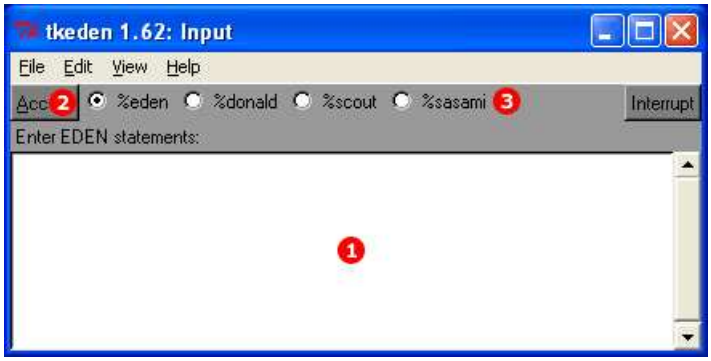
CS405 Introduction to Empirical Modelling
Lab 1 – Getting started with tkeden

This lab session is designed to give you a brief introduction to the tools used in the Empirical Modelling course.

1 Starting tkeden

We will be primarily using the tkeden tool in the lab sessions. On the DCS Linux machines you can start the tool by typing "`tkeden`" in a shell. When you start the tkeden tool it will look like this:



This window is called the 'input window'. The key elements of the interface are: the text entry box (1), the accept button (2) and the notation switches (3). Statements are entered into the text entry box (1) and then executed by clicking on the accept button (2). The environment enables the modeller to switch between notations on-the-fly by selecting the notation switches (3).

The EDEN, DoNaLD and SCOUT notations in the screenshot above are standard notations which are automatically loaded at start-up. EDEN is the Engine for DEfinitive Notations and is the primary notation for specifying dependencies. DoNaLD and SCOUT are two other commonly-used notations for line drawing and window layout respectively. Further notations can be added to the environment for specialist tasks.

Other useful features in the environment include the ability to view the history of your interaction ('View' → 'View history...') which contains all the commands typed into the input window. To cycle through previous commands in the input window text area, hold down 'Ctrl' and press the up or down arrow.

Basic guides to each of the standard notations are available from the Help menu.

2 Dependencies

A definitive notation is a notation (or language) that maintains dependencies between observables (or variables). As an example, if we enter the EDEN definition:

```
a is b + c;
```

we mean that observable `a` *is always* the sum of `b` and `c`. Whenever the value of `b` or `c` changes, the value of `a` is automatically recalculated. This is called a **dependency**. We can apply this to more complicated

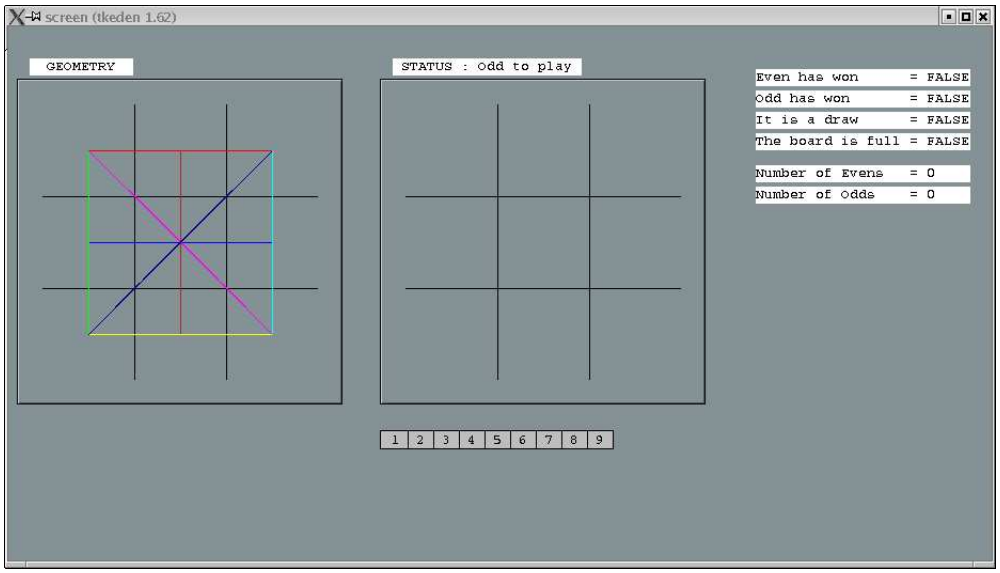


Figure 1: Tic-tac-toe model

formulas that involve functions and conditionals.

Another example:

```
d is (e == 1) ? f : g;
```

In this case, if `e` equals 1, then the value of `d` is the same as `f`, otherwise the value of `d` is the same as `g`. If either `e`, `f` or `g` change then `d` is re-evaluated.

3 Tic-tac-toe

This week we'll be getting started with tkeden and the EDEN notation by playing a variation on the game tic-tac-toe. However, the model is broken and requires some dependencies for the game to play correctly.

Start the tool 'tkeden' and load the model with the following commands:

```
cd /dcs/emp/empublic/projects/tictactoeRoe2002/
tkeden Run.e
```

If you cannot see the entire screen (see figure 1) then either resize the window with the mouse, or type "`windowSize = 200;`" into the input window and click on 'Accept'.

3.1 Rules of the game

This game is similar to tic-tac-toe (noughts and crosses) in that the object is to get three counters in a line. The difference is that instead of placing noughts and crosses the player places numbers in the grid. A player wins if they place the last counter in a line and the sum of the values in that line is equal to the target (which is 15).

Players take it in turns to place numbers in the grid. One player (Odd) can only play odd numbers and they always start. The other player (Even) can only play even numbers. They always go second. No number can be used more than once.

3.2 The model

Play with the model and have a go at the game. You should notice that the information windows on the right of the screen do not change. You can also cheat! There is no restriction on playing only odd or even numbers. There is no restriction on playing the same number twice.

The Task

The aim of this task is to add some dependencies to the model to get the information windows working correctly and place restrictions on which numbers can be played next. The variables that need to be defined have been set up in the model already. By defining them correctly you can get the model to work properly. These are the variables that need to be defined:

1. **n1valid, n2valid, n3valid, n4valid, n5valid, n6valid, n7valid, n8valid, n9valid** – defines whether the first, second, etc. number can be selected.
2. **nofeven** – defines the number of even numbers on the board.
3. **nofodd** – defines the number of odd numbers on the board.
4. **evenwon** – defines whether Even has won.
5. **oddwon** – defines whether Odd has won.
6. **full** – defines whether the board is full.
7. **draw** – defines whether the game is a draw.

To redefine one of these you need to type in a definition (e.g. `n1valid is TRUE;`) into the tkeden input window and click on the 'Accept' button.

Resources for the task

The existing model already has some definitions and functions present. In this exercise you may need to use some of the observables (variables) in this list:

1. `nofsquares` – is the number of squares in the grid (i.e. in this example 9).
2. `lastselection` – is the number that was last entered into the grid.
3. `target` – is the number that each player is trying to get in a line.
4. `player` – the person whose turn it is to put a number on the board.
5. `evenplayer` – a constant.
6. `oddplayer` – a constant.

To find out the current value and definition of an observable, you can use the query operator '?'. For example, to find out the current value of `nofsquares`, enter into the input window:

```
?nofsquares;
```

The definition and current value will be printed on the console:

```
nofsquares is allsquares#; /* current value of nofsquares is 9 */
nofsquares ~> [linesthrusingle]; /* nofsquares last changed by initialisation */
```

In addition to the observables above, you may need to use some of the following functions:

1. `not_used(n);` n is a number. This function will give you back a value of TRUE if the number n does not appear in the grid. This function will give you back a value of FALSE if the number n appears in the grid.

2. `countevens();` will return the number of squares in the grid that contain even numbers.

3. `countodds();` will return the number of squares in the grid that contain odd numbers.

4. `isodd(n);` n is a number. This function will give you back a value of TRUE if the number n is odd. The function will give you back a value of FALSE if the number n is even.

5. `iseven(n);` n is a number. This function will give you back a value of TRUE if the number n is even. The function will give you back a value of FALSE if the number n is odd.

6. `checklines ( target );` target is the number which each player is trying to get in a line. This function will give you back a value of TRUE if there is a line containing three numbers which adds up to the value target. If there is not a line then this function will return a value of FALSE.

4 Summary

You should by working through this lab sheet:

- know how to start tkeden, and execute definitive scripts;
- have an understanding of EDEN, and be able to modify and query EDEN variables;
- know how to write a dependency in EDEN to link two or more observables; and,

5 Revision history

The tic-tac-toe model was developed by Chris Roe as an extension of the original oxo model. The worksheet was adapted from Chris Roe's exercises from the EM 2002 course, and was revised by Antony Harfield for the CS405 course in 2006.