



UNIVERSITY OF
LIVERPOOL

Richard Laurence Whitty
200502059

Supervisor: Prof. Frank Wolter
Simulation of Turmites

Liverpool, 2009

Abstract

Turmites are an extension of Turing Machines into two dimensions. They differ in that they maintain an orientation of the head and the move commands are replaced with rotations about an axis. This report presents a piece of software to design and experiment with multi-dimensional Turing machines and Turmites. It provides a graphical user interface to design the transition function and visualise their execution in two or three dimensions with a wide range of options customisable at run-time.

Contents

1	Introduction	1
1.1	The Project	1
1.2	Aims and Objectives	2
1.3	Challenges	2
1.4	The Solution	2
2	Background	4
2.1	Background of the problem to be solved	4
2.1.1	What is a Turing Machine?	4
2.1.2	Extending into Two Dimensions	6
2.1.3	Generalising to n Dimensions	7
2.2	Existing Approaches	7
2.3	Research	8
2.4	Requirements	8
3	Design	10
3.1	General Components	10
3.1.1	Matrix	11
3.1.2	Vector	11
3.1.3	Coord	11
3.1.4	Dimensionality	11
3.1.5	The Rotation Class	13
3.1.6	The Grid	13
3.1.7	CoordSet	14
3.2	The Turmites	15
3.2.1	The Transition Function	15
3.3	The View	16
4	Realisation	22
4.1	Hardware and Software Used	22
4.2	Methodology	23

4.3	Changes to the Design	23
4.3.1	User Interface Changes	24
4.3.2	Traditional Turing Machines	24
4.4	Implementation of Components	24
4.4.1	The Grid	24
4.4.2	The View	25
4.4.3	The User Interface	25
4.4.4	The Visualisations	26
4.5	Problems and Solutions	28
4.5.1	Head Orientation	28
4.5.2	Version Control	29
4.5.3	GridView Inheritance Heirachy	30
5	Evaluation	31
5.1	Evaluation Criteria	31
5.2	Assessment	31
5.3	Ideas for Extensions	33
5.3.1	Hybrid Grid	33
5.3.2	Attractor Detection	33
5.3.3	Undo Transition	34
5.3.4	Multiple Turmites	34
5.3.5	Z ordering in GridView	34
6	Learning Points	35
6.1	Writing Defect-Free Concurrent Software	36
7	Professional Issues	37
7.1	Code of Conduct	37
7.1.1	The Public Interest	37
7.1.2	Duty to Relevant Authority	37
7.1.3	Duty to the Profession	37
7.1.4	Professional Competence and integrity	37
7.2	Code of Practice	37
7.2.1	Act Professionally as a Specialist	37
7.2.2	Manage your Workload Efficiently	38
7.2.3	When Managing a Programme of Work	38
7.2.4	When Managing Project Risks	38
7.2.5	When Closing a Project	38
7.2.6	When Designing New Systems	39
7.2.7	When Programming	39

A	Instructions for Compiling and Running the Software	43
A.1	Running	43
A.2	Compiling	43
B	User Guide	44
B.1	Defining a Turmite	44
B.2	Configuration	45
C	Source Listings	46
D	Specification	144
D.1	Project Description	145
D.2	Statement of Deliverables	145
D.2.1	Features	145
D.2.2	Desirable features	146
D.2.3	Experiments	146
D.2.4	Evaluation	146
D.3	Conduct of the Project and Plan	146
D.4	Preparation	146
D.4.1	Design Stage	147
D.4.2	Milestones	147
D.4.3	Hardware to be used	147
D.4.4	Software to be used	147
D.5	Risk assessment	148
D.5.1	Major challenges	148
D.5.2	New skills	148
E	Original Design	149
E.1	abstract	150
E.2	What is a Turmite?	150
E.2.1	What is a Turing Machine?	150
E.2.2	Extending into two dimensions	151
E.2.3	Generalising to n dimensions	152
E.3	Design	152
E.3.1	General Components	152
E.3.2	The Grid	154
E.3.3	The Turmites	155
E.3.4	The View	157
E.4	Evaluation Design	157
E.5	Plan	158

List of Figures

2.1	Simple state machine	4
2.2	A simple Turmite	6
2.3	Its output	7
3.1	Basic Mathematical Constructs	17
3.2	The model classes	18
3.3	Classes used to design the transition functions	19
3.4	Classes used by the user interface during execution.	20
3.5	The different stages of execution	21
E.1	Basic mathematical constructs	159
E.2	The different stages of execution	160

Listings

3.1	Sine and Cosine Implementations from Dimensionality	12
3.2	Givens Rotation Matrix Calculation from Dimensionality	13
3.3	Grid interface	14
3.4	CoordSet	14
4.1	Abridged GridView interface	26
4.2	archive.sh	29
C.1	multidimensional/Factory.java	47
C.2	multidimensional/ITurmite.java	51
C.3	multidimensional/util/HashArray.java	52
C.4	multidimensional/util/CoordSet.java	52
C.5	multidimensional/util/MakeDiagram.java	53
C.6	multidimensional/util/TreeArray.java	55
C.7	multidimensional/util/WrapAroundStack.java	55
C.8	multidimensional/util/ZCoordSet.java	56
C.9	multidimensional/test/Matcher.java	59
C.10	multidimensional/test/SerialTest.java	60
C.11	multidimensional/view/Multitest.java	61
C.12	multidimensional/view/Pallete.java	62
C.13	multidimensional/view/CoordComparator.java	63
C.14	multidimensional/view/CubeRepresentation.java	63
C.15	multidimensional/view/MouseHandler.java	64
C.16	multidimensional/view/GridView2d.java	66
C.17	multidimensional/view/ViewProxy.java	70
C.18	multidimensional/view/ElementRepresentation.java	70
C.19	multidimensional/view/GridView.java	71
C.20	multidimensional/view/GridView3d.java	72
C.21	multidimensional/view/SphereRepresentation.java	76
C.22	multidimensional/gui/ActionDesigner.java	76
C.23	multidimensional/gui/Main.java	79
C.24	multidimensional/gui/widgets/DirectionPicker.java	82
C.25	multidimensional/gui/widgets/InputBox.java	83

C.26 multidimensional/gui/widgets/SpinEdit.java	84
C.27 multidimensional/gui/widgets/FileBox.java	86
C.28 multidimensional/gui/widgets/UndoButton.java	87
C.29 multidimensional/gui/widgets/LabelledComponent.java	88
C.30 multidimensional/gui/widgets/MultipleChoice.java	89
C.31 multidimensional/gui/widgets/DebugResizeListener.java	89
C.32 multidimensional/gui/widgets/SpinEdit2.java	90
C.33 multidimensional/gui/Menu.java	91
C.34 multidimensional/gui/LoadSave.java	98
C.35 multidimensional/gui/MyWindowListener.java	101
C.36 multidimensional/gui/RunPanel.java	102
C.37 multidimensional/gui/ActionList.java	106
C.38 multidimensional/gui/ConfigPanel.java	107
C.39 multidimensional/gui/TransitionListModel.java	109
C.40 multidimensional/basics/Matrix.java	111
C.41 multidimensional/basics/Dimensionality.java	116
C.42 multidimensional/basics/Coord.java	120
C.43 multidimensional/basics/Grid.java	122
C.44 multidimensional/basics/Vector.java	124
C.45 multidimensional/basics/Rotation.java	125
C.46 multidimensional/basics/MatrixTest.java	126
C.47 multidimensional/basics/SimpleTest.java	127
C.48 multidimensional/basics/GridListener.java	127
C.49 multidimensional/Transition.java	128
C.50 multidimensional/TransitionFunction.java	129
C.51 multidimensional/TurmiteException.java	131
C.52 multidimensional/TurmiteListener.java	132
C.53 multidimensional/GridImpl.java	132
C.54 multidimensional/Turmite.java	133
C.55 multidimensional/TransitionFunctionDoubleMap.java	136
C.56 multidimensional/TransitionFunctionHash.java	137
C.57 multidimensional/TuringMachine.java	139
C.58 multidimensional/TurmiteEvent.java	141

Chapter 1

Introduction

This chapter describes what problem the solution addresses and discusses some of the challenges encountered. It then describes the solution and gives a brief statement of its effectiveness.

1.1 The Project

Turing machines are a theoretical model of computation [1]. They are a class of symbol-based state machines into which there has been extensive research.

A turing machine is a finite state automaton with an infinite amount of storage. This takes the form of a tape extending in both directions with a movable head. Using the head the machine can read and write symbols. A transition function determines which action should be taken by the machine by examining the current state of the machine (which turing referred to as the M-configuration) and the scanned symbol. It has been shown that Turing Machines are capable of universal computation, even when the set of possible symbols on the tape is only '1' and '0'.

There have been a number of extensions of these machines, for example adding additional heads and/or tapes and having them operate on a lattice or grid-like structure. The extension this project is concerned with is the Turmite (though it also supports standard Turing machines.)

Turmites operate on a two dimensional grid and instead of specifying the direction to move, the head moves forward one step each iteration and a rotation is applied by the transition rule. The rotation accumulates so if a machine turns left, then left again, on the next step it shall move in the opposite direction to the first.

The problem addressed by this project is to provide a user interface that can be used to define the transition function of Turing Machines and Tur-

mites. However the solution is not limited to two dimensions and can simulate arbitrary systems. Additionally the solution provides graphical visualisations to study Turmites and Turing machines in two and three dimensions.

1.2 Aims and Objectives

The required aims of the project were as follows:

1. To develop a model for specifying and simulating Turing Machines in 2 and more dimensions.
2. To create visualisations for the simulations.
3. To develop efficient data structures to store the data on which they operate.
4. To have a usable user interface to specify the automata in a sensible way.

1.3 Challenges

The main challenges which needed to be dealt with were as follows:

Finding efficient structures to store the contents of the tape Part of the definition of Turing machines is that the tape is of infinite length, this is clearly not possible on a real computer. With some Turmites the space is mostly empty after it has been running for a certain time so simply allocating static arrays will not suffice.

3D Graphics in Java Java does not provide an API for three dimensional graphics. There are a number of third party libraries which depend on native libraries, which causes issues with portability (one of the main advantages of using Java).

Effectively communicating my progress I will have to concisely and accurately keep my supervisor up to date on my progress.

1.4 The Solution

The solution produced is a graphical application written in the Java programming language for the purposes of designing and experimenting with

multidimensional Turing machines. It provides a user interface to specify the machine, and can visualise the execution of them using either a two dimensional grid view, or hardware accelerated three dimensional graphics.

Instead of specifying a fixed size area for the machine to operate the space will grow as the automaton writes more symbols. It is only bounded by the limits of the platform (see section 4.4.1). The view can also be zoomed and translated in order to better inspect the machine.

The solution can also generate state transition diagrams for the automata designed in it. These are output in `dot` format to be used by GraphViz. It also supports rule strings for the creation of 1-state-n-colour automata. It can also load and save the automata to files on disk for later use.

The application produced is a success as it met all the stated aims. In addition the architecture of the system is highly extensible and modular which will aid future development. The only piece of functionality not present is the ability to load the initial state of the grid, which is necessary to perform any meaningful computation.

Chapter 2

Background

2.1 Background of the problem to be solved

Before the solution can be described, it is necessary to define the problem.

2.1.1 What is a Turing Machine?

uring machines are a fundamental model of computation. Originally proposed by Alan Turing in his 1937 paper “On Computable Numbers, with an Application to the Entscheidungsproblem” [1].

A Turing Machine is a theoretical model which is not intended for real-world use. It is a type of finite state automaton with an infinite tape (which is equivalent to the memory of modern computers) which can store a set of symbols.

The solution he proposes is a simple abstract state machine. All modern computers are simply incredibly complex deterministic finite state automata.

To take a simple example, the state machine shown in Figure 2.1 will compute the same thing as the C code shown in Listing 2.1.1

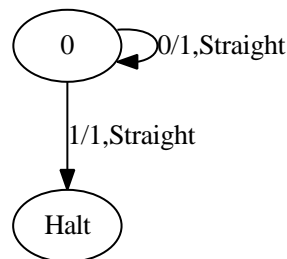


Figure 2.1: Simple state machine

Simple C program

```

char* ptr=0x0000;
while (!* ptr) {
    *ptr++ = 1;
}

```

This model can be formalised as a 7-tuple:

$$M = \{Q, \Gamma, b, \Sigma, \delta, q_0, F\}$$

- Q is a finite set of states.
- Γ is a finite set of symbols on the tape.
- $b \in \Gamma$ is the blank symbol, which exists in all elements of the tape which have not been set to a defined value.
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of input symbols, or ones which the machine responds to.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ the transition function, which changes the state and moves the head left (L) or right (R).
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final or accepting states.

For the purposes of the simulation of such systems on a modern computer we can make a few simplifications. When dealing with the set of integers (\mathbb{I}), will be taken to mean the range of integers expressable by the computer as a 32 bit integer. ($-2, 147, 483, 648$ to $+2, 147, 483, 647$)

- $\Gamma \equiv \Sigma \equiv \mathbb{I}$ The set of possible inputs is the set of possible values on the tape, which is also the set of integers.
- $b = 0$ The blank symbol is defined as the integer 0.
- $q_0 = 0$ The initial state is also 0.
- $Q \equiv \mathbb{I}$ The set space is as large as the computer can handle.
- Accepting states are ones which would otherwise lie outside the domain of the transition function.

So, in order to simulate a Turing machine with these simplifications, we end up with a 5-tuple:

- the set of possible direction changes (in this case, left and right - or forward and back)
- The location of the head on the tape.
- The contents of the tape.
- The state register.
- The transition function.

2.1.2 Extending into Two Dimensions

One of the more well-known two-dimensional Turing machine (or Turmite) was discovered by a number of people independently. Possibly the most well known example of a 2d Turing Machine (or Turmite) can be credited to C Langton[2] with “Langton’s Ant”, however the approach taken there was to create a cellular automaton system with a large number of states.

In 1989, A. K. Dewdney published an article in Scientific American describing the work of Greg Turk[3] in which an equivalent system was proposed. Here the head of a 1 state 2 symbol Turing machine moves on the tape with turtle-like motion similar to the Logo programming language, but with only 90° turns (Figure 2.2). This is the model used in this system. When this model is run it displays some interesting emergent behavior as seen in Figure 2.3.

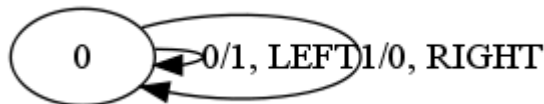


Figure 2.2: A simple Turmite

The model is extended into two dimensions by:

- Exchanging the tape for a two-dimensional grid.
- Extending the head pointer to be a 2-tuple which contains the coordinates on the grid.
- Extending the set of directions in the grid to include up, down, left and right.
- The movement of the head is constant, meaning it always moves one step forward. The input from the transition function affects the orientation of the head, and so the direction it moves on the grid.

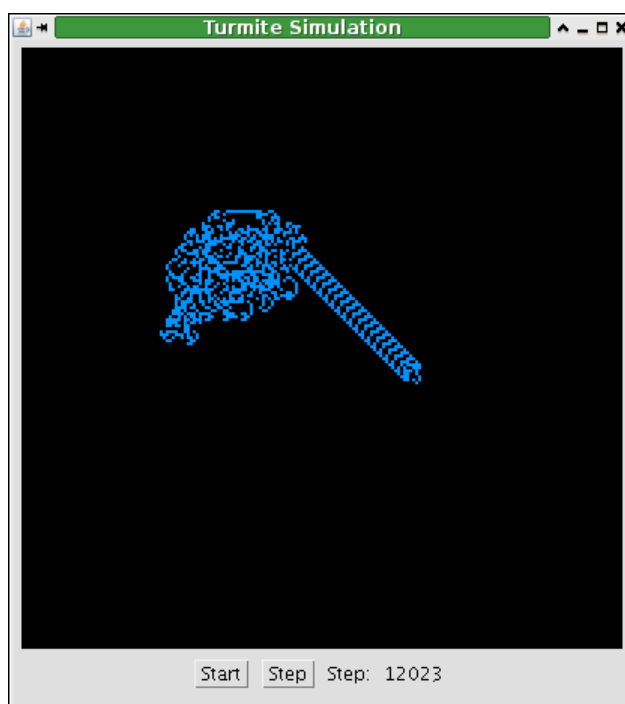


Figure 2.3: Its output

When dealing with changes of direction, the orientation of the head is cumulative and always 'steps' forward.

2.1.3 Generalising to n Dimensions

Here we can infer a number of interesting points:

- For each dimension, we add two more possible changes of direction.
- For N dimensions, the tape can be represented by an N dimensional grid of elements.
- Similarly, the head position is an N -tuple.
- Movement of the head is always parallel to one of the axis and is of unit distance.

2.2 Existing Approaches

There are many existing solutions which model and display two dimensional Turing machines, however relatively few work with three dimensions.

Some take the cellular automata route while others use a simpler mechanism which keeps track of the position and orientation of the 'ant'. One example of a 3d implementation is Heiko Hamann's paper [4]. His simulation is limited to three dimensions, and does not provide a user interface for designing, nor a way to load/save to a file. Hamann's system is also limited to a fixed-size grid.

My solution supports many (more than three) dimensions at the simulation layer and 2 or 3 dimensions for displaying them. It also has a user interface to design the transition function. Transition functions may be saved to a file on disk for later examination.

Rather than use a fixed-size array to store the data, the system was implemented as a sparse (or associative) array. These provide a mapping between key-value pairs. The key being the coordinates on the grid, the value being the current symbol stored at that location.

It should be noted that whilst Hamann's paper is also based on matrix mathematics, the solution presented here was arrived at independently as the paper was not found until late in the project.

The solution presented is larger and more complex than other solutions, but provides much more by way of functionality.

2.3 Research

A fair amount of reading took place to understand both the initial problem, and how existing approaches worked. This continued during implementation, continuing to seek out relevant articles and library documentation. I have read a number of papers extensively, making good use of tools such as CiteSeer¹ and SpringerLink². This has helped find information on existing approaches and what their references were.

2.4 Requirements

The project requirements were as follows;

Provide a GUI to configure the Turmite and run the simulation This was accomplished. Many implementation options are customisable at runtime using a drop-down menu.

¹<http://citeseer.ist.psu.edu>

²<http://www.springerlink.com>

Be able to load/save transition function definitions to/from a file

This was accomplished using the Java Serialisation API.

Support both fixed size and dynamically-sized grids

This is actually supported, but there seems no good reason for supporting fixed-size systems when effectively unbounded ones are configured.

Extend into three or more dimensions

The visualisation supports two or three dimensions. The model can support any number, but requires more static storage for each additional dimension as the size of each matrix increases by one in each direction, and each dimension adds two additional matrices.

The ability to load the initial state of the grid

I experimented with this in the two dimensional simulation. Originally I had planned to use a palette based image file (i.e. 16/256 colours) and use the palette index to populate the grid.

However actually creating and reading the file effectively posed problems. The colour mappings in the system and the pixel art program (e.g; ms paint, kolourpaint) were different and a key would need to be provided.

These difficulties removed the usefulness of this method.

Have a Variety of Visualisation Options

Only two visualisations are provided, they are however customisable.

- It is possible to load the palette information from a file, setting the colours for each symbol.
- For the 3d visualisation the user can enable/disable lighting and anti-aliasing in case the graphics hardware on the computer is not powerful enough.

Provide a User Guide

The user guide is distributed as a PDF file. A dialog by dialog description is provided, along with step-by-step guides for the basic tasks.

Chapter 3

Design

The design is split into three main areas:

General Components This provides the basic mathematical constructs for the simulation - such as Vectors, Coordinates, Matrices and the N-Dimensional area for them to operate on.

The Model This provides the simulation layer - Transition function handling and the Turmites themselves.

The View This is the user interface. It provides the GUI for designing and configuring the simulation, as well as the implementation for the graphical display modes.

3.1 General Components

The simulation relies heavily on matrix and vector mathematics. Hence it requires a number of support classes in which these operations are implemented.

To aid robustness, `Coord` and `Matrix` are immutable [5], that is they are not altered after construction. This is because `Coord`, in particular, is used as a key to Map structures. If the key were changed after insertion it would have unpredictable results.

`Matrix` is, in one small case, mutable - where it is necessary to load values into it (e.g. constructing the rotation matrices.) This is expected to be performed during initialisation and not at some arbitrary point during execution. (or rather, not when there might be more than one reference to the object)

`Grid` is mutable, but can be locked against concurrent access using the `synchronized` keyword A UML class diagram is shown in figure 3.1.

3.1.1 Matrix

This is an n by m integer matrix class which provides the basic matrix operations;

- Matrix and scalar multiplication.
- Matrix and scalar addition.

Its purpose is to handle the orientation and rotation of the Turmite heads in N-space. All operations return a new matrix as the result, apart from the `set()` method as that would be unduly wasteful of resources.

Whenever there is a bad value passed, an exception will be thrown. The intention is not that they should be caught, but should never be caused to be thrown.[6] For example trying to multiply two incompatible matrices.

3.1.2 Vector

This is simply a special case of **Matrix** which has only one column. It inherits from **Matrix**.

The multiply method is overridden to return a **Vector** reference instead of a **Matrix**. This is valid because the only reasonable multiplication of a **Vector** is by a rotation matrix and the result will be a vector of the same order.

3.1.3 Coord

This class represents an n-tuple for the purposes of reading and writing points in the grid. It supports addition with other **Coords** and **Vectors**.

Additionally the instances of the class must be comparable - for example implementing the `Comparable<Coord>` interface in Java. This is so they can be inserted into container classes and sorted (for example it may be desirable to produce a more complicated scenegraph implementation).

3.1.4 Dimensionality

This class provides information to the rest of the system which concerns the ones mentioned above. It is a singleton, as the information it provides is the same to any part of the system. This means that initialisation of this class is one of the first tasks the software should perform.

It performs the following actions:

Listing 3.1: Sine and Cosine Implementations from Dimensionality

```

/**
 * calculates the cosine of the angle. the provided angle is a shorthand
 * type where:
 * 0 = 0 degrees.
 * 1 = 90 degrees.
 * 2 = 180 degrees.
 * 3 = 270 degrees.
 * because it is only multiples of pi/2 we can use a switch statement.
 */
private static int cos(final int i) {
    switch (i % 4) {
        case 0:
            return 1;
        case 2:
            return -1;
        case 1:
        case 3:
            return 0;
        default: // unnecessary, but the compiler complains otherwise.
            throw new IllegalArgumentException(
                "somehow the modulus failed to work...");
    }
}
private static int sin(final int i) {
    return Dimensionality.cos(i + 1);
}

```

- It acts as a matrix cache. As there are only so many rotation matrices that are applicable for a given number of dimensions, it does not make sense to create a new one each time it is needed. The instance precalculates all the matrices which are relevant and caches them upon initialisation.
- It contains factory methods for `Coord` and `Vector` to provide default values with the correct number of dimensions, and caches those as well.
- It provides convenience methods for the matrices through the first 3 dimensions. This is mostly to aid in testing before the GUI is complete.
- Calculating a Givens[7] rotation matrix for a set number of dimensions and a certain rotation about a specified axis pair.

Constructing the Rotation Matrices

Before the matrix is constructed, a shortcut may be taken working out the sine and cosine of the angle. Because all movement is of unit direction parallel

Listing 3.2: Givens Rotation Matrix Calculation from Dimensionality

```
private static Matrix givens(final int size, final int i, final int k,
    final int cos, final int sin) {
    final Matrix m = new Matrix(size, size);
    m.set(i, i, cos);
    m.set(k, k, cos);
    m.set(i, k, sin);
    m.set(k, i, -sin);
    return m;
}
```

to one of the axis the calculations can be done with a simple switch statement as seen in Listing 3.1.

The matrices are then calculated using the method shown in Listing 3.2.

3.1.5 The Rotation Class

The `Rotation` (Listing C.45) was added to solve a problem:

When the user defines the transition function, they had to select the change in direction from a drop down box. Because all the matrices are generated during initialisation in order to pool memory they are tested for pointer equality rather than reference equality.

When a saved transition function is loaded, the Java serialisation API creates a new instance, and does not point it to the original one. This meant that the user interface had no way of knowing which matrix it was referring to.

`Rotation` was introduced to make it easier to edit transitions, as it attaches a name to each matrix, and can allow `Turmites` loaded from a file to use the matrices generated by that instance of the application rather than specifying its own. This also ensures that the machine will continue to function when run with a higher number of dimensions (but not less).

3.1.6 The Grid

The grid is the part of the model which stores the symbols written by the `Turmite` for retrieval. Because the model needs to be extensible into many dimensions an abstract representation of the coordinates must be provided.

The class `Coord` is to be implemented using variadic functions, and shall provide basic vector operations, such as addition. The symbols in the individual cells shall be represented by integers (as this is the basic unit of computation)

The Grid interface consists of three methods (See Listing C.19 for the entire class):

Listing 3.3: Grid interface

```
public abstract class Grid {
    abstract public int get(Coord pos);
    abstract public void set(Coord pos, int i);
    public void addGridListener(final GridListener l);
}
```

The use of the `GridListener` is important, because it allows the view to keep track of the grid contents in a format appropriate to its own implementation. It also removes the need to poll a whole area of the grid when the display window is moved/re-sized.

The downside to this is that it requires two copies of the grid data to be stored in memory. However, this is important for reliability, as shown here:

Thread 0 : The Turmite updates its grid position.

Thread 0 : The grid locks and starts calling all the listeners with the new value.

Thread 1 : The view class receives a resize or move message which requires it to call the grid, it locks itself.

Thread 1 : It calls the Grid but cannot yet acquire the lock so waits.

Thread 0 : The grid calls the View, but cannot acquire the lock so waits.

This causes a deadlock.

3.1.7 CoordSet

The grid is implemented in terms of another interface which makes it easier to have multiple implementations of the collection without complicating the Grid implementations. It is shown in Listing C.4:

Listing 3.4: CoordSet

```
public interface CoordSet<T> {
    public abstract T get(Coord pos);
    public abstract void set(Coord pos, T i);
}
```

It presents the same interface as an array, but indexed by `Coord`. It stores the default value and synchronizes the access.

3.2 The Turmites

I define a Turmite as a 5-tuple:

$$\{state, position, orientation, transition_function, grid\}$$

- The state is defined as an integral value.
- Position is a `Coord` indicating the current location on the grid.
- Orientation is a unit vector in the direction the Turmite is pointing.
- `transition_function` is a dual-indexed associative array which maps a $(state, colour)$ pair to `Transition` object.
- Grid is a reference to the grid on which the Turmite operates.

3.2.1 The Transition Function

A `Transition` is a structure with 5 elements:

$$\{old_colour, old_state, new_colour, new_state, rotation\}$$

The colour and state values are integers, the rotation is represented as a matrix which can be multiplied with the Turmite's orientation to give the resultant direction. For comparison purposes, they are determined to be equal if the initial state and initial colour are equal. Because of this a number of methods from `java.lang.Object` must be overridden:

compareTo This ensures proper ordering of transitions for the list views.

hashCode This generates a hash of the distinguishing features of the transition, namely the original state and the colour that was read from the grid.

equals This is overridden to only compare the same values as `hashCode`.

The transition function itself is a collection of `Transitions`.

The Turmite class is constructed by providing the grid, transition function and the initial position. The initial state is defined to be 0. The method that performs the actual work is `act()`.

Like the Grid, the Turmites produce events for listeners (Listing C.52). These are for a number of intended purposes, mainly providing information to the user during execution. This will save the user interface having to continuously poll them.

3.3 The View

The view includes the user interface for constructing Turmites as well as the means of displaying them. There will be three main windows for the graphical interface;

Definition Window Here the user will define the transition function. There will be a list of the currently defined transitions on the left. On the right will be a form in which to input the 5 values needed.

At the bottom there will be buttons to load and save the definition to/from a file, as well as to proceed with the simulation. (Figure 3.5(a))

Configuration Window Here the user can select options such as:

- The initial size of the simulation window.
- the time to wait between iterations.

Simulation Window This will show the simulation running. It will include an iteration counter and a single-stepping function, as well as the option to run the simulation at the previously prescribed rate.

The actual view presented will also depend on the previous choice. For a two dimensional Turmite, could look like Figure 3.5(b), but for one running in three dimensions it it more likeFigure 3.5(c).¹

¹Also, it is possible to view a two dimensional system using the three dimensional view.

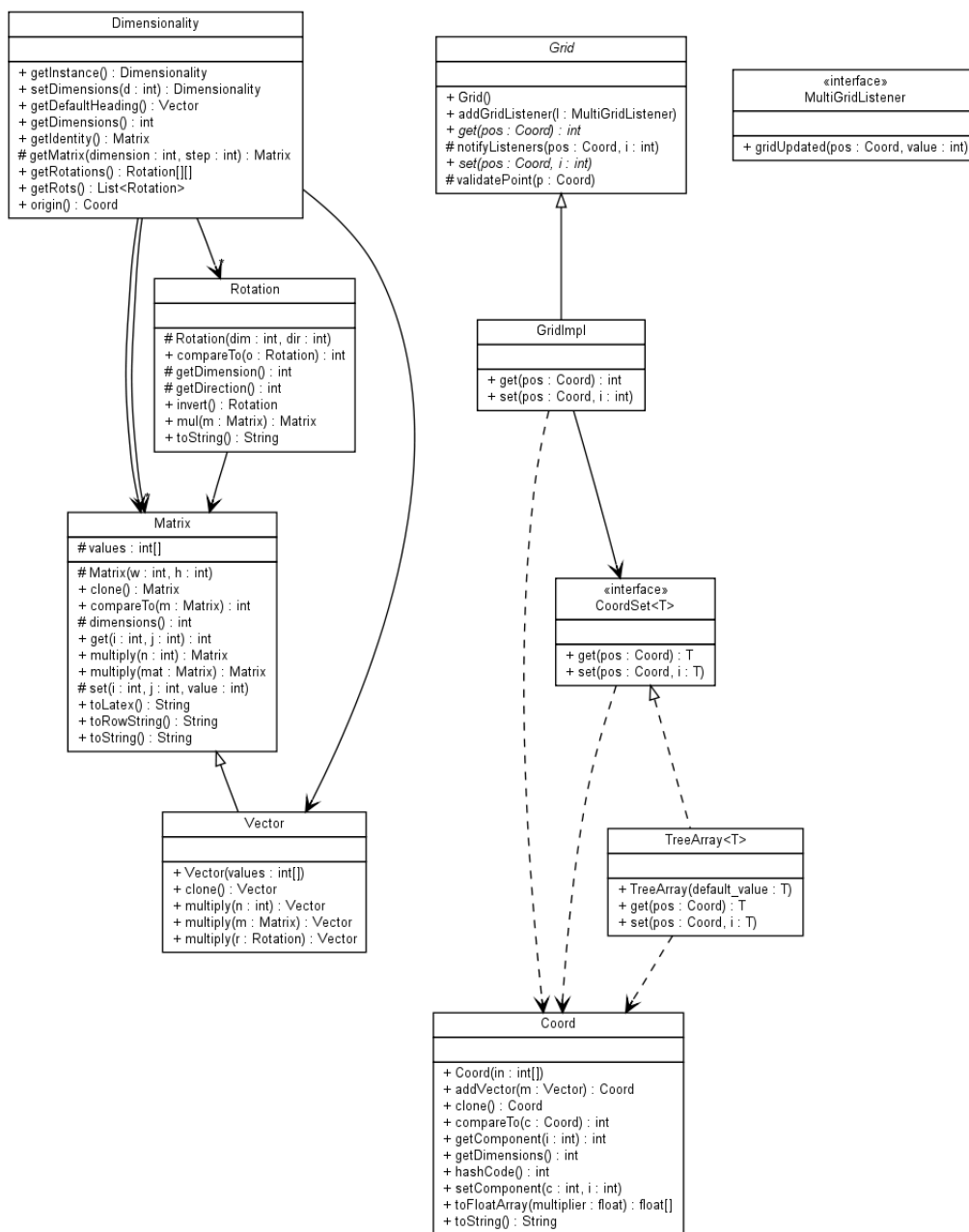


Figure 3.1: Basic Mathematical Constructs

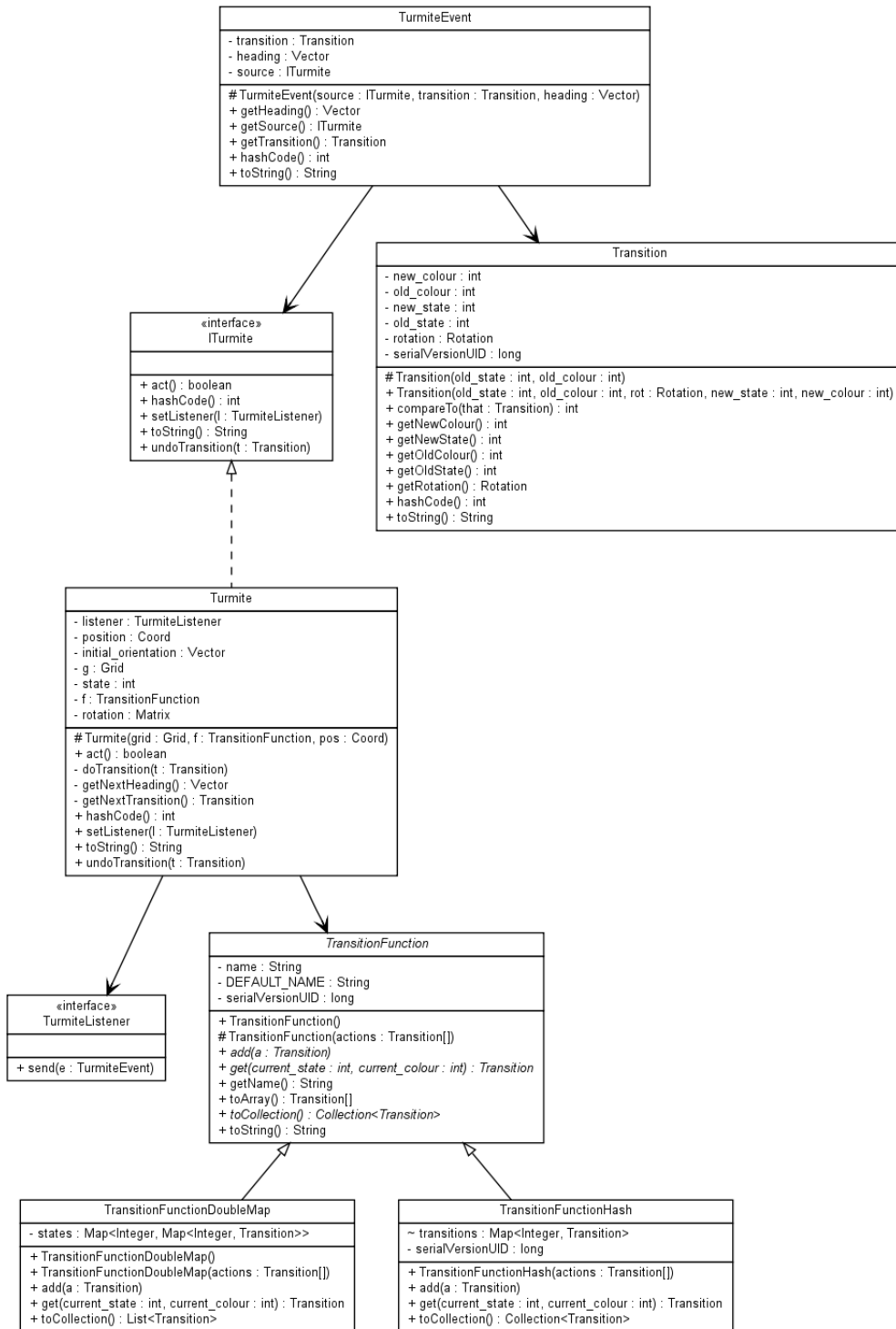


Figure 3.2: The model classes

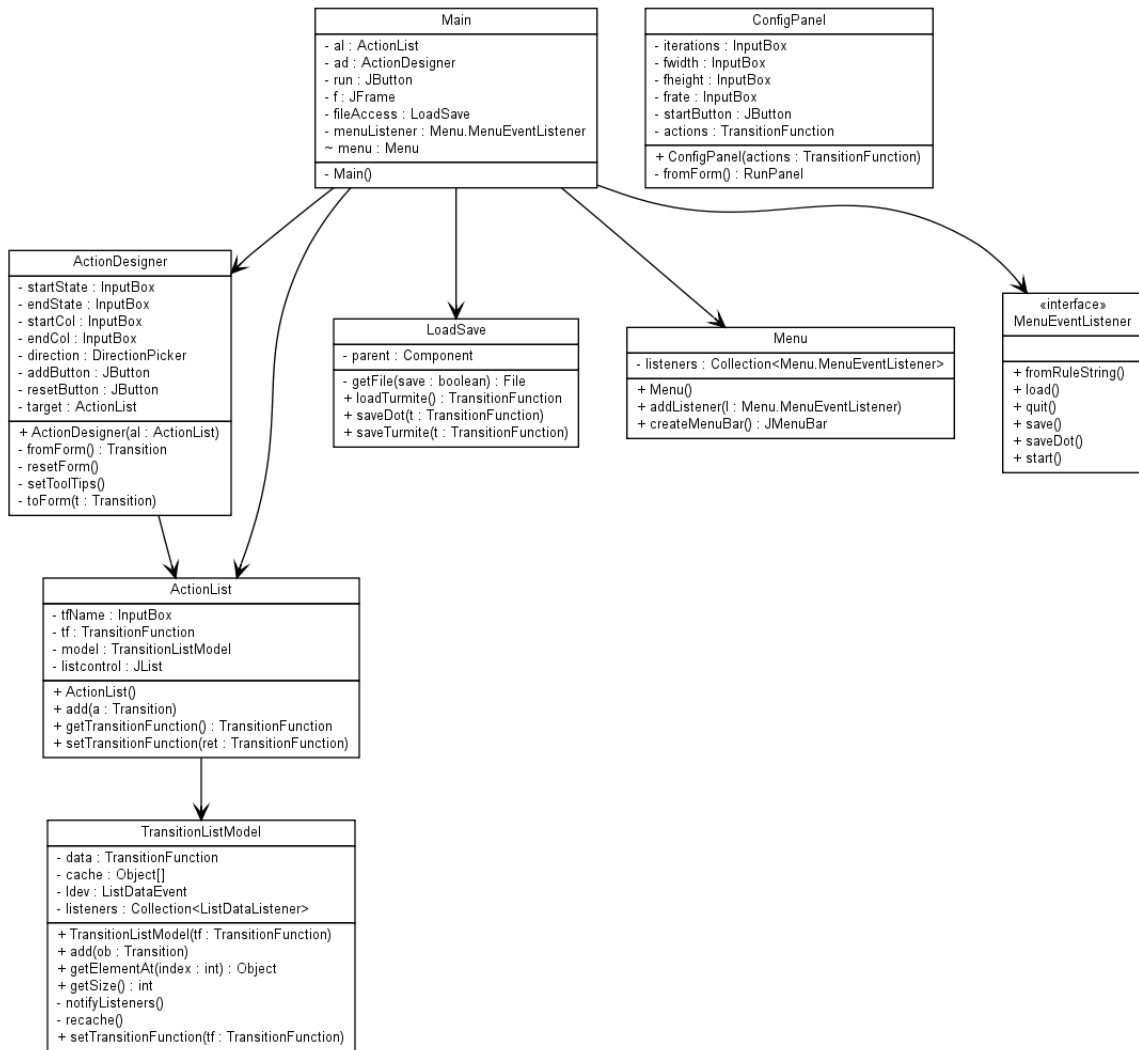


Figure 3.3: Classes used to design the transition functions

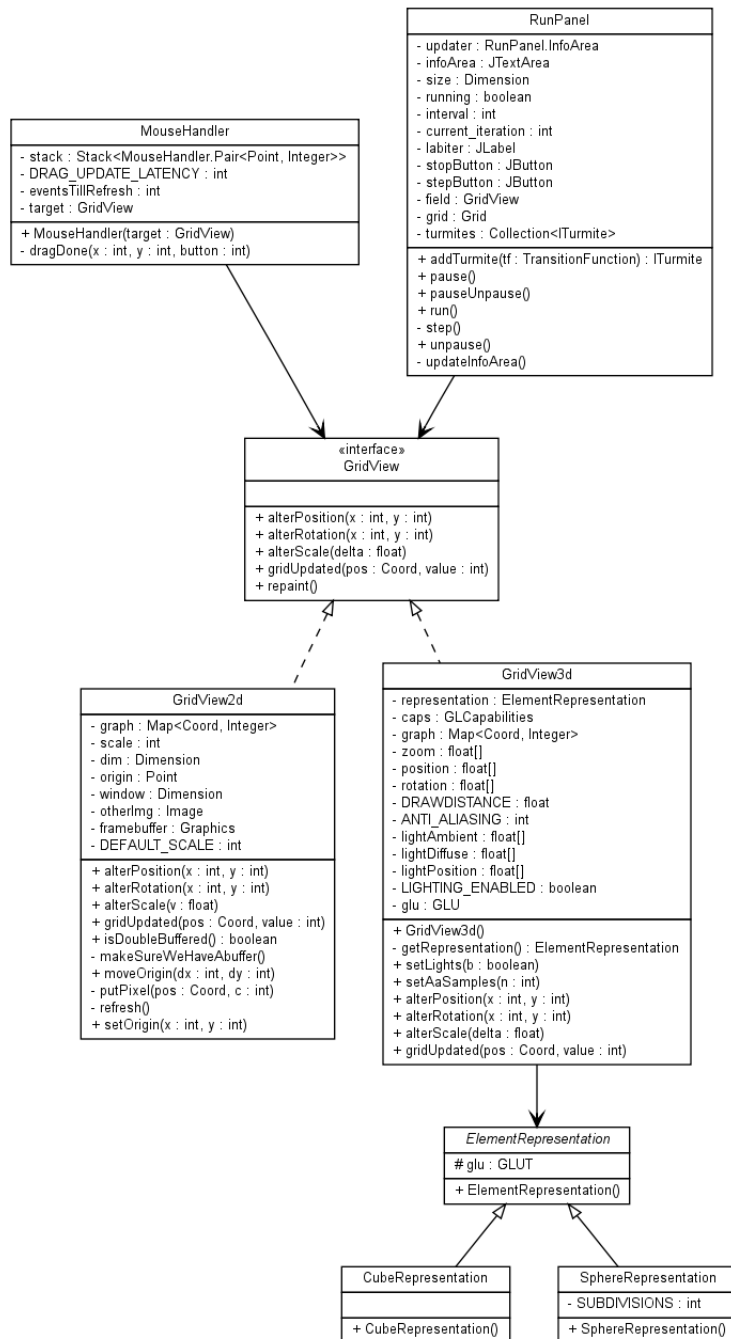
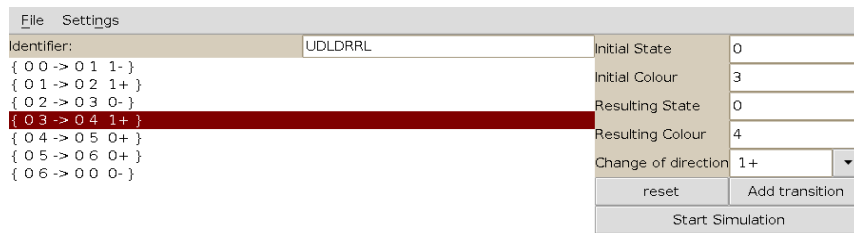
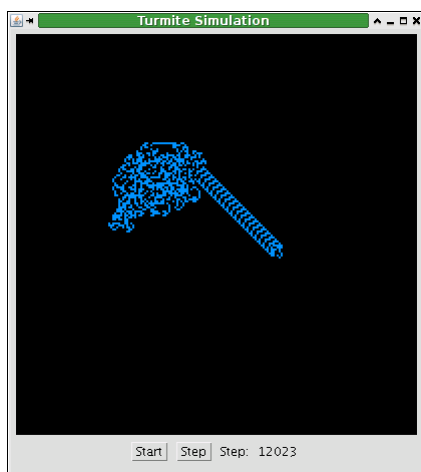


Figure 3.4: Classes used by the user interface during execution.



(a) definition window

Transition: { 0 1 -> 0 2 1+ }
 Heading: [-1][0][0]



(b) Langton's ant running in a 2d view



(c) a 3d Turmite

Figure 3.5: The different stages of execution

Chapter 4

Realisation

This chapter describes how the software was implemented, the methodology used to ensure it was completed and the problems encountered during development.

4.1 Hardware and Software Used

The system was developed using the following combination of hardware and software:

Hardware

- Intel Core 2 Quad Q6600 overclocked to 3.2GHz.
- 4096 megabytes of DDR2 memory.
- Asus P5K-E motherboard.
- Nvidia Geforce 8500GT graphics card.

Software

The Eclipse IDE This is a capable IDE for Java. It works on multiple platforms (it is itself written in Java) and provides compiler/debugger integration. It also has a lot of refactoring tools built in (e.g. rename symbol, encapsulate constructor/method/field etc).

JOGL - OpenGL bindings for Java This is a set of OpenGL bindings for Java. It was chosen after some comparisons with similar libraries.

Gentoo Linux was the operating system. It provides much more control over the system than many other distributions, so is ideal for a development workstation.

LaTeX and Kile LaTeX is a software typesetting package, particularly well suited to academic or technical writing. Kile is an IDE for LaTeX for the K Desktop Environment.

UMLgraph UMLGraph is a Java application for creating UML diagrams from Java source code. It outputs files to be read by graphviz, which actually draws the diagrams.

The Bourne Again Shell Bash is the default shell on most Linux distributions and is easily installable on other Unixes (e.g. Solaris, FreeBSD). During development of the software I needed a number of scripts for various tasks. Bash fits the bill nicely.

Graphviz Graphviz takes a plain-text description of a directed or undirected graph and renders it to an image. It was used to generate all of the diagrams for this report. It is also the target format for the state diagram generator provided in the solution.

4.2 Methodology

Before starting the development of this system, I constructed a prototype implementation which only supported two dimensions. Even though no code from this was used in the final version it was useful to see what sort of structural challenges existed. Afterwards I spent a lot of time reading about existing implementations and examining their shortcomings and strengths. Only then did the actual design take place.

Once the design was complete I wrote all the stubs¹ for the classes. After that was complete I progressed with a bottom-up methodology - first implementing the `Matrix`(Listing C.40), `Coord`(Listing C.42) and `Vector`(Listing C.44) classes, as everything else is built on top of these.

4.3 Changes to the Design

The design needed to be revised in a number of ways:

¹A stub class is one which contains all the interface declarations, but no implementation. This is to allow you to write and compile the other code which depends on it

4.3.1 User Interface Changes

As more features were implemented, continuing to add buttons to the lower part of the main windows was not an adequate design - it became cluttered and unfriendly.

The decision was made to change it to use a menu bar rather than the buttons, and expose more options at run-time (such as the implementations of various interfaces) rather than only at build-time. This is handled by the `Menu` class (Listing C.33) and made it easier to add more options to customise the application.

4.3.2 Traditional Turing Machines

There is a class `TuringMachine` which implements a standard TM. This means that it does not hold an orientation, and simply moves one step in whatever direction the transition specifies. This was added because it was so simple to implement.

4.4 Implementation of Components

4.4.1 The Grid

The grid is intended to be treated as an interface in order to allow experimentation with different implementations. However some functionality is common across them all, so it is actually implemented as an abstract base class (Listing C.43).

Limitations

At this point it is worth mentioning the limits of the system. Using a typical 32 bit operating system on commodity hardware the total amount of memory available to user-mode applications is only a certain proportion of the address space.

Taking 32-bit Linux running on IA-32 (x86) as an example, it maps the first gigabyte of physical memory into all address spaces for use by the kernel at `0xC0000000`. [8] This means only 3 gigabytes (3×2^{31} bytes) of virtual memory are available for user-mode tasks - reading or writing to the kernel area would cause a general protection fault. [9]

Even if one were to assume the whole of this memory was available for the grid data structure, the virtual address space would be exhausted faster than the range of coordinates in the structure.

For example, a two-dimensional grid (even if stored as an array) would run out of address space around 768×2^{20} cells, which is far less than the 2^{64} which can be addressed using the coordinates.

When the memory overhead of the structure itself is taken into account the exhaustion of resources would occur much more quickly. This overhead is not insignificant, if we assume a simple binary search tree implementation:

```
struct TreeNode {
    TreeNode *left ,*right; // pointers to children
    int32_t key [2]; // the coord
    int32_t value; // the value of the cell
};
```

For systems with 32-bit pointers, this gives an overhead of 4 words, or 16 bytes for every word of information stored.

When using a 64-bit architecture, the virtual memory space is far larger so the exhaustion will not occur until a later point. However even then it is worth mentioning that for the example above it would add another 8 bytes onto the amount of overhead for each element stored.

The reason this is a good idea at all is based on the observation that most of the space available (if one were to draw a bounding box) is empty, that is has not been modified from it's initial configuration.

If one were to allocate a static array, for many Turmites which build 'highways', after it has been going for a while, the vast majority of the bounding box has not even been touched by the Turmite. This can be seen in Figure 3.5(b)

This is only for a two dimensional model, and the problem is exasperated when working with more dimensions.

4.4.2 The View

The implementation of the view was further separated into two portions. First the user interface for designing the transition functions, then the visualisations for displaying the simulation on screen.

4.4.3 The User Interface

The user interface is based on the Java Swing toolkit. This is Java's native toolkit and is mostly compatible across supported platforms.

During development there were a number of user interface patterns that were required in multiple locations. To reduce code size and improve cohesion

these were constructed as a number of components - for example a field with a label to the left of it, or a combobox with a label.

All elements of the interface are provided with tooltips where appropriate.

4.4.4 The Visualisations

There are two visualisations to choose from, and some functionality shared between both.

Common Functionality

All implementations conform to the same interface and are referenced as such. It is required that they inherit from `JPanel` and implement (or their superclass implements) the following methods:

Listing 4.1: Abridged `GridView` interface

```
public interface GridView extends Grid.MultiGridListener {
    /**
     * @see Grid.MultiGridListener#gridUpdated(Coord, value)
     */
    public void gridUpdated(Coord pos, int value);

    // these are called by the mouse handler
    public void alterPosition(int x, int y);
    public void alterRotation(int x, int y);
    public void alterScale(float delta);
}
```

The control of these implementations is handled by a common module, the `MouseHandler` class (Listing C.15).

This is to ensure that all the view controls respond in the same way to user interaction, and provide a consistent experience. The

Two Dimensional

The two dimensional visualisation was straightforward to design and implement. It plots each value as a colour on a grid, which can be zoomed and translated.

It was implemented by extending a `JPanel` (as this is a precondition of the interface) and overriding the `paint()`.

Three Dimensional

The three dimensional visualisation took more work, because java does not include a 3D API. There are a number of third party APIs to accomplish this, as well as the possibility of creating my own. Here are some of the advantages/disadvantages of each approach:

Roll My Own The option was present to build my own set of bindings. Having worked with JNI (Java Native Interface) in the past, and reading about the AWT Native Interface ², which i would have to use for graphical output it was decided that this would be a last resort.

The advantage would be that only the sufficient and necessary functionality would be exposed to Java, with the possibility of writing the more performance-critical parts in C. [10]

The main disadvantage would have been needing to write and test (potentially) dissimilar implementations for multiple platforms. This would inevitably cause issues and divert time and effort from the main project.

Java3D This is a retained-mode API³ produced by Sun (the creators and maintainers of Java). However, for this project (where rendering is based on data rather than a scene) being locked into a scenegraph model was too restrictive and would have overcomplicated the system.

JOGL - Java OpenGL This is a set of bindings for OpenGL, which is the main cross-platform graphics library. It provides a 1:1 mapping for Java calls to OpenGL calls, and a minimum of overhead. It was developed under the Java Community Process (JSR231) ⁴

LWJGL - Light-Weight Java Game Library This is a library geared towards game development, and has a lot of external dependencies. The graphics portion is built on top of JOGL, so there seemed no reason to use this instead.

All of these required external libraries, and while Java3D looks the most likely to become part of the standard in future, it's API did not lend itself to what I wanted to do (draw a set of objects at essentially random positions). As I was already familiar with OpenGL, I chose to use JOGL.

²http://java.sun.com/j2se/1.5.0/docs/guide/awt/1.3/AWT_Native_Interface.html

³<https://java3d.dev.java.net/>

⁴<http://jcp.org/en/jsr/detail?id=231>

It would be possible to support multiple graphics back-ends as the code is highly modular; in the best case only one class would need to be replaced (in this case, the `GridView` implementation). However in this case I have decided to only support one.

There is support for defining a number of different representations. I have chosen to use a cube because it's simple (and it's a cubic grid).

The view is lit using a specular light so it is possible to tell the difference between adjacent elements. There is also support for anti-aliasing at a user-defined level. Both of these options are configurable at run-time.

4.5 Problems and Solutions

In this section I shall describe some of the problems I encountered during design and implementation, and how these were corrected.

4.5.1 Head Orientation

Originally the starting orientation was generated with the following algorithm (from Listing C.41:

```

1 Vector makeDefaultHeading() {
2     final int [] out = new int [this.dimensions];
3     // all elements are zero
4     out[0] = 1; // first element one.
5     return new Vector(out);
6 }
```

Resulting in a column vector such as:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

With certain transition functions, it is possible to get into a state where the head does not rotate even when a rotation is applied.

For example; If one tries to rotate about the YZ axis:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Which is not the desired behaviour. The solution was to change the initial vector to be zero, but one in the highest dimension, as per this algorithm e.g. change line 4 of the above listing to:

Listing 4.2: archive.sh

```
#!/bin/bash
VERBOSE="FALSE"
STAMP=" `date +%F` "
NAME="turmites"
SRCDIR="src"
ENDNAME="${NAME}-${STAMP}"
cp "$SRCDIR" "$ENDNAME" -r
if [[ "$?" != "0" ]]
then
    rm -fr "$ENDNAME"
    echo "Could not create backup. Exiting." >2
    exit 1
fi
pushd "$ENDNAME"
javas.sh clean
popd

tar czf "${ENDNAME}.tar.gz" "$ENDNAME"
# remove the copy we made.
rm -rf "$ENDNAME"
```

4 out[**this**.dimensions-1] = 1; // last element one.

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$$

This produces the correct effect.

4.5.2 Version Control

Normally when starting a project, the first thing I do is create a repository on my Subversion server for it, so there is a complete history available. However this time things did not go to plan. One of the main issues was that I was using Eclipse, whereas for any language other than Java I would use a normal text editor. Despite there being a Subversion plug-in available, I experienced difficulties getting it to work correctly. For the first couple of revisions I manually updated and checked in files. However after a number of refactorings (e.g. renaming/moving classes) the repository fell out of sync with the current project.

This was dangerous as I no longer had a good record of all the changes made. In order to remedy this a bash script was constructed (Listing 4.2) Following this, changes continued to be made incrementally - ensuring that each day had a working version. Any time a change was to be made the source was archived. However, it was far from ideal. In future I shall ensure that the version control system is correctly set up before development proceeds.

4.5.3 GridView Inheritance Heirachy

In order to connect `MouseListener`(Listing C.15) to the view implementations, it is necessary to register `MouseWheelListener`, `MouseListener` and `MouseMotionListener` with the component. However because they do not necessarily both descend from the same class (e.g. `JPanel`) a runtime cast was necessary to add these listeners. It worked for the two provided implementations but was too fragile.

To fix this, the methods were added to `GridView` and that allowed the calls to travel up the other inheritance tree.

Chapter 5

Evaluation

5.1 Evaluation Criteria

The system was evaluated to the following criteria:

Correctness The software performs as expected, and produces results verifiable by other implementations if they exist.

Robustness The software does not crash. If bad data is input then a message is displayed informing the user. It does not simply quit to the console with a stack trace. Even fatal errors are reported via the user interface.

Maintainability The software should still have room to be extended at the end of the project. This should be done by suggesting possible extensions that would not be problematic to implement on top of the existing codebase.

Portability The software will work on a variety of platforms.

Performance It should not make too much use of system resources (processor time or memory). In so much as it should not cause the computer running the simulation to run out of either of these resources.

Usability It should not be difficult or cumbersome to use the software. This is a very subjective measure, but important in graphical applications.

5.2 Assessment

Each of these is assessed below:

Correctness The software does as is expected of it. Rule definitions as published by others perform identically on this system.

Usability is a subjective measure. Users of the software can reasonably be expected to have a basic understanding of Turing machines. Tooltips are provided for all components in the designer interface. On suggestion from peers, many options were moved to the menu bar rather than buttons in the main window.

Performance is possibly an issue. As discussed above, there is an upper bound on the size of the grid not greater than the available virtual memory. In addition to this, some platforms will page anonymous memory to disk (swap) which will severely impact performance as the working set grows.

To compound matters, there is no simple and portable method to find out if the system is running out of memory. The JVM will throw an `OutOfMemoryError` in certain situations¹ however certain operating systems do not report the memory issue to even native applications (i.e. they overcommit memory) and just pretend to allocate it. [11]

It is not possible to avoid such situations without the means to accurately determine the details of the operating environment.

Additionally, the 3D visualisation requires hardware OpenGL-compatible acceleration. Running without this, the system runs unusably slowly even on high specification systems.²

However, because the system is maintainable and the concrete types are not referenced throughout the project it is possible that better performing implementations can be constructed at a later date.

Additionally, there is no way to iterate over a certain section of the Grid data structures. This was an oversight. However the system does not appear to slow down as they become more full.

Maintainability is a subjective quality. I have worked to keep interface separate from implementation, and where there are interdependancies between modules these are documented. It is relatively simple to add new functionality such as a new visualisation or

¹<http://java.sun.com/j2se/6/docs/api/java/lang/OutOfMemoryError.html>

²The machine in question is a 3.2GHz quad core with software rendering, or rather the hardware acceleration disabled.

Portability is only limited by that of the platform (the JVM) and the JOGL library. No assumptions have been made about directory structure or file naming conventions. Even line endings are produced for the correct system.³

For a short time it appeared that there was a problem with JOGL and certain 32 bit Linux distributions, however I have traced it back to a faulty graphics driver⁴.

The system works perfectly on 32 bit Windows and 64 bit Linux.

5.3 Ideas for Extensions

This section describes a number of possible extensions to the system. Some of them were partially implemented while others were not. The intention is to show that the software is maintainable and extensible.

5.3.1 Hybrid Grid

For many Turmites the grid remains mostly empty. However some exhibit a lot of activity in a certain area. The idea is that part of the grid could be allocated as a static array, and coordinates outside that would be accessed in the same way as the existing implementations. This could potentially lead to improved time and space efficiency.

5.3.2 Attractor Detection

The original reason for the `TurmiteListener` mechanism was to try and detect when a Turmite enters a repetitive state (e.g. building highways). I had an idea about applying either a modified compression or a string matching algorithm to the history of what state transitions were applied. This would be useful for large-scale automated testing of the machines.

For example you could iterate through every possible ruleset and run each one for a large number of iterations (Without graphical output this would not take as long). At the end you would get a list of those which halted, those which formed repetitive patterns within the allotted time, and those which did not.

³For historical reasons different operating systems use different control codes to signal the end of a line of text. For example, UNIX-like systems use the line-feed character (decimal 10), whereas Microsoft DOS and Windows systems use carriage return (decimal 13) followed by the line feed.

⁴The `i915` driver from the vanilla Linux kernel version 2.6.28.

5.3.3 Undo Transition

When simulating the machines, it was often frustrating to wait for a large number of iterations for some pattern to emerge, then miss the start of it. Again using the `TurmiteListener` functionality it should be possible to 'undo' a transition to get to the critical point. This is partially implemented in `Turmite` but there is no user interface support.

5.3.4 Multiple Turmites

The model and the view both support multiple Turmites on the same grid. However there is no user interface to add more than one.

5.3.5 Z ordering in GridView

At the moment, if a three-dimensional Turmite is viewed using the two-dimensional visualisation, it does not appear as it should. This is because the `CoordSet` implementations have no concept of depth. If one were to create a Z-buffer for these classes, it would be possible to use the two dimensional view to monitor a three dimensional turmite from two or more angles. It would appear that one could then use a three dimensional visualisation to monitor the execution of a four dimensional Turmite.

Chapter 6

Learning Points

This section describes the main lessons learnt during the development of the system. During the course of the project, I have learnt to effectively use the \LaTeX typesetting program¹. The main advantages of \LaTeX are the easy integration of bibliography utilities (e.g. BibTeX) and the flexibility of structuring the document. There is a rather steep learning curve to using \LaTeX in place of graphical desktop publishing software.

While I was already familiar with Java and its standard class libraries, I had never worked with externally provided ones before. It was cumbersome to get the development environment set up correctly for JOGL. Because parts of it are native code it also caused minor portability issues between the target systems. Creating distributables of Java software is not something I had previously done - instead simply distributing the source. With this project, it is not straightforward to build because it requires a different set of libraries on each platform I constructed some archives to enable people to run it without difficulty. In addition the use of Java Web Start is not something I had previous experience of, nor that of signing jars for use with it.

One thing I have definitely learnt is the importance of being able to clearly communicate what is happening. The system presented is a non-trivial piece of software. As software grows it becomes more difficult to keep the complexity under control. From the outset the design was intended to assist with this and it has done. There is clear distinction between the logical parts of the system.

¹<http://www.latex-project.org>

6.1 Writing Defect-Free Concurrent Software

Working with external Java libraries

constructing more complicated user interfaces

writing defect free concurrent software

Chapter 7

Professional Issues

This chapter examines how the BCS Code of Practice [12] and Code of Conduct [13] relate to this project and what was done to ensure compliance.

7.1 Code of Conduct

I have determined the following clauses apply to this project:

7.1.1 The Public Interest

7.1.2 Duty to Relevant Authority

- You shall not misrepresent or withhold information on the performance of products, systems or services, or take advantage of the lack of relevant knowledge or inexperience of others.

The system does what it claims to, and attempts to explain the limitations it imposes.

7.1.3 Duty to the Profession

7.1.4 Professional Competence and integrity

7.2 Code of Practice

7.2.1 Act Professionally as a Specialist

- Evaluate new products, assess their potential benefit and recommend their use where appropriate.

I evaluated a number of languages and libraries before commencing development. For the stated goals the combination of Java and JOGL was appropriate.

7.2.2 Manage your Workload Efficiently

- Do not undertake, or commit to, more assignments than you can reasonably expect to meet in a given time.
- Ensure that you have the necessary resources to complete assignments within the given time scale.

I was confident that the project would be completed at least to the original specification over the course of the year.

7.2.3 When Managing a Programme of Work

- Advise your customer if, in your opinion, any stage in the programme will not deliver the anticipated benefits.

The interim report was submitted explaining that the project was on schedule.

7.2.4 When Managing Project Risks

- Devise mitigation actions that will reduce the chances of the most serious risks happening.

The desired features were prioritised, so were completed in that order. I also adopted an incremental development methodology, similar to agile development [14].

Once the initial version was written and the structure produced, I made small changes to add functionality or restructure code ensuring that everything kept working and perform regression tests.

7.2.5 When Closing a Project

- Honestly summarise the mistakes made, good fortune encountered and lessons learnt.

This is important for any project in order to aid the spread of knowledge. In this report I have detailed a number of problems and their solutions.

- Recommend changes that will be of benefit to later projects.

This report details a number of future directions that this work could be taken.

7.2.6 When Designing New Systems

- Resist the pressure to build in-house when there may be more cost-effective solutions available externally.

The choice of development language was heavily influenced by this. If I had used C++ I would have had to develop large amounts of additional code and used conditional compilation features to have the software portable between systems. This would be made worse when creating a GUI application.

If I had gone and learnt a C++ framework, such as Qt ¹. This would have aided portability, but also demanded a steep learning curve to get to the level of competence as I am with Java/AWT/Swing.

Again, when investigating 3D graphics APIs, the option to create a set of bindings in-house was present, and would have made certain things easier and allowed me to improve performance in parts. However when factored against the additional programming and debugging time it was not worthwhile.

7.2.7 When Programming

- Strive to produce well-structured code that facilitates testing and maintenance.

I always strive to produce maintainable code. I have been involved in maintaining a number of projects where the original developers had given little thought to readability or maintenance. The system has been designed to allow for changing requirements.

- Follow programming guidelines appropriate to the language and encourage your colleagues to do likewise.
- Produce code that other programmers will find easy to maintain; use meaningful naming conventions and avoid overly complex programming techniques where not strictly necessary.

¹<http://www.qtsoftware.com>

All code is commented, except where this would detract from the readability of the code. The principles of object oriented design have been applied where appropriate - most significantly programming to interfaces and data hiding [15].

I followed the Sun Java style guidelines [16], and made use of other common good practices where appropriate (e.g from books such as Effective Java [17]). In addition, automated tools were used to keep the formatting and conventions consistent.²

The construction of major components of the system is performed by an abstract factory class, which returns either an interface or abstract class reference. The selection of which type to return is provided through enumerated types which can easily be set at either run-time or build-time. In many cases, constructors for concrete types are not visible outside that namespace.

- Make yourself aware of the limitations of the the platform (operating system and hardware) and avoid programming techniques that will make inefficient use of the platform.

The major limitations of the platform are described in section 4.4.1, showing I am more than aware of them. The use of memory is perhaps not the most efficient, but the flexibility provided in terms of maintainability offset the performance issues.

- Wherever possible, avoid platform specific techniques that will limit the opportunities for subsequent upgrades.

The choice of Java as a platform restricts the amount of platform-specific techniques that can be employed. However using native libraries (as is the case with JOGL) means care must be taken that the functionality employed is consistent across different operating environments.

²Such as Eclipse's "Clean-up" and refactoring functionality

Bibliography

- [1] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. 1936.
- [2] Chris G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. pages 12–37, 1991.
- [3] A.K Dewdney. Two-dimensional Turing machines and turmites make tracks on a plane. 261(3):124–??, September 1989.
- [4] H. Hamann. Definition and behavior of langton’s ant in three dimensions. *Complex Systems*, 14:263–268, 2003.
- [5] Joshua Bloch. *Effective Java*, chapter 4, page 73. Addison Wesley, 2nd edition, 2008.
- [6] Sharon Zakhour, Scott Hommel, Jacob Royal, Isaac Rabinovitch, Tom Risser, and Mark Hoeber. Unchecked exceptions - the controversy. In *The Java Tutorial: A Short Course on the Basics*, chapter 12. Sun Microsystems, 4th edition, 2008.
- [7] W. Givens. Computation of plane unitary rotations transforming a general matrix to triangular form. 6:26–50, 1958.
- [8] Mel Gorman. *Understanding the Linux Virtual Memory Manager*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.
- [9] *IA32 Architecture Software Developers Manual*, volume 3, chapter 5. Intel Corporation, 2004.
- [10] Sheng Liang. *The Java™ Native Interface Programmer’s Guide and Specification*. Sun Microsystems.
- [11] Mel Gorman. *Understanding the Linux Virtual Memory Manager*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

- [12] *British Computer Society code of practice*, pages 102–110. Chapman & Hall, Ltd., London, UK, UK, 1996.
- [13] *British Computer Society code of conduct (1992)*, pages 93–96. Chapman & Hall, Ltd., London, UK, UK, 1996.
- [14] Stephen R. Palmer and John M. Felsing. *A Practical Guide to Feature-Driven Development (The Coad Series)*. Prentice Hall PTR, February 2002.
- [15] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*, chapter 1. Addison-Wesley, January 1995.
- [16] Sun Microsystems. Code conventions for the java programming language. <http://java.sun.com/docs/codeconv/>.
- [17] Joshua Bloch. *Effective Java*. Addison Wesley, 2nd edition, 2008.
- [18] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

Appendix A

Instructions for Compiling and Running the Software

A.1 Running

In order to run the software you need to have a current Java Virtual Machine installed (i.e. newer than 1.5). To run the software via Java Web Start, open a web browser and navigate to <http://www.csc.liv.ac.uk/~cs6rlw/turmites> and click on the “Web Start” item.

A.2 Compiling

In order to compile the software, download the source tarball from <http://www.csc.liv.ac.uk/~cs6rlw/turmites> or from the CD. Open a terminal and extract it. You will also need the JOGL packages from <http://jogl.dev.java.net/> for your platform if you are using something other than Windows or 64 bit Linux (with a 64-bit Java installation). Once you have this, extract the jar and shared library files to the `lib/` directory in the source distribution.

Once you have the prerequisites, type:

```
turmites $ ./javas.sh rebuild
```

This will build all the class files required to execute the program. If you are on Windows then you will need to use your own method of compiling Java.

To run the program, use the script `run3d.bat` or `run3d.sh` to execute the Java virtual machine with the correct arguments. If you are on 32-bit Linux, you may need to edit the class path and native library path settings in `run3d.sh` to set it to the JOGL libraries you downloaded.

Appendix B

User Guide

B.1 Defining a Turmite

The first window shown when you open the program is the definition window. To the left you can see the list of rules present, and on the right the panel to define an individual rule. Here I shall explain the options:

Initial State This is the state the machine must be in for this rule to come into effect.

Initial Colour This is the colour under the head. This, along with the initial state determines the actions the machine undertakes.

New State The state the turmite transitions into following application of this rule.

New Colour The colour the turmite sets at the current grid location before moving.

Change of Direction This differs depending on whether a Turmite or traditional Turing Machine is being constructed. For a Turmite this is the rotation to apply. For a Turing Machine it is the offset. They are named in the following manner: the number indicates an axis of the grid to either move along or rotate about. The +/- signs indicate the direction (left or right).

For each transition, enter all the relevant information and click “Add Transition” below.

You can also load Transition functions from a file on disk. Select the “File” menu and click “Open“. There is also a function to use a rule string, as suggested by Heiko Hamann.

B.2 Configuration

In the settings menu you can choose various implementation options, however the only ones which should matter to normal users are:

View Impl This determines whether or not to use a two or three dimensional view. The default is three.

Anti Aliasing Whether to use anti-aliasing for the three dimensional view. This improves visual quality, but may reduce performance on less powerful machines.

Use lighting in 3d Whether to use lighting for the three dimensional view. This makes it easier to distinguish between adjacent cells, but may reduce performance on less powerful machines.

Machine Impl This allows the user to select between a standard Turing Machine or a Turmite.

Appendix C

Source Listings

Listing C.1: multidimensional/Factory.java

```

package multidimensional;

import java.awt.Dimension;
import java.util.Collection;

import multidimensional.basics.Coord;
import multidimensional.basics.Dimensionality;
import multidimensional.basics.Grid;
import multidimensional.basics.Rotation;
import multidimensional.util.CoordSet;
import multidimensional.util.HashArray;
import multidimensional.util.TreeArray;
import multidimensional.view.GridView;
import multidimensional.view.GridView2d;
import multidimensional.view.GridView3d;

/**
 * This abstracts away construction of the main types so as not to expose
 * implementation details outside of the package.
 *
 * @author richard
 *
 */
public abstract class Factory {
    /** The types for the CoordSet implementation. @hidden */
    public static enum EGridImpl {
        Tree, Hash
    }

    /** The grid view impementation @hidden */
    public static enum EGridViewImpl {
        TWOD, THREEED
    }

    /** Whether to do 'proper' turing machines or Turmites. @hidden */
    public static enum EMachineImpl {
        Turmite, Classic
    }

    /** Transition function implementation. @hidden */
    public static enum ETFImpl {
        DoubleMap, Hashing
    }

    private static EGridImpl gImpl = EGridImpl.Hash;

    private static EGridViewImpl gvImpl = EGridViewImpl.THREEED;

    private static ETFImpl tfImpl = ETFImpl.Hashing;
    private static EMachineImpl mImpl = EMachineImpl.Turmite;

    /**
     * Constructs an empty transition function.
     *
     * @return
     */
    public static TransitionFunction emptyTF() {
        switch (Factory.getTfImpl()) {
            default:
            case DoubleMap:

```

```

        return new TransitionFunctionDoubleMap();
    case Hashing:
        return new TransitionFunctionHash();
    }
}

/**
 * This creates a TransitionFunction based on a 'rule string'. A lot of
 * other simulations have similar features and is included mostly for
 * completeness.
 *
 * @param s
 *         The rule string to construct it from.
 * @return TransitionFunction corresponding to the rule.
 * @throws TurmiteException
 *         if there is a syntactic error.
 */
public static TransitionFunction fromRuleString(final String s)
    throws TurmiteException {
    final int states = s.length();
    final Dimensionality d = Dimensionality.getInstance();
    // use a few aliases for convenience.
    final Rotation[][] rots = d.getRotations();
    final Rotation LEFT = rots[1][0];
    final Rotation RIGHT = rots[1][1];
    final Rotation UP = rots[2][0];
    final Rotation DOWN = rots[2][1];
    final String upperString = s.toUpperCase();
    final TransitionFunction ret = Factory.emptyTF();
    for (int i = 0; i < states; ++i) {
        Transition a;
        switch (upperString.charAt(i)) {
            case 'L':
                a = new Transition(0, i, LEFT, 0, (i + 1) % states);
                break;
            case 'R':
                a = new Transition(0, i, RIGHT, 0, (i + 1) % states);
                break;
            case 'U':
                a = new Transition(0, i, UP, 0, (i + 1) % states);
                break;
            case 'D':
                a = new Transition(0, i, DOWN, 0, (i + 1) % states);
                break;
            default:
                throw new TurmiteException(
                    "Rule_string_should_only_contain_[LRUD]._Found_%c_at_
                    _index_%d",
                    upperString.charAt(i), i);
        }
        ret.add(a);
    }
    ret.setName(s);
    return ret;
}

// accessors for the implementations.
private static EGridImpl getGImpl() {
    return Factory.gImpl;
}

```

```

private static EGridViewImpl getGvImpl() {
    return Factory.gvImpl;
}

private static ETFImpl getTfImpl() {
    return Factory.tfImpl;
}

/**
 * Constructs a Grid object.
 *
 * @return the grid of the correct type.
 */
public static Grid grid() {
    CoordSet<Integer> s;
    switch (Factory.getGImpl()) {
        case Tree:
            s = new TreeArray<Integer>(0);
        default:
        case Hash:
            s = new HashArray<Integer>(0);
    }
    return new GridImpl(s);
}

/**
 * Constructs a grid view.
 *
 * @param windowSize
 *           The initial window size.
 * @return the correct view instance.
 */
static public GridView gridView(final Dimension windowSize) {
    switch (Factory.getGvImpl()) {
        case TWO.D:
            return new GridView2d(windowSize);
        default:
        case THREE.D:
            return new GridView3d(windowSize);
    }
}

// set the implementation of the various interfaces.
public static void setGImpl(final EGridImpl gImpl) {
    Factory.gImpl = gImpl;
}

public static void setGvImpl(final EGridViewImpl gvImpl) {
    Factory.gvImpl = gvImpl;
}

public static void setMachineImpl(final EMachineImpl mImpl) {
    Factory.mImpl = mImpl;
}

public static void setTfImpl(final ETFImpl tfImpl) {
    Factory.tfImpl = tfImpl;
}

/**
 * Constructs a transition function.
 *

```

```

    * @param t
    *           the transitions it should contain.
    * @return the correct function.
    */
    public static TransitionFunction TF(final Collection<Transition> t) {
        switch (Factory.getTfImpl()) {
            default:
            case DoubleMap:
                return new TransitionFunctionDoubleMap(t);
            case Hashing:
                return new TransitionFunctionHash(t);
        }
    }

    /**
     * copy constructor for transition functions.
     *
     * @param t
     * @return
     */
    public static TransitionFunction TF(final TransitionFunction t) {
        return Factory.TF(t.toCollection());
    }

    public static TransitionFunction tf2dint(final int i)
        throws TurmiteException {
        final char[] d = { 'l', 'r' };
        return Factory.fromRuleString(Factory.toBin(i, d)).setName(
            String.format("Turmite_%d.", i));
    }

    public static TransitionFunction tf3dint(final int i)
        throws TurmiteException {
        final char[] d = { 'l', 'r', 'u', 'd' };
        return Factory.fromRuleString(Factory.toBin(i, d)).setName(
            String.format("Turmite_%d.", i));
    }

    /**
     * converts a positive integer to a string representation of it's binary
     * value.
     *
     * @param n
     *           the number.
     * @param data
     *           the number base characters.
     */
    private static String toBin(final int n, final char[] data) {
        int x = n;
        final StringBuilder sb = new StringBuilder();
        while (x > 0) {
            final int y = x % data.length;
            sb.append(data[y]);
            x /= data.length;
        }
        sb.reverse();
        // System.out.printf("RS: %s%n", sb);
        return sb.toString();
    }

    /**
     * Constructs a machine as it should.

```

```

*
* @param grid
*         the grid to operate on.
* @param f
*         the transition function.
* @param pos
*         the initial position.
* @return the finished product.
*/
public static ITurmite turmite(final Grid grid, final TransitionFunction
    f, final Coord pos) {
    switch (Factory.mImpl) {
    case Turmite:
        return new Turmite(grid, f, pos);
    case Classic:
        return new TuringMachine(grid, f, pos);
    default:
        throw new RuntimeException(
            "Somehow an enum with two values isn't set to either.");
    }
}
}
}

```

Listing C.2: multidimensional/ITurmite.java

```

package multidimensional;

/**
 * This is the interface for the machines simulated. Concrete types are
 * constructed by Factory. Do not use them directly.
 *
 * @author richard
 *
 */
public interface ITurmite {
    /**
     * Performs one transition.
     *
     * @return true if halted, false otherwise.
     */
    public boolean act();

    /**
     * @see java.lang.Object#equals(Object)
     */
    public boolean equals(final Object o);

    /**
     * @see java.lang.Object#hashCode()
     */
    public int hashCode();

    /**
     * Assigns a turmitelister to this machine.
     *
     * @param l
     *         the listener to receive events.
     */
}

```

```

    public void setListener(final TurmiteListener l);

    /**
     * @see java.lang.Object#toString()
     */
    public abstract String toString();

    /**
     * reverses the transition specified.
     *
     * @param t
     *         the last transition executed (hopefully).
     */
    public void undoTransition(Transition t);
}

```

Listing C.3: multidimensional/util/HashArray.java

```

package multidimensional.util;

import java.util.HashMap;
import java.util.Map;

import multidimensional.basics.Coord;

public class HashArray<T> implements CoordSet<T> {

    private final Map<Coord, T> grid = new HashMap<Coord, T>();

    private final T default_value;

    public HashArray(final T default_value) {
        this.default_value = default_value;
    }

    public T get(final Coord c) {
        final T i = this.grid.get(c);
        return (i == null) ? this.default_value : i;
    }

    public void set(final Coord c, final T i) {
        if (i.equals(this.default_value)) {
            this.grid.remove(c);
        } else {
            this.grid.put(c, i);
        }
    }
}

```

Listing C.4: multidimensional/util/CoordSet.java

```

package multidimensional.util;

import multidimensional.basics.Coord;

/**
 * this presents the interface of an array of T.
 *
 * @author richard
 */

```

```

* @param <T>
*/
public interface CoordSet<T> {
    public abstract T get(Coord pos);

    public abstract void set(Coord pos, T i);
}

```

Listing C.5: multidimensional/util/MakeDiagram.java

```

package multidimensional.util;

import java.io.File;
import java.util.Collection;
import java.util.HashSet;
import java.util.Set;

import multidimensional.Factory;
import multidimensional.Transition;
import multidimensional.TransitionFunction;
import multidimensional.basics.Dimensionality;

/**
 * Creates a state transition diagram for the turmite provided.
 *
 * @author richard
 */
public class MakeDiagram {
    public static void main(final String[] args) throws Exception {
        Dimensionality.setDimensions(3);
        boolean rule = false;
        for (final String arg : args) {
            TransitionFunction tf = null;
            if (arg.equals("--rule")) {
                rule = true;
                continue;
            }
            if (rule) {
                tf = Factory.fromRuleString(arg);
                rule = false;
            } else {
                tf = new multidimensional.gui.LoadSave(null)
                    .loadSetFromFile(new File(arg));
            }
            System.out.printf("%s\n", MakeDiagram.toDot(tf));
        }
        if (rule) {
            System.err.printf("Error: _expected_rule_string\n");
            System.exit(1);
        }
    }

    /**
     * This does the 'classic' turing machine diagram
     *
     * @param tf
     * @return
     */
}

```

```

static public String toDot(final TransitionFunction tf) {
    final StringBuilder sb = new StringBuilder();
    final Collection<Transition> trans = tf.toCollection();
    final Set<Integer> nodes = new HashSet<Integer>();
    sb.append(String.format("digraph_%s_{%n", tf.getName()));
    for (final Transition t : trans) {
        nodes.add(t.getOldState());
    }
    for (final Integer i : nodes) {
        sb.append(String.format("\tState%d_[label=\"%d\"];%n", i, i));
    }
    for (final Transition t : trans) {
        final String destState = nodes.contains(t.getNewState()) ?
            String
                .format("State%d", t.getNewState()) : "Halt";
        sb.append(String.format("\tState%d->_%s_"
            + "[label=\"%d/%d,%s\"];%n", t.getOldState(), destState,
                t
                    .getOldColour(), t.getNewColour(), t.getRotation()));
    }
    sb.append("}\n");
    return sb.toString();
}

/**
 * This one adds a box for each transition and puts the actions in there
 *
 * @param tf
 * @return
 */
static public String toDot2(final TransitionFunction tf) {
    final StringBuilder sb = new StringBuilder();
    final Transition[] trans = tf.toArray();
    final Set<Integer> nodes = new HashSet<Integer>();
    sb.append(String.format("digraph_%s_{%n", tf.getName()));
    for (int x = 0; x < trans.length; ++x) {
        final Transition t = trans[x];
        // build the list of states
        nodes.add(t.getOldState());
        // add nodes for the transitions.
        sb.append(String.format("\tTrans%d_"
            + "[label=\"%Write_%d\\nRotate_%s\"][shape=box];%n", x, t
                .getNewColour(), t.getRotation().toString()));
    }
    // print the states
    for (final Integer i : nodes) {
        sb.append(String.format("\tState%d;%n", i));
    }
    for (int x = 0; x < trans.length; ++x) {
        final Transition t = trans[x];
        sb.append(String.format("\tState%d->_Trans%d_"
            + "[label=\"%d\"];%n", t.getOldState(), x, t
                .getOldColour()));
        sb.append(String.format("\tTrans%d->_State%d;%n", x, t
            .getNewState()));
    }
    sb.append("}\n");
    return sb.toString();
}
}

```

Listing C.6: multidimensional/util/TreeArray.java

```

package multidimensional.util;

import java.util.Map;
import java.util.TreeMap;

import multidimensional.basics.Coord;

/**
 * This extends the SparseArray class into N dimensions, and replaces the
 * hash
 * lookup with a tree (as we can't really hash N integers into one).
 *
 * @author richard
 *
 * @param <T>
 *         the type to store.
 */
public class TreeArray<T> implements CoordSet<T> {

    private final Map<Coord, T> grid = new TreeMap<Coord, T>();

    private final T default_value;

    public TreeArray(final T default_value) {
        this.default_value = default_value;
    }

    @Override
    synchronized public T get(final Coord pos) {
        final T i = this.grid.get(pos);
        return (i == null) ? this.default_value : i;
    }

    /*
     * (non-Javadoc)
     *
     * @see multidimensional.util.CoordSet#set(multidimensional.Coord, T)
     */
    synchronized public void set(final Coord pos, final T i) {
        if (i.equals(this.default_value)) {
            this.grid.remove(pos);
        } else {
            this.grid.put(pos, i);
        }
    }
}

```

Listing C.7: multidimensional/util/WrapAroundStack.java

```

package multidimensional.util;

/** @hidden */
public class WrapAroundStack<T> {
    public static void main(final String[] args) {
        final WrapAroundStack<Integer> s = new WrapAroundStack<Integer>(7);
        for (int i = 0; i < 10; ++i) {
            System.out.printf("Pushing %d\n", i);
            s.push(i);
        }
    }
}

```

```

        for (int i = 0; i < 10; ++i) {
            final int x = s.pop();
            System.out.printf("popped_%d%n", x);
        }
    }

    private final Object data[];

    private int size, start;

    public WrapAroundStack(final int len) {
        if (len < 1) {
            throw new IllegalArgumentException(
                "Length_must_be_greater_than_zero");
        }
        this.data = new Object[len];
        this.size = 0;
        this.start = 0;
    }

    public T pop() {
        if (this.size == 0) {
            throw new IllegalStateException("Stack_underflow");
        }
        int index = this.start + this.size - 1;
        index %= this.data.length;
        --this.size;
        return (T) this.data[index];
    }

    public void push(final T ob) {
        if (this.size == this.data.length) {
            this.data[this.start++] = ob;
            this.start %= this.data.length;
        } else {
            this.data[(this.start + this.size) % this.data.length] = ob;
            ++this.size;
        }
    }

    public int size() {
        return this.size;
    }
}

```

Listing C.8: multidimensional/util/ZCoordSet.java

```

package multidimensional.util;

import java.util.Map;
import java.util.SortedSet;
import java.util.TreeMap;
import java.util.TreeSet;

import multidimensional.basics.Coord;

public class ZCoordSet<T> {
    static class OrderedItem<T> implements Comparable<OrderedItem<T>> {
        final int order;
    }
}

```

```

final T value;

public OrderedItem(final int order, final T value) {
    this.order = order;
    this.value = value;
}

@Override
public int compareTo(final OrderedItem<T> o) {
    return this.order - o.order;
}

@Override
public boolean equals(final Object o) {
    if (o == null) {
        return false;
    }
    if (o instanceof OrderedItem) {
        return this.order == ((OrderedItem) o).order;
    }
    return false;
}

}

@Override
public int hashCode() {
    return this.order;
}
}

public static class Test {
    public static void printf(final String fmt, final Object... args) {
        System.out.printf(fmt, args);
    }

    ZCoordSet<Integer> data = new ZCoordSet<Integer>(0, 1);

    public Test() {
        this.testTrunc(new Coord(1, 2, 3, 4));
        this.testSet(new Coord(1, 2, 3, 4), 5);
        this.testSet(new Coord(1, 4, 3, 4), 4);
        this.testSet(new Coord(1, 9, 3, 4), 3);
        this.testSet(new Coord(1, -9, 3, 4), 11);
    }

    public void get(final Coord c) {
    }

    public void testSet(final Coord c, final int i) {
        this.testTrunc(c);
        Test.printf("Setting %s to %d%n", c, i);
        final boolean s = this.data.set(c, i);
        Test.printf("Set returned %b%n", s);
    }

    public void testTrunc(final Coord c) {
        Test.printf("trunc: %s -> %s%n", c, this.data.mkItem(c));
    }
}

```

```

private static class ZComponent<T> {
    SortedSet<OrderedItem<T>> data = new TreeSet<OrderedItem<T>>();

    public OrderedItem<T> get() {
        if (this.data.size() == 0) {
            return null;
        } else {
            return this.data.first();
        }
    }

    public boolean put(final int index, final T ob) {
        final OrderedItem<T> o = new OrderedItem<T>(index, ob);
        this.data.add(o);
        final OrderedItem<T> ret = this.get();

        // System.out.printf("Inserted %d, got back %d\n", index,
        // ret.order);
        return o.equals(ret);
    }

    public void remove(final int index) {
        final OrderedItem<T> o = new OrderedItem<T>(index, null);
        this.data.remove(index);
    }
}

public static void main(final String[] args) {
    new Test();
}

Map<Coord, ZComponent<T>> data = new TreeMap<Coord, ZComponent<T>>();
private final int index;
private final boolean MAX = true;
private final boolean MIN = false;

boolean minMax = this.MAX;

T default_value;

ZCoordSet(final T defaultValue, final int index) {
    this.default_value = defaultValue;
    this.index = index;
}

// @Override
public T get(final Coord pos) {
    final ZComponent<T> col = this.data.get(this.mkItem(pos));
    if (col == null) {
        return this.default_value;
    }
    final T ret = col.get().value;
    return ret == null ? this.default_value : ret;
}

private Coord mkItem(final Coord pos) {
    final int d = pos.getDimensions();
    if (this.index >= d) {
        throw new IllegalStateException(
            "Coord_doesn't_have_enough_dimensions.");
    }
}

```

```

    final int [] ret = new int [d - 1];
    {
        int i, curr;
        for (i = 0, curr = 0; i < d; ++i) {
            if (i == this.index) {
                continue;
            }
            ret[curr++] = pos.getComponent(i);
        }
    }
    return new Coord(ret);
}

// @Override
public boolean set(final Coord pos, final T i) {
    final int order = pos.getComponent(this.index);
    final Coord realPos = this.mkItem(pos);
    ZComponent<T> col = this.data.get(realPos);
    if (col == null) {
        col = new ZComponent<T>();
        this.data.put(realPos, col);
    }

    return col.put(order, i);
}
}

```

Listing C.9: multidimensional/test/Matcher.java

```

package multidimensional.test;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.util.List;
import java.util.Vector;

import javax.swing.JFrame;
import javax.swing.WindowConstants;

import multidimensional.Factory;
import multidimensional.ITurmite;
import multidimensional.Transition;
import multidimensional.TransitionFunction;
import multidimensional.TurmiteEvent;
import multidimensional.TurmiteException;
import multidimensional.TurmiteListener;
import multidimensional.basics.Dimensionality;
import multidimensional.gui.RunPanel;

/**
 * @hidden
 * @author richard
 *
 */
public class Matcher extends JFrame implements TurmiteListener {
    public static void main(final String [] args) throws TurmiteException {
        Dimensionality.setDimensions(3);
        final int i = args.length < 1 ? 444 : Integer.parseInt(args[0]);
        final Matcher m = Matcher.Matcher2d(i);
    }
}

```

```

static public Matcher Matcher2d(final int i) throws TurmiteException {
    return new Matcher(Factory.tf2dint(i));
}

static public Matcher Matcher3d(final int i) throws TurmiteException {
    return new Matcher(Factory.tf3dint(i));
}

private final TransitionFunction t;

private final List<Transition> history = new Vector<Transition>();

boolean running = false;
RunPanel rp;

Dimension window = new Dimension(600, 600);

ITurmite turm;

public Matcher(final TransitionFunction t) {
    super(t.getName());
    this.t = t;
    this.running = true;
    this.rp = new RunPanel(this.window, 5);
    this.rp.addTurmite(t).setListener(this);
    this.setLayout(new BorderLayout());
    this.add(this.rp);
    this.pack();
    this.setVisible(true);
    this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
}

@Override
public void send(final TurmiteEvent e) {
    final Transition t = e.getTransition();
    this.history.add(t);
    System.out.printf("%4X%n", t.hashCode());
    System.out.flush();
}
}

```

Listing C.10: multidimensional/test/SerialTest.java

```

package multidimensional.test;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.util.List;

import multidimensional.Factory;
import multidimensional.Transition;
import multidimensional.TransitionFunction;
import multidimensional.basics.Dimensionality;

```

```

import multidimensional.basics.Rotation;

/**
 * @hidden
 *
 * @author richard
 *
 */
public class SerialTest {
    public static void main(final String[] args) throws Exception {
        final String fileName = "tha_ant2";
        Dimensionality.setDimensions(3);
        final TransitionFunction tf = Factory.emptyTF();
        final List<Rotation> rot = Dimensionality.getInstance().getRots();
        tf.add(new Transition(0, 0, rot.get(2), 0, 1));
        tf.add(new Transition(0, 1, rot.get(3), 0, 0));

        final OutputStream os = new FileOutputStream(fileName);
        final ObjectOutput oo = new ObjectOutputStream(os);
        oo.writeObject(tf);
        oo.close();

        // Reading Objects

        final InputStream is = new FileInputStream(fileName);
        final ObjectInput oi = new ObjectInputStream(is);
        final TransitionFunction newObj = (TransitionFunction) oi.readObject
            ();

        System.out.println(tf.equals(newObj) ? "WIN" : "LOSE");

        System.out.printf("old:\t%s%nnew:\t%s%n", tf, newObj);
        oi.readObject();
        oi.close();
    }
}

```

Listing C.11: multidimensional/view/Multitest.java

```

package multidimensional.view;

import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import multidimensional.Factory;
import multidimensional.ITurmite;
import multidimensional.TransitionFunction;
import multidimensional.basics.Coord;
import multidimensional.basics.Dimensionality;
import multidimensional.basics.Grid;

/** @hidden */
public class Multitest implements Runnable {
    static Thread displayT = new Thread(new Multitest());
    static boolean bQuit = false;
    static String rulestring = "LLRLLRRUDUDDL";

    public static void main(final String[] args) {
        Dimensionality.setDimensions(3);
        if (args.length > 0) {

```

```

        Multitest.rulestring = args[0];
    }
    Multitest.displayT.start();
}

public void run() {
    try {
        final Frame frame = new Frame("3d_Turmite_display_test");
        // GLCanvas canvas = new GLCanvas();
        final GridView3d view = new GridView3d();
        final Grid grid = Factory.grid();
        grid.addGridListener(view);

        final TransitionFunction trans = Factory
            .fromRuleString(Multitest.rulestring);
        final ITurmite mite = Factory.turmite(grid, trans, new Coord(0,
            0, 0));

        // canvas.addGLEventListener(view);
        frame.add(view);
        frame.setSize(640, 480);
        // frame.setUndecorated(true);
        int size = frame.getExtendedState();
        size |= Frame.MAXIMIZED_BOTH;
        frame.setExtendedState(size);

        frame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(final WindowEvent e) {
                Multitest.bQuit = true;
                System.exit(0);
            }
        });
        frame.setVisible(true);
        // frame.show();
        view.requestFocus();

        while (!Multitest.bQuit) {
            view.display();
            mite.act();
            Thread.yield();
        }
    } catch (final Exception e) {
        e.printStackTrace(System.out);
        return;
    }
}
}

```

Listing C.12: multidimensional/view/Pallete.java

```

package multidimensional.view;

import java.awt.Color;

/** @hidden */
public class Pallete {
    private static float [][] colours = new float [][] {

```

```

new float [] { 0.0f, 0.0f, 0.0f }, new float [] { 1.0f, 0.0f, 0.0f },
    new float [] { 0.0f, 1.0f, 0.0f }, new float [] { 0.0f, 0.0f, 1.0f
    },
    new float [] { 1.0f, 1.0f, 0.0f }, new float [] { 1.0f, 0.0f, 1.0f
    },
    new float [] { 0.0f, 1.0f, 1.0f }, new float [] { 0.5f, 1.0f, 0.0f
    },
    new float [] { 0.5f, 0.5f, 0.0f }, new float [] { 0f, 0.5f, 0.2f
    },
    new float [] { 1.0f, 0.0f, 0.0f },
    new float [] { 0.59f, 0.3f, 0.0f },
    new float [] { 1.0f, 0.0f, 0.0f }, new float [] { 1.0f, 0.0f, 0.0f
    },
    new float [] { 1.0f, 0.0f, 0.0f }, new float [] { 1.0f, 0.0f, 0.7f
    },
    new float [] { 1.0f, 0.0f, 0.0f } };

public static Color getColour(final int i) {
    final float [] ret = Palette.getFloatArr(i);
    return new Color(ret[0], ret[1], ret[2]);
}

public static float [] getFloatArr(final int i) {
    if (i > Palette.colours.length - 1 || i < 0) {
        throw new IllegalArgumentException(String.format(
            "Colour index must in range %d-%d. Got %d.", 0,
            Palette.colours.length, i));
    }
    return Palette.colours[i];
}
}

```

Listing C.13: multidimensional/view/CoordComparator.java

```

package multidimensional.view;

import java.util.Comparator;

import multidimensional.basics.Coord;

/** @hidden */
public final class CoordComparator implements Comparator<Coord> {

    @Override
    public int compare(final Coord o1, final Coord o2) {
        for (int i = o1.getDimensions() - 1; i >= 0; --i) {
            final int cmp = o1.getComponent(i) - o2.getComponent(i);
            if (cmp != 0) {
                return cmp;
            }
        }
        return 0;
    }
}

```

Listing C.14: multidimensional/view/CubeRepresentation.java

```

package multidimensional.view;

```

```

import javax.media.opengl.GL;

public class CubeRepresentation extends ElementRepresentation {

    @Override
    public void drawElement(final int value, final GL gl) {
        ElementRepresentation.doColour(value, gl);
        ElementRepresentation.glu.glutSolidCube(1.0f);
    }
}

```

Listing C.15: multidimensional/view/MouseHandler.java

```

package multidimensional.view;

import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import java.util.Stack;

import javax.swing.event.MouseInputListener;

public class MouseHandler implements MouseInputListener, MouseWheelListener
{
    /** @hidden */
    private static class Pair<A, B> {
        A a;
        B b;

        public Pair(final A a, final B b) {
            this.a = a;
            this.b = b;
        }
    }

    private final Stack<Pair<Point, Integer>> stack = new Stack<Pair<Point,
        Integer>>();
    // as we get a lot of drag events, don't refresh each time.
    private static int DRAG.UPDATELATENCY = 5;
    /**
     * how many drag events remain before we update.
     */
    private int eventsTillRefresh = MouseHandler.DRAG.UPDATELATENCY;
    /**
     * The component that we are manipulating.
     */
    private final GridView target;

    /**
     * Constructs a mouse listener for the specified GridViewFixed. Note
     * that
     * this class automatically adds itself as the relevant listeners, as it
     * implements a number of interfaces and they are subject to change.
     *
     * @param target
     * the GridViewFixed that this listener should operate on.
     */
    public MouseHandler(final GridView target) {
        this.target = target;
    }
}

```

```

        target.addMouseMotionListener(this);
        target.addMouseWheelListener(this);
        target.addMouseListener(this);
    }

    private void dragDone(final int x, final int y, final int button) {
        // System.out.printf("Drag done: %3d,%3d%n",x,y);
        switch (button) {
            case 1:
                this.target.alterPosition(x, y);
                break;
            case 3:
                this.target.alterRotation(x, y);
                break;
            case 2:
                this.target.alterScale((float) y / 3);
                break;
            default:
                break;
        }
    }
}

@Override
public void mouseClicked(final MouseEvent e) {
}

@Override
public void mouseDragged(final MouseEvent e) {
    final Pair<Point, Integer> item = this.stack.peek();
    if (--this.eventsTillRefresh > 0) {
        return;
    }
    final int dx = item.a.x - e.getX();
    final int dy = item.a.y - e.getY();
    this.dragDone(dx, dy, item.b);
    item.a = e.getPoint();
    // System.out.printf("%d,%d%n", dx, dy);
    this.eventsTillRefresh = MouseHandler.DRAG.UPDATELATENCY;
    return;
}

// don't use these but they're required by the interfaces.
@Override
public void mouseEntered(final MouseEvent e) {
}

@Override
public void mouseExited(final MouseEvent e) {
}

@Override
public void mouseMoved(final MouseEvent e) {
}

@Override
public void mousePressed(final MouseEvent e) {
}

```

```

        this.stack.push(new Pair<Point, Integer>(e.getPoint(), e.getButton()
            ));
    }

    @Override
    public void mouseReleased(final MouseEvent e) {
        final Pair<Point, Integer> el = this.stack.pop();
        // at the end of a drag, calculate the total movement and tell the
        // target.
        final Point p = e.getPoint();
        final int dx = el.a.x - p.x;
        final int dy = el.a.y - p.y;
        this.dragDone(dx, dy, el.b);
        // target.alterPosition(dx, dy);
    }

    @Override
    public void mouseWheelMoved(final MouseWheelEvent e) {
        // zoom in and out using the mouse wheel.
        final int delta = e.getWheelRotation();
        this.target.alterScale(((float) -delta));
    }
}

```

Listing C.16: multidimensional/view/GridView2d.java

```

package multidimensional.view;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Point;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

import javax.swing.JPanel;

import multidimensional.basics.Coord;

public class GridView2d extends JPanel implements GridView {
    // / @TODO need to do some z ordering.
    private final Map<Coord, Integer> graph = new TreeMap<Coord, Integer>(
        new CoordComparator());

    /**
     * How big each cell on the grid is on the screen (as a square).
     */
    private int scale = GridView2d.DEFAULT_SCALE;

    /**
     * The size of the desired panel in pixels.
     */
    private Dimension dim;

    /**

```

```

    * The co-ordinate of the top-left point.
    */
    volatile private Point origin;
    /**
     * the number of cells to display
     */
    private Dimension window;
    /**
     * the offscreen buffer. (to prevent flicker)
     */
    private Image otherImg;
    /**
     * Graphics context for the offscreen buffer.
     */
    private Graphics framebuffer;
    private static final int DEFAULT.SCALE = 4;

    public GridView2d(final Dimension window) {
        this.origin = new Point(0, 0);
        this.setBackground(Pallete.getColour(0));
        new MouseHandler(this);
        this.dim = (Dimension) window.clone();
        this.setGridSize(window, this.scale);
        this.setPreferredSize(this.dim);
        // handle the resize correctly
        this.addComponentListener(new ComponentAdapter() {
            @Override
            public void componentResized(final ComponentEvent e) {
                final Dimension s = GridView2d.this.getSize();

                GridView2d.this.setGridSize(s, GridView2d.this.scale);
            }
        });
    }

    // the position of the camera relative to the origin.
    /*
     * (non-Javadoc)
     *
     * @see multidimensional.view.GridView#alterPosition(int, int)
     */
    public void alterPosition(final int x, final int y) {
        this.moveOrigin(x, y);
        this.repaint();
    }

    /*
     * (non-Javadoc)
     *
     * @see multidimensional.view.GridView#alterRotation(int, int)
     */
    public void alterRotation(final int x, final int y) {
        // the 2d view doesn't rotate
        return;
    }

    /*
     * (non-Javadoc)
     *

```

```

    * @see multidimensional.view.GridView#alterScale(float)
    */
    public void alterScale(final float v) {
        this.setGridSize(this.dim, this.scale + (int) v);
        this.repaint();
    }

    /*
     * (non-Javadoc)
     * @see multidimensional.view.GridView#gridUpdated(multidimensional.
     *   Coord,
     *   int)
     */
    @Override
    synchronized public void gridUpdated(final Coord pos, final int value) {
        if (value == 0) {
            this.graph.remove(pos);
        } else {
            this.graph.put(pos, value);
        }
        this.putPixel(pos, value);
        this.repaint();
    }

    @Override
    public boolean isDoubleBuffered() {
        return true;
    }

    /**
     * we can't allocate an offscreen buffer before there is a drawing
     * context
     * hence we need to check before we start drawing.
     */
    synchronized private void makeSureWeHaveAbuffer() {
        if (this.otherImg == null) {
            this.otherImg = this.createImage(this.dim.width, this.dim.height);
        }

        if (this.otherImg == null) {
            throw new RuntimeException(
                "Tried to allocate offscreen buffer before we have a
                graphics context.");
        }
        this.framebuffer = this.otherImg.getGraphics();
        this.refresh();
    }

    synchronized public void moveOrigin(final int dx, final int dy) {
        final int _dx = dx / this.scale;
        final int _dy = dy / this.scale;

        // System.err.printf("Applied (%3d,%3d) transform to the origin%n",
        //   _dx,
        //   _dy);
        this.setOrigin(this.origin.x + _dx, this.origin.y + _dy);
    }

```

```

@Override
synchronized public void paint(final Graphics g) {
    this.makeSureWeHaveAbuffer();
    g.drawImage(this.otherImg, 0, 0, this);
}

synchronized private void putPixel(final Coord pos, final int c) {
    final int w2 = this.window.width / 2;
    final int h2 = this.window.height / 2;
    final int x = pos.getComponent(0) - this.origin.x;
    final int y = pos.getComponent(2) - this.origin.y; // dimensions -1?
    // System.out.printf("Scale=%d\n", scale);
    if (x < -w2 || x >= w2 || y < -h2 || y >= h2) {
        return;
    }
    // /throw new RuntimeException("sss");
    this.makeSureWeHaveAbuffer();
    this.framebuffer.setColor(Palette.getColour(c));
    this.framebuffer.fillRect((x + w2) * this.scale, (y + w2) * this.
        scale,
        this.scale, this.scale);
}

synchronized private void refresh() {
    this.framebuffer.setColor(Palette.getColour(0));
    this.framebuffer.fillRect(0, 0, this.dim.width, this.dim.height);
    final Set<Map.Entry<Coord, Integer>> values = this.graph.entrySet();
    for (final Map.Entry<Coord, Integer> curr : values) {
        this.putPixel(curr.getKey(), curr.getValue());
    }
    this.repaint();
}

// Synchronise everything touching the data as it can change at any time
// and be called from multiple threads (main and awt-event).
synchronized private void setGridSize(final Dimension window,
    final int scale) {
    this.scale = scale < 1 ? 1 : scale;
    this.dim = (Dimension) window.clone();
    this.setPreferredSize(this.dim);
    // if (window != this.window)
    // this.window = (Dimension) window.clone();
    this.window = new Dimension(window.width / this.scale, window.height
        / this.scale);
    this.otherImg = null;
    System.err.printf("New scale=%d\n", this.scale);
    System.out.printf("dim; %s\nwin:%s\n", this.dim, this.window);
    //
}

synchronized public void setOrigin(final int x, final int y) {
    this.origin = new Point(x, y);
    this.refresh();
}

/**
 * Overridden so awt doesn't cause flicker by blanking the panel.
 */

```

```

    @Override
    synchronized public void update(final Graphics g) {
        this.paint(g);
    }
}

```

Listing C.17: multidimensional/view/ViewProxy.java

```

package multidimensional.view;

import java.util.Map;
import java.util.TreeMap;

import javax.media.opengl.GL;

/** @hidden */
public class ViewProxy extends ElementRepresentation {
    private final Map<Integer, ElementRepresentation> map = new TreeMap<
        Integer, ElementRepresentation>();
    private final ElementRepresentation DEFAULTREP = new CubeRepresentation
        ();

    // private static class IntRange implements Comparable<IntRange>{
    // private int min,max;
    // private Map<IntRange, ElementRepresentation> map=new
    // Map<IntRange, ElementRepresentation>();
    // @Override
    // public int compareTo(IntRange o) {
    // if((o.max > this.min && o.min < this.max)
    // || (o.min < this.max && o.max > this.min))
    // //
    // throw new RuntimeException("These ranges overlap!");
    // if(o.min > this.max) return -1;
    // if(o.max < this.min) return 1;
    // //
    // return 0;
    // }
    // }
    // }
    synchronized public void add(final int value, final
        ElementRepresentation e) {
        this.map.put(value, e);
    }

    @Override
    synchronized public void drawElement(final int value, final GL gl) {
        ElementRepresentation e = this.map.get(value);
        if (e == null) {
            e = this.DEFAULTREP;
        }
        e.drawElement(value, gl);
    }
}

```

Listing C.18: multidimensional/view/ElementRepresentation.java

```

package multidimensional.view;

```

```

import javax.media.opengl.GL;

import com.sun.opengl.util.GLUT;

/**
 * Classes that implement this interface provide the three-dimensional
 * visual
 * for an element or cell.
 *
 * This is to allow multiple possibilities and for them to be selectable at
 * run-time.
 *
 * @author richard
 */
public abstract class ElementRepresentation {
    protected static final GLUT glu = new GLUT();

    protected static void doColour(final int i, final GL gl) {
        final float [] c = Palette.getFloatArr(i);
        gl.glColor4f(c[0], c[1], c[2], 0.5f);
        gl.glMaterialfv(GL.GLFRONT.AND.BACK, GL.GLDIFFUSE, c, 0);
    }

    public abstract void drawElement(int value, GL gl);
}

```

Listing C.19: multidimensional/view/GridView.java

```

package multidimensional.view;

import java.awt.event.KeyListener;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseWheelListener;

import multidimensional.basics.Coord;
import multidimensional.basics.Grid;
import multidimensional.basics.GridListener;

public interface GridView extends GridListener {
    public void addKeyListener(KeyListener l);

    // these are from component, but seeing as we can't use a generic
    // parameter
    // as a superclass I have to duplicate them here.
    // see java.awt.Component/Panel
    public void addMouseListener(MouseListener l);

    public void addMouseMotionListener(MouseMotionListener l);

    public void addMouseWheelListener(MouseWheelListener l);

    // the position of the camera relative to the origin.
    /**
     * Moves the view around by the specified x/y position.
     *
     * @param x
     *         the horizontal offset.
     * @param y
     *         the vertical offset.
     */
}

```

```

    */
    public void alterPosition(int x, int y);

    /**
     * Rotates the view with the x/y offset from the mouse. Optional method.
     * Do
     * not throw an exception if this is not supported. Just ignore it.
     *
     * @param x
     *         the horizontal offset.
     * @param y
     *         the vertical offset.
     */
    public void alterRotation(int x, int y);

    /**
     * Zooms in or out.
     *
     * @param delta
     *         the amount. This can be treated differently by the
     *         different
     *         views.
     */
    public void alterScale(float delta);

    /**
     * @see Grid.GridListener#gridUpdated(Coord, value)
     */
    public void gridUpdated(Coord pos, int value);

    public void repaint();
}

```

Listing C.20: multidimensional/view/GridView3d.java

```

package multidimensional.view;

import java.awt.Dimension;
import java.util.Map;
import java.util.TreeMap;

import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLJPanel;
import javax.media.opengl.glu.GLU;

import multidimensional.basics.Coord;

public class GridView3d extends GLJPanel implements GLEventListener,
    GridView {

    private final ElementRepresentation representation;
    private final GLCapabilities caps = new GLCapabilities();
    private final Map<Coord, Integer> graph = new TreeMap<Coord, Integer>(
        new CoordComparator());

    private final float zoom[] = new float[] { 0.0f, 0.0f, -50f };

    private final float position[] = new float[] { 0.0f, 0.0f, 0.0f };
}

```

```

private final float rotation [] = new float [] { 30.0f, 0.0f, 30.0f };

/**
 * The space between each element. The elements are taken to be of unit
 * size.
 */
private static final float DRAWDISTANCE = 1.3f;

private static int ANTLALIASING = -1;

private static final ElementRepresentation getRepresentation() {
    final int i = 1;
    switch (i) {
        case 0:
            return new SphereRepresentation();
        case 1:
        default:
            return new CubeRepresentation();
    }
}

public static void setLights(final boolean b) {
    GridView3d.LIGHTING_ENABLED = b;
}

// the light positions and colours (white)
private final float [] lightAmbient = { 0.5f, 0.5f, 0.5f, 1.0f };
private final float [] lightDiffuse = { 1.0f, 1.0f, 1.0f, 1.0f };

private final float [] lightPosition = { 0.0f, 0.0f, 2.0f, 1.0f };
private static boolean LIGHTING_ENABLED = true;
private static final GLU glu = new GLU();

public static void setAaSamples(final int n) {
    GridView3d.ANTLALIASING = n < 2 ? -1 : n;
}

public GridView3d() {
    this.caps.setDoubleBuffered(true);
    this.representation = GridView3d.getRepresentation();
    new MouseHandler(this);
    this.addGLEventListener(this);
}

public GridView3d(final Dimension size) {
    this();
    this.setPreferredSize(size);
    this.setSize(size);
}

// the position of the camera relative to the origin.
/*
 * (non-Javadoc)
 *
 * @see multidimensional.view.GridView#alterPosition(int, int)
 */
public void alterPosition(final int x, final int y) {
    this.position[0] -= 0.1 * (float) x;
    this.position[1] += 0.1 * (float) y;
    this.repaint();
}

```

```

/*
 * (non-Javadoc)
 *
 * @see multidimensional.view.GridView#alterRotation(int, int)
 */
public void alterRotation(final int x, final int y) {
    this.rotation[0] -= 0.1 * (float) y;
    this.rotation[1] -= 0.1 * (float) x;
    this.repaint();
}

/*
 * (non-Javadoc)
 *
 * @see multidimensional.view.GridView#alterScale(float)
 */
public void alterScale(final float delta) {
    this.zoom[2] += delta;
    this.repaint();
}

@Override
public void display(final GLAutoDrawable drawable) {
    final GL gl = drawable.getGL();
    gl.glLoadIdentity();
    // camera position and orientation
    this.translate(gl, this.zoom);
    this.rotate(gl, this.rotation);
    this.translate(gl, this.position);

    // push it so we don't need to keep doing the above translation.
    gl.glPushMatrix();
    gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    synchronized (this.graph) {
        for (final Map.Entry<Coord, Integer> e : this.graph.entrySet())
        {
            // peek
            gl.glPopMatrix();
            gl.glPushMatrix();
            final float [] curr = e.getKey().toFloatArray(
                GridView3d.DRAWDISTANCE);
            // go to the right place
            this.translate(gl, curr);
            this.representation.drawElement(e.getValue(), gl);
        }
    }
    gl.glPopMatrix(); // re-align matrix stack
    gl.glFlush();
}

@Override
public void displayChanged(final GLAutoDrawable drawable,
    final boolean modeChanged, final boolean deviceChanged) {
    // don't really care
}

/*
 * (non-Javadoc)
 *

```

```

* @see multidimensional.view.GridView#gridUpdated(multidimensional.
  Coord,
* int)
*/
@Override
public void gridUpdated(final Coord pos, final int value) {
    synchronized (this.graph) {
        if (value == 0) {
            this.graph.remove(pos);
        } else {
            this.graph.put(pos, value);
        }
    }
}

@Override
public void init(final GLAutoDrawable drawable) {
    final GL gl = drawable.getGL();
    gl.glShadeModel(GL.GL_SMOOTH);
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    gl.glClearDepth(1.0f);
    gl.glEnable(GL.GL_DEPTH_TEST);
    gl.glDepthFunc(GL.GL_LEQUAL);
    gl.glHint(GL.GL_PERSPECTIVE_CORRECTION_HINT, GL.GL_NICEST);
    if (GridView3d.ANTI_ALIASING > 1) {
        this.caps.setSampleBuffers(true);
        this.caps.setNumSamples(GridView3d.ANTI_ALIASING);
    }
    // set up the lights if we want them.
    if (GridView3d.LIGHTING_ENABLED) {
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_AMBIENT, this.lightAmbient, 0);
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_DIFFUSE, this.lightDiffuse, 0);
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPECULAR, this.lightDiffuse, 0);
        ;
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_POSITION, this.lightPosition,
            0);
        gl.glEnable(GL.GL_LIGHT1);
        gl.glEnable(GL.GL_LIGHTING);
    }
    // let it cull the faces.
    gl.glCullFace(GL.GL_BACK);
    gl.glEnable(GL.GL_CULL_FACE);
}

@Override
public void reshape(final GLAutoDrawable drawable, final int x,
    final int y, final int width, int height) {
    final GL gl = drawable.getGL();
    if (height <= 0) {
        height = 1;
    }
    final float h = (float) width / (float) height;
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();
    GridView3d.glu.gluPerspective(40f, h, 0.1f, 500.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
}

private void rotate(final GL gl, final float[] r) {
    // gl.glRotatef(50f, r[0], r[1], r[2]);
    gl.glRotatef(r[0], 1, 0, 0);
}

```

```

        gl.glRotatef(r[1], 0, 1, 0);
        gl.glRotatef(r[2], 0, 0, 1);
    }

    /**
     * Translates the current modelview matrix by the array provided.
     * Accounts
     * for the case where the order is <math>3</math>.
     *
     * @param gl
     *         the JOGL instance.
     * @param t
     *         the vector to move it by.
     */
    private void translate(final GL gl, final float[] t) {
        if (t.length == 3) {
            gl.glTranslatef(t[0], t[1], t[2]);
        } else { // handle the case where it may be a one or two dimensional
            // simulation.
            final float[] temp = new float[3];
            for (int i = t.length - 1; i >= 0; --i) {
                temp[i] = t[i];
            }
            this.translate(gl, temp);
        }
    }
}

```

Listing C.21: multidimensional/view/SphereRepresentation.java

```

package multidimensional.view;

import javax.media.opengl.GL;

public class SphereRepresentation extends ElementRepresentation {
    private static final int SUBDIVISIONS = 32;

    @Override
    public void drawElement(final int value, final GL gl) {
        ElementRepresentation.doColour(value, gl);
        ElementRepresentation.glu.glutSolidSphere(0.5f,
            SphereRepresentation.SUBDIVISIONS,
            SphereRepresentation.SUBDIVISIONS);
    }
}

```

Listing C.22: multidimensional/gui/ActionDesigner.java

```

package multidimensional.gui;

import java.awt.Component;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JList;
import javax.swing.JOptionPane;

```

```

import javax.swing.JPanel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import multidimensional.Transition;
import multidimensional.basics.Rotation;
import multidimensional.gui.widgets.DirectionPicker;
import multidimensional.gui.widgets.InputBox;

/**
 * This form designs a state transition, and adds it to the list provided.
 *
 * @author richard
 *
 */
public class ActionDesigner extends JPanel implements ActionListener,
    ListSelectionListener {
    private final InputBox startState = new InputBox("Initial_State", 5, "0"
    );
    private final InputBox endState = new InputBox("Resulting_State", 5, "0"
    );
    private final InputBox startCol = new InputBox("Initial_Colour", 5, "0"
    );
    private final InputBox endCol = new InputBox("Resulting_Colour", 5, "0"
    );

    private final DirectionPicker direction = new DirectionPicker();
    private final JButton addButton = new JButton("Add_transition");
    private final JButton resetButton = new JButton("reset");

    private ActionListener target = null;

    public ActionDesigner(final ActionListener al) {
        this.target = al;
        this.setLayout(new GridLayout(0, 1));
        final JPanel p = new JPanel();
        p.setLayout(new GridLayout(0, 2));
        p.add(this.resetButton);
        p.add(this.addButton);
        for (final Component c : new Component[] { this.startState,
            this.startCol, this.endState, this.endCol, this.direction, p
            }) {
            this.add(c);
        }
        this.resetButton.addActionListener(this);
        this.addButton.addActionListener(this);
        this.target.addListSelectionListener(this);
        this.setToolTips();
    }

    @Override
    public void actionPerformed(final ActionEvent e) {
        if (e.getSource() == this.resetButton) {
            this.resetForm();
        } else if (e.getSource() == this.addButton) {
            final Transition a = this.fromForm();
            if (a != null) {
                this.target.add(a);
                System.out.println(a);
            }
        }
    }
}

```

```

/**
 * Creates a transition from the contents of the form.
 *
 * @return the resultant transition, or null if there was an error.
 */
private Transition fromForm() {
    int oldstate, newstate;
    int oldcolour, newcolour;
    Transition retval = null;
    String errmsg = "THIS_IS_A_BUG_IF_YOU_SEE_THIS!!!_(and_a_highly_
        impossible_one)";
    try {
        errmsg = "initial_state";
        oldstate = Integer.parseInt(this.startState.getText());
        errmsg = "initial_colour";
        newstate = Integer.parseInt(this.endState.getText());
        errmsg = "new_state";
        oldcolour = Integer.parseInt(this.startCol.getText());
        errmsg = "new_colour";
        newcolour = Integer.parseInt(this.endCol.getText());
        retval = new Transition(oldstate, oldcolour,
            (Rotation) this.direction.getSelectedItemAt(newstate,
                newcolour);
    } catch (final NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Bad_numeric_input_for_"
            + errmsg, "Error", JOptionPane.ERROR_MESSAGE, null);
    }
    return retval;
}

/**
 * This resets the form to the 'default' values.
 */
private void resetForm() {
    this.endState.setText("0");
    this.startCol.setText("0");
    this.endCol.setText("0");
    this.startState.setText("0");
    this.direction.setSelectedIndex(0);
}

/**
 * Sets the tooltips for the various controls.
 */
private void setToolTips() {
    this.startState
        .setToolTipText("The_initial_state_of_the_turmite_for_this_
            rule");
    this.endState
        .setToolTipText("The_state_to_change_to_after_this_rule_is_
            used.");
    this.startCol
        .setToolTipText("The_colour_under_the_turmite_to_begin_with.
            ");
    this.endCol
        .setToolTipText("The_colour_to_set_at_the_current_location."
            );
    this.direction
        .setToolTipText("The_rotation_to_apply_to_the_turmite_during_
            this_rule.");
}

```

```

        this.addButton
            .setToolTipText("Add_this_rule_to_the_transition_function.")
            ;
        this.resetButton.setToolTipText("Resets_this_side_of_the_form.");
    }

    /**
     * This sets the contents of the controls to reflect the Transition
     * passed.
     *
     * @param t
     */
    private void toForm(final Transition t) {
        this.startState.setText(String.format("%d", t.getOldState()));
        this.endState.setText(String.format("%d", t.getNewState()));
        this.startCol.setText(String.format("%d", t.getOldColour()));
        this.endCol.setText(String.format("%d", t.getNewColour()));
        this.direction.setSelectedItem(t.getRotation());
    }

    @Override
    public void valueChanged(final ListSelectionEvent e) {
        // get the selected object from the list
        final Object sel = (((JList) e.getSource()).getSelectedValue());
        // to work around an issue in swing, it's possible there's a string
        // rather than a transition.
        if (sel instanceof Transition) {
            this.toForm((Transition) sel);
        }
    }
}

```

Listing C.23: multidimensional/gui/Main.java

```

package multidimensional.gui;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.UIManager;

import multidimensional.Factory;
import multidimensional.TransitionFunction;
import multidimensional.TurmiteException;

/**
 * The entry point for the program.
 *
 * @author richard
 */
public class Main extends JPanel {
    /**
     * @param args
     */
}

```

```

*          Filename of transition function to pre-load, optional.
*          This is
*          for file association etc.
*/
public static void main(final String args[]) {
    multidimensional.basics.Dimensionality.setDimensions(3);
    String finalLookAndFeel = null;
    // make it look better, this also fixes some rendering
    // issues on Linux with certain driver/WM combinations
    // where the system would just draw grey windows.
    for (final String lf : new String[] {
        UIManager.getSystemLookAndFeelClassName(),
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel",
        "com.sun.java.swing.plaf.gtk.GTKLookAndFeel" }) {
        try {
            UIManager.setLookAndFeel(lf);
            finalLookAndFeel = lf;
        } catch (final Exception e) {
            // may as well inform the user.
            System.err.printf("Non-fatal_exception_setting_l&f:%n_%s%n",
                e
                    .getMessage());
        }
    }
    if (finalLookAndFeel == null) {
        System.out
            .printf("No_look_and_feel_was_selected.%n"
                + "Problems_possible,_but_the_system_should_"
                + "still_have_the_default.%n");
    } else {
        System.out.printf("Using_look&feel:_%s%n", finalLookAndFeel);
    }
    if (args.length > 0) {
        new Main(args[0]);
    } else {
        new Main();
    }
}

private final ActionListener al;
private final ActionDesigner ad;
private final JButton run = new JButton("Start_Simulation");
private final JFrame f;
private final LoadSave fileAccess;

private final Menu.MenuEventListener menuListener = new Menu.
    MenuEventListener() {
        @Override
        public void fromRuleString() {
            final String rule = (String) JOptionPane.showInputDialog(Main.
                this,
                "Input_the_rule_string:", "Construct_from_rule_string",
                JOptionPane.QUESTION_MESSAGE, null, null, "LR");
            try {
                final TransitionFunction out = Factory.fromRuleString(rule);
                Main.this.al.setTransitionFunction(out);
            } catch (final TurmiteException ex) {
                JOptionPane.showMessageDialog(Main.this, String.format(
                    "Error:_%s", ex.getMessage()), "Error",
                    JOptionPane.ERROR_MESSAGE);
            }
        }
    }

```

```

}

@Override
public void load() {
    try {
        TransitionFunction ret = null;
        ret = Main.this.fileAccess.loadTurmite();
        if (ret != null) {
            Main.this.al.setTransitionFunction(ret);
        }
    } catch (final TurmiteException e1) {
        JOptionPane.showMessageDialog(Main.this, e1.getMessage());
        return;
    }
}

@Override
public void quit() {
    System.exit(0);
    // this.f.dispose();
}

@Override
public void save() {
    try {
        final TransitionFunction out = Main.this.al
            .getTransitionFunction();
        Main.this.fileAccess.saveTurmite(out);
    } catch (final TurmiteException e1) {
        JOptionPane.showMessageDialog(Main.this, e1.getMessage());
        // e1.printStackTrace();
        return;
    }
}

@Override
public void saveDot() {
    try {
        Main.this.fileAccess.saveDot(Main.this.al
            .getTransitionFunction());
    } catch (final TurmiteException e) {
        JOptionPane.showMessageDialog(Main.this, e.getMessage());
    }
}

@Override
public void start() {
    try {
        final TransitionFunction am = Main.this.al
            .getTransitionFunction();
        final JFrame f = new JFrame("Configuration");
        f.setLayout(new BorderLayout());
        f.addWindowListener(MyWindowListener.getInstance());
        f.add(new ConfigPanel(am));
        f.pack();
        f.setVisible(true);
    } catch (final TurmiteException e1) {
        JOptionPane.showMessageDialog(Main.this, e1.getMessage());
        // e1.printStackTrace();
        return;
    }
}

```

```

    }
};
final Menu menu;

private Main() {
    this.f = new JFrame("Turmite_editor");
    this.menu = new Menu();
    this.f.setJMenuBar(this.menu.createMenuBar());
    this.menu.addListener(this.menuListener);
    this.al = new ActionListener();
    this.ad = new ActionDesigner(this.al);
    this.ad.add(this.run);
    this.run.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(final ActionEvent e) {
            Main.this.menuListener.start();
        }

    });
    this.run.setToolTipText("Start_the_simulation.");
    this.setLayout(new BorderLayout());
    this.add(this.al, BorderLayout.CENTER);
    this.add(this.ad, BorderLayout.EAST); //this.add(run, BorderLayout.
        SOUTH);

    this.fileAccess = new LoadSave(this);

    this.f.setLayout(new BorderLayout());
    this.f.addWindowListener(MyWindowListener.getInstance());
    this.f.add(this);
    this.f.pack();
    this.f.setVisible(true);
}

/**
 * This constructor creates the window and loads a turmite from a file.
 *
 * @param filename
 *         the file to load from.
 */
private Main(final String filename) {
    this();
    try {
        final TransitionFunction tf = this.fileAccess
            .loadSetFromFile(new File(filename));
        this.al.setTransitionFunction(tf);
    } catch (final TurmiteException e) {
        JOptionPane.showMessageDialog(this, String.format(
            "An_error_occurred_loading_the_definition_from_%s' : %s"
            ,
            filename, e.getMessage()), "Error",
            JOptionPane.ERROR_MESSAGE);
    }
}
}

```

Listing C.24: multidimensional/gui/widgets/DirectionPicker.java

```
package multidimensional.gui.widgets;
```

```

import javax.swing.JComboBox;

import multidimensional.basics.Dimensionality;
import multidimensional.basics.Rotation;

public class DirectionPicker extends LabelledComponent {

    private Rotation[] directions;
    private JComboBox direction;

    public DirectionPicker() {
        super("Change_of_direction");
        directions= Dimensionality.getInstance().getRots().toArray(new
            Rotation[0]);
        // directions = new Rotation[rots.size()];
        // for (int i = rots.size() - 1; i >= 0; --i)
        //     directions[i] = rots.get(i);
        direction = new JComboBox(directions);

        this.add(direction);
    }

    public Rotation getSelectedItem() {
        return (Rotation) direction.getSelectedItem();
    }

    public void setSelectedItem(Object anObject) {
        // this requires some jiggery-pokery.
        // when we load from a file, the rotation objects aren't the ones
        // from the factory.
        // as the list to
        for(Rotation dir:directions) {
            if(anObject.equals(dir)) {
                System.out.printf("Got %s and %s matching %n", anObject, dir);
                direction.setSelectedItem(dir);
                break;
            }
        }
    }

    public void setSelectedIndex(int i) {
        direction.setSelectedIndex(i);
    }
}

```

Listing C.25: multidimensional/gui/widgets/InputBox.java

```

package multidimensional.gui.widgets;

import javax.swing.JTextField;

public class InputBox extends LabelledComponent {
    private static final int DEFAULT.WIDTH = 10;
    private JTextField tf = new JTextField(DEFAULT.WIDTH);

    public InputBox(String label, int width, String defaultValue) {
        super(label);
        this.tf = width<0 ? new JTextField(defaultValue) : new JTextField(
            defaultValue, width);
    }
}

```

```

        this.add(this.tf);
    }

    public String getText() {
        return tf.getText();
    }

    public void setText(String s) {
        tf.setText(s);
    }

    public void setEnabled(boolean enabled) {
        tf.setEnabled(false);
    }
}

```

Listing C.26: multidimensional/gui/widgets/SpinEdit.java

```

package multidimensional.gui.widgets;

import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Frame;
import java.awt.Label;
import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;

/**
 * Provides a component similar to the spinedit control on windows.
 *
 * @author richard
 * @hidden
 */
public class SpinEdit extends Panel implements ActionListener,
    MouseWheelListener {
    private static final int FIELD_WIDTH = 6;

    // simple test
    public static void main(final String[] args) {
        final Frame f = new Frame("spinedit_test");
        f.add(new SpinEdit(0, 10, 5, true));
        f.setVisible(true);
        f.pack();
    }

    private final boolean bounded;

    private final int min, max;

    private int val;

    private final Label textbox = new Label("_____");

    // not very pretty... sort something else out at some point.
    private final Button b_up = new Button("/\\\"), b_down = new Button("\\\"
    );

    /**
     * constructs a spinedit with no limitations on the value.

```

```

*
* @param value
*/
public SpinEdit(final int value) {
    this(0, 0, value, false);
}

/**
*
* @param min
*         minimum value.
* @param max
*         maximum value.
* @param initial
*         the initial value.
* @param bounded
*         whether we should respect the min/max.
*/
public SpinEdit(final int min, final int max, final int initial,
                final boolean bounded) {
    this.min = min;
    this.max = max;
    this.val = initial;
    this.bounded = bounded;
    this.b.down.addActionListener(this);
    this.textbox.setEnabled(false);
    this.b.up.addActionListener(this);
    // for(Component c : new Component[]{b_down, b_up, textbox}) {
    this.textbox.addMouseWheelListener(this);
    this.b.down.addMouseWheelListener(this);
    this.b.up.addMouseWheelListener(this);
    this.addMouseWheelListener(this);
    // c.addActionListener(this);
    // }
    this.setLayout(new BorderLayout());
    this.add(this.b.up, BorderLayout.NORTH);
    this.add(this.b.down, BorderLayout.SOUTH);
    this.add(this.textbox, BorderLayout.CENTER);
    this.updateTextBox();
}

public void actionPerformed(final ActionEvent e) {
    if (e.getSource() == this.b.up) {
        this.inc();
    } else if (e.getSource() == this.b.down) {
        this.dec();
    } else {
        throw new RuntimeException(
            String
                .format(
                    "SpinEditControl %s registered for _
                    unknown_action_provider: %s",
                    this, e.getSource()));
    }
    this.updateTextBox();
}

private void dec() {
    if (this.bounded) {
        if ((this.val - 1) >= this.min) {
            --this.val;
        }
    }
}

```

```

        }
        } else {
            --this.val;
        }
    }

    /**
     * Gets the value stored in this control.
     *
     * @return the integer value.
     */
    public int getValue() {
        return this.val;
    }

    private void inc() {
        if (this.bounded) {
            if ((this.val + 1) <= this.max) {
                ++this.val;
            }
        } else {
            ++this.val;
        }
    }

    @Override
    public void mouseWheelMoved(final MouseEvent e) {
        int delta = e.getWheelRotation();
        System.err.printf("Got mouse wheel: %d%n", delta);
        if (delta < 0) {
            while (delta++ != 0) {
                this.inc();
            }
        } else if (delta > 0) {
            while (delta-- != 0) {
                this.dec();
            }
        }
        this.updateTextBox();
        // TODO Auto-generated method stub
    }

    // save repeating the same code... hopefully the JIT will inline it.
    private void updateTextBox() {
        this.textbox.setText(String.format("%" + SpinEdit.FIELD_WIDTH + "d",
            this.val));
    }
}

```

Listing C.27: multidimensional/gui/widgets/FileBox.java

```

package multidimensional.gui.widgets;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JButton;

```

```

import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

/** @hidden */
public class FileBox extends JPanel implements ActionListener {
    private final JButton pick = new JButton("!");

    private final JTextField filename = new JTextField(12);

    private final JLabel label = new JLabel("File");

    boolean save = false;

    public FileBox() {
        this.setLayout(new BorderLayout());
        this.add(this.label, BorderLayout.WEST);
        this.add(this.pick, BorderLayout.EAST);
        this.filename.setEnabled(false);
        this.pick.addActionListener(this);
        this.add(this.filename, BorderLayout.CENTER);
    }

    @Override
    public void actionPerformed(final ActionEvent ev) {
        final JFileChooser fc = new JFileChooser();
        File file = null;
        final int returnVal = this.save ? fc.showSaveDialog(this) : fc
            .showOpenDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            file = fc.getSelectedFile();
        }
        if (file == null) {
            return;
        }
        try {
            this.filename.setText(file.getCanonicalPath());
        } catch (final Exception e) {
            System.err.println(e.getMessage());
        }
    }

    public File getFile() {
        final String fn = this.filename.getText().trim();
        return (fn.length() < 1) ? null : new File(fn);
    }

    public void setLabel(final String s) {
        this.label.setText(s);
    }
}

```

Listing C.28: multidimensional/gui/widgets/UndoButton.java

```

package multidimensional.gui.widgets;

import java.awt.event.ActionEvent;

```

```

import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JOptionPane;

import multidimensional.ITurmite;
import multidimensional.Transition;
import multidimensional.TurmiteEvent;
import multidimensional.TurmiteListener;
import multidimensional.util.WrapAroundStack;

public class UndoButton extends JButton implements TurmiteListener,
    ActionListener {
    private final WrapAroundStack<Transition> s;
    private final ITurmite t;

    public UndoButton(final ITurmite t, final int levels) {
        super("Undo");
        this.addActionListener(this);

        this.t = t;
        this.s = new WrapAroundStack<Transition>(levels);
        t.setListener(this);
    }

    @Override
    public void actionPerformed(final ActionEvent e) {
        if (this.s.size() < 1) {
            JOptionPane.showMessageDialog(this,
                "No more undo levels available");
        } else {
            this.t.undoTransition(this.s.pop());
        }
    }

    @Override
    public void send(final TurmiteEvent e) {
        this.s.push(e.getTransition());
    }
}

```

Listing C.29: multidimensional/gui/widgets/LabelledComponent.java

```

package multidimensional.gui.widgets;
import java.awt.GridLayout;
import javax.swing.JLabel;
import javax.swing.JPanel;
/**
 * This consolidates some common code for the various input fields.
 * It presents a label and the layout for them. The subclass then adds the
 * relevant field.
 * @author richard
 *
 */
public class LabelledComponent extends JPanel {
    private JLabel label;
    public LabelledComponent(String label) {
        this.label = new JLabel(label);
        this.setLayout(new GridLayout(0, 2));
    }
}

```

```

        this.add(this.label);
    }
    public void setLabel(String s) {
        label.setText(s);
    }
}

```

Listing C.30: multidimensional/gui/widgets/MultipleChoice.java

```

package multidimensional.gui.widgets;

import javax.swing.JComboBox;

public class MultipleChoice extends LabelledComponent {

    private JComboBox tf;

    public MultipleChoice(String label, String[] values) {
        super(label);
        this.tf = new JComboBox(values);
        this.add(this.tf);
    }

    public String getText() {
        return (String) tf.getSelectedItem();
    }

    public int getIndex() {
        return tf.getSelectedIndex();
    }

    public void setEnabled(boolean enabled) {
        tf.setEnabled(false);
    }
}

```

Listing C.31: multidimensional/gui/widgets/DebugResizeListener.java

```

package multidimensional.gui.widgets;

import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;

import javax.swing.JFrame;

/**
 * Test class to see where the events land. not important.
 *
 * @author somewhere in the jdk docs.
 * @hidden
 */
final class DebugResizeListener implements ComponentListener {
    private static void displayMessage(final String s) {
        System.out.println(s);
    }

    public void componentHidden(final ComponentEvent e) {
        DebugResizeListener.displayMessage(e.getComponent().getClass()
            .getName()
            + " _Hidden");
    }
}

```

```

    }

    public void componentMoved(final ComponentEvent e) {
        DebugResizeListener.displayMessage(e.getComponent().getClass()
            .getName()
            + " _-----_Moved");
    }

    public void componentResized(final ComponentEvent e) {
        if (e.getSource() instanceof JFrame) {
            ((JFrame) e.getSource()).pack();
        }
        DebugResizeListener.displayMessage(e.getComponent().getClass()
            .getName()
            + " _-----_Resized_");
    }

    public void componentShown(final ComponentEvent e) {
        DebugResizeListener.displayMessage(e.getComponent().getClass()
            .getName()
            + " _-----_Shown");
    }

}

```

Listing C.32: multidimensional/gui/widgets/SpinEdit2.java

```

package multidimensional.gui.widgets;

import java.awt.Adjustable;
import java.awt.Frame;
import java.awt.Label;

import javax.swing.JPanel;
import javax.swing.JScrollBar;

/**
 * Provides a component similar to the spinedit control on windows.
 *
 * @hidden
 * @author richard
 *
 */
public class SpinEdit2 extends JPanel {
    private static final int FIELD_WIDTH = 6;

    // simple test
    public static void main(final String[] args) {
        final Frame f = new Frame("spinedit_test");
        f.add(new SpinEdit2(0, 10, 5, true));
        f.setVisible(true);
        f.pack();
    }

    private final boolean bounded;

    private final int min, max;

    private final int val;

```

```

private final Label textbox = new Label("        ");

// not very pretty... sort something else out at some point.
private final JScrollBar scroll;

/**
 * constructs a spinedit with no limitations on the value.
 *
 * @param value
 */
public SpinEdit2(final int value) {
    this(0, 0, value, false);
}

/**
 *
 * @param min
 *         minimum value.
 * @param max
 *         maximum value.
 * @param initial
 *         the initial value.
 * @param bounded
 *         whether we should respect the min/max.
 */
public SpinEdit2(final int min, final int max, final int initial,
                 final boolean bounded) {
    this.min = min;
    this.max = max;
    this.val = initial;
    this.bounded = bounded;
    this.scroll = new JScrollBar(Adjustable.VERTICAL, this.val, 0, min,
                                max);
    this.textbox.setEnabled(false);
    this.add(this.textbox);
    this.add(this.scroll);
}

// save repeating the same code... hopefully the JIT will inline it.
private void updateTextBox() {
    this.textbox.setText(String.format("%" + SpinEdit2.FIELD_WIDTH + "d"
    ,
        this.val));
}
}

```

Listing C.33: multidimensional/gui/Menu.java

```

package multidimensional.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.util.Collection;
import java.util.Vector;

import javax.swing.AbstractButton;
import javax.swing.ButtonGroup;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JMenu;

```

```

import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JRadioButtonMenuItem;

import multidimensional.Factory;
import multidimensional.view.GridView3d;

/**
 * This is an incredibly ugly class. First class functions would be nice.
 *
 * @author richard
 * @see Main
 */
public class Menu {
    /**
     * Called when menu events happen.
     *
     * @author richard
     */
    /**
     * Generates a transition function from a user-specified rule-string
     */
    public void fromRuleString();

    /**
     * Loads a transition function from a file.
     */
    public void load();

    /**
     * Exits the program.
     */
    public void quit();

    /**
     * Saves the current transition function to a file.
     */
    public void save();

    /**
     * Saves the current transition function as a dot diagram.
     */
    public void saveDot();

    /**
     * Starts the simulation.
     */
    public void start();
}

/**
 *
 * @return a menu containing the anti-aliasing options.
 */
private static JMenu aaSetting() {
    final JMenu submenu = new JMenu("Anti-aliasing");
    submenu.setMnemonic(KeyEvent.VK_I);

    final ButtonGroup buttonGroup = new ButtonGroup();

```

```

JRadioButtonMenuItem radiobutton = new JRadioButtonMenuItem("
    Disabled");
radiobutton.setMnemonic(KeyEvent.VK_0);
radiobutton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent e) {
        GridView3d.setAaSamples(-1);
    }
});
buttonGroup.add(radiobutton);
submenu.add(radiobutton);
radiobutton = new JRadioButtonMenuItem("2x");
radiobutton.setMnemonic(KeyEvent.VK_2);
radiobutton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent e) {
        GridView3d.setAaSamples(2);
    }
});
buttonGroup.add(radiobutton);
submenu.add(radiobutton);

radiobutton = new JRadioButtonMenuItem("4x");
radiobutton.setMnemonic(KeyEvent.VK_4);
radiobutton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent e) {
        GridView3d.setAaSamples(4);
    }
});
buttonGroup.add(radiobutton);
submenu.add(radiobutton);
radiobutton = new JRadioButtonMenuItem("8x");
radiobutton.setMnemonic(KeyEvent.VK_8);
radiobutton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent e) {
        GridView3d.setAaSamples(8);
    }
});
buttonGroup.add(radiobutton);
submenu.add(radiobutton);

radiobutton.doClick();
return submenu;
}

/**
 *
 * @return menu allowing selection of transition function implementation
 */
private static JMenu functionImp() {
    final JMenu submenu = new JMenu("Function_impl");
    submenu.setMnemonic(KeyEvent.VK_S);
    final ButtonGroup buttonGroup = new ButtonGroup();

    JRadioButtonMenuItem radiobutton = new JRadioButtonMenuItem(
        "Double_tree");
    radiobutton.setMnemonic(KeyEvent.VK_D);
    buttonGroup.add(radiobutton);
    submenu.add(radiobutton);

```

```

        radiobutton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                Factory.setTfImpl(Factory.ETFImpl.DoubleMap);
            }
        });
        radiobutton = new JRadioButtonMenuItem("Hash_table");
        radiobutton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                Factory.setTfImpl(Factory.ETFImpl.Hashing);
            }
        });
        radiobutton.setMnemonic(KeyEvent.VK.H);
        radiobutton.setSelected(true);
        buttonGroup.add(radiobutton);
        radiobutton.doClick();
        submenu.add(radiobutton);
        return submenu;
    }

    /**
     *
     * @return menu to select which grid implementation to use.
     */
    private static final JMenu gridImpl() {
        final JMenu submenu = new JMenu("Grid_impl");
        submenu.setMnemonic(KeyEvent.VK.G);

        final ButtonGroup buttonGroup = new ButtonGroup();
        JRadioButtonMenuItem radiobutton = new JRadioButtonMenuItem(
            "Hash_table");
        radiobutton.setMnemonic(KeyEvent.VK.H);
        radiobutton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                Factory.setGImpl(Factory.EGridImpl.Hash);
            }
        });
        buttonGroup.add(radiobutton);
        submenu.add(radiobutton);
        radiobutton.doClick();
        radiobutton = new JRadioButtonMenuItem("Tree_Map");
        radiobutton.setMnemonic(KeyEvent.VK.T);
        radiobutton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                Factory.setGImpl(Factory.EGridImpl.Tree);
            }
        });
        buttonGroup.add(radiobutton);
        submenu.add(radiobutton);
        return submenu;
    }

    /**
     *
     * @return a menu to select whether to use a standard Turing machine or
     *         a
     *         Turmite.
     */
    private static JMenu machineImpl() {

```

```

final JMenu submenu = new JMenu("Machine_Impl");
submenu.setMnemonic(KeyEvent.VK_M);

final ButtonGroup buttonGroup = new ButtonGroup();

JRadioButtonMenuItem radiobutton = new JRadioButtonMenuItem(
    "Classic_Turing_Machine");
radiobutton.setMnemonic(KeyEvent.VK_C);
buttonGroup.add(radiobutton);
submenu.add(radiobutton);
radiobutton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final(ActionEvent e) {
        Factory.setMachineImpl(Factory.EMachineImpl.Classic);
    }
});
radiobutton = new JRadioButtonMenuItem("Turmite");
radiobutton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final(ActionEvent e) {
        Factory.setMachineImpl(Factory.EMachineImpl.Turmite);
    }
});
radiobutton.setMnemonic(KeyEvent.VK_T);
radiobutton.setSelected(true);
buttonGroup.add(radiobutton);
radiobutton.doClick();
submenu.add(radiobutton);
return submenu;
}

private static JMenu viewImpl() {
    final JMenu submenu = new JMenu("View_impl");
    submenu.setMnemonic(KeyEvent.VK_V);

    final ButtonGroup buttonGroup = new ButtonGroup();
    JRadioButtonMenuItem radiobutton = new JRadioButtonMenuItem("2D");
    radiobutton.setMnemonic(KeyEvent.VK_2);
    radiobutton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(final(ActionEvent e) {
            Factory.setGvImpl(Factory.EGridViewImpl.TWOD);
        }
    });
    buttonGroup.add(radiobutton);
    submenu.add(radiobutton);

    radiobutton = new JRadioButtonMenuItem("3D");
    radiobutton.setMnemonic(KeyEvent.VK_3);
    radiobutton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(final(ActionEvent e) {
            Factory.setGvImpl(Factory.EGridViewImpl.THREED);
        }
    });
    buttonGroup.add(radiobutton);
    submenu.add(radiobutton);
    radiobutton.setSelected(true);
    radiobutton.doClick();
    return submenu;
}

```

```

private final Collection<MenuEventListener> listeners = new Vector<
    MenuEventListener>();

public void addListener(final MenuEventListener l) {
    this.listeners.add(l);
}

public JMenuBar createMenuBar() {
    JMenuBar menuBar;
    JMenu menu;
    final JMenu submenu;
    JMenuItem menuItem;
    final JRadioButtonMenuItem radiobutton;
    JCheckBoxMenuItem cbMenuItem;
    final ButtonGroup buttonGroup;
    // Create the menu bar.
    menuBar = new JMenuBar();

    // Build the first menu.
    menu = new JMenu("File");
    menu.setMnemonic(KeyEvent.VK_F);
    menu.getContext().setAccessibleDescription(
        "Deal_with_this_turmite");

    menuBar.add(menu);
    menuItem = new JMenuItem("Run", KeyEvent.VK_R);
    menuItem.getContext().setAccessibleDescription(
        "Runs_this_turmite");
    menuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(final ActionEvent e) {
            for (final MenuEventListener l : this.listeners) {
                l.start();
            }
        }
    });
    menu.add(menuItem);
    menu.addSeparator();

    menuItem = new JMenuItem("Rule_string", KeyEvent.VK_S);
    menuItem.getContext().setAccessibleDescription(
        "Generate_the_transition_function_from_a_rule_string.");
    menuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(final ActionEvent e) {
            for (final MenuEventListener l : this.listeners) {
                l.fromRuleString();
            }
        }
    });

    menu.add(menuItem);
    // a group of JMenuItem
    menuItem = new JMenuItem("Open", KeyEvent.VK_O);
    menuItem.getContext().setAccessibleDescription(
        "Loads_a_definition");
    menuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(final ActionEvent e) {
            for (final MenuEventListener l : this.listeners) {
                l.load();
            }
        }
    });
}

```

```

    }
  });

  menu.add(menuItem);
  menu.addSeparator();

  menuItem = new JMenuItem("Save", KeyEvent.VK_S);
  menuItem.getAccessibleContext().setAccessibleDescription(
    "Save_the_definition");
  menuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent e) {
      for (final MenuEventListener l : Menu.this.listeners) {
        l.save();
      }
    }
  });
  menu.add(menuItem);
  menuItem = new JMenuItem("Save_diagram", KeyEvent.VK_D);
  menuItem.getAccessibleContext().setAccessibleDescription(
    "Save_a_transition_diagram");
  menuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent e) {
      for (final MenuEventListener l : Menu.this.listeners) {
        l.saveDot();
      }
    }
  });
  menu.add(menuItem);
  menu.addSeparator();

  menuItem = new JMenuItem("Quit", KeyEvent.VK_Q);
  menuItem.getAccessibleContext().setAccessibleDescription("Exits");
  menuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent e) {
      for (final MenuEventListener l : Menu.this.listeners) {
        l.quit();
      }
    }
  });
  menu.add(menuItem);

  // Build second menu in the menu bar.
  menu = new JMenu("Settings");
  menu.setMnemonic(KeyEvent.VK_N);
  menu.getAccessibleContext().setAccessibleDescription(
    "Configuration_Options");
  menuBar.add(menu);
  menu.add(Menu.machineImpl());
  menu.add(Menu.functionImpl());

  menu.addSeparator();
  menu.add(Menu.viewImpl());
  menu.add(Menu.gridImpl());
  menu.add(Menu.aSetting());
  // a group of check box menu items
  menu.addSeparator();
  cbMenuItem = new JCheckBoxMenuItem("Use_Lighting_in_3D");
  cbMenuItem.addActionListener(new ActionListener() {

```

```

        public void actionPerformed(final ActionEvent event) {
            final AbstractButton aButton = (AbstractButton) event
                .getSource();
            final boolean selected = aButton.getModel().isSelected();
            GridView3d.setLights(selected);
        }
    });
    cbMenuItem.setSelected(true);
    cbMenuItem.setMnemonic(KeyEvent.VK_L);
    menu.add(cbMenuItem);

    return menuBar;
}
}

```

Listing C.34: multidimensional/gui/LoadSave.java

```

package multidimensional.gui;

import java.awt.Component;
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.io.ObjectStreamException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

import multidimensional.TransitionFunction;
import multidimensional.TurmiteException;
import multidimensional.util.MakeDiagram;

public class LoadSave {

    private final Component parent;

    public LoadSave(final Component parent) {
        this.parent = parent;
    }

    private File getFile(final boolean save) throws TurmiteException {
        try {
            final JFileChooser fc = new JFileChooser();
            File file = null;
            final int returnVal = save ? fc.showSaveDialog(this.parent) : fc
                .showOpenDialog(this.parent);

            if (returnVal == JFileChooser.APPROVE_OPTION) {
                file = fc.getSelectedFile();
            }
            return file;
        } catch (final Exception e) {

```

```

        throw new TurmiteException(e.getMessage());
    }
}

/**
 * Loads a transition function from a file.
 *
 * @param f
 *         the file to load from.
 * @return the transition function loaded.
 * @throws TurmiteException
 *         if some error occurs while loading it.
 */
public TransitionFunction loadSetFromFile(final File f)
    throws TurmiteException {
    ObjectInput oi = null;
    TransitionFunction input = null;
    try {

        oi = new ObjectInputStream(new FileInputStream(f));
        input = (TransitionFunction) oi.readObject();

    } catch (final EOFException e1) {
        throw new TurmiteException("Unexpected_end_of_file");
    } catch (final ClassNotFoundException e) {
        throw new TurmiteException("Could_not_find_the_correct_class.");
    } catch (final InstantiationException e) {
        throw new TurmiteException("Could_not_instantiate_the_class."
            + "Possibly_you_are_trying_to_load_an_old_file?");
    } catch (final ObjectStreamException e) {
        throw new TurmiteException(
            "error_reading_stream._Maybe_incorrect_file_type?");
        // } catch (final FileNotFoundException e) {
        // throw new TurmiteException(e.getMessage());
    } catch (final IOException e) {
        throw new TurmiteException("Could_not_open_file %s", e.
            getMessage());
    } finally {
        if (oi != null) {
            try {
                oi.close();
            } catch (final IOException e) {
                System.err.printf("Could_not_close_input_stream: %s%n",
                    e.
                        getMessage());
                e.printStackTrace();
            }
        }
    }
    return input;
}

public TransitionFunction loadTurmite() throws TurmiteException {
    final File f = this.getFile(false);

    if (f == null) {
        return null;
    }
    return this.loadSetFromFile(f);
}

```

```

public void saveDot(final TransitionFunction t) throws TurmiteException
{
    final File f = this.getFile(true);
    if (f == null) {
        return;
    }
    PrintWriter p;
    try {
        p = new PrintWriter(new OutputStreamWriter(new FileOutputStream(
            f,
            false)));
    } catch (final Exception e) {
        throw new TurmiteException("Could not open file %s", e.
            getMessage());
        // } catch (final FileNotFoundException e) {
        // throw new TurmiteException("Could not open file %s",
        // e.getMessage());
    }
    p.write(MakeDiagram.toDot(t));
    p.flush();
    p.close();
}

public void saveTurmite(final TransitionFunction t) throws
TurmiteException {
    final String name = (String) JOptionPane.showInputDialog(this.parent
        ,
        "What shall we call this turmite?", "name.it",
        JOptionPane.QUESTION_MESSAGE, null, null, t.getName());
    if (name == null) {
        return;
    }
    t.setName(name);
    final File f = this.getFile(true);
    if (f == null) {
        return;
    }
    this.writeSetToFile(t, f);
}

private void writeSetToFile(final TransitionFunction turmite, final File
f)
throws TurmiteException {
    OutputStream os = null;
    ObjectOutput oo = null;
    try {
        os = new FileOutputStream(f);
        oo = new ObjectOutputStream(os);
        oo.writeObject(turmite);
    } catch (final Exception e) { // don't really care what type
        throw new TurmiteException("Error saving turmite: %s", e
            .getMessage());
    } finally {
        // close them in reverse order...
        // Destructors would be nice :/
        try {
            if (oo != null) {
                oo.close();
            }
        } catch (final IOException e) {
            // nothing can really be done... if there's an error it's
            // already been written anyway.

```



```

        // TODO Auto-generated method stub
    }
}

```

Listing C.36: multidimensional/gui/RunPanel.java

```

package multidimensional.gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.Vector;

import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;

import multidimensional.Factory;
import multidimensional.ITurmite;
import multidimensional.TransitionFunction;
import multidimensional.TurmiteEvent;
import multidimensional.TurmiteListener;
import multidimensional.basics.Dimensionality;
import multidimensional.basics.Grid;
import multidimensional.view.GridView;

public class RunPanel extends JPanel implements Runnable, ActionListener {
    /**
     * This class is in charge of keeping track of the data for all the
     * Turmites
     * and providing the text for the info area.
     *
     * @hidden
     * @author richard
     */
    private static class InfoArea implements TurmiteListener {
        // one would expect HashSet<TurmiteListener.Event> to work just as
        // well
        // It doesn't. :/
        private final Map<ITurmite, TurmiteEvent> status = new HashMap<
            ITurmite, TurmiteEvent>();

        /**
         * Gets the status string associated with the turmite provided.
         *
         * @param t
         *         the machine to get the info for.
         * @return String representing the current state of the machine.
         * @throws IllegalArgumentException
         */
    }
}

```

```

        *           if the ITurmite is not there.
        */
    public String get(final ITurmite t) {
        final TurmiteEvent e = this.status.get(t);
        if (e == null) {
            throw new IllegalArgumentException(String.format(
                "The_turmite_%s_is_not_in_the_set:/.", t.toString()
            ));
        }
        return e.toString();
    }

    @Override
    public void send(final TurmiteEvent t) {
        this.status.put(t.getSource(), t);
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder();
        for (final TurmiteEvent e : this.status.values()) {
            sb.append(e);
        }
        // System.err.printf("Done info box: %s%n", sb.toString());
        return sb.toString();
    }
}

private final InfoArea updater = new InfoArea();

private final JTextArea infoArea;

private final Dimension size;
/**
 * Whether the thread is running.
 */
private volatile boolean running = false;
/**
 * construction parameters.
 */
private final int interval;
/**
 * How many iterations have been run.
 */
private int current_iteration;
/**
 * gui components.
 */
private final JLabel labiter = new JLabel("Step:");
private final JButton stopButton = new JButton("Start");
private final JButton stepButton = new JButton("Step");
/**
 * The grid visualisation.
 */
private final GridView field;
/**
 * The grid the turmites run on.
 */
private final Grid grid;

/**
 * The turmites we're using on the board.

```

```

*/
private final Collection<ITurmite> turmites = new Vector<ITurmite>();

/**
 *
 * @param size
 *         the size of the viewing window
 * @param interval
 *         the time in milliseconds between updates (you need
 *         something
 */
public RunPanel(final Dimension size, final int interval) {
    this.size = (Dimension) size.clone();

    this.interval = interval;
    this.grid = Factory.grid();

    this.field = Factory.gridView(this.size);
    this.grid.addGridListener(this.field);
    this.setLayout(new BorderLayout());
    final Panel buttons = new Panel();
    buttons.setLayout(new FlowLayout());
    this.add((JComponent) this.field, BorderLayout.CENTER);
    buttons.add(this.stopButton);
    buttons.add(this.stepButton);
    buttons.add(this.labiter);
    this.add(buttons, BorderLayout.SOUTH);
    this.stopButton.addActionListener(this);
    this.stepButton.addActionListener(this);
    this.current_iteration = 0;
    this.infoArea = new JTextArea(3, 30);
    // infoArea.setEnabled(false);
    this.infoArea.setEditable(false);
    this.add(this.infoArea, BorderLayout.NORTH);
}

@Override
public void actionPerformed(final ActionEvent e) {
    if (e.getSource() == this.stopButton) {
        this.pauseUnpause();
    } else if (e.getSource() == this.stepButton) {
        this.step();
    }
}

/**
 * Adds a turmite constructed from the ActionMap to the field.
 *
 * @param tf
 *         the transition map of the turmite.
 * @return the turmite instance created.
 */
public ITurmite addTurmite(final TransitionFunction tf) {
    final ITurmite t = Factory.turmite(this.grid, tf, Dimensionality
        .getInstance().origin());
    this.turmites.add(t);
    t.setListener(this.updater);
    System.out.printf("Added turmite: %s%n", t);
    return t;
}

```

```

public void pause() {
    if (this.running) {
        this.stopButton.setText("Start");
        this.running = false;
        this.stepButton.setEnabled(true);
    }
}

/**
 * Pause or unpause the simulation.
 */
public void pauseUnpause() {

    if (this.running) {
        this.pause();
    } else {
        this.unpause();
    }

}

public void run() {
    System.out.printf("Thread_started...%n");

    while (this.running) {
        // if the window has been disposed of, then get rid of this
        // thread.
        // as it's a panel, we can't hook the dispose method of the
        // frame.
        if (!this.isDisplayable()) {
            this.pauseUnpause();
        }
        // wait the allotted time.
        try {
            Thread.sleep(this.interval);
        } catch (final InterruptedException e) {
            // don't really know how to test this... so I don't know if
            // the
            // thread is terminated or what. Stop it to make sure.
            this.pause();
            JOptionPane.showMessageDialog(this, String.format(
                "Simulation_was_interrupted: %s", e.getMessage()),
                "Error", JOptionPane.ERROR_MESSAGE, null);
        }

        // proceed with the simulation.
        if (this.running) {
            this.step();
        }
    }
}

/**
 * Advances the simulation one timestep.
 */
private void step() {
    boolean halted = false;
    // update the turmites
    for (final ITurmite t : this.turmites) {
        if (t.act()) {

```

```

        if (this.turmites.size() == 1) { // if it's the only one
            then
                // die nicely.
                this.pause();
                halted = true;
        }
        System.out.printf("Turmite_%s_is_halted.%n", t);
    }
}
this.labiter
    .setText(String.format("Step:_%6d", this.current_iteration))
    ;
if (!halted) {
    ++this.current_iteration;
}

this.updateInfoArea();
this.repaint();
Thread.yield();
}

public void unpause() {
    if (!this.running) {
        this.stopButton.setText("Stop");
        this.running = true;
        this.stepButton.setEnabled(false);
        new Thread(this).start();
    }
}

private void updateInfoArea() {
    this.infoArea.setText(this.updater.toString());
    // final ITurmite t = this.turmites.toArray(new ITurmite[] {})[0];
    // this.infoArea.setText(String.format("Transition: %s%nHeading: %s%n",
    // t
    // .getNextTransition(), t.getNextHeading().toRowString()));
}
}
}

```

Listing C.37: multidimensional/gui/ActionList.java

```

package multidimensional.gui;

import java.awt.BorderLayout;

import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionListener;

import multidimensional.Factory;
import multidimensional.Transition;
import multidimensional.TransitionFunction;
import multidimensional.TurmiteException;
import multidimensional.gui.widgets.InputBox;

public class ActionList extends JPanel {

```

```

private final InputBox tfName = new InputBox("Identifier:", 20, "
    Untitled");

private TransitionFunction tf;

private final TransitionListModel model;

private final JList listcontrol;

public ActionList () {
    this.tf = Factory.emptyTF(); // new TransitionFunction();
    this.tfName.setToolTipText("The_name_of_the_turmite.");
    this.model = new TransitionListModel(this.tf);
    this.listcontrol = new JList(this.model);
    this.listcontrol
        .setToolTipText("The_rules_defined_for_the_transition_
            function.\n"
                + "Click_one_to_load_to_the_opposite_pane.");
    this.setLayout(new BorderLayout());
    this.add(this.tfName, BorderLayout.NORTH);
    this.add(this.listcontrol);
    this.listcontrol.setSelectionMode(ListSelectionMode.
        SINGLE_SELECTION);
}

public void add(final Transition a) {
    this.model.add(a);
}

public void addListSelectionListener(final ListSelectionListener l) {
    this.listcontrol.addListSelectionListener(l);
}

public TransitionFunction getTransitionFunction() throws
    TurmiteException {
    if (this.tf.toArray().length < 1) {
        throw new TurmiteException("No_rules_specified");
    }
    this.tf.setName(this.tfName.getText());
    return this.tf;
}

public void setTransitionFunction(final TransitionFunction ret) {
    this.tf = ret;
    this.model.setTransitionFunction(ret);
    this.tfName.setText(ret.getName());
}
}

```

Listing C.38: multidimensional/gui/ConfigPanel.java

```

package multidimensional.gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

```

```

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import multidimensional.TransitionFunction;
import multidimensional.gui.widgets.InputBox;

public class ConfigPanel extends JPanel implements ActionListener {
    // private final InputBox iterations = new InputBox(
    // "Iterations (0 for infinite):", -1, "0");
    private final InputBox fwidth = new InputBox("View_width:", -1, "500");
    private final InputBox fheight = new InputBox("View_height:", -1, "500")
        ;
    private final InputBox frate = new InputBox("Tick_interval_(ms):", -1, "
        50");
    private final JButton startButton = new JButton("Start");

    private final TransitionFunction actions;

    public ConfigPanel(final TransitionFunction actions) {
        this.actions = actions;
        final JPanel right = new JPanel();
        System.out.printf("ConfigPanel_got_actions:%n%s%n", actions.toString
            ());
        right.setLayout(new GridLayout(0, 1));
        // right.add(this.iterations);
        right.add(this.frate);
        right.add(this.fwidth);
        right.add(this.fheight);
        // right.add(input_data);
        // input_data.setLabel("Input data:");
        // rules.setColumns(20);
        // rules.setText(actions.toString());
        // rules.setEnabled(false);
        this.startButton.addActionListener(this);
        this.setLayout(new BorderLayout());
        // add(new JScrollPane(rules),BorderLayout.WEST);
        this.add(right, BorderLayout.CENTER);
        this.add(this.startButton, BorderLayout.SOUTH);
    }

    @Override
    public void actionPerformed(final ActionEvent e) {
        if (e.getSource() == this.startButton) {
            final RunPanel p = this.fromForm();
            if (p == null) {
                return;
            }
            System.out.printf("Got_panel_%s%n", p);
            System.out.printf("Actions:_%s%n", this.actions);
            p.addTurmite(this.actions);
            // for some reason here, JFrame doesn't seem to pass on the
            // resize
            // events....
            final JFrame f = new JFrame("Turmite_Simulation");

            // f.addComponentListener(new DebugResizeListener());
            f.setLayout(new BorderLayout());
            f.add(p);
            f.addWindowListener(MyWindowListener.getInstance());
        }
    }
}

```

```

        f.pack();
        f.setVisible(true);
    }
}

/**
 * This constructs a RunPanel from the information in the form. If there
 * is
 *
 * @return
 */
private RunPanel fromForm() {
    RunPanel retval = null;
    String errMsg = "BADBADBAD";
    int width, height, rate;
    try {
        errMsg = "width_of_field";
        width = Integer.parseInt(this.fwidth.getText());
        errMsg = "height_of_field";
        height = Integer.parseInt(this.fheight.getText());
        errMsg = "time_between_iterations";
        rate = Integer.parseInt(this.frate.getText());

        retval = new RunPanel(new Dimension(width, height), rate);
    } catch (final NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Bad_numeric_input_for_the_"
            + errMsg, "Error", JOptionPane.ERROR_MESSAGE, null);
    } catch (final Exception e) {
        JOptionPane.showMessageDialog(this, e.getMessage(), "Error",
            JOptionPane.ERROR_MESSAGE, null);
        e.printStackTrace(System.err);
    }
    return retval;
}
}
}

```

Listing C.39: multidimensional/gui/TransitionListModel.java

```

package multidimensional.gui;

import java.util.Arrays;
import java.util.Collection;
import java.util.Vector;

import javax.swing.ListModel;
import javax.swing.event.ListDataEvent;
import javax.swing.event.ListDataListener;

import multidimensional.Transition;
import multidimensional.TransitionFunction;

/**
 * This implements the listmodel for the ActionListener control.
 *
 * Because we are displaying information from a set, the default model is
 * inadequate. This uses TransitionFunction as it's backend, though this is
 * not
 * technically necessary. The issue is that even NavigableSet does not allow

```

```

* access by index, so we need to keep a cache locally and sort it.
*
* @author richard
*
*/
public class TransitionListModel implements ListModel {
    private TransitionFunction data;

    /**
     * a local cache of the transitions. one can only assume toArray is
     * horribly
     * slow.
     */
    private Object[] cache;
    /**
     * The list data event that gets sent to the listeners. Only one event
     * ever
     * gets sent - so keep hold of it.
     */
    private final ListDataEvent ldev = new ListDataEvent(this,
        ListDataEvent.CONTENT_CHANGED, 0, 0);
    /**
     * The listeners for the ListDataEvents.
     */
    private final Collection<ListDataListener> listeners = new Vector<
        ListDataListener>();

    /**
     * Constructs the control with the specified function.
     *
     * @param tf
     *         The transition function to represent in the view.
     */
    public TransitionListModel(final TransitionFunction tf) {
        this.setTransitionFunction(tf);
    }

    /**
     * Adds a transition to the function, replacing another if it has the
     * same
     * initial conditions.
     *
     * @param ob
     *         the transition to add.
     */
    public void add(final Transition ob) {
        this.data.add(ob);
        this.recache();
        this.notifyListeners();
    }

    @Override
    public void addListDataListener(final ListDataListener l) {
        this.listeners.add(l);
        l.contentsChanged(this.ldev);
    }

    @Override
    public Object getElementAt(final int index) {
        final Object ob = this.cache.length == 0 ? "....."
            : this.cache[index];

```

```

        // System.out.printf("Returning %s as %d%n", ob, index);
        return ob;
    }

    @Override
    public int getSize() {
        // The list control doesn't bother with
        final int l = this.cache.length == 0 ? 10 : this.cache.length;
        return l;
    }

    /**
     * Notifies the ListDataListeners of a change.
     */
    private void notifyListeners() {
        for (final ListDataListener l : this.listeners) {
            l.contentsChanged(new ListDataEvent(this,
                ListDataEvent.CONTENT_CHANGED, 0, 0));
            // System.out.printf("Notified listener %s of change%n", l);
        }
    }

    /**
     * Gets the cache data from the TransitionFunction reference. As this
     * was
     * being done in a couple of places it made sense to separate it into a
     * routine. It also sorts the resulting array, this is desirable from a
     * usability standpoint. Also the overhead will be negligible compared
     * to
     * the actual simulation.
     */
    private void recache() {
        this.cache = this.data.toArray();
        Arrays.sort(this.cache);
    }

    @Override
    public void removeListDataListener(final ListDataListener l) {
        this.listeners.remove(l);
    }

    /**
     * Sets the transition function displayed in the list (e.g. when you
     * load
     * one).
     *
     * @param tf
     *         the one to use.
     */
    public void setTransitionFunction(final TransitionFunction tf) {
        this.data = tf;
        this.recache();
        this.notifyListeners();
    }
}

```

Listing C.40: multidimensional/basics/Matrix.java

```
package multidimensional.basics;
```

```

/**
 * This is a matrix which can be of arbitrary size. It is intended to be
 * immutable, however there is clearly a need to sometimes load them with
 * certain values. Hence all the actual operations besides set() return a
 * new
 * Matrix. I may have done things differently if Java had operator
 * overloading.
 */
public class Matrix implements Comparable<Matrix>, Cloneable {
    private final int w, h;
    protected int [] values;

    /**
     * Constructs a matrix with the specified width and height. If it is a
     * square matrix, it is initialised to the identity matrix. Otherwise
     * all
     * cells are 0.
     *
     * @param w
     *         the width of the matrix, in elements.
     * @param h
     *         the height of the matrix in elements.
     */
    protected Matrix(final int w, final int h) {
        this.w = w;
        this.h = h;
        this.values = new int[w * h];
        if (w == h) {
            for (int i = 0; i < w; ++i) {
                this.set(i, i, 1);
            }
        }
    }

    @Override
    public Matrix clone() {
        final Matrix m = new Matrix(this.w, this.h);
        m.values = this.values.clone();
        return m;
    }

    public int compareTo(final Matrix m) {
        if (this.w != m.w) {
            return this.w - m.w;
        }
        if (this.h != m.h) {
            return this.h - m.h;
        }
        for (int i = 0; i < this.values.length; ++i) {
            final int x = this.values[i] - m.values[i];
            if (x != 0) {
                return x;
            }
        }
        return 0;
    }

    protected int dimensions() {
        return this.h;
    }
}

```

```

@Override
public boolean equals(final Object o) {
    if (o == null) {
        return false;
    }
    if (o instanceof Matrix) {
        return this.compareTo((Matrix) o) == 0;
    }
    return false;
}

/**
 * Gets the value at the provided row/column.
 *
 * @param i
 *         the column.
 * @param j
 *         the row.
 * @return the value.
 */
public int get(final int i, final int j) {
    return this.values[this.getIndex(i, j)];
}

/**
 * Calculates the index into the data array based on the row/column.
 *
 * @param i
 *         row.
 * @param j
 *         column.
 * @return index into values array.
 */
private int getIndex(final int i, final int j) {
    final int index = (i * this.w) + j;
    if (index >= this.values.length) {
        throw new IllegalArgumentException("Matrix_not_big_enough.");
    }
    return index;
}

/**
 * Performs a scalar multiplication of the matrix.
 *
 * @param n
 *         the number to multiply by.
 * @return a new matrix containing result.
 */
public Matrix multiply(final int n) {
    final Matrix ret = new Matrix(this.w, this.h);
    for (int i = 0; i < this.values.length; ++i) {
        ret.values[i] = this.values[i] * n;
    }
    return ret;
}

// /**
// *
// * @return the height of the matrix in cells.
// */
// public int height() {

```

```

// return this.h;
// }

/**
 * Returns a new matrix which is the product of this one and the one
 * provided.
 *
 * @param mat
 *         the multiplicand.
 * @return New matrix containing the product.
 */
public Matrix multiply(final Matrix mat) {
    if (this.h != mat.w) {
        throw new RuntimeException(
            "number_of_columns_must_match_number_of_rows");
    }
    final int m = this.w;
    final int n = mat.w;
    final int p = mat.h;
    final Matrix ret = new Matrix(m, p);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < p; ++j) {
            int cell = 0;
            for (int r = 0; r < n; ++r) {
                cell += this.get(i, r) * mat.get(r, j);
            }
            ret.set(i, j, cell);
        }
    }
    return ret;
}

protected void set(final int i, final int j, final int value) {
    this.values[this.getIndex(i, j)] = value;
}

public String toLatex() {
    final StringBuilder sb = new StringBuilder();
    sb.append("\\left(-\\begin{array}{")
    for (int i = 0; i < this.w; ++i) {
        sb.append('c');
    }
    sb.append("}\\n");
    for (int y = 0; y < this.h; ++y) {
        for (int x = 0; x < this.w; ++x) {
            if (x > 0) {
                sb.append("&");
            }
            sb.append(String.format("%3d", this.get(x, y)));
        }
        if (y < this.h - 1) {
            sb.append(String.format("\\\\ \\ \\ \\ \\ %n"));
        }
    }
    sb.append("\\end{array}-\\right)\\n");
    return sb.toString();
}

// @Deprecated
// public Matrix resize(int w, int h) {
// Matrix m = new Matrix(w, h);

```

```

// int x1 = Math.min(this.w, w);
// int y1 = Math.min(this.h, h);
// for (int x = 0; x < x1; ++x)
// for (int y = 0; y < y1; ++y)
// m.set(x, y, this.get(x, y));
// return m;
// }
// /**
// * Returns a new matrix containing the sum of this and the one
//   provided.
// * Such that this.get(i,j) + m.get(i,j) == returnedMatrix.get(i,j)
// *
// * @param m
// * the one to add to this.
// * @return sum of the two matrices.
// */
// public Matrix add(final Matrix m) {
// if (m.w != this.w || m.h != this.h) {
// throw new IllegalArgumentException(
// "Matricies must be the same size");
// }
// final Matrix ret = new Matrix(this.w, this.h);
// for (int i = 0; i < this.values.length; ++i) {
// ret.values[i] = this.values[i] + m.values[i];
// }
// return ret;
// }

public String toRowString() {
    final StringBuilder sb = new StringBuilder();
    for (int x = 0; x < this.w; ++x) {
        for (int y = 0; y < this.h; ++y) {
            sb.append(' ');
            sb.append(String.format("%3d", this.get(x, y)));
            sb.append(' ');
        }
        sb.append(String.format("%n"));
    }
    return sb.toString();
}

@Override
public String toString() {
    final StringBuilder sb = new StringBuilder();
    for (int y = 0; y < this.h; ++y) {
        for (int x = 0; x < this.w; ++x) {
            sb.append(' ');
            sb.append(String.format("%3d", this.get(x, y)));
            sb.append(' ');
        }
        sb.append(String.format("%n"));
    }
    return sb.toString();
}

//
// // this isn't used anywhere - candidate for removal?
// public Matrix transpose() {
// final Matrix ret = new Matrix(this.h, this.w);
// for (int i = 0; i < this.h; ++i) {
// for (int j = 0; j < this.w; ++j) {
// ret.set(j, i, this.get(i, j));

```

```

// }
// }
// return ret;
// }

// /**
// *
// * @return the width of the matrix in elements.
// */
// public int width() {
// return this.w;
// }
}

```

Listing C.41: multidimensional/basics/Dimensionality.java

```

package multidimensional.basics;

import java.util.List;

public class Dimensionality {
    private static Dimensionality instance;

    // as we are only rotating in multiples of pi/2 then I can take a
    // shortcut
    // here.
    /**
     * Calculates the cosine of the angle. The provided angle is a shorthand
     * type where: 0 = 0 degrees. 1 = 90 degrees. 2 = 180 degrees. 3 = 270
     * degrees.
     *
     * Because it is only multiples of pi/2 we can use a switch statement.
     */
    private static int cos(final int i) {
        switch (i % 4) {
            case 0:
                return 1;
            case 2:
                return -1;

            case 1:
            case 3:
                return 0;
            default:
                throw new IllegalArgumentException(
                    "Somehow the modulus failed to work...");
        }
    }

    public static Dimensionality getInstance() {
        if (Dimensionality.instance == null) {
            throw new RuntimeException(
                "The singleton has not yet been created."
                + " Call Dimensionality.setDimensions() first.");
        }
        return Dimensionality.instance;
    }
}

```

```

/**
 * Create the Given's matrix with the specified parameter.
 *
 * @param size
 * @param i
 * @param k
 * @param cos
 *         the cosine of the angle.
 * @param sin
 *         the sine of the angle
 * @return matrix, like I said.
 */
private static Matrix givens(final int size, final int i, final int k,
                             final int cos, final int sin) {
    final Matrix m = new Matrix(size, size);
    m.set(i, i, cos);
    m.set(k, k, cos);
    m.set(i, k, sin);
    m.set(k, i, -sin);
    // System.err.printf("%d -> i=%d k=%d cos=%d\n", size, i, k, cos);
    // System.err.println(m);
    return m;
}

/**
 * Creates all the rotation objects that are applicable to this
 * invocation.
 *
 * @see multidimensional.gui.widgets.DirectionPicker
 * @see Rotation
 */
private static void makeRots() {
    final Dimensionality d = Dimensionality.getInstance();
    final int dimensions = d.getDimensions();
    d.rotation = new Rotation[dimensions][2];
    d.rotationList = new java.util.Vector<Rotation>();
    // make forwards and backwards.
    {
        final Matrix temp = new Matrix(dimensions, dimensions);
        Rotation r = new Rotation(temp, "Straight");
        d.rotation[0][0] = r;
        d.rotationList.add(r);
        r = new Rotation(temp.multiply(-1), "About-turn");
        d.rotation[0][1] = r;
        d.rotationList.add(r);
    }
    // then the actual rotations.
    for (int i = 0; i < d.getDimensions() - 1; ++i) {
        for (int dir = 0; dir < 2; ++dir) {
            d.rotation[i + 1][dir] = new Rotation(i, dir);
            d.rotationList.add(d.rotation[i + 1][dir]);
        }
    }
}

/**
 * This initialises the singleton class. Once the number of dimensions
 * has
 * been defined you can start using it properly. This should be one of
 * the
 * first things to be done in the software.
 *
 */

```

```

    * @param d
    *           the number of dimensions to support.
    * @return the dimensionality instance.
    */
    public static Dimensionality setDimensions(final int d) {
        if (Dimensionality.instance != null) {
            if (d < Dimensionality.instance.getDimensions()) {
                System.err.printf(
                    "Setting %Dimensions.dimensions to a value lower"
                    + " than it was previously: was %d is %d\n",
                    d,
                    Dimensionality.instance.getDimensions());
            }
        }
        // but do it anyway. normally this won't happen without cause...
        Dimensionality.instance = new Dimensionality(d);
        return Dimensionality.instance;
    }

    /**
     * Calculates the sine of the angle. The provided angle is a shorthand
     * type
     * where: 0 = 0 degrees. 1 = 90 degrees. 2 = 180 degrees. 3 = 270
     * degrees.
     *
     * This is implemented in terms of cos() above.
     */
    private static int sin(final int i) {
        return Dimensionality.cos(i + 1);
    }

    /**
     * The initial heading. This is always going to be zero in all but the
     * final
     * element which will be 1.
     */
    private final Vector defaultHeading;
    /**
     * The number of dimensions supported in this invocation of the program.
     */
    private final int dimensions;

    /**
     * The identity matrix.
     */
    private final Matrix identity;
    /**
     * All the matrices that are needed for the simulation.
     */
    private final Matrix matrices [][];
    /**
     * All the rotation objects required. The array is organised so the
     * first
     * index is the dimension and the second is the rotation about it (
     * either 0
     * or 1 for left/right).
     */
    private Rotation rotation [][];
    /**
     * Because the user interface uses a list view to display the various
     * rotations, provide that list here.

```

```

    */
    private List<Rotation> rotationList;

    private Dimensionality(final int dimensions) {
        this.dimensions = dimensions;
        this.matrices = new Matrix[dimensions - 1][2];
        int dim = 0;
        for (int j = 0; j < dimensions - 1; ++j) {
            for (int k = j + 1; k < dimensions; ++k) {
                // System.err.printf("j=%d\ tk=%d%n", j, k);
                this.matrices[dim][0] = Dimensionality.givens(dimensions, j
                    ,
                    k, Dimensionality.cos(1), Dimensionality.sin(1));
                this.matrices[dim][1] = Dimensionality.givens(dimensions, j
                    ,
                    k, Dimensionality.cos(3), Dimensionality.sin(3));
            }
            ++dim;
        }
        this.defaultHeading = this.makeDefaultHeading();
        this.identity = new Matrix(dimensions, dimensions);

        // rotations=Dimensionality.makeRots();
    }

    public Vector getDefaultHeading() {
        return this.defaultHeading.clone();
    }

    public int getDimensions() {
        return this.dimensions;
    }

    public Matrix getIdentity() {
        return this.identity.clone();
    }

    protected Matrix getMatrix(final int dimension, final int step) {
        Matrix ret = null;
        switch (step) {
            case 0:
                ret = this.matrices[dimension][0];
                break;
            case 1:
                ret = this.matrices[dimension][1];
                break;
            default:
                break;
        }
        if (ret == null) {
            throw new IllegalArgumentException("Invalid matrix specification
                ");
        }
        return ret;
    }

    /**
     * This returns the raw array, so be careful!
     *
     * @see #rotation
     * @return the array of rotations.
     */
}

```

```

public Rotation [][] getRotations() {
    if (this.rotation == null) {
        Dimensionality.makeRots();
    }
    return this.rotation;
}

public List<Rotation> getRots() {
    if (this.rotationList == null) {
        Dimensionality.makeRots();
    }
    return this.rotationList;
}

private Vector makeDefaultHeading() {
    final int [] out = new int[this.dimensions];
    for (int i = 0; i < this.dimensions - 1; ++i) {
        out[i] = 0;
    }
    out[this.dimensions - 1] = 1;
    // out[0] = 1;
    return new Vector(out);
}

/**
 * Creates a coordinate pointing at the origin of space with the correct
 * number of dimensions.
 *
 * @return
 */
public Coord origin() {
    return new Coord(new int[this.dimensions]);
}
}

```

Listing C.42: multidimensional/basics/Coord.java

```

package multidimensional.basics;

/**
 * This class represents a coordinate on an n-ary plane. It provides methods
 * to
 * interact with Vectors for the purposes of addition, and other stuff so it
 * can
 * be used in various collections.
 *
 * @author richard
 */
public class Coord implements Comparable<Coord> {

    /**
     * The internal data.
     */
    private final int data[];

    /**
     * Constructs a Coord with the specified values.
     *
     * @param in
     *         the data to set.
     */
}

```

```

    */
    public Coord(final int... in) {
        this.data = in.clone();
    }

    /**
     * Adds the value of a vector to this coordinate.
     *
     * @param m
     *         the vector to add.
     * @return the result of the computation.
     */
    public Coord addVector(final Vector m) {
        if (m.dimensions() != this.data.length) {
            throw new RuntimeException("Incorrect_vector_length");
        }
        final Coord ret = new Coord(this.data);
        for (int i = 0; i < this.data.length; ++i) {
            ret.data[i] += m.get(0, i);
        }
        return ret;
    }

    @Override
    public Coord clone() {
        return new Coord(this.data);
    }

    @Override
    public int compareTo(final Coord c) {
        if (c.data.length != this.data.length) {
            throw new IllegalArgumentException(String.format(
                "passed_coordinates_did_not_match_in_length: %s != %s",
                this, c));
        }
        for (int i = 0; i < this.data.length; ++i) {
            final int v = this.data[i] - c.data[i];
            if (v != 0) {
                return v;
            }
        }
        return 0;
    }

    @Override
    public boolean equals(final Object c) {
        if (c instanceof Coord) {
            return this.compareTo((Coord) c) == 0;
        }
        return false;
    }

    /**
     * Gets the ith component
     *
     * @param i
     *         the number of the dimension.
     * @return the value of that component.
     */
    public int getComponent(final int i) {
        return this.data[i];
        // return i < data.length ? data[i]:0;
    }

```

```

    }

    // java will throw an exception here anyway, and I don't want to be
    // encouraged to catch it. no point doubling the work of the JVM.
    // public void setComponent(final int c, final int i) {
    //     this.data[c] = i;
    // }

    /**
     *
     * @return the number of dimensions in which this coordinate exists.
     */
    public int getDimensions() {
        return this.data.length;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        for (final int i : this.data) {
            hash *= 31;
            hash += i;
        }
        return hash;
    }

    /**
     * This converts the co-ordinate to an array of floats, scaling by the
     * specified factor. This method is for the benefit of the view classes
     * and
     * shouldn't be used in the model, however it is useful to keep code and
     * data together at times like this.
     *
     * @param multiplier
     *         is multiplied with each element before storing in the new
     *         array.
     * @return float array containing the values of the coordinates scaled
     *         by
     *         the multiplier.
     */
    public float[] toFloatArray(final float multiplier) {
        final float[] ret = new float[this.data.length];
        for (int i = this.data.length - 1; i >= 0; --i) {
            ret[i] = (float) this.data[i] * multiplier;
        }
        return ret;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder();
        sb.append('(');
        sb.append(String.format("%d", this.data[0]));
        for (int i = 1; i < this.data.length; ++i) {
            sb.append(String.format(",%d", this.data[i]));
        }
        sb.append(')');
        return sb.toString();
    }
}

```

Listing C.43: multidimensional/basics/Grid.java

```

package multidimensional.basics;

import java.util.Collection;

/**
 * This class is an extension of the Grid class for the 2 dimensional
 * turmites.
 * It behaves much the same.
 *
 * @author richard
 *
 */
public abstract class Grid {

    /**
     * The listeners to be notified of a change.
     */
    // chances are there'll only be one - so don't waste too much space.
    private final Collection<GridListener> listeners = new java.util.Vector<
        GridListener>(
            1);

    /**
     * Adds a listener to this grid.
     *
     * @param l
     */
    final public void addGridListener(final GridListener l) {
        synchronized (this.listeners) {
            this.listeners.add(l);
        }
    }

    /**
     * Gets the value stored at the coordinate provided.
     *
     * @param pos
     *         the coordinate to check.
     * @return the value of the element referenced.
     */
    abstract public int get(Coord pos);

    /**
     * Notifies all the MultiGridListeners of the updated value.
     *
     * @param pos
     *         the coordinate that has been updated.
     * @param i
     *         the value it was set to.
     */
    final protected void notifyListeners(final Coord pos, final int i) {
        synchronized (this.listeners) {
            for (final GridListener l : this.listeners) {
                l.gridUpdated(pos, i);
            }
        }
    }

    /**
     * Sets the location specified to the provided value.

```

```

*
* @param pos
* @param i
*/
abstract public void set(Coord pos, int i);

/**
 * Checks that a point is suitable to use with this grid.
 *
 * @param p
 */
protected void validatePoint(final Coord p) {
    final int d = p.getDimensions();
    final int thisd = Dimensionality.getInstance().getDimensions();
    if (d > thisd) {
        throw new RuntimeException(String
            .format("Dimension count mismatch: expecting <%d. Got %d",
                thisd, d));
    }
}
}

```

Listing C.44: multidimensional/basics/Vector.java

```

package multidimensional.basics;

/**
 * This class is a special case of a matrix in that it only has one column.
 * The
 * operation methods are overridden to return a vector.
 *
 * @author richard
 */
public class Vector extends Matrix implements Cloneable {
    public Vector(final int... values) {
        super(1, values.length);
        for (int i = 0; i < values.length; ++i) {
            this.set(i, 0, values[i]);
        }
    }

    @Override
    public Vector clone() {
        return new Vector(this.values);
    }

    @Override
    public Vector multiply(final int n) {
        return new Vector(super.multiply(n).values);
    }

    @Override
    public Vector multiply(final Matrix m) {
        return new Vector(super.multiply(m).values);
    }

    public Vector multiply(final Rotation r) {
        return new Vector(super.multiply(r.getMatrix()).values);
    }
}

```

}

Listing C.45: multidimensional/basics/Rotation.java

```

package multidimensional.basics;

import java.io.Serializable;

public class Rotation implements Comparable<Rotation>, Serializable {
    private static final long serialVersionUID = 1;

    private final int direction;

    private final int dimension;

    private String identifier = "ID_NOT_SET";

    /**
     * The rotation matrix.
     */
    private Matrix m;

    protected Rotation(final int dim, final int dir) {
        this.dimension = dim;
        this.direction = dir;
        this.identifier = String.format("%2d%c", dim, dir == 0 ? '-' : '+');
    }

    /**
     * Constructs a rotation object without using the matrices in
     * Dimensionality. (for example, forward and backwards)
     *
     * @param m
     *         the matrix to use.
     * @param s
     *         the identifier to give it.
     */
    protected Rotation(final Matrix m, final String s) {
        this.dimension = this.direction = -1;
        this.identifier = s;
        this.m = m;
    }

    @Override
    public int compareTo(final Rotation o) {
        if (this.getDimension() == -1 && this.getDirection() == -1) {
            if (o.getDimension() == -1 && o.getDirection() == -1) {
                return this.identifier.compareTo(o.identifier);
            }
            return -1;
        }
        if (this.getDimension() != o.getDimension()) {
            return this.getDimension() - o.getDimension();
        }
        return this.getDirection() - o.getDirection();
    }

    @Override
    public boolean equals(final Object o) {
        if (o == null) {

```

```

        return false;
    }
    if (!(o instanceof Rotation)) {
        return false;
    }
    return this.compareTo((Rotation) o) == 0;
}

protected int getDimension() {
    return this.dimension;
}

protected int getDirection() {
    return this.direction;
}

/**
 * Gets the rotation matrix associated with the direction and dimension.
 *
 * @return said matrix.
 */
Matrix getMatrix() {
    return (this.m != null) ? this.m : Dimensionality.getInstance()
        .getMatrix(this.getDimension(), this.getDirection());
}

public Rotation invert() {
    final Dimensionality d = Dimensionality.getInstance();
    final int ndir = this.direction == 0 ? 1 : 0;
    final int ndim = this.dimension == -1 ? 0 : this.dimension + 1;
    return d.getRotations()[ndir][ndim];
}

public Matrix mul(final Matrix m) {
    return this.getMatrix().multiply(m);
}

@Override
public String toString() {
    return this.identifier;
}
}

```

Listing C.46: multidimensional/basics/MatrixTest.java

```

package multidimensional.basics;

/** @hidden */
public class MatrixTest {
    public static void main(final String args[]) {
        Dimensionality.setDimensions(3);
        final Rotation r = Dimensionality.getInstance().getRotations()
            [2][1];
        final Vector v = new Vector(0, 0, 1);
        System.out.printf("%s\nTIMES%n%s\nEQUALS%n%s\n", v, r.getMatrix()
            .toLatex(), v.multiply(r.getMatrix()).toLatex());
    }
}

```

Listing C.47: multidimensional/basics/SimpleTest.java

```

package multidimensional.basics;

import multidimensional.util.CoordSet;
import multidimensional.util.TreeArray;

/**
 * This runs langons ant on the new n-dimensional stuff (although it's still
 * a
 * 2d thingy)
 * @hidden
 * @author richard
 *
 */
public class SimpleTest {
    private static final int DIMENSIONS = 2;
    Coord position = new Coord(50, 50);
    Vector direction = new Vector(1, 0);
    static Dimensionality rot = Dimensionality.setDimensions(DIMENSIONS);
    CoordSet<Integer> grid = new TreeArray<Integer>(0);
    Matrix l = rot.getMatrix(0, 1);
    Matrix r = rot.getMatrix(0, 3);

    void act() {
        int val = grid.get(position);
        if (val == 0) {
            System.out.println("Turmite_got_0");
            direction = direction.multiply(l);
            System.out.println("Got_new_direction_after_turning_left:");
            System.out.println(direction);
            grid.set(position, 1);
        } else {
            System.out.println("Turmite_got_" + val);
            direction = direction.multiply(r);
            System.out.println(direction);
            grid.set(position, 0);
        }
        System.out.println("Position_was:");
        System.out.println(position);
        position = position.addVector(direction);
        System.out.println("Position_is:");
        System.out.println(position);
        System.out.println("xxxxxx");
        // System.out.printf("Grid size=%d\n", grid.grid.size());
    }

    public static void main(String args[]) {
        Vector dir = new Vector(1, 0);
        for (int i = 0; i < 4; ++i) {
            System.err.printf("***%d\n", i);
            System.out.printf("%d: \n%s\n", i, dir.multiply(rot.getMatrix(0,
                i)));
        }
        SimpleTest t = new SimpleTest();
        for (int i = 0; i < 15000; ++i)
            t.act();
    }
}

```

Listing C.48: multidimensional/basics/GridListener.java

```

package multidimensional.basics;

public interface GridListener {
    /**
     * This is called when a new value has been set on the grid.
     *
     * @param pos
     *           The grid location of the change.
     * @param value
     *           The new value.
     */
    public void gridUpdated(Coord pos, int value);
}

```

Listing C.49: multidimensional/Transition.java

```

package multidimensional;

import java.io.Serializable;

import multidimensional.basics.Rotation;

public class Transition implements Serializable, Comparable<Transition> {
    private final int new_colour, old_colour, new_state, old_state;
    private final Rotation rotation;

    private static final long serialVersionUID = 2;

    protected Transition(final int old_state, final int old_colour) {
        this.old_colour = old_colour;
        this.old_state = old_state;
        this.new_colour = this.new_state = -1;
        this.rotation = null;
    }

    public Transition(final int old_state, final int old_colour,
                     final Rotation rot, final int new_state, final int new_colour) {
        this.old_colour = old_colour;
        this.old_state = old_state;
        this.new_colour = new_colour;
        this.new_state = new_state;
        this.rotation = rot;
    }

    @Override
    public int compareTo(final Transition that) {
        final int x = this.getOldState() - that.getOldState();
        return (x != 0) ? x : this.getOldColour() - that.getOldColour();
    }

    @Override
    public boolean equals(final Object o) {
        if (o == null) {
            return false;
        }
        if (o instanceof Transition) {
            final Transition t = (Transition) o;
            return this.old_state == t.old_state

```

```

        && this.old_colour == t.old_colour;
    }
    return false;
}

public int getNewColour() {
    return this.new_colour;
}

public int getNewState() {
    return this.new_state;
}

public int getOldColour() {
    return this.old_colour;
}

public int getOldState() {
    return this.old_state;
}

//
// protected Matrix getRotationMatrix() {
//     return getRotation().getMatrix();
// }

public Rotation getRotation() {
    return this.rotation;
}

@Override
public int hashCode() {
    return ((this.old_state << 16) | (this.old_colour & 0xffff));
}

@Override
public String toString() {
    return String.format("%d_%d->%d_%d_%s", this.getOldState(),
        this
            .getOldColour(), this.getNewState(), this.getNewColour(),
            this
                .getRotation().toString());
}
}

```

Listing C.50: multidimensional/TransitionFunction.java

```

package multidimensional;

import java.io.Serializable;
import java.util.Collection;

/**
 * This class provides a mapping of state, colour combinations to state
 * transitions.
 *
 * @author richard
 *
 */
public abstract class TransitionFunction implements Serializable {

```

```

/**
 * the name assigned to the function for the purposes of serialization.
 */
private String name = null;

/**
 * The default name if one isn't set.
 */
private static final String DEFAULTNAME = "Untitled";

private static final long serialVersionUID = 2;

/**
 * Constructs an empty transition table.
 */
public TransitionFunction() {
}

protected TransitionFunction(final Collection<Transition> actions) {
    this();
    for (final Transition t : actions) {
        this.add(t);
    }
}

// another reason I wish java had templates.
/**
 * Constructs a transition table comprising of the provided actions.
 *
 * @param actions
 */
protected TransitionFunction(final Transition... actions) {
    this(); // call the default in case I ever need to put anything
           // there.
    for (final Transition a : actions) {
        this.add(a);
    }
}

/**
 * Adds the action to the transition table, replacing anything that was
 * previously in the slot for that state, colour combination.
 *
 * @param a
 *         the action to add.
 */
public abstract void add(Transition a);

/**
 * Get the action relating to to provided state, colour pair.
 *
 * @param current_state
 *         the current state of the target TM.
 * @param current_colour
 *         the current colour underneath the TM.
 * @return the action. If there was an element matching the state but
 *         not
 *         the colour, it behaves as though the colour is 0. If there
 *         was no
 *         corresponding action found, returns NULL.
 */
public abstract Transition get(int current_state, int current_colour);

```

```

/**
 *
 * @return the name of the transition function.
 */
public String getName() {
    return this.name == null ? TransitionFunction.DEFAULTNAME : this.name;
}

/**
 * Sets the name of this transition function.
 *
 * @param name
 *         the name to set.
 * @return reference to this.
 */
public TransitionFunction setName(final String name) {
    this.name = name;
    return this;
}

/**
 * Returns an array representation of the transition function. (that is,
 * an
 * array of all the transitions).
 *
 * @return array containing all the transitions in this function.
 */
public Transition[] toArray() {
    return this.toCollection().toArray(new Transition[0]);
}

/**
 * Returns a collection of all the Actions which make up the turmite.
 *
 * @return vector of it all.
 */
public abstract Collection<Transition> toCollection();

@Override
public String toString() {
    final StringBuilder sb = new StringBuilder();
    sb.append("Transition_function:");
    sb.append(this.getName());
    sb.append('\n');
    for (final Transition a : this.toCollection()) {
        sb.append(a.toString());
        sb.append('\n');
    }
    return sb.toString();
}
}

```

Listing C.51: multidimensional/TurmiteException.java

```

package multidimensional;

/**
 * Indicates an error somewhere.

```

```

*
* @author richard
*
*/
public class TurmiteException extends Exception {
    /**
     * Constructs the exception. Same form as printf.
     *
     * @param fmt
     *         format string.
     * @param args
     *         argument list.
     */
    public TurmiteException(final String fmt, final Object... args) {
        super(String.format(fmt, args));
    }
}

```

Listing C.52: multidimensional/TurmiteListener.java

```

package multidimensional;

/**
 * Classes implementing this interface receive TurmiteEvents from the
 * turmite
 * they are registered with. This is intended to be useful for displaying
 * the
 * current state etc.
 *
 * @author richard
 *
*/
public interface TurmiteListener {

    /**
     * Receives an event from the turmite.
     *
     * @param e
     *         the event sent.
     */
    public void send(TurmiteEvent e);
}

```

Listing C.53: multidimensional/GridImpl.java

```

package multidimensional;

import multidimensional.basics.Coord;
import multidimensional.basics.Grid;
import multidimensional.util.CoordSet;

/**
 * This class is an extension of the Grid class for the 2 dimensional
 * turmites.
 * It behaves much the same.
 *
 * @author richard
 *
*/

```

```

public class GridImpl extends Grid {
    /**
     * This actually stores the data.
     */
    private final CoordSet<Integer> grid;

    protected GridImpl(final CoordSet<Integer> grid) {
        this.grid = grid;
    }

    @Override
    public int get(final Coord pos) {
        this.validatePoint(pos);
        return this.grid.get(pos);
    }

    @Override
    public void set(final Coord pos, final int i) {
        this.validatePoint(pos);
        this.grid.set(pos, Integer.valueOf(i));
        this.notifyListeners(pos, i);
    }
}

```

Listing C.54: multidimensional/Turmite.java

```

package multidimensional;

import multidimensional.basics.Coord;
import multidimensional.basics.Dimensionality;
import multidimensional.basics.Grid;
import multidimensional.basics.Matrix;
import multidimensional.basics.Vector;

/**
 * This class represents a Turmite, which is essentially an instance of a
 * transition function.
 *
 * @author richard
 */
class Turmite implements ITurmite {

    private TurmiteListener listener = null;
    /**
     * The current location of the Turmite on the grid.
     */
    private Coord position;
    /**
     * The basis for the matrix math.
     */
    private final Vector initial_orientation;
    /**
     * The grid on which the turmite operates.
     */
    private final Grid g;
    /**
     * The state register.
     */
    private int state = 0;
}

```

```

/**
 * The mapping of the current colour under the turmite and the current
 * state
 * to the transition to be performed.
 */
private final TransitionFunction f;
/**
 * The accumulated rotations up to this point.++++
 */
private Matrix rotation;

/**
 * Constructs a turmite.
 *
 * @param grid
 *         the grid that it should use.
 * @param f
 *         the transition function.
 * @param pos
 *         the initial position.
 */
protected Turmite(final Grid grid, final TransitionFunction f,
                  final Coord pos) {
    this.g = grid;
    this.f = f;
    this.position = pos;
    final Dimensionality dimen = Dimensionality.getInstance();
    this.initial_orientation = dimen.getDefaultHeading();
    this.rotation = dimen.getIdentity();
}

/*
 * (non-Javadoc)
 *
 * @see multidimensional.ITurmite#act()
 */
@Override
public boolean act() {
    // get the relevant transition for this state, colour pair.
    final Transition t = this.getNextTransition();
    if (t == null) {
        return true;
    }
    this.doTransition(t);
    return false;
}

private void doTransition(final Transition t) {
    this.g.set(this.position, t.getNewColour());
    // update the state register.
    this.state = t.getNewState();
    // accumulate the rotation
    this.rotation = t.getRotation().mul(this.rotation);
    // convert it to a vector.
    final Vector heading = this.getNextHeading();
    // update the position
    this.position = this.position.addVector(heading);
    if (this.listener != null) {
        this.listener.send(new TurmiteEvent(this, t, heading));
    }
}

```

```

}

/*
 * (non-Javadoc)
 * @see multidimensional.ITurmite#equals(java.lang.Object)
 */
@Override
public boolean equals(final Object o) {
    return this == o;
}

private Vector getNextHeading() {
    return this.initial_orientation.multiply(this.rotation);
}

private Transition getNextTransition() {
    return this.f.get(this.state, this.g.get(this.position));
}

/*
 * (non-Javadoc)
 * @see java.lang.Object#hashCode()
 */
@Override
public int hashCode() {
    return this.f.getName().hashCode();
}

/*
 * (non-Javadoc)
 * @see multidimensional.ITurmite#setListener(multidimensional.Turmite.
 * TurmiteListener)
 */
@Override
public void setListener(final TurmiteListener l) {
    this.listener = l;
}

@Override
public String toString() {
    return this.f.getName();
}

// no gui for this :(
@Override
public void undoTransition(final Transition t) {

    // update the state register.
    this.state = t.getOldState();
    // undo the rotation
    this.rotation = t.getRotation().invert().mul(this.rotation);
    // convert it to a vector.
    final Vector heading = this.getNextHeading();
    // update the position (going backwards)
    this.position = this.position.addVector(heading.multiply(-1));
    this.g.set(this.position, t.getOldColour());
}
}

```

Listing C.55: multidimensional/TransitionFunctionDoubleMap.java

```

package multidimensional;

import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Vector;

/**
 * This implementation of TransitionFunction uses a double Map to do it's
 * work.
 * Probably not the best way to do it.
 *
 *
 * @author richard
 *
 */
public class TransitionFunctionDoubleMap extends TransitionFunction {
    private final Map<Integer, Map<Integer, Transition>> states = new
        HashMap<Integer, Map<Integer, Transition>>();

    /**
     * Constructs an empty transition table.
     */
    public TransitionFunctionDoubleMap() {
    }

    public TransitionFunctionDoubleMap(
        final Collection<Transition> actions) {
        for (final Transition t : actions) {
            add(t);
        }
    }

    /**
     * another reason I wish java had templates.
     */
    /**
     * Constructs a transition table comprising of the provided actions.
     *
     * @param actions
     */
    public TransitionFunctionDoubleMap(final Transition... actions) {
        for (final Transition a : actions) {
            add(a);
        }
    }

    /**
     * Adds the action to the transition table, replacing anything that was
     * previously in the slot for that state, colour combination.
     *
     * @param a
     *         the action to add.
     */
    @Override
    public void add(final Transition a) {
        final int state = a.getOldState();
        final int colour = a.getOldColour();
    }
}

```

```

        if (states.containsKey(state)) {
            states.get(state).put(colour, a);
        } else {
            final Map<Integer, Transition> colourmap = new HashMap<Integer,
                Transition>();
            colourmap.put(colour, a);
            states.put(state, colourmap);
        }
    }

    /**
     * Get the action relating to to provided state, colour pair.
     *
     * @param current_state
     *     the current state of the target TM.
     * @param current_colour
     *     the current colour underneath the TM.
     * @return the action. If there was an element matching the state but
     *     not
     *     the colour, it behaves as though the colour is 0. If there
     *     was no
     *     corresponding action found, returns NULL.
     */
    @Override
    public Transition get(final int current_state,
        final int current_colour) {
        final Map<Integer, Transition> colourmap = states
            .get(current_state);
        if (colourmap == null) {
            return null;
        }
        Transition transition = colourmap
            .get(current_colour);
        if (transition == null) {
            transition = colourmap.get(0);
        }
        return transition;
    }

    @Override
    public List<Transition> toCollection() {
        final Collection<Map<Integer, Transition>> inside = states
            .values();
        final List<Transition> ret = new Vector<Transition>(
            inside.size());
        for (final Map<Integer, Transition> it : inside) {
            ret.addAll(it.values());
        }
        return ret;
    }
}
}

```

Listing C.56: multidimensional/TransitionFunctionHash.java

```

package multidimensional;

import java.util.Collection;
import java.util.HashMap;

```

```

import java.util.Map;

/**
 * This implementation uses a single map and hashes the transitions.
 *
 * @author richard
 *
 */
public class TransitionFunctionHash extends TransitionFunction {
    Map<Integer, Transition> transitions = new HashMap<Integer, Transition>
        >();

    private static final long serialVersionUID = 2;

    //
    // private static Integer hashInt(final int i1,
    // final int i2) {
    // return Integer
    // .valueOf(((i1 << 16) | (i2 & 0xffff)));
    // }

    public TransitionFunctionHash(final Collection<Transition> actions) {
        super(actions);
    }

    // another reason I wish java had templates.
    /**
     * Constructs a transition table comprising of the provided actions.
     *
     * @param actions
     */
    public TransitionFunctionHash(final Transition... actions) {
        super(actions);
    }

    /**
     * Adds the action to the transition table, replacing anything that was
     * previously in the slot for that state, colour combination.
     *
     * @param a
     *         the action to add.
     */
    @Override
    public void add(final Transition a) {
        // final int state = a.getOldState();
        // final int colour = a.getOldColour();
        // // final Integer h = TransitionFunctionHash.hashInt(state, colour
        // );
        this.transitions.put(a.hashCode(), a);
    }

    /**
     * Get the action relating to to provided state, colour pair.
     *
     * @param current_state
     *         the current state of the target TM.
     * @param current_colour
     *         the current colour underneath the TM.
     * @return the action. If there was an element matching the state but
     *         not

```

```

        *         the colour, it behaves as though the colour is 0. If there
        *         was no
        *         corresponding action found, returns NULL.
        */
    @Override
    public Transition get(final int current_state, final int current_colour)
    {
        Transition t = this.transitions.get(new Transition(current_state,
            current_colour).hashCode());
        if (t == null) {
            t = this.transitions.get(new Transition(current_state, 0)
                .hashCode());
        }
        return t;
    }

    /**
     * Returns a collection of all the Actions which make up the turmite.
     *
     * @return vector of it all.
     */
    @Override
    public Collection<Transition> toCollection() {
        return this.transitions.values();
    }
}

```

Listing C.57: multidimensional/TuringMachine.java

```

package multidimensional;

import multidimensional.basics.Coord;
import multidimensional.basics.Dimensionality;
import multidimensional.basics.Grid;
import multidimensional.basics.Vector;

/**
 * This class implements a traditional turing machine, without the
 * accumulation
 * of orientation.
 *
 * @author richard
 */
class TuringMachine implements ITurmite {

    private TurmiteListener listener = null;
    /**
     * The current location of the Turmite on the grid.
     */
    private Coord position;
    /**
     * The basis for the matrix math.
     */
    private final Vector initial_orientation;
    /**
     * The grid on which the turmite operates.
     */
    private final Grid g;
}

```

```

    * The state register.
    */
    private int state = 0;
    /**
     * The mapping of the current colour under the turmite and the current
     * state
     * to the transition to be performed.
     */
    private final TransitionFunction f;

    /**
     * Constructs a turmite.
     *
     * @param grid
     *         the grid that it should use.
     * @param f
     *         the transition function.
     * @param pos
     *         the initial position.
     */
    protected TuringMachine(final Grid grid, final TransitionFunction f,
        final Coord pos) {
        this.g = grid;
        this.f = f;
        this.position = pos;
        final Dimensionality dimen = Dimensionality.getInstance();
        this.initial_orientation = dimen.getDefaultHeading();
    }

    /*
     * (non-Javadoc)
     *
     * @see multidimensional.ITurmite#act()
     */
    @Override
    public boolean act() {
        // get the relevant transition for this state, colour pair.
        final Transition t = this.getNextTransition();
        if (t == null) {
            return true;
        }
        this.doTransition(t);
        return false;
    }

    private void doTransition(final Transition t) {
        this.g.set(this.position, t.getNewColour());
        // update the state register.
        this.state = t.getNewState();
        // translate the position.
        final Vector heading = this.initial_orientation.multiply(t
            .getRotation());
        // update the position
        this.position = this.position.addVector(heading);
        if (this.listener != null) {
            this.listener.send(new TurmiteEvent(this, t, heading));
        }
    }

    /*
     * (non-Javadoc)

```

```

    *
    * @see multidimensional.ITurmite#equals(java.lang.Object)
    */
    @Override
    public boolean equals(final Object o) {
        return this == o;
    }

    private Vector getNextHeading(final Transition t) {
        return this.initial_orientation.multiply(t.getRotation());
    }

    private Transition getNextTransition() {
        return this.f.get(this.state, this.g.get(this.position));
    }

    /*
    * (non-Javadoc)
    *
    * @see java.lang.Object#hashCode()
    */
    @Override
    public int hashCode() {
        return this.f.getName().hashCode();
    }

    /*
    * (non-Javadoc)
    *
    * @see multidimensional.ITurmite#setListener(multidimensional.Turmite.
    * TurmiteListener)
    */
    @Override
    public void setListener(final TurmiteListener l) {
        this.listener = l;
    }

    @Override
    public String toString() {
        return this.f.getName();
    }

    // no gui for this :(
    @Override
    public void undoTransition(final Transition t) {

        // update the state register.
        this.state = t.getOldState();
        // undo the rotation
        // convert it to a vector.
        final Vector heading = this.getNextHeading(t);
        // update the position (going backwards)
        this.position = this.position.addVector(heading.multiply(-1));
        this.g.set(this.position, t.getOldColour());
    }
}

```

Listing C.58: multidimensional/TurmiteEvent.java

```
package multidimensional;
```

```

import multidimensional.basics.Vector;

public class TurmiteEvent {
    private final Transition transition;
    private final Vector heading;
    private final ITurmite source;

    /**
     *
     * @param source
     *         The turmite that the event originated from.
     * @param transition
     *         The transition that was executed.
     * @param heading
     *         the direction the head is pointing.
     */
    protected TurmiteEvent(final ITurmite source, final Transition
        transition,
        final Vector heading) {
        this.source = source;
        this.transition = transition;
        this.heading = heading;
    }

    @Override
    public boolean equals(final Object that) {
        if (that != null) {
            if (that instanceof TurmiteEvent) {
                return this.source == ((TurmiteEvent) that).source;
            }
        }
        return false;
    }

    public Vector getHeading() {
        return this.heading;
    }

    public ITurmite getSource() {
        return this.source;
    }

    public Transition getTransition() {
        return this.transition;
    }

    @Override
    public int hashCode() {
        return this.source.toString().hashCode(); // just need something
        // that won't change
        // during execution.
    }

    @Override
    public String toString() {
        return String.format("%s:%nTransition: %s%nHeading: %s%a", this.
            source
            .toString(), this.transition, this.heading.toString());
    }
}

```



Appendix D

Specification

D.1 Project Description

The aim of this project is to create a generic piece of software to facilitate the simulation and study of two dimensional Turing Machines (Turmites). It is being written for Frank Wolter under his supervision. I propose a solution allows a user to define and analyse different Turmites. The user interface should be cleanly separated from the back-end. If possible I would like to extend the model into 3 or more dimensions. Additionally I will attempt to compute something using these machines, as they have been shown to be equivalent to normal Turing machines.

D.2 Statement of Deliverables

The supplied documentation will consist of:

- User Guide explaining how to use the software, and to serve as a reference.
- Technical design document (general overview and layout of source).
- The documentation automatically-generated from the source code.
- The software will be conceptually split into three parts:
 - The 'field' on which the turmites operate. The interface provided should be that of an array of the correct dimension, but the implementation should be cleanly separated for the purposes of experimentation.
 - The turmites themselves, and their transition table. There should be support for loading and saving these to a file so the user does not need to recreate them each time the program is run.
 - The user interface, which will provide a means to design a transition function and to run the simulation.

Each of these can be tested independently. For all but the GUI it should be possible to construct unit tests using an appropriate framework.

D.2.1 Features

- Have support for both fixed size and dynamically-sized fields for the turmites.
- Be able to save/load the transition function definitions to/from a file.

- Provide a GUI to configure the turmite and run the simulation.

D.2.2 Desirable features

- Extend the simulation into 3 or more dimensions.
- Be able to load the initial state of the grid (e.g. from an image file).
- Have a variety of visualisation options.

D.2.3 Experiments

Experiments will be conducted to test the viability of different programming approaches. This will involve separating the interface of each major component with its implementation so that they can be substituted at a later date. They will be mostly subjective for the visual components to see if there is any improvement in the clarity of information presented. For performance-based concerns, unless there is a noticeable improvement or decrease I shall profile it using an appropriate tool for the language.

D.2.4 Evaluation

- I shall evaluate the product using the following criteria:
- It must not crash, even when given bad input. A message should be displayed if appropriate.
- It must run on a variety of systems (old and new, windows and unix based).
- It must be correct. This should be verifiable by executing well-known turmites and examining the output. As it progresses this will turn into hand-executing code and comparing this with both the expected and actual output.

D.3 Conduct of the Project and Plan

D.4 Preparation

The background research will span a number of areas:

- Turmites themselves, how they are defined, how they operate.

- Algorithms for manipulating initially sparse, but potentially eventually dense collections of data.
- The documentation for the libraries to provide the graphics and user interface. I have already read about each of these, and have preexisting experience with graphical interfaces and algorithms.

D.4.1 Design Stage

An initial prototype was constructed using Java to familiarize myself with what is required. I plan to use an object oriented language for implementation, however more research is needed in order to decide which.

D.4.2 Milestones

1. Turmite simulation works in 2 dimensions, no visualisation. Needs to be stepped through either in a debugger or using print statements on a test case.
2. Visualisation for 2 dimensions, try and extend model to 3 or more.
3. GUI for creating transition tables.
4. Turmite serialization, experiment with 3 dimensional visualisations.

D.4.3 Hardware to be used

The development system has the following components, and runs 64 bit Gentoo Linux:

- Intel Core 2 Quad Q6600 overclocked to 3.2ghz
- 4096 megabytes of DDR2 memory
- Asus P5K-E Wifi/AP motherboard
- Nvidia Geforce 8500GT graphics card.

D.4.4 Software to be used

- the Eclipse integrated development environment for java. This allows simple integration with a debugger and refactoring tools.
- The Sun JDK 1.6.
- Sun xVM VirtualBox for testing on 32 bit BSD and Windows.

The code will be tested on both 32 and 64-bit Linux, as well as 32 bit Windows and FreeBSD. If Java is not the final implementation language, I shall substitute it for the appropriate development environment and compiler.

D.5 Risk assessment

D.5.1 Major challenges

- Finding an efficient way to store the data the turmites produce/work on.
- Finding an effective way to visualize 3 dimensional data.
- Creating a user interface that is both easy to use and versatile.

D.5.2 New skills

- Working with 3 dimensional graphics.
- Thinking in many dimensions

Appendix E

Original Design

E.1 abstract

Turmites are an extension of Turing machines to use a two-dimensional tape (or plane).

This document provides the detailed design for my multi-dimensional Turing machine simulation. The product shall simulate them in any number of dimensions and provide a method of visualisation for one, two or three dimensional systems.

E.2 What is a Turmite?

Before I can describe my solution to the problem, we must first understand the conceptual model of a multi-dimensional Turing machine.

E.2.1 What is a Turing Machine?

A traditional Turing machine (TM) is a finite state automaton with a one dimensional storage array first proposed by Alan Turing[1]. This model can be formalised as a 7-tuple[18]:

$$M = \{Q, \Gamma, b, \Sigma, \delta, q_0, F\}$$

- Q is a finite set of states.
- Γ is a finite set of possible values under the head.
- $b \in \Gamma$ is the blank symbol, which exists in all elements of the tape which have not been set to a defined value.
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of input symbols, or ones which the machine responds to.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ the transition function, which changes the state and moves the head left (L) or right (R).
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final or accepting states.

For the purposes of the simulation of such systems on a modern computer we can make a few simplifications. When dealing with the set of integers (\mathbb{I}), will be taken to mean the range of integers expressible by the computer as a 32 bit integer. $(-2, 147, 483, 648$ to $+2, 147, 483, 647)$

- $\Gamma \equiv \Sigma \equiv \mathbb{I}$ The set of possible inputs is the set of possible values on the tape, which is also the set of integers.
- $b = 0$ The blank symbol is defined as the integer 0.
- $q_0 = 0$ The initial state is also 0.
- $Q \equiv \mathbb{I}$ The set space is as large as the computer can handle.
- Accepting states are ones which would otherwise lie outside the domain of the transition function.

So, in order to simulate a Turing machine with these simplifications, we end up with a 5-tuple:

- the set of possible direction changes (in this case, left and right - or forward and back)
- The location of the head on the tape.
- The contents of the tape.
- The state register.
- The transition function.

E.2.2 Extending into two dimensions

Possibly the most well known example of a 2d Turing Machine (or Turmite) can be credited to C Langton[2] with “Langton’s Ant”. However it was also discovered by at least three other people.

We can extend this model into two dimensions by:

- Exchanging the tape for a two-dimensional array, or a grid.
- Extending the head pointer to be a 2-tuple which contains the coordinates on the grid.
- Extending the set of directions in the grid to include up, down, left and right.
- The movement of the head is constant, meaning it always moves one step forward. The input from the transition function affects the orientation of the head, and so the direction it moves on the grid.

I shall be basing my model on the work of A.K Dewdney[3].

E.2.3 Generalising to n dimensions

Here we can infer a number of interesting points:

- For each dimension, we add two more possible changes of direction.
- For N dimensions, the tape can be represented by an N dimensional grid of elements.
- Similarly, the head position is an N-tuple.
- Movement of the head always occurs in units of 1, parallel to one of the axis.

E.3 Design

E.3.1 General Components

The simulation relies heavily on matrix and vector mathematics. Hence there needs to be some groundwork layed for this to work and for the rest to build on. I have decided to make these classes immutable as it simplifies the interactions and run-time dependencies between them[5].

A UML class diagram is shown in figure E.1.

Matrix

This is an n by m matrix class which provides common matrix operations, including:

- Matrix and scalar multiplication.
- Matrix and scalar addition.

Its purpose is to handle the orientation and rotation of the Turmite heads in N-space. All operations return a new matrix as the result, apart from the `get()/set()` methods as that would be unduly wasteful of resources.

Whenever there is a bad value passed, an exception will be thrown. The intention is not that they should be caught, but should never be caused to be thrown. For example trying to multiply two incompatible matrices.

Vector

This is simply a special case of **Matrix** which has only one column. It inherits from **Matrix**.

The multiply method is overridden to return a **Vector** reference instead of a **Matrix**. This is valid because the only reasonable multiplication of a **Vector** is by a rotation matrix and the result will be a vector of the same order.

Coord

This class represents an n-tuple for the purposes of reading and writing points in the grid. It supports addition with other **Coords** and **Vectors**.

Additionally the instances of the class must be comparable - for example implementing the **Comparable<Coord>** interface in Java, or overloading the **operator<>** in C++. This is so they can be inserted into container classes and sorted (for example it may be desirable to produce a more complicated scenegraph implementation).

Dimensionality

This class provides information to the rest of the system which concerns the ones mentioned above. It is a singleton, as the information it provides is the same to any part of the system.

It performs the following actions:

- It acts as a matrix cache. As there are only so many rotation matrices that are applicable for a given number of dimensions, it does not make sense to create a new one each time it is needed. The instance precalculates all the matrices which are relevant and caches them upon initialization.
- It contains factory methods for **Coord** and **Vector** to provide default values with the correct number of dimensions, and caches those as well.
- It provides convenience methods for the matrices through the first 3 dimensions. This is mostly to aid in testing before the GUI is complete.
- Calculating a Givens[7] rotation matrix for a set number of dimensions and a certain rotation about a specified axis pair.

E.3.2 The Grid

The grid is the part of the model which stores the symbols written by the Turmite for retrieval. Because the model needs to be extensible into many dimensions an abstract representation of the coordinates must be provided.

The class `Coord` is to be implemented using variadic functions, and shall provide basic vector operations, such as addition. The symbols in the individual cells shall be represented by integers (as this is the basic unit of computation)

Interface

The Grid interface shall consist of two methods:

$$\text{get}(\text{pos} : \text{Coord}) : \text{Integer}$$

$$\text{set}(\text{pos} : \text{Coord}, \text{value} : \text{Integer}) : \text{void}$$

These get or set the value stored at the specified co-ordinate. In addition there needs to be an efficient way to update the display - simply iterating over the visible area is a bad use of resources (as many of the elements are likely to be empty)

$$\text{addGridChangeListener}(l : \text{GridChangeListener}) : \text{void}$$

Where `GridChangeListener` is an interface with the following method:

$$\text{gridUpdated}(\text{pos} : \text{Coord}, \text{value} : \text{Integer}) : \text{void}$$

which is called each time a value on the grid is changed.

Design Considerations

The use of the `GridChangeListener` is important, because it allows the view to keep track of the grid contents in a format appropriate to it's own implementation. It also removes the need to poll a whole area of the grid when the display window is moved/re-sized.

Implementation

The grid shall be treated as an interface in order to allow experimentation with different implementations. However a simple one may be an associative array of `Coords` to integers. For example in the Java programming language:

```
Map<Coord,Integer> grid = new TreeMap<Coord,Integer>();
```

The get and set methods can simply wrap around this; with get returning the 'empty' value if there is no entry in the map for that coordinate and set notifying the listeners.

At this point it is worth mentioning the limits of the system. Using a typical 32 bit operating system on commodity hardware the total amount of memory available to user-mode applications is 2 gigabytes 2^31 . This is because the operating system kernel reserves half of the address space for it's own data structures - this is true on windows and linux at least.

Even in an ideal model, where we make the following assumptions:

- All of the 2 gigabytes is made available for this single structure.
- Instead of the tree nodes holding references to the data they hold the data itself.
- The Coord is of 3 dimensions.

```
struct TreeNode {
TreeNode *left,*right; // pointers to children
int key[3]; // the coord
int value; // the value of the cell
};
```

On 32-bit architectures pointers (variables containing addresses) are 4 bytes, as are integers. This causes each tree node to be $6*8$ or 42 bytes in length.

$$\frac{2,147,483,647}{42} = 51130563$$

Which is a mere 51 million opposed to 2^{31^N} - where N is the number of dimensions. However, the point of using a sparse array is that most of the cells will be empty for most of the time. If this limit is encountered it is possible to use a `Grid` implementation that is based on an actual multidimensional array which will be more space-efficient.

On 64 bit architectures (e.g. AMD64) this will take far longer to occur due to the 41 bit address bus.

E.3.3 The Turmites

I define a Turmite as a 5-tuple:

$$\{state, position, orientation, transition_function, grid\}$$

- The state is defined as an integral value.

- Position is a `Coord` indicating the current location on the grid.
- Orientation is a unit vector in the direction the Turmite is pointing.
- `transition_function` is a dual-indexed associative array which maps a $(state, colour)$ pair to `Transition` object.
- Grid is a reference to the grid on which the Turmite operates.

The Transition Function

A `Transition` is a structure with 5 elements:

$$\{old_colour, old_state, new_colour, new_state, rotation\}$$

The colour and state values are integers, the rotation is represented as a matrix which can be multiplied with the Turmite's orientation to give the resultant direction.

Interface

The Turmite class is constructed by providing the grid, transition function and the initial position. The initial state is defined to be 0. The method that performs the actual work is `act()`.

Implementation

Here is a possible java implementation for the `act()` method of a Turmite.

```
public void act() {
    // get the current colour from the grid.
    int curr = g.get(position);
    // get the relevant transition for this state,colour pair.
    Transition t = transitionfunction.get(state, curr);
    // set the new colour at the current location.
    this.g.set(position, t.new_colour);
    // update the state register.
    this.state=t.new_state;
    // accumulate the rotation
    this.rotation=t.rotation.multiply(rotation);
    // convert it to a vector.
    Vector heading = initial_orientation.multiply(rotation);
    // update the position
    this.position = position.addVector(heading);
}
```

E.3.4 The View

The view includes the user interface for constructing Turmites as well as the means of displaying them. There will be three main windows for the graphical interface;

Definition Window Here the user will define the transition function. There will be a list of the currently defined transitions on the left. On the right will be a form in which to input the 5 values needed.

At the bottom there will be buttons to load and save the definition to/from a file, as well as to proceed with the simulation. (Figure E.2(a))

Configuration Window Here the user can select options such as: (figure E.2(b))

- how large the simulation window should be.
- the specific visualisation to use.
- the time to sleep between iterations.

Simulation Window This will show the simulation running. It will include an iteration counter and a single-stepping function, as well as the option to run the simulation at the previously prescribed rate.

The actual view presented will also depend on the previous choice. For a two dimensional Turmite, it might be similar to that in figure E.2(c), but for one running in three dimensions it could look more like figure E.2(d).

Implementation

The implementation of the view depends heavily on the chosen programming language. The partitioning I have performed will lend itself to being implemented with a variety of graphical toolkits - for example Qt (C++) or AWT/Swing (Java).

E.4 Evaluation Design

The evaluation shall look at, in order of decending priority.

Correctness The software should perform as expected within it's operational parameters.

Robustness The software should not crash or cause undefined behaviour. This applies both during normal use and when the system is intentionally provided with invalid data.

Maintainability The software should still have room to be extended at the end of the project. This should be done by suggesting possible extensions that would be problematic to implement on top of the existing codebase.

Portability The software should run on a number of operating systems and CPU architectures. Including (but not limited to) Linux, FreeBSD and Microsoft Windows. Any libraries used must have equivalents on most platforms.

Performance It should not make too much use of system resources (processor time or memory). In so much as it should not cause the computer running the simulation to run out of either of these resources.

Usability It should not be difficult or cumbersome to use the software. This is a very subjective measure, but important in graphical applications.

E.5 Plan

The project is currently on schedule. Java is currently the implementation language of choice - however in order to provide a three dimensional view it will be necessary to use an external library such as JOGL or Java3D. Both of these are implemented as JNI packages which means the binaries are platform dependant.

This raises the question of whether the main advantage of java holds true. If I need to distribute seperate builds for windows, linux, bsd etc (and my initial experiments show they do not behave the same). However I am familiar with the Swing and AWT programming interfaces.

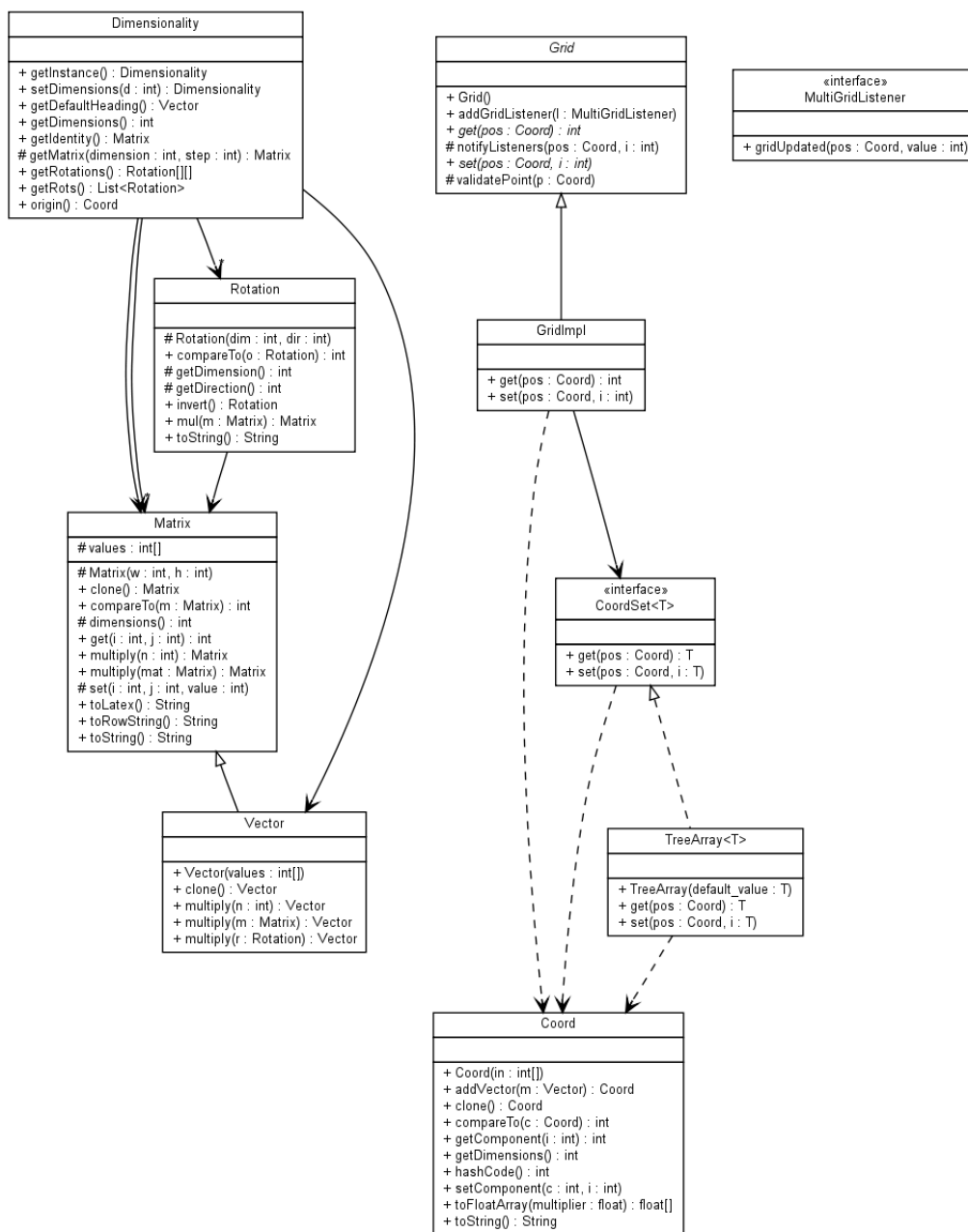
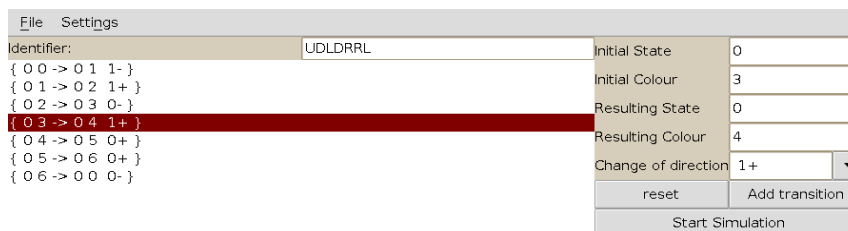
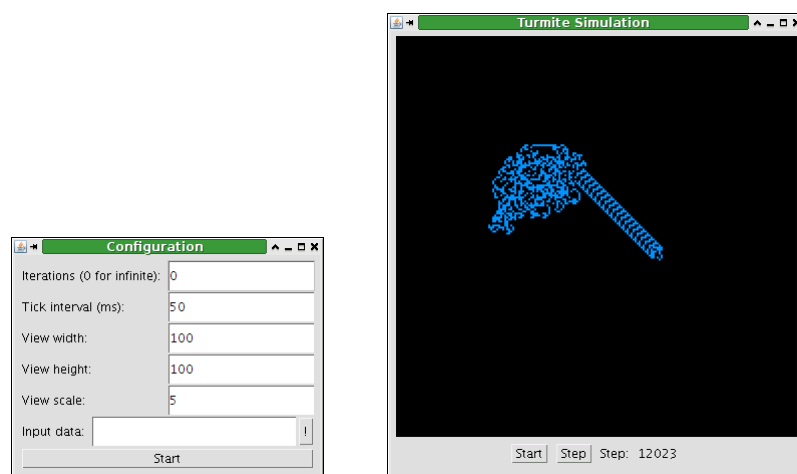


Figure E.1: Basic mathematical constructs



(a) definition window



(b) Configuration window

(c) Langton's ant running in a 2d view

Transition: { 0 1 -> 0 2 1+ }
 Heading: [-1][0][0]



(d) a 3d turmite

Figure E.2: The different stages of execution