }

   public void startMapEditor(){
      filterBox.setSelected(false);
      mainMenu.this.msim2=null;
      mainMenu.this.msim2=new simWorldCreator();
      int tabCount=test.tpane.getTabCount();

      if (tabCount>1) {
         test.tpane.removeTabAt(1);
      }
      JPanel mapPanel=new JPanel();//new BorderLayout());
      JSeparator sep=new JSeparator(JSeparator.VERTICAL);

      mapPanel.setBorder(mainMenu.this.loweredetched);
      mapPanel.add(mainMenu.this.msim2,BorderLayout.WEST);
      sep.setBorder(mainMenu.this.loweredetched);
      mapPanel.add(sep,BorderLayout.WEST);
      mapPanel.add(mainMenu.this.toolbar,BorderLayout.WEST);
      mapPanel.add(gridSize,BorderLayout.EAST);
      mapPanel.setBorder(mainMenu.this.loweredetched);
      test.tpane.addTab("Map Editor",mapPanel);
      //  test.tpane.insertTab("Map editor",null,
mainMenu.this.msim2,null,1);
      test.tpane.setSelectedIndex(1);

      clearMap.setEnabled(true);
   }
}
```

Appendix B – User manual

The part of the simulator software coded thus far can be found on the attached cd-rom disk. Under the directory named 'Project' is a java 'jar' file named "My.jar". This software must be run on a computer with Java version 1.4 installed or greater in order to function. For details on installing Java and setting paths details please check details at http://www.sun.com/java.

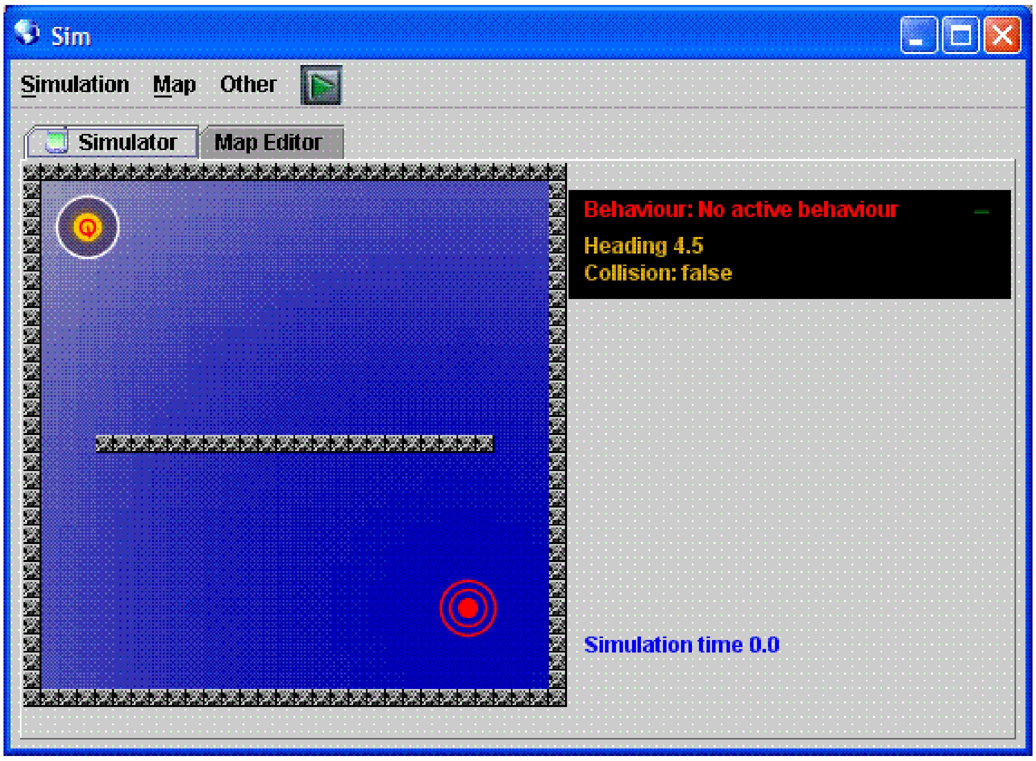
If everything works correctly a graphical user interface will appear. It consists of one tab panel displaying the simulator and a menu bar. The user can run the default simulation or create a new environment first by creating it in the map editor or loading it from a file. The user can access the map editor to create simulator environments by selecting the 'Map editor' under the map bar. The map editor will open up in another tab.

Various editing options are available in the map editor. The user can select to create a simulator world with a source or target for the robot and any obstacles, which are single grid cells the robot cannot travel through. The user can erase/add obstacle, or change the size of the simulator environment. After creating an environment he/she is then able to store this via the save command or by pressing 'Ctrl-S'.

To load this into simulation immediately without saving the user can select to press Ctrl-I or by choosing the load immediate selection under the menu bar. The user can also select to load from a file by selecting the 'load command' or by pressing 'Ctrl-L'. When either of these two actions are performed the simulator will then automatically display the robot in this new environment.

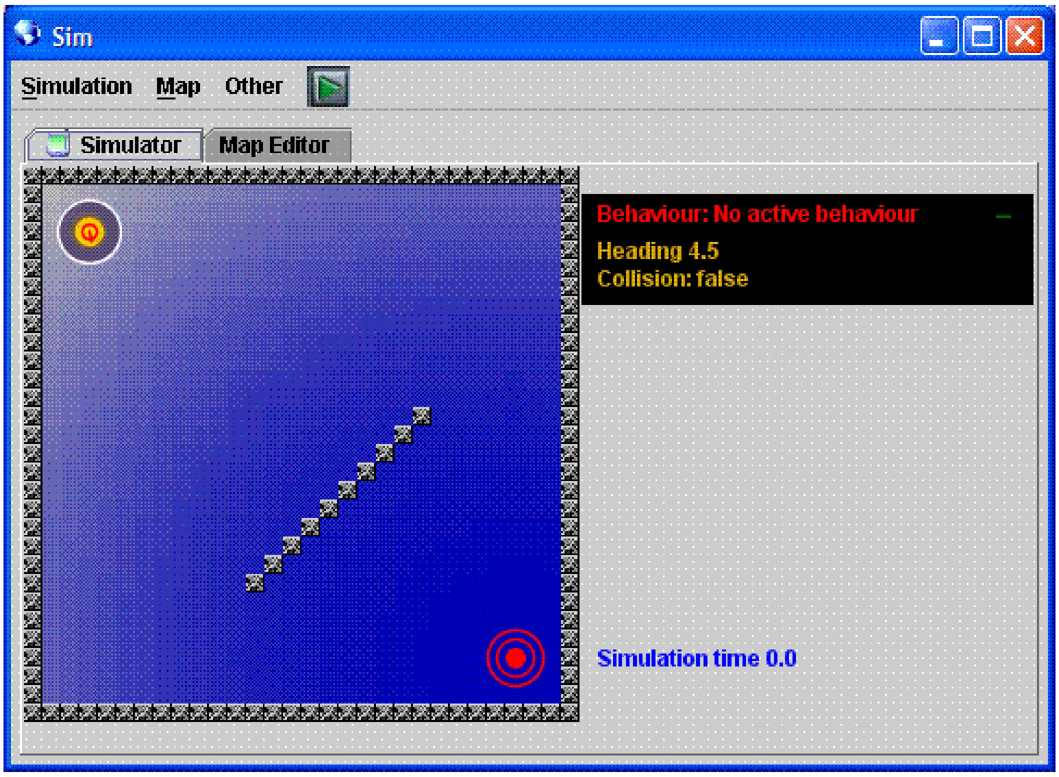
Currently there is only one algorithm hard-coded into the robot. This is the 3rd algorithm mentioned in the evaluation section. To run the simulation the user can either press the 'play' button to the right of the menu bar, pressing Ctrl-P (pause/unpause), or selecting 'Resume simulation' under the menu bar. The simulation can be reset by pressing Ctrl-T or selecting 'Re-Start simulation' from the menu.

Appendix C – Configuration pictures



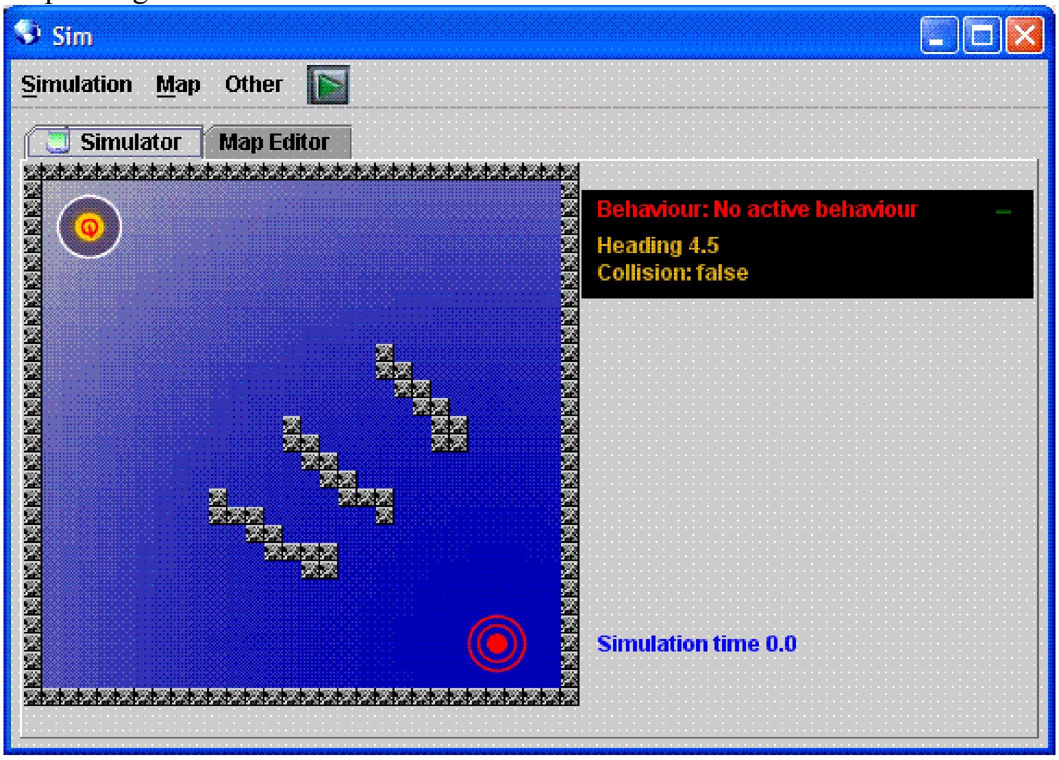
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 1



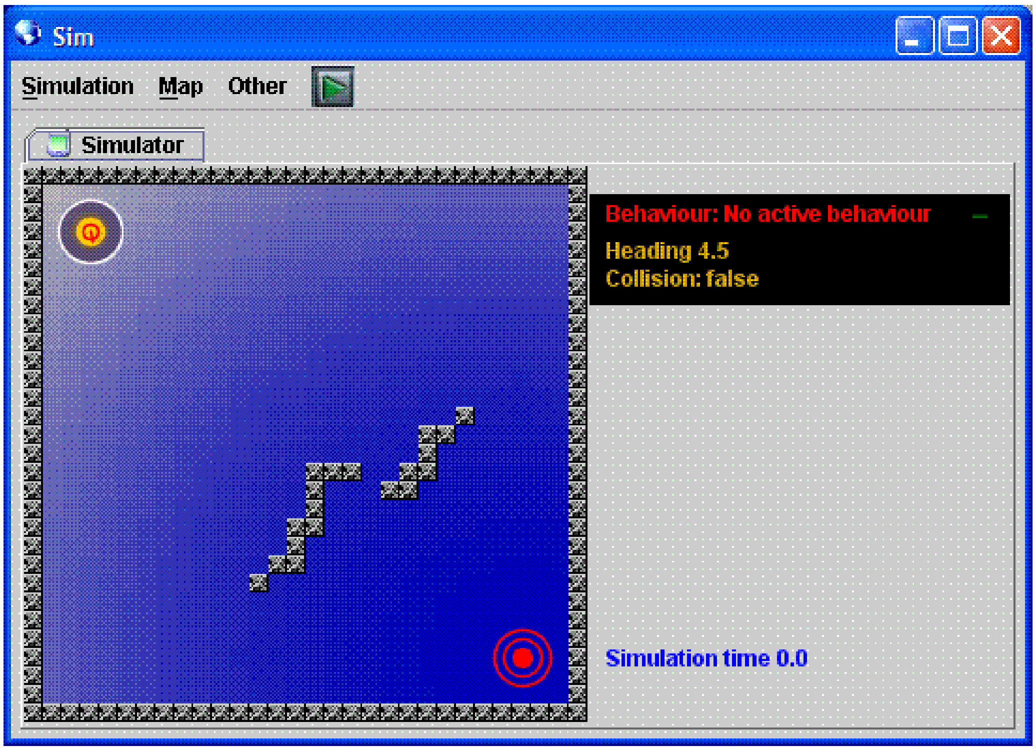
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 2



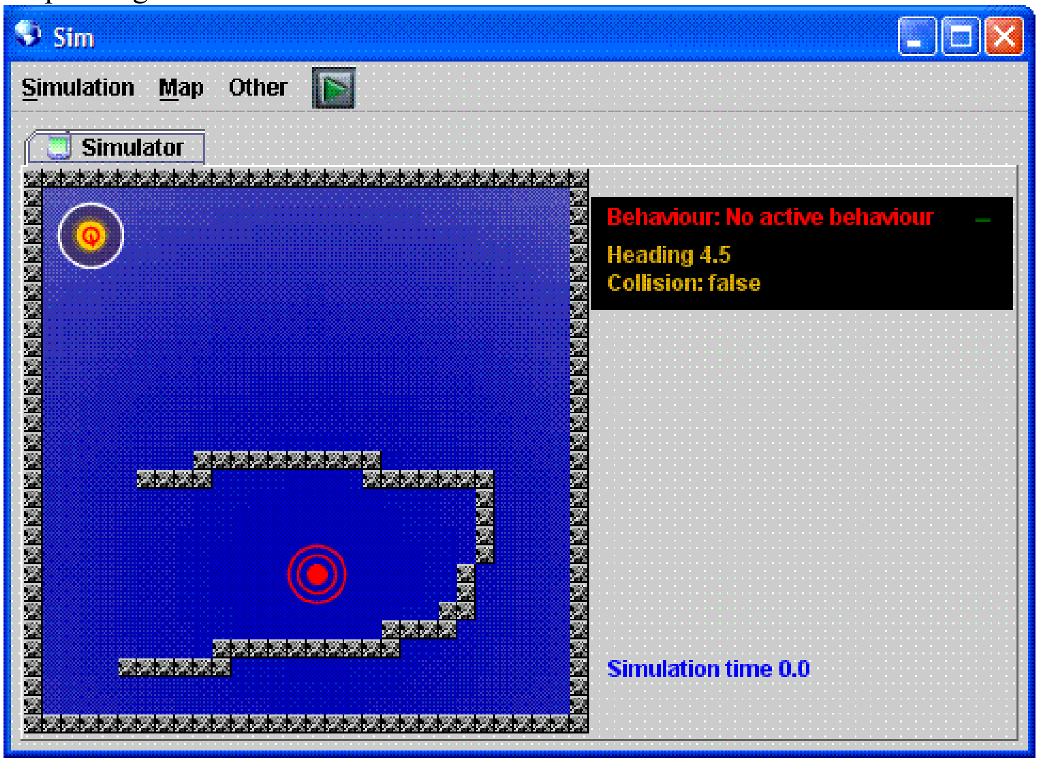
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 3



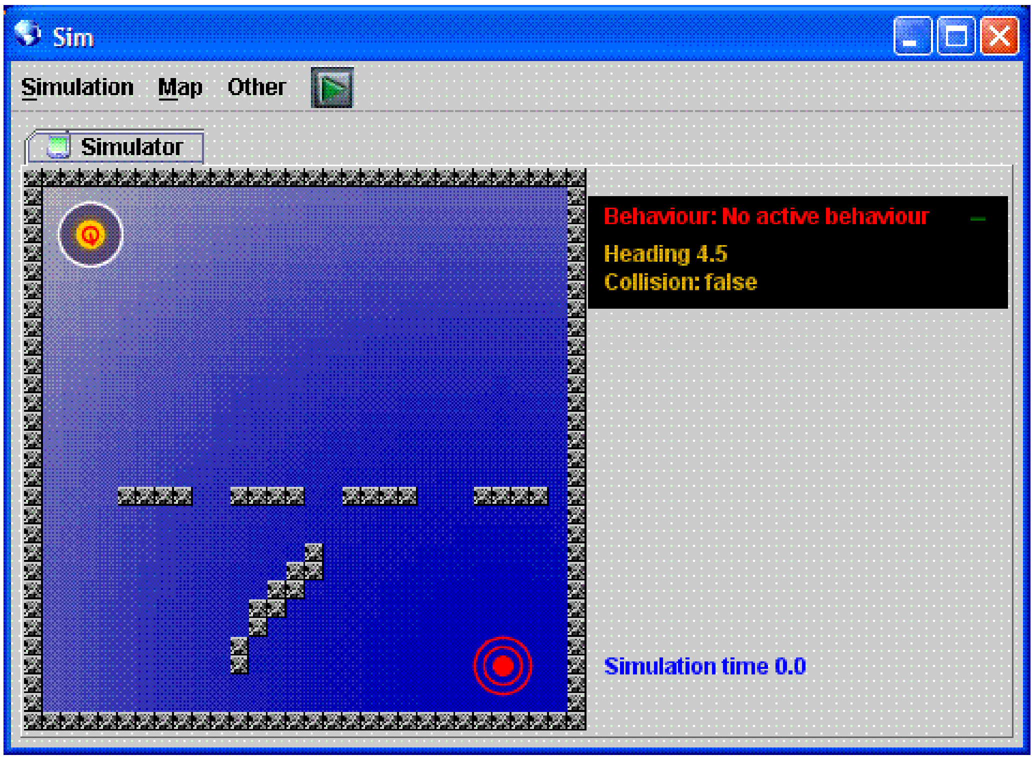
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 4



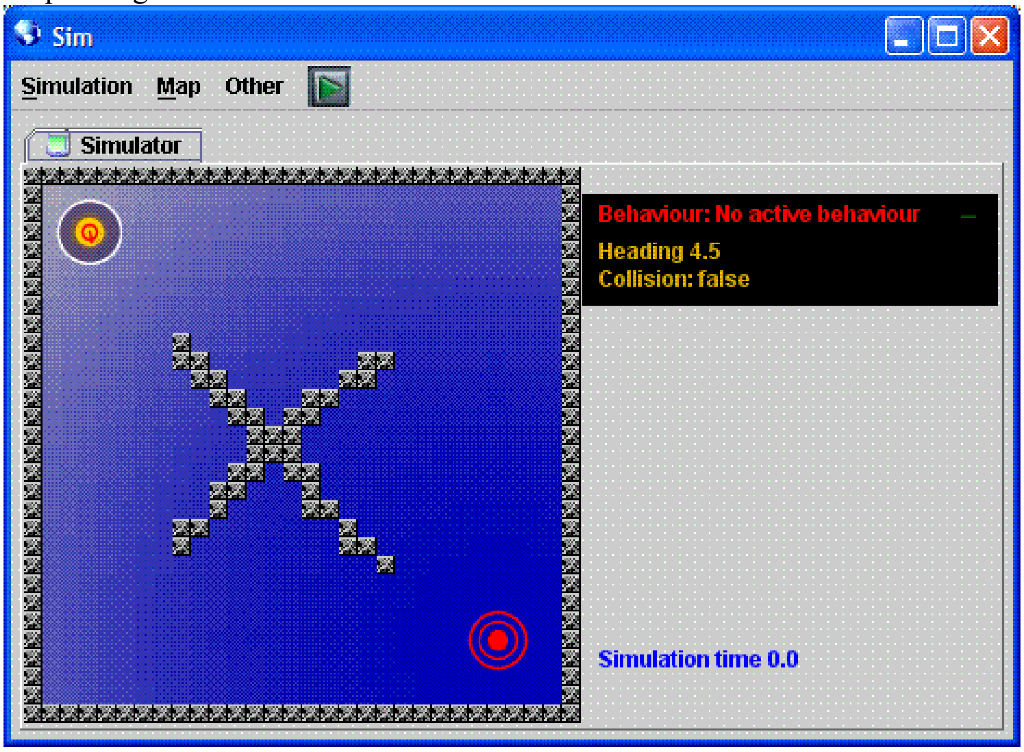
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 5



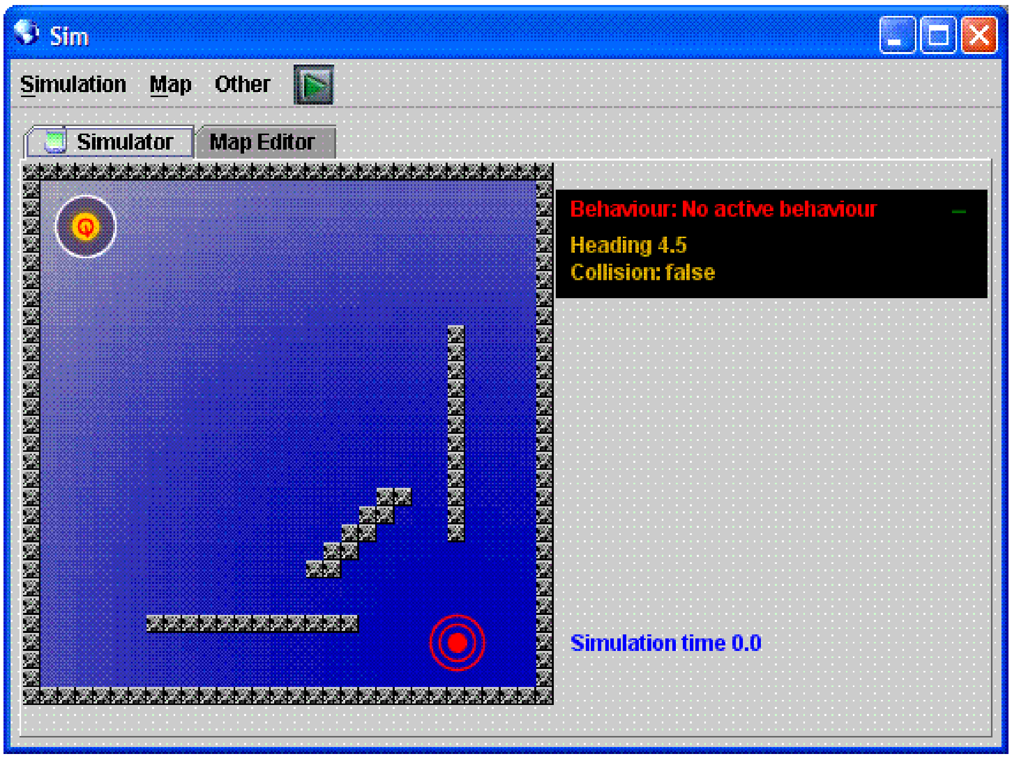
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 6



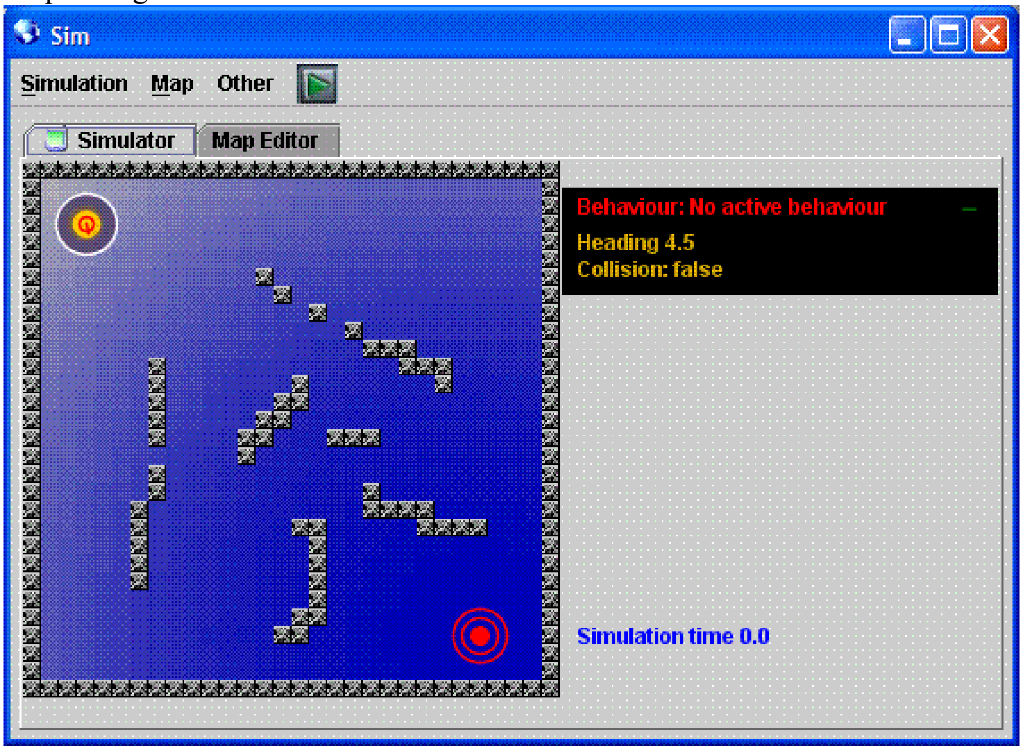
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 7



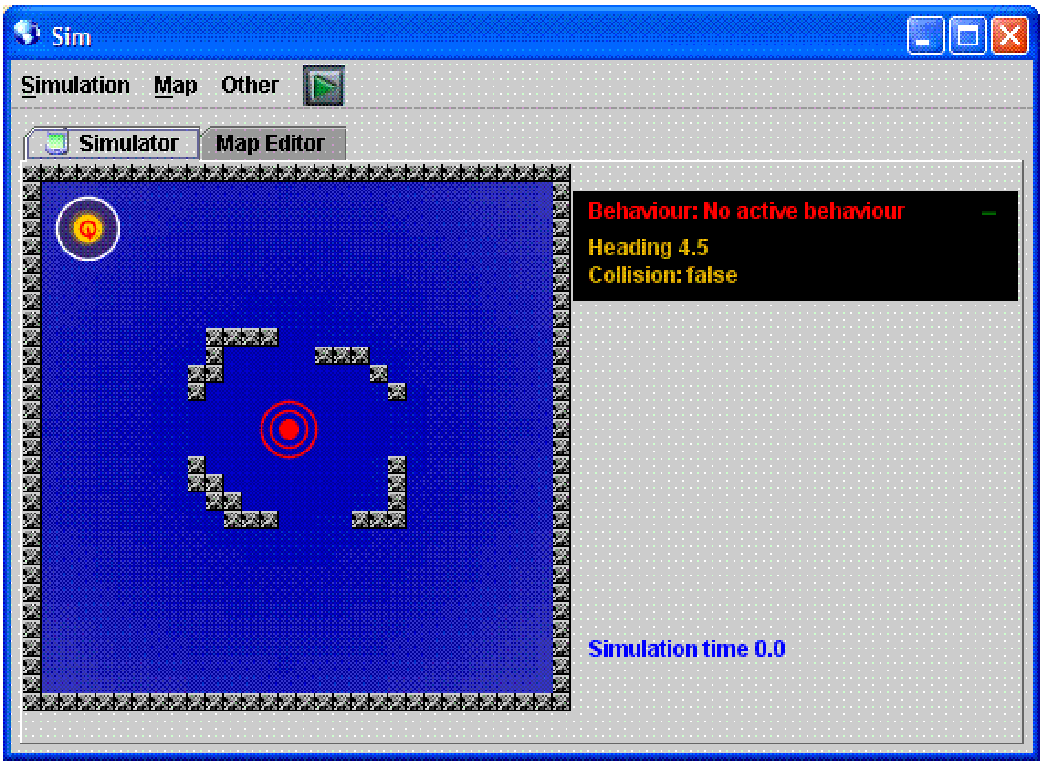
Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 8



Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 9



Behaviour: No active behaviour
Heading 4.5
Collision: false
Simulation time 0.0

Map configuration 10