University of Edinburgh
School of Informatics

Web MENACE: A Links Demonstration of a Classic Learning Machine

4th Year Project Report
Computer Science

Samuel Corbett

April 3, 2009

Abstract: The Matchbox Educable Noughts-and-Crosses Engine, a 1959 invention of Biologist and Computer Scientist Donald Michie, was one of the earliest examples of reinforcement learning. Capable of incrementally learning to play noughts-and-crosses with a combination of matchboxes, beads and reinforcements, MENACE was by no means a miracle of artificial intelligence but was instead a proof of concept designed to educate Michie's AI-sceptic contemporaries. The task of this project was to create Web-MENACE: an appealing and interactive Links-based demonstration of Michie's original construction. In this report I describe how users may play MENACE, create and train their own machines, and then explore the contents of its matchboxes. I then test MENACE's performance in different scenarios and suggest and test potential improvements to the model.

Acknowledgements

Ian Stark for his suggestions, criticisms and encouragement, Sam Lindley for his technical assistance and Kylie Hill for being thoroughly great.

Contents

1. Introduction

The Matchbox Educable Noughts and Crosses Engine, MENACE for short, was the 1959 invention of Donald Michie. Created to prove incorrect a colleague's assertion that machines could never 'learn', MENACE used a combination of matchboxes, beads and reinforcements to play games of noughts-and-crosses, over time attaining a good standard of play. One of the earliest examples of a machine able to learn, it directly preceded the development of BOXES, the first working demonstration of reinforcement learning by computer. [15, 18]

This project's task was to create Web-MENACE: an interactive demonstration of MENACE using the Links programming language. Links is a strongly typed, strict, functional web-based language under development at the University of Edinburgh, designed to eliminate the 'impedance mismatch' problem by providing a single language for the three tiers of web development (web-browser, web-server and database) [9]. Secondary goals of the project included the creation of user-customised MENACEs and the exploration of MENACE's digital matchboxes, both past and present.

To achieve the primary task I have written a program that allows a user to play against a MENACE machine himself, or observe it playing one of of a set of predefined opponents that range in difficulty from extremely simple to quite tricky. As the user plays repeated games against MENACE he will see it adapt to his strategies, eventually becoming a well-rounded player.

I provide a facility for users to create, customise and train their own MENACE machines. Customisation is achieved by altering the reward scheme it uses and changing each matchbox's initial provision of beads. 'Training' a machine entails it quickly playing many games against computer opponents, to raise it to a more advanced level than in its initial state. Both the number of games and the machine's opponents can be user-specified.

The 'exploration' of MENACE, described in Chapter 4, allows a user to step through a game MENACE might face and observe both its current preferences and how they have been previously altered at each stage. The complete program is publicly available at `http://groups.inf.ed.ac.uk/links/examples/menace/index.links`.

In this report I first describe the historical background to MENACE and give a brief introduction to the features of Links. I then detail my implementation of Web-MENACE, noting implementation details and design choices in the process. In Chapter 5 I test MENACE's performance against various opponents and suggest how it could be adjusted to increase its efficacy.

2. Background

In this chapter I give the historical background to MENACE, describing its conception and original design and defining it in terms of a Markov Decision Process. I then give an introduction to Links, detailing its notable features and explaining how it differs from the most widely used web-programming languages.

2.1 MENACE

During World War II Donald Michie worked at Bletchley Park, notably aiding efforts to solve the 'Tunny' cipher by improving the Colossus computer. While there he befriended Alan Turing (being one of few of Turing's ability at chess [20]) and the pair, along with Jack Good, formed a discussion group around Turing's 'child machine' concept, in which Turing proposed constructing an intelligent machine by first building a general learning program and then training it as you would a child. By the end of the war Michie's interest in machine intelligence had been captured – he later commented:

> I resolved that I should spend my life on the pursuit of machine intelligence as soon as such an enterprise became feasible. [18]

The wait took fifteen years, during which Michie worked as a geneticist, creating chess-playing paper machines in his spare time. In 1959 Michie was challenged by a colleague that machines able to learn were, in principle, impossible. Since digital computers were still scarce his response was the construction of MENACE, a contraption of beads and matchboxes that could learn to play noughts-and-crosses. It won the bet with his colleague and an invitation from the US Office of Naval Research to visit Stanford.

At Stanford Michie adapted MENACE to create a general trial-and-error learner that became known as BOXES. It was the first ever working demonstration of reinforcement learning by machine and preceded the explosion of British interest in artificial intelligence that was only tempered by the 1973 publication of the Lighthill report and the subsequent cut in government funding for artificial intelligence research. [13, 18]

2.1.1 The MENACE Machine

The operation of MENACE is simple: A matchbox for each of the 287 distinct boards the machine might face (rotations and reflections taken into account – see

Figure 2.1: Symmetry and reflection reduce the number of boards MENACE must consider. Here the red Xs indicate potential moves and grey Xs indicate their ignorable symmetries.

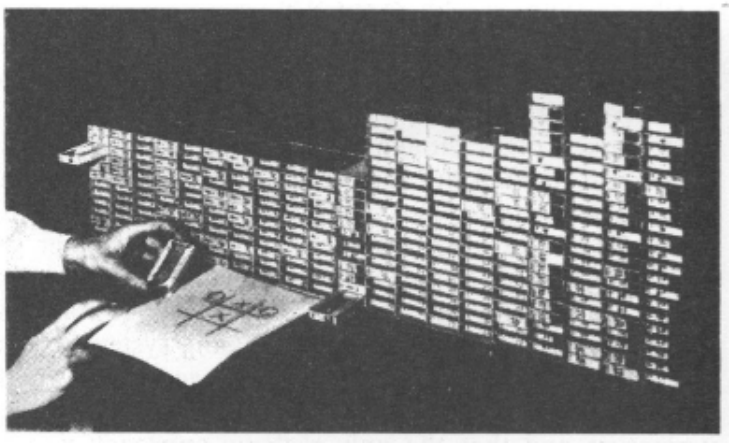


Figure 2.2: The Original MENACE Machine [15]. Used with permission.

Figure 2.1 for an example) contains a number of differently coloured beads corresponding to the unoccupied squares that might be played on that board. When it is MENACE's turn to play the human operator takes the board's corresponding box, gives it a shake and tilts it forward, making the beads fall into a V-shaped fence attached. The machine plays in the square corresponding to the colour of bead at the V's apex.

At the end of the game reinforcements are applied. If the machine has won, three beads of the appropriate colour are added to each of the matchboxes used. If it has drawn, one bead is added, and if it has lost it is 'punished' by removing the bead used. As the machine plays many games it will come to favour those moves that lead to winning positions. [15]

In Michie's first tournament with MENACE it abandoned all openings but the corner after seventeen games. By the twentieth game the machine was consis-

tently drawing so Michie began using 'unsound variations' of strategies to win, a useful tactic until it learnt to deal with these too. After the 220^{th} game the machine had beaten Michie eight times in ten games so Michie retired from the tournament. [12]

2.1.2 More formally

Formally, the game of noughts-and-crosses can be thought of as a finite Markov decision process (MDP). These are "the specification of a sequential decision problem for a fully observable environment with a Markovian transition model and additive rewards" [17]. Deconstructed term-by-term and related to noughts-and-crosses this means:

- That there are only a limited number of possible games;
- That the quality of a player is determined by the decisions it makes during a game;
- That a player will always be able to know the current state of the board (and all states that might follow);
- The probability of transitioning from one state to another is only dependent upon the *current* state – the moves played before reaching this board are irrelevant.
- The 'utility' of a particular move is the sum of rewards received. In MENACE's case this is simply the number of beads in the move's matchbox.

MDPs are defined by four components, given in terms of their definition for one player of a game of noughts-and-crosses:

- The *state space*, S: All boards the player could face.
- An *action space*, A: In this case actions are 'move in square x', where $x \in \{1..9\}$.
- A *transition model*, $P_a(s, s') = P(s'|s, a)$: The probability of moving to state s' given that the model is in state s and the player performs action a. The model should reflect how the opponent plays.
- A *reward function*: Player specific. In MENACE's case applied at the end of a game – +3 beads in a matchbox for a win, +1 bead for a draw, -1 beads for a loss.

MENACE can be thought of as providing a *policy* for this MDP, adjusted so that rewards are only distributed at the end of an iteration rather than after each step.

A policy $\pi(s)$ denotes the action recommended by π in state s and in MENACE's case represents shaking a matchbox and selecting the bead at the apex of the V.

The stochastic element of the opposing player means that each game will lead to a different state history, so policies are measured in terms of the expected utility of the state history. An optimal policy is one that maximises this utility. MENACE clearly does not fit this criteria when new, since it is effectively a random player. Whether the reinforcements to its matchboxes (adapting the policy) bring it closer to this maximum depend on the difficulty of the opponent it is playing: too easy and MENACE will settle for a solution that is 'good enough', too difficult and MENACE might never find a useful policy.

2.1.3 Related Work

Relatively few implementations of MENACE exist, at least publicly on the internet. Two implementations worthy of note are 'MENACE in C++' [21] and a version written in Visual Basic [19]. Both are interested in MENACE as something of a programming exercise, with the bonus of an engaging history. This project is apparently unique in its attempt to make MENACE visually understandable and explorable.

2.2 Links

As the Internet has matured and the number of people 'connected' has exploded there have been a glut of new technologies for web-programming and a sizeable increase in the number of people trying their hand. A novice is lured in by the promise of a simple Wordpress blog install [7] and is soon battling with PHP, HTML, cascading stylesheets (CSS) and probably Asynchronous Javascript and XML (AJAX) techniques too, in this new 'Web 2.0' age.

Similarly, professional developers are expected to be able to combine 'traditional' static content with personalised services that update in real-time, recommend items of interest based on user habits and combine interactive Flash based services like video and music streaming, all while making sure the service is scalable and doesn't overload the server.

The troubles encountered when attempting to link so many disparate technologies has been phrased as the 'impedance mismatch' problem [4] – it is difficult, especially in large systems, to ensure that each tier of a web-service receives the data it expects. For example, responding to an AJAX request to display information contained in a database in a client's browser could easily require three different formats for the data: the database → a format suitable for the server

language, the server language → JSON, for the AJAX library, and JSON → XML, to display in the browser. It is the programmer's responsibility to manage these requirements.

Links is a language designed to ease this problem by providing a single point of call for all tiers of a project. A program written in Links is translated into three components – Javascript, to run on the client, Links, to run on the server, and SQL, for communication with a database. Programs may move seamlessly between the three tiers as they execute.

In contrast with many languages used for web-programming, for example PHP and Ruby, Links is functional, has a strong-typing system and is strict. The type-safe abstractions such languages allow provide the grounds for Links to introduce some powerful and novel concepts to the field of web-programming, notably 'Formlets' and abstractions over query fragments, both features described below.

The following is a brief overview of the features of Links.

The client/server relationship One of the fundamental features of Links is its capability to transparently pass execution of a program between the client and the server. Functions can be annotated with `client` or `server` tags that guarantee the function will be be run on that tier (if the tag is omitted the function will be compiled for both).

Client-server calls are made as asynchronous `XMLHTTPRequests` and server-client calls are responses containing the result of the function call and a representation of the server's state, to be applied the next time the server is called. It is important to note that when a program is executing on the client the server maintains *no* state – it is in this sense that Links programs are scalable, since when the server is idle it consumes no resources [9].

Databases The third 'tier' of a Links application is the database. Run from the server, queries are executed with a list comprehension construct. Connecting to a database, formatting queries as SQL and managing results are all handled by Links transparently from the programmer. The use of databases is described in more detail later in this chapter.

XML Links includes XML as a native datatype and provides a set of primitives that, given an element of the active document and some XML, make the appropriate change to the underlying structure of the document. These functions include things like `appendChildren`, `replaceNode` and `removeNode`, and can only

be run from the client. Nodes in a document can be referenced with the function `getNodeById`.

For example, to append an item telling of MENACE's latest win to a list defined in a document as:

```
<ul id="result-list"></ul>
```

One can simply write:

```
appendChildren(
   <li>Menace wins!</li>,
   getNodeById("result-list")
)
```

If the XML has no enclosing element (the `li` tags in the example above) an XML 'forest' can be used instead, denoted by the tags `<#></#>`.

XML and Links code can be recursively nested within each other, allowing a degree of 'templating' – a generic structure can be created that inserts the results of various function calls to customise the appearance of the page.

Interactivity Links allows event handlers to be attached to page elements. These have the form:

```
l:onxxx="{ function }"
```

And are fired whenever `xxx` occurs. Examples include `click`, `mouseover` and `submit` (useful for altering the default action of a form). They are active as long as the element they are attached to remains in the Document Object Model (DOM, a language and platform independent model for representing XML and HTML documents).

Though they may be the start of any particular computation, handlers are most often used as the start of a chain of function calls to specifically respond to an action by the user, with visual feedback provided with an update to the DOM.

Handling events In the scope of each event handler is a special variable called `event`, accessible by special functions corresponding to the Yahoo! Web UI library [8]. These functions allow things like `getTarget`, returning the DomNode where the event occurred, and `getPageX`, returning the x coordinate of the event, relative to the top-left corner of the page.

Formlets Links introduces the concept of 'Formlets' for meaningful abstraction of forms. They allow the HTML representation of a form to be separated from

the supplied data's internal representation and may be combined and reused to create larger forms without losing any semantic meaning. They are an interesting concept but have not been used in this project. [10, 11]

The creation of MENACE has involved particular use of concurrency and databases, two aspects of Links I now explain in more detail.

2.2.1 Concurrency

Links makes forking a program into many threads of control particularly easy. A new process is spawned with the simple:

```
var pid = spawn { function(..) };
```

This will run `function` in a new process until it terminates (if ever). Processes have 'mailboxes' and can be sent messages with:

```
pid ! Message
```

Messages arrive in a first in, first out order, and are received either by calling `recv()` or, more usefully, by switching over the possibilities. For example, a process waiting for a human to make a move in a game of noughts-and-crosses might:

```
receive {
   case MovedSquare(index: Int) -> { .. }
   case Quit -> { .. }
}
```

This mailbox has the variant type `[| MovedSquare:Int | Quit |]` – if it is sent any other kind of message the program will simply not compile. The unknown case can be handled with the _ wildcard.

Links masks the fact that Javascript makes no provision for concurrent programs by inserting explicit context switches between processes, and eliminates the call stack in between these switches since it will only contain tail calls to which execution will never return [9].

Concurrency is of particular use when handling user interaction – one can spawn a process that waits for certain events, display a page with event handlers attached to the appropriate elements and have these handlers send the process messages when they fire, the process handling the event asynchronously to any other actions occurring on the page.

2.2.2 Database interaction

A major component of Links is its provision for database operations. Links supports MySQL, PostgreSQL and SQLite and permits a reduced subset of SQL expressions – the `select`, `insert`, `update` and `delete` operations. Connections to a database are parameterised by the specification of a username, password, server and port and they can be accessed with table-handles, which define the table name and its type, in the form of a record. For example, a table that tracks the results of a MENACE machine's games could be defined as:

```
table "games"
with (machine_id: Int, game_num: Int,
      opponent:   Int, result:   Char)
from (database "web_menace")
```

Retrieving data from a database occurs through list comprehensions that *must* be executing on the server. To get a list of all the opponents a MENACE machine has beaten using the table above one might:

```
for (game <-- games)
where (game.machine_id == some_id && game.result == 'W')
   [(opponent = game.opponent, game = game.game_num)]
}
```

The long arrow `<--` indicates the right hand side of the argument is a table and guarantees that the comprehension will execute as a *single* SQL query. In this case the query is likely to be equivalent to:

```
SELECT opponent, game_num
FROM   games
WHERE  games.machine_id == some_id
AND    games.result == 'W'
```

Queries must return a record of 'base' types (`Bool`, `Int`, `Char`, `Float`, `Xml` or `Database`).

The reduced subset of SQL supported by Links means that a query like:

```
length(for (x <-- table) where condition)
```

Cannot be translated into:

```
SELECT count(*) FROM table WHERE condition
```

Instead one must pull all items matching `condition` into a list in Links and run `length` on that.

As of version 0.5 Links supports higher-order queries – a query may be described according to an abstract condition that Links will not know until runtime, but Links guarantees at compile time that the query will execute as a single SQL statement. For example, to match a condition in the table `games`, one can write:

```
fun matching_games(condition) {
   for (game <-- games) where (condition(game))
       [records]
}
```

The `condition` could be a single function, it could be many complex conditions combined dynamically according to some user-defined input; if it compiles Links guarantees a single SQL query. It does this by "analysing the program to ensure that callers [..] never pass a function that Links cannot translate to SQL in the context of the query." [5] Such functions are generally those that are recursive, use 'wild' primitives (basic functions provided by Links that are known to be untranslatable to SQL), or that do not return a list of records with fields of base type.

3. Creating Web-MENACE

The aim of this project was to create Web-Menace: an appealing, interactive demonstration of Donald Michie's original Menace that runs in a web-browser and is written in Links. To that extent I have constructed a system that allows a user to play Menace himself, or have Menace play one of a number of predefined opponents. By playing many games the user can observe Menace learning to counter and combat the alternate strategies thrown its way.

Further goals of the project included the creation of arbitrary machines, a facility to explore Menace's innards and an implementation of Martin Gardner's Hexapawn [12], a simplified matchbox-learner that is perhaps easier to understand because of its smaller scope. Of these three goals I completed the first pair, eventually running out of time to create a worthy version of Hexapawn.

In this chapter I describe my implementation of the basic Menace machine, initially giving a high-level overview of its specification, before describing its implementation in Links. The creation of user-defined machines is described at the end of this chapter and encompasses alternate reward schemes, different initial provision of beads and 'training' machines by having them automatically play many games against other opponents.

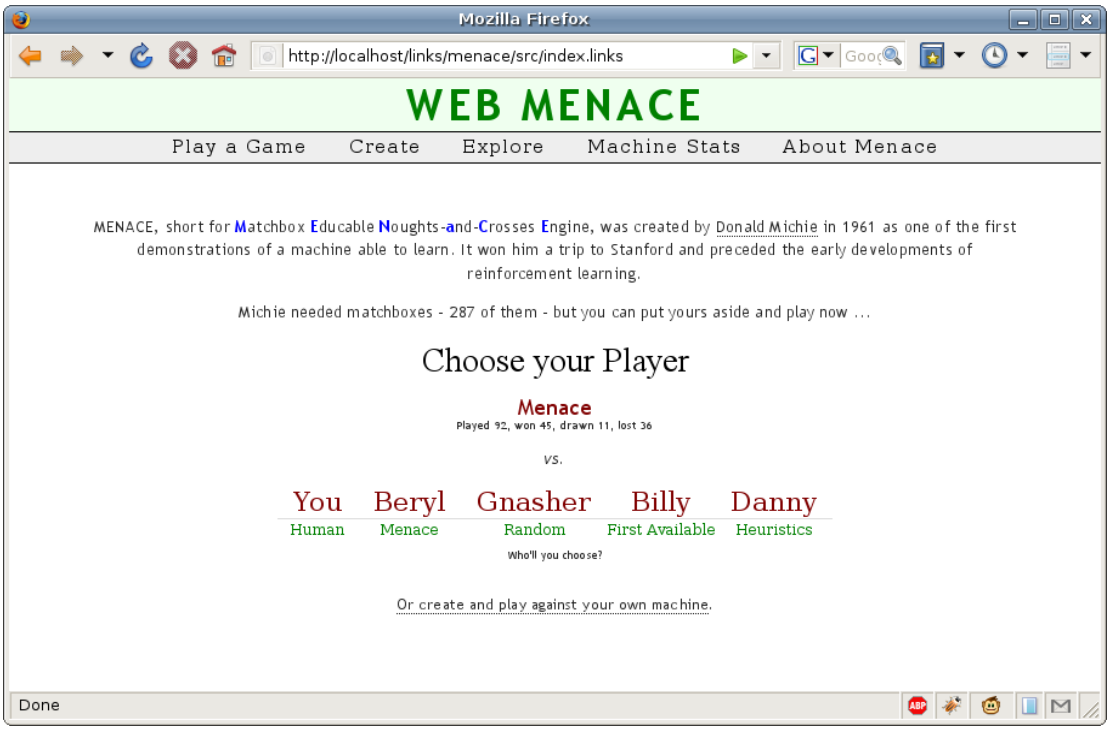
The 'exploration' of Menace's matchboxes is described in the next chapter. It includes the presentation of their current state and the visualisation of how the number of beads inside has been altered over the many games it plays.

I use a few items of terminology throughout this and later sections of this report:

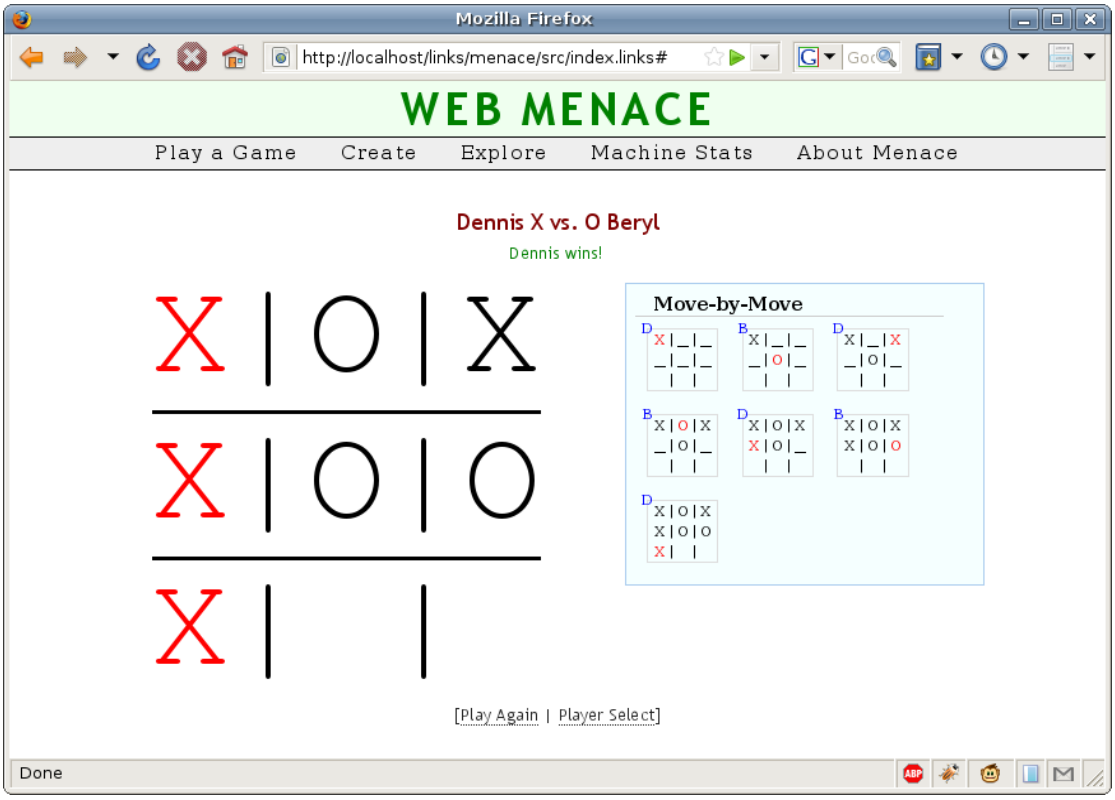
- Menace is Donald Michie's original construction.
- Web-Menace is the project being described.
- A **machine** is any computer-player.
- Menace has a number of **standard opponents** it can play and be trained against. They are described in Section 3.3.1.
- A Menace machine's **state** refers to its collection of matchboxes and the beads contained within.

3.1 A Brief Overview

There are five distinct sections to Web-Menace – playing a game, creating new machines, exploring Menace, viewing general machine stats (like games won,



(a) The introductory Web-MENACE page shows a brief introduction to MENACE and gives the user the choice of several opponents to play against Dennis.



(b) Dennis outwits Beryl. The winning line is highlighted in red and the progress of the game is shown step-by-step on the right.

Figure 3.1: The interface for playing games.

drawn and lost rather than beads in matchboxes) and an 'about' section describing the history and operation of MENACE.

Pictured in Figures 3.1(a) and 3.1(b) are screenshots of the first of these. Figure 3.1(a) is an introductory screen, giving a brief overview of MENACE and prominently displaying an option to play the 'standard' MENACE (nicknamed Dennis), a machine that follows Michie's original design, against either the user or one of the standard opponents.

Choosing one of the opponents immediately starts a game and replaces this introductory page with a large board to contain the current state of the game and a section that will be filled with smaller boards indicating each separate move as the game progresses. If the user chose one of the standard opponents to play MENACE then the game progresses independent of his actions. If however he chose to play MENACE himself, he takes his moves by clicking the square of the large board in which he wishes to play. Figure 3.1(b) is an example of how this all appears at the end of a game.

3.2 Program Setup

Though Web-MENACE is treated by the browser as a single page and runs under a single URL it was developed as several individual pages that were merged at the end of development. The advantage of merging comes when moving between sections that were otherwise independent – the new 'page' loads instantly rather than incurring a thirty-second delay as Links parses and type checks the file, and so forth. To achieve this, each item in the menu at the top of the page has an `onclick` handler attached that swaps the page's current content with that of the new section.

Links provides a very basic mechanism for code reuse by allowing the combination of `include` statements in a program with a preprocessor script, run before Links starts it work, that literally replaces the include statement with the contents of the requested file. Though a little ungainly this method allows a certain amount of refactoring and is undeniably better than there being no mechanism at all. It was used to share common constructs like type definitions between files and to keep logically distinct sections of the program apart.

3.2.1 The User Interface

Pictured in Figures 3.1(a) and 3.1(b) are the introductory Web-MENACE screen and the end of an example game. General details were already mentioned in Section 3.1, here I give a few implementation details.

The introductory screen Is mostly static HTML. Hovering the mouse over one of the opponents' names fires an event handler to display a short line indicating the number of games the machine has played and how many of them were won, lost and drawn. Clicking a name fires another event handler to begin a game of noughts-and-crosses between Dennis and the chosen opponent.

The Game Upon choosing an opponent the game is launched and the introductory page is replaced to show the game in action. On the left is a large board to show the current state of play, and on the right is a step-by-step list of the moves taken in the game, each latest move highlighted in red.

If the user chose one of the standard opponents to play MENACE, the large board is simply plain HTML. When the user plays MENACE himself, though, event handlers are attached to each square with actions for the `onmouseover`, `onmouseout` and `onclick` events. The `mouseover` and `mouseout` handlers simply display and hide the user's game character to give a clear indication of where the game thinks he is about to play if he clicks. Doing so sends a message containing the integer index of the square to the 'manager', described in Section 3.3.2.

3.2.2 Storing Data

Web-MENACE stores all information in a database. The following is an overview of the tables it contains.

Machines Tracks the attributes of each computer player. These include its name, its type, whether it was human created or is one of the standard machines and the number of games it has played, won and drawn (with the number lost inferred).

Game history A record of every game a machine plays is kept, listing the opponent and the result.

Rewards When a user creates his own MENACE machine (see Section 3.4), he has the option of supplying an alternate reward scheme. They are stored in this table and default to Donald Michie's original scheme of three for a win, one for a draw and less one for a loss.

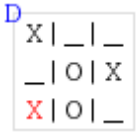


Figure 3.2: An example board

Initial bead provisions A second element of customisation afforded the user is to alter the number of beads provided for each potential move of a previously unencountered board. Values default to the scheme used by Michie, which follows the formula $\lfloor F/2 \rfloor$, where F indicates the number of unplayed squares on the board. For example, each move in Figure 3.2 would be given 2 beads.

Boards Each board MENACE encounters is stored with a unique identifier to avoid duplicating data in other tables. They are represented as a nine-character string with underscores to represent blank squares. For example, the board in Figure 3.2 would be represented as '`X___OXXO_`'.

Matchboxes Each MENACE machine represents its current 'state' (as described above) as many records in this table. A single record is appropriate to a single machine and represents the number of beads associated with a free square of a particular board. It is referenced whenever MENACE plays a move and when first initialised is set to the according value in the 'initial bead provisions' table.

Matchbox histories Complementing the table of current machine states is a second table to record every transition a matchbox passes through, combined with the result of each game in which it was used. From this one can read 'in the n^{th} game that MENACE A encountered board B it played in square C. The result was D and afterwards the matchbox contained E beads.'

3.3 Playing a Game

Playing a game is a matter of choosing an opponent for MENACE and then either taking turns as need be or watching its computer opponent play.

Web-MENACE defines a number of types to represent a game of noughts-and-crosses. A `Square` is an (index, filling character) tuple and a `Board` is a list of squares. Boards are defined in this manner, rather than simply as a list of characters, so that reflections and rotations are easily reversible when translating information from the database back to the context of the game.

Players (human and computer alike) are represented by the `PlayerInfo` type, effectively a container for the machine's information as stored in the database. The two competitors in a game are given type `GamePlayerInfo`, which holds the `PlayerInfo` information, an abbreviation to use for displaying status updates mid-game (the shortest non-identical subsequence of characters between the two machine's names) and information needed by MENACE to perform its reinforcements at the end of the game, wrapped in a `Maybe` type so that non-MENACEs can ignore it.

3.3.1 Computer Opponents

Web-MENACE provides four 'standard' opponents that are useful for training new machines and for observing how MENACE copes against different styles of play. They can all run on either the client or the server.

A second MENACE (Beryl) An alternate version of MENACE that plays according to Michie's original design, but as second player rather than first. When both machines are new and are repeatedly played against each other Beryl has a far harder time learning to playing well.

Heuristics (Danny) The 'heuristics' machine plays according to the following rules, given in order of preference:

1. Make a winning move;
2. Block the opponent from winning;
3. Play in the center square;
4. Play in the first available corner square (left-to-right, top-to-bottom);
5. Play in the first available remaining square.

Note that this machine is deterministic and is trivially beatable (see Figure 5.2(a) on page 36).

MENACE finds Danny a tough opponent, especially when 'fresh' and effectively plays randomly, when Danny tends to pummel MENACE into submission, MENACE only learning because it loses so many games that only a handful of squares in boards met early in play contain any beads.

Random (Gnasher) and First Available Square (Billy) The least interesting pair of opponents play exactly as their names suggest. Gnasher is useful

for a few initial rounds of training when creating a machine; it allows MENACE to explore possible moves and hopefully find some useful strategies without penalising it too heavily for simple mistakes.

Billy is useful for showing MENACE's limitations – it often wins after MENACE has played many games against 'better' opponents. That said, MENACE is generally quick to learn how beat it.

3.3.2 The Manager

Games are controlled by a 'manager', running in a separate process created at the start of any series of games between two opponents and receiving triggers from the user interface. When spawned it is provided with two variables of type `GamePlayerInfo`, indicating the participants in the game. Each turn it acts as follows:

1. Decide whose turn it is to play;
2. Have this player make a move:
    - If the player is human wait for a `HumanClick` message;
    - Otherwise switch on the player's type and call the appropriate function. MENACE's turns are described in more detail below.
3. Display the move;
4. Work out the new game state – one of `GameWon`, `GameDrawn` or `GameInProgress`.
    - If the game has finished, display an appropriate status message, highlight any winning lines in red and perform the end-game database updates described in Section 3.3.4.
    - Otherwise repeat this whole process.

I initially used three processes to control a game – one for the manager and one for each player, but eventually decided the program was simply clearer if the manager acted on behalf of the two players.

3.3.3 Menace's Turns

Described only briefly above, a typical MENACE move is rather more involved than a standard opponent or human move. It proceeds as follows:

1. If there is only one square remaining simply play there and ignore the following steps;

2. Get the board's ID from the database, rotating and reflecting as necessary. If the board has not been encountered before (so isn't in the database) add it and set default bead values, accounting for any symmetries the board may contain;

3. Get the machine's bead values for this board ID;

4. Choose a square with probability proportional to the number of beads it contains and the number of beads contained by all free squares;

5. Play in the square and make a note of the board id/square combination for end-game updates.

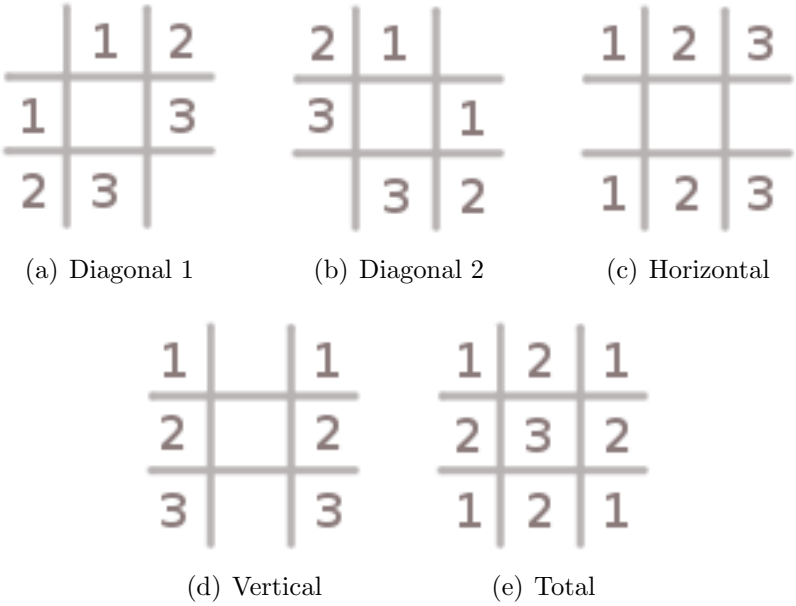
To calculate playable squares Web-MENACE first looks for any symmetries in a board by checking the four variants (a)–(d) shown in Figure 3.3. If one of the patterns is matched the symmetric squares are removed from consideration and the playable squares are then any remaining empty squares. If all patterns are matched the board is fully symmetric and only three squares need be considered, as shown in (e).

To decide its move in step 4, MENACE generates a random number between 1 and the number of beads in the matchbox, then repeatedly subtracts the number of beads kept for each square from this value until the result is zero or less. The square that took MENACE over the threshold is the one that is played. If a matchbox contains no beads then MENACE simply plays randomly, an important distinction from Michie's design which instead would have been understood as MENACE resigning from play.

For a long time during the development of Web-MENACE, this procedure used random numbers between 0 and the number of beads in the matchbox - 1. When there were no beads in the matchbox this created a random number between 0 and 0 – 0 – and the first square examined took MENACE to the threshold! By this simple oversight if a matchbox was empty MENACE would simpy play in the first available square. Its effect is mentioned in Section 5.1.1.

3.3.4 The End Game

At the end of the game a number of database updates are made. MENACE updates the matchboxes (or database rows) it used according to its reward scheme and it adds each new square/bead-count combination to its table tracking matchbox histories. A record is added to the table of game histories indicating who played who and which player won, and the table of machines is updated to reflect the two participant's new played, won and drawn values.



(a) Diagonal 1 (b) Diagonal 2 (c) Horizontal

(d) Vertical (e) Total

Figure 3.3: The four kinds of symmetry encounterable in a game of noughts-and-crosses and a fifth special case. MENACE uses these symmetries to reduce the number of possible moves it must consider. Example code to decide (a) is given in Figure 3.4.

3.4 Creating Other MENACEs

If a user does not want to play Dennis, the default MENACE, he can create his own machine. The only restriction is that its name is unique. To allow an element of customisation and the examination of MENACE's performance under differing circumstances, alternate reward schemes may be supplied and the default number of beads provided for each move may be altered.

Changing the initial number of beads MENACE assigns to each possible move on a board effectively alters the number of chances it is allowed to take with the move. Make it high and MENACE can explore a lot and is less likely to be punished for experimenting when playing a decent opponent, make it low and MENACE becomes far more conservative, quickly ignoring moves that have previously led it to a loss.

Rewards may be anything between -128 and 127 beads (though such extreme values only really make sense if the default number of beads in a matchbox is also made a lot larger) and there is no restriction that a win must have a positive reward and a loss a negative reward, so a machine that attempts to lose could also be created. Rewards are specified for each move of the game.

```
typename Square = (Int, Char);
typename Board  = [Square];

# True if all square pairings are equal
sig symm : (Board, [(Int, Int)]) ~> Bool
fun symm(board, indices) {
   all(fun ((x,y)) {
      sq_equal(select(board, x), select(board, y))
   }, indices)
}

sig sq_equal : (Square, Square) ~> Bool
fun sq_equal(s1, s2) {
   second(s1) == second(s2)
}

# Symmetry through top-left/bottom-right diagonal line
sig tlbr_symmetric : (Board) ~> Bool
fun tlbr_symmetric(board) {
   symm(board, [(1, 3), (2, 6), (5, 7)])
}
```

Figure 3.4: Function `tlbr_symmetric` returns true if the board is symmetric as shown in Figure 3.3(a). Checking for other symmetries proceeds in a similar manner.

The create-a-machine page also allows the user to play any other machine, or play any two machines against themselves.

3.4.1 Training a machine

New machines may be 'trained' by playing many games against other computer opponents chosen by the user. The games are all played on the server and feedback about the machine's performance is provided every twenty games with a list of wins, draws and losses, and a chart graphing this information (in the form described in Section 4.4). Any number of games may take place against any combination of opponents – the choice is the user's. Ten games takes roughly four and a half seconds.

This feature is especially useful for quickly examining MENACE's performance given different customisations and was used heavily for the tests in Chapter 5.

Figure 3.5 shows the interface for training a MENACE machine. Opponents to play are displayed in a list monitored by a process tracking the ids of the elements containing opponent information, so that information can be collated when the user has finished. Clicking the 'Add another' button sends a message to this process, adding and tracking another list item. The 'Remove' link sends the manager a message containing the ID of the list item to delete from the DOM and remove from those being tracked.

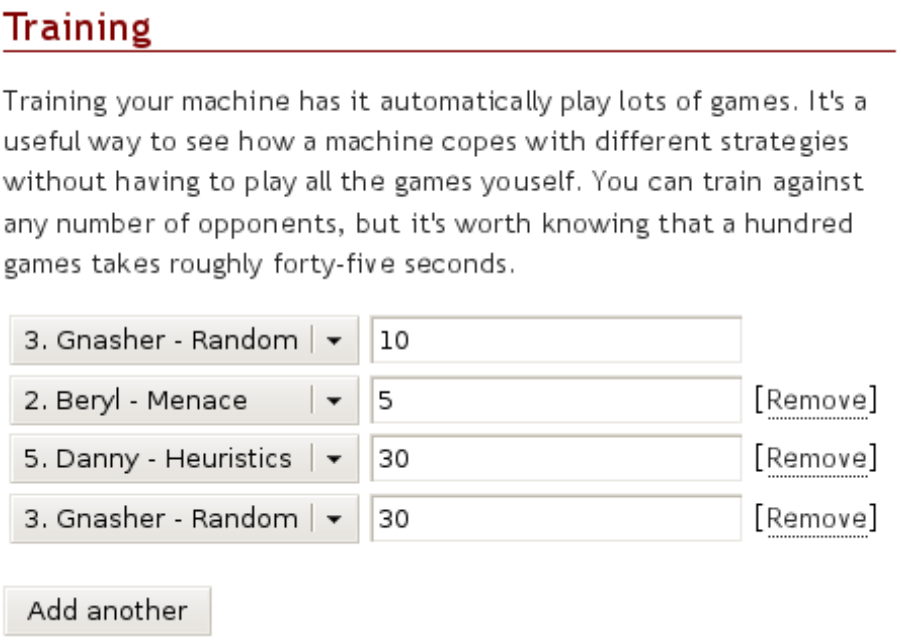


Figure 3.5: The interface for training a MENACE machine.

4. Exploring MENACE

This chapter describes how a user of Web-MENACE may explore a MENACE machine's matchboxes and come to an understanding of how such a simple creation learns to play noughts-and-crosses.

I have focused on two aspects of MENACE – representing the current state of its matchboxes and giving an illustration of how they have been adjusted through the many games it plays. They are combined on screen to view simultaneously as a user steps through the game.

Also provided is a tool for viewing individual machine statistics. These include the number of games a machine has played, won, lost and drawn, a graph of its progress and its 'favourite' and most 'disliked' opponents, judged as the opponents against which it has had the largest number of wins and defeats respectively.

4.1 An Overview

To explore MENACE's matchboxes the user is presented with a screen as pictured in Figure 4.1. Its premise is to allow a user to step through a game of noughts-and-crosses between either MENACE and a human or two MENACEs, at each move presenting the squares MENACE may decide between for its move and providing an easily interpretable guide to how likely the machine is to choose between each one, as well as how these likelihoods have changed though many games.

Clicking one of the possibilities is treated as MENACE moving in that square and loads information for the next stage of the game. Stepping through to the end of a game presents the user with information on how MENACE would have been rewarded had it really played the match.

The charts shown are created with the Google Chart API, a handy service whose use I now describe.

4.2 Charts

A lot of the information tracked by Web-MENACE is excellent for displaying as a chart – for instance, in such a format it is easy to see when a machine has played well, or has started storing many beads in a particular matchbox. However, since Links is a very young language there are few options for tasks that stretch beyond

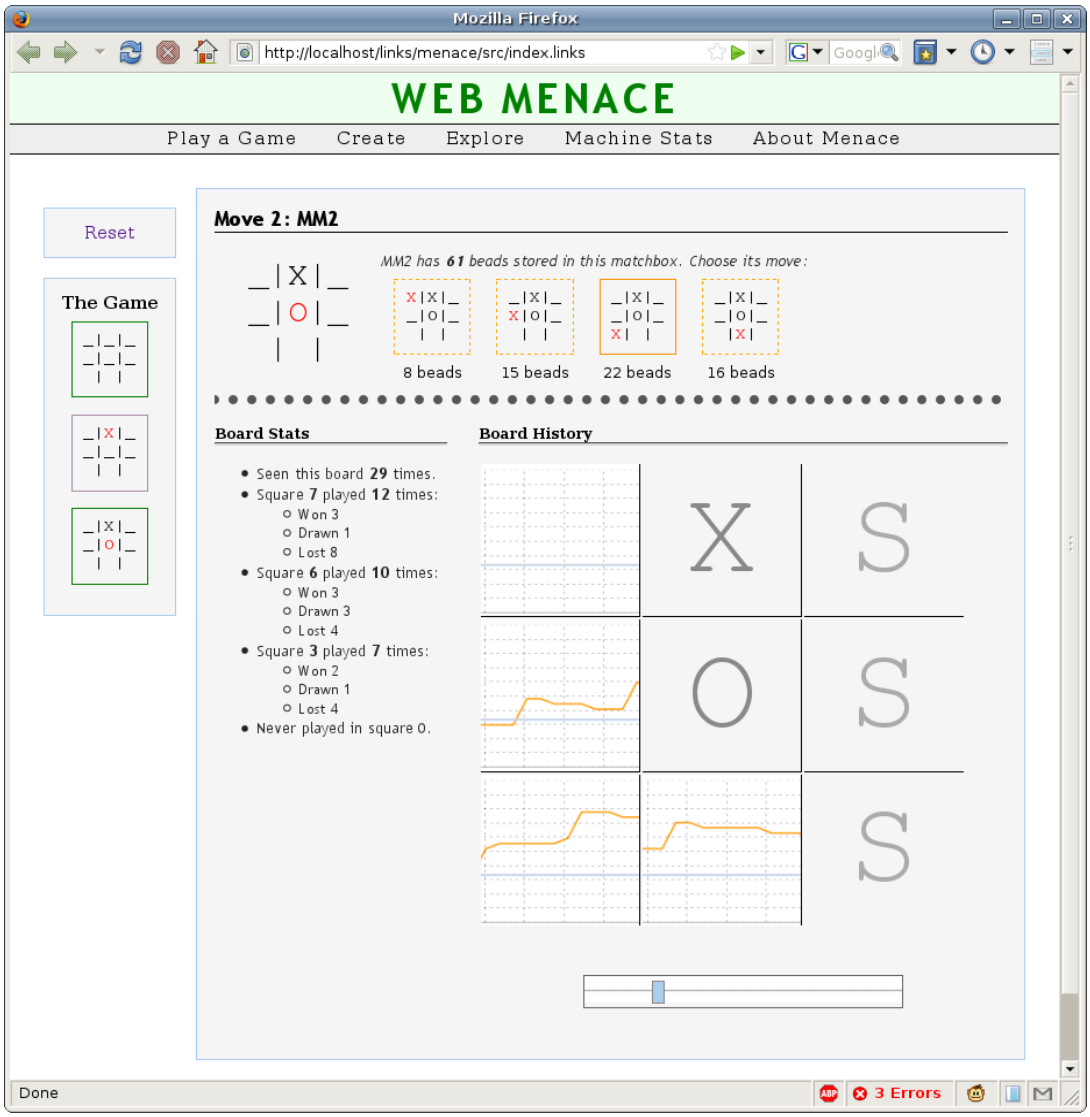


Figure 4.1: The interface for exploring MENACE's matchboxes.

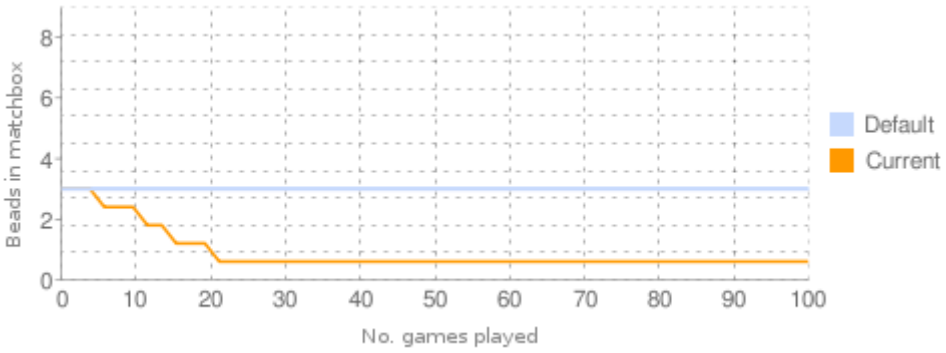


Figure 4.2: The y-axis implies there are approximately 0.8 beads in this matchbox – there are in fact zero.

its current reaches. To create charts I instead used the Google Chart API [3], a service that dynamically creates PNG-encoded images given data encoded in a URL. The choice of an external service was part because Links has no way to call external programs from the server, so creating and displaying charts with Gnuplot [2] was not an option, and part for its simplicity – the program simply needs to provide the data; all image processing is offloaded.

Many different kinds of chart can be created and there are tenfold more customisation settings, but my use of the service was entirely satisfied by the simple line-chart. Parameters like scale, axes and colours can be set as extra arguments in the URL.

The service proved extremely useful, albeit at the cost of a loss of fine-grained control. One particular quirk was the placement of axis labels – they would often imply different information to that specified by the data. See Figure 4.2 for an example.

The sheer amount of data contained in some graphs (for example, performance histories of machines that have played thousands of games) meant that the URLs generated began to be far longer than browsers can handle. Tests indicated the maximum URL length allowed by the API to be 2,076 characters, though there is no official confirmation of this by Google.[1]

To circumvent the problem, each function creating a chart has to specify a cut-off point along with the data. If the length of the data is larger than this cut-off point then the items will be sampled at a reduced granularity (every $(\texttt{length(data)}/\texttt{cut-off}) + 1$ items). Since the charts created are particularly linear this works well at reducing the length of large charts without losing a lot of detail.

[1] As a marker, Internet Explorer can handle URLs up to 2,083 characters long [6].

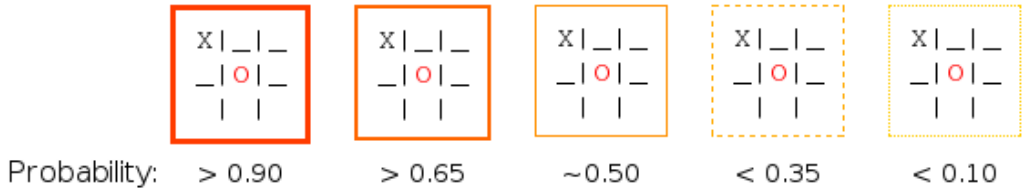


Figure 4.3: Border weights emphasise how likely the move is to be made.

4.3 Current and historical preferences

Web-MENACE provides two ways to examine a MENACE machine, giving information about the current state of its matchboxes and a visualisation of how its preferences have changed over the games it has played.

4.3.1 Current preferences

Showing MENACE's current preferences is simply a matter of consulting the 'matchboxes' table it uses when playing a game. If MENACE has not encountered the board before it is added to the database and default beads are set.

To emphasise MENACE's preferences its choices are displayed with a border relating to the likelihood of the square being played. The borders range from thick-red, meaning a probability greater than 0.9, to a very thin dotted yellow, meaning a probability smaller than 0.1.

Figure 4.3 shows the range of possibilities, and Figure 4.4 shows how this information is presented on screen.

4.3.2 Historical preferences

To compare how MENACE's preference for particular squares has altered as it has played many games I display a table like that shown in Figure 4.5. An entry in the table is either an `X` or a `O`, if the square has already been played, an `S` if the square is symmetric with another and can therefore be ignored, or a chart indicating how the number of beads kept for square has varied over time.

The procedure to create them is as follows:

1. Initialise n lists with the default number of beads for this board (where n is the number of playable squares);

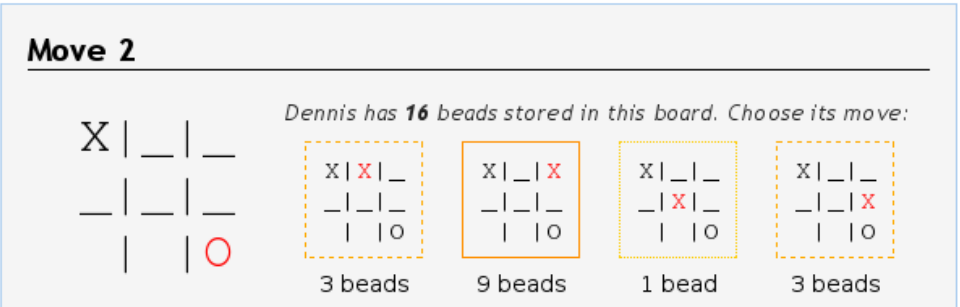


Figure 4.4: An example MENACE's current preferences. Note that though there are seven free squares, three are symmetric with others and can be discounted.

2. Fetch all square/bead information for this machine and board combination from the matchbox histories table and form a list of (square, bead) tuples;
3. Take the head of the list and add the number of beads to the front of the appropriate individual square list;
4. Repeat the head items of each of the other square lists;
5. Repeat steps 3 and 4 until there are no more items to process;
6. Reduce the granularity of long lists and finally reverse.

To ensure each chart has the same scale the maximum number of beads in any matchbox is tracked and the value used as the scale for all charts.

This all occurs on the server to take advantage of its more efficient list processing.[2]

4.3.3 User Interface

These two sets of information are combined in a page that lets a user step through a game of noughts-and-crosses, examining a chosen MENACE machine's current and past preferences at each move.

Given a board the machine might face, its choices and preferences towards them are displayed as in Figure 4.4, with event handlers attached to each possibility

[2]Challenged on this assertion I ran two tests: one created the URLs for three charts for a board that had been encountered 500 times on the client, the second did the same on the server. Each test was run five times. The average time to completion on the client was just over twelve seconds, the average for the server was just over three.

The client is hindered by the creation of new lists in steps 3 and 4 of the process – lists on the client are implemented as Javascript arrays, so appending an item to a list on the client actually creates an entirely new structure and sorting it runs in the order of $n^2log(n)$ operations rather than $nlog(n)$. The same operations on the server can take advantage of efficient pointer operations.

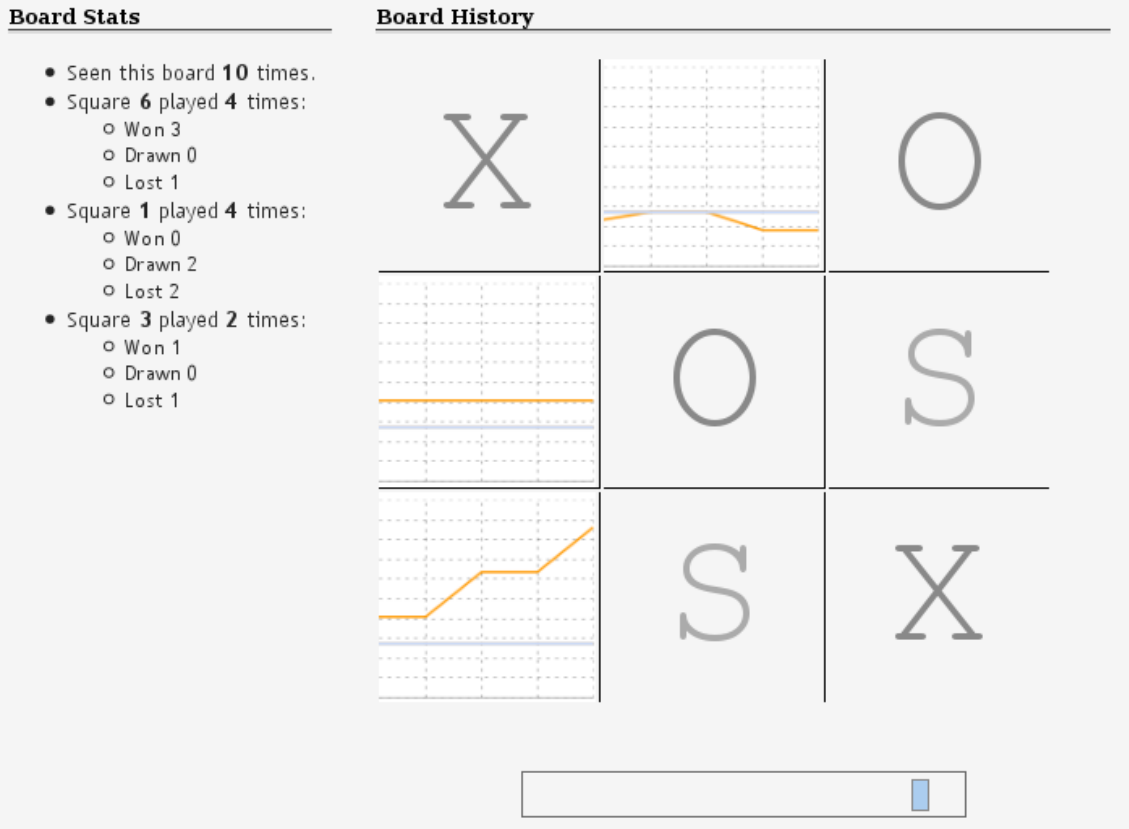


Figure 4.5: A MENACE machine's bead history for a board encountered midway though a game. Each `S` indicates a square symmetric with another. Hovering over an image displays it in full and the slider at the bottom moves all images in tandem. This MENACE clearly favours playing in the bottom left-hand corner.

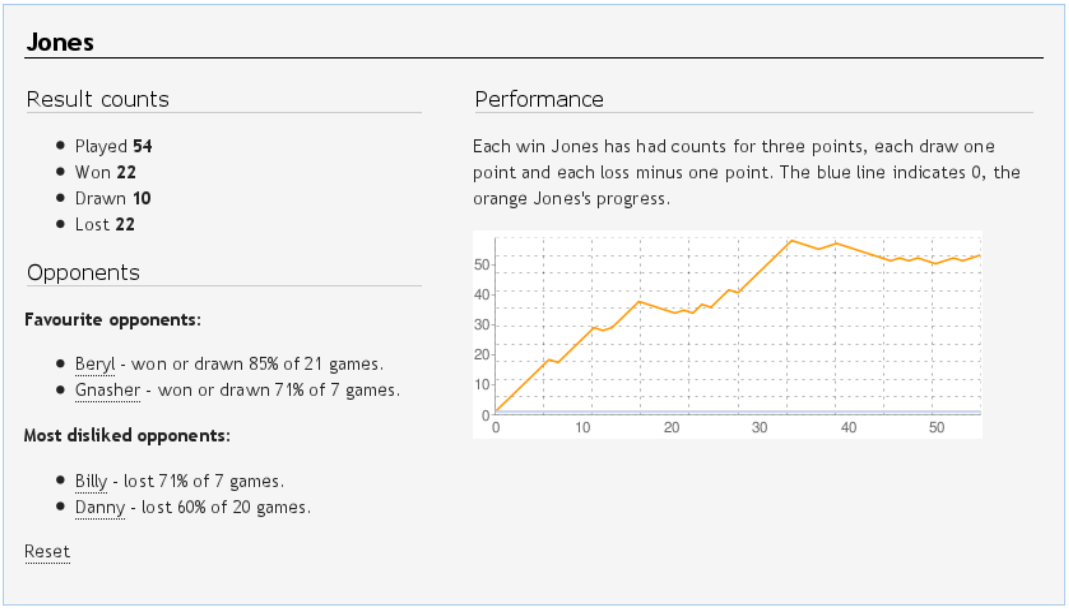


Figure 4.6: The presentation of machine statistics.

set to register a click as a signal to 'make this move'. Historical preferences are as pictured in Figure 4.5 and only show a limited section of the chart – hovering over a single image displays it in full, while a 'slider' beneath the table moves all the images in tandem, hopefully giving a good impression of how the balance of beads in the entire matchbox changes rather than simply how an individual bead-count has been affected.

As a final aid to understanding, a number of statistics about the board are displayed. These include the number of times the machine has encountered the board during play and the number of times the machine has won, lost and drawn after playing each possibility.

If the user steps through to the end of a game its result is displayed and he is informed of how MENACE would be rewarded had just played the game.

4.4 Individual machine statistics

Web-MENACE also provides more general information about individual machines, with data on the number of games played, won, drawn and lost, the machine's 'favourite' and 'most disliked' opponents (classed as the top three win/draw and loss percentages respectively) and a performance chart graphing its progress presented as shown in Figure 4.6.

5. Observations and Statistics

Donald Michie's first tournament with MENACE spanned 220 games played over two eight-hour sessions. Adopting a 'best strategy', he found the machine quickly settled into a safe drawing line of play in response, so began using theoretically unsound tactics to lure the machine into unfamiliar territory, at the risk of losing the game. These paid off until after 150 games, when the machine coped with any tactic used against it. [14] Michie continued attempting to outdo the machine, commenting:

> The machine was by then exploiting unsound variations with increasing acumen, so that I would have done better to return to 'best strategy' and put up with an endless series of draws. [..] It is likely, however, that my judgement was sometimes impaired by fatigue.

After the 220^{th} game MENACE had beaten Michie eight times in ten so Michie retired from the tournament, considering the machine to have mastered noughts-and-crosses.

Fortunately, fifty years of Moore's law[1] and the advent of personal computing mean such exercises can now be completed in a matter of seconds rather than days. In this chapter I describe a number of tests performed on the original MENACE and detail its successes and why it can be prone to failure, before suggesting and testing potential improvements to the machine, in the form of different reward schemes and alternate initial provision of beads to matchboxes. At the end of the chapter I give a few comments on how I found working with Links.

5.1 Testing MENACE

All the tests described here were performed through the 'create-a-machine' part of Web-MENACE (see Section 3.4). The result graphs were generated with a Python script that fetched information from the database, formatted it appropriately and then displayed it with Gnuplot.

[1] Moore's law suggests that the number of transistors that can be placed on an integrated circuit doubles roughly every two years. It was first stated in 1965 [16] and is expected to continue well into the next decade.

5.1.1 Michie's MENACE

Described below is a set of five tests against opponents of different abilities. Each set incorporated several MENACEs, always playing first and with Michie's standard reward scheme, whose results were averaged to account for any statistical anomalies. The results are presented in Table 5.2 and include the standard deviation to give an indication of consistency. The tests were as follows:

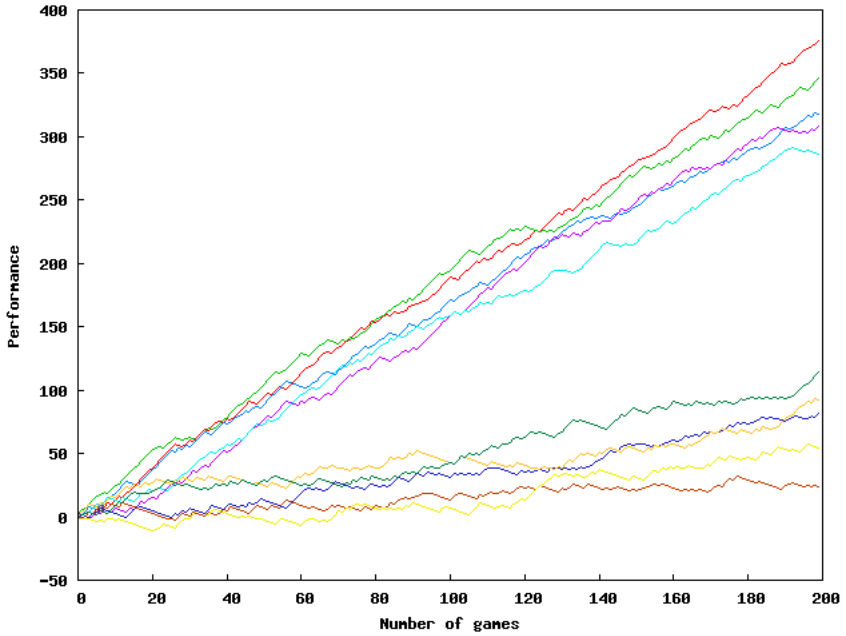
- **A**: Five lots of 50 games against Billy;
- **B**: Five lots of 75 games against Gnasher;
- **C**: Ten lots of 200 games against Danny;
- **D**: Ten lots of 100 games against Gnasher followed by 150 games against Danny;
- **E**: Five lots of 200 games against another new MENACE.

Table 5.1: The performance of Michie's MENACE

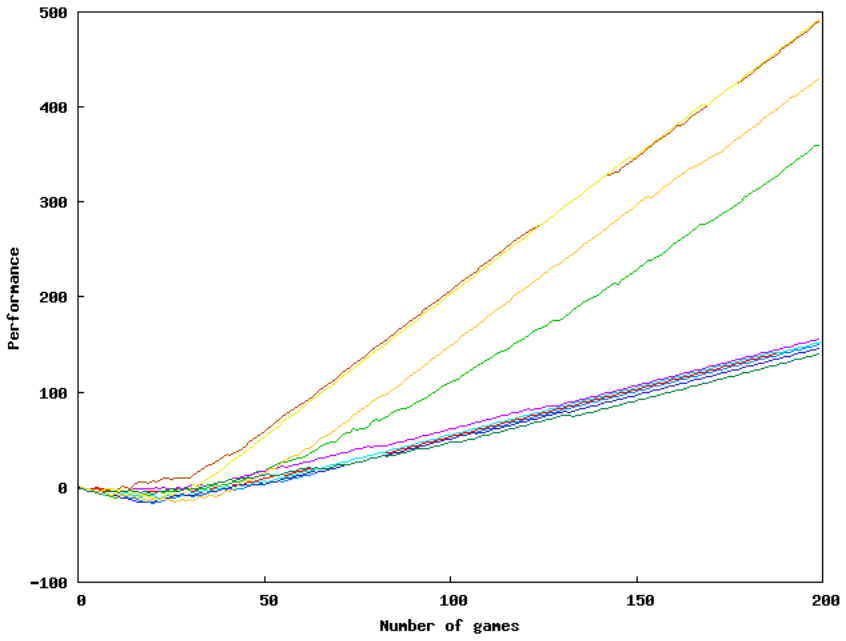
		Average			Standard Deviation		
Test	Played	Won	Drawn	Lost	Won	Drawn	Lost
A	50	41.2	2.8	6.0	2.32	2.32	1.67
B	75	51.8	6.8	16.4	3.66	3.00	2.58
C	200	53.8	123.4	23.0	68.66	67.7	1.79
D	350	71.90	115.0	64.7	15.35	16.4	8.43
E	200	111.6	35.4	53.2	12.22	9.35	12.97

Consistency MENACE is clearly very consistent – I was surprised by quite how low many of the standard deviations are. The results suggest it is efficient at finding a strategy and exploiting it as much as possible, especially against simple opponents like Billy and Gnasher.

The low standard deviation for games lost against Danny (Test C) is interesting, especially when viewed with the information presented in Figure 5.1(b), which shows the performance graphs for the machines involved. On average, MENACE loses 23 games, always at the beginning of the series. If, in this time, the machine discovers a strategy for beating Danny it will exploit it until the end, otherwise it will almost always play to a draw. Experimentation indicated that in these first twenty games Danny beats MENACE so often that its first matchbox empties. From that point on it plays randomly and will adopt the first strategy it discovers with a positive reward, be it the good fortune of a win or the likelier draw.

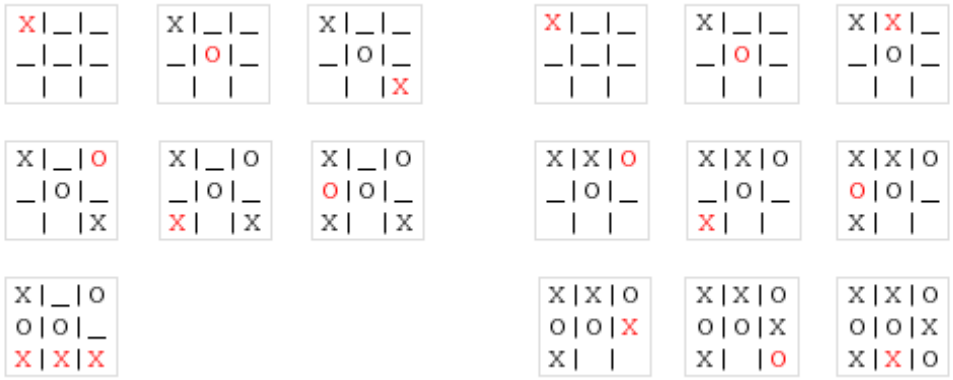


(a) Results of MENACE vs. MENACE. The top set of lines are the machines that took the first turn, the bottom set played second.



(b) Results for MENACE vs. Danny. Four of the ten machines learned the strategy to beat him, the rest had to settle for draws. The first 20 or so games are paramount to MENACE's success.

Figure 5.1: MENACE is very consistent against its opponents. 'Performance' is graphed as per Michie's original reward scheme of 3 points for a game won, 1 for a game drawn and -1 for a game lost.



(a) How to beat Danny

(b) MENACE instead learns to draw

Figure 5.2: Danny, the 'heuristics' player (playing O in both games here), is trivially beatable but MENACE has difficulty discovering his weakness and instead plays for a draw.

Trouble against Danny The results against Danny (Tests C and D) give an indication of how MENACE struggles against better opponents, yet its exertions feel rather futile to a knowledgeable observer since it is quickly apparent that Danny is trivially beatable. Figure 5.2(a) gives a possible strategy one could use against the machine.

Simple probability would suggest that a brand new MENACE – one with no changes made to its matchboxes – should chance upon the strategy to beat Danny just over once in every hundred games.[2] MENACE's trouble is that it is only provided with twelve beads for the first matchbox and is far likelier to run out of beads than chance upon this strategy. Finding a drawing strategy is a lot likelier, but we have already seen how MENACE finds a strategy and exploits it as much as possible, so it ends up rewarding draws so much it doesn't give itself a chance to explore further. Both issues are discussed further below.

In Section 3.3.3 I mentioned an incorrect use of random numbers that resulted in MENACE playing in the first available square if its matchbox for a board was empty. This manifested itself against Danny, when a combination of his skill and determinism meant MENACE would always fluke a drawing strategy after thirty moves. The thirty moves gave MENACE time to lose the twelve beads in its first matchbox and the twelve beads in its second matchbox (so it would always start a game by playing in the first two squares of the board) and eight games to discover a drawing strategy from that more constrained position. This sequence of play

[2] Three choices of move for the first board, four for the second, three for the third and three for the fourth: $1/3 * 1/4 * 1/3 * 1/3 = 1/108$

is illustrated in Figure 5.2(b).

The advantage of playing first Test E indicates the advantage MENACE has from playing first – on average in a series of games between two MENACEs the first player wins half the games and draws the next quarter. The difference is shown most clearly in Figure 5.1(a). The worst performer of the second players was beaten so many times it had few beads in its matchboxes and essentially played randomly. One would expect results to eventually tail off into a series of draws.

5.1.2 Adjusting MENACE

Alternate reward schemes Michie's reward scheme was designed for simplicity. He reasoned that if a machine lost after playing its fourth move, the move was poor without qualification and there would be no point ever repeating it, so giving each possible play at MENACE's fourth move a single bead meant that removing it would banish the move forever. Working backwards from the fourth move he decided a machine should have two tries for moves encountered on its third turn, three for those encountered on its second move and four for those on its first.

If winning is paramount and MENACE is playing an opponent with an inherent weakness, like Danny, one might consider only rewarding wins. However, such a strategy would make MENACE a far slower learner against an intelligent opponent, and since noughts-and-crosses is a zero-sum game – if both players play perfectly the game can only be drawn – if MENACE were played against an opponent with perfect strategy it would be unable to adjust its matchboxes, forever remaining a random player.

More interesting would be to weight reinforcements according to the stage of the game a matchbox was used – the first move of a game clearly has less influence over the result than the last, and rewards could be weighted appropriately.

Rewards based on a succession of games It often felt that MENACE was too ready to reward itself for 'yet another' draw, and sometimes did so to the point that it was extremely unlikely to play a move clearly advantageous to an observer. A sensible tweak would be to reinforce identical results either up to a limit, after which no adjustments are made, or to use diminishing returns, so that successive games add slightly fewer beads to the matchbox.

Rewarding a machine for drawing after a succession of wins can be considered rather nonsensical – it would be wiser to reward a draw after a series of losses

instead. Similarly, a win after several losses should be reinforced a lot more strongly than one after many other wins.

Bead provision Play MENACE against any machine of a decent standard and it is quickly apparent that the initial provision of four beads for each of the three playable squares of the first move of the game is thoroughly insufficient – once it has lost twelve games the matchbox is empty and MENACE resorts to playing randomly.

Michie himself observed this, writing:

> It turned out that the allotment of only twelve beads to the first box [..] gave the machine scarcely sufficient resources to withstand repeated discouragements in the early stages of play against an expert. [14]

Such scenarios suggest a lighter penalty for losing would assist MENACE.

Testing these suggestions I used the features for customising new MENACE machines to test the suggestions of alternate reward schemes and bead provisions.

Rewards were set to:

- Five beads for all moves of a win;
- No beads for the first move for a draw, one bead for the rest;
- Less two beads for the third move of a loss, less one for the rest.

My reasoning being that a win is always worth investigating so should be rewarded highly, the first move of a draw counts so little towards the eventual result that it should not be rewarded at all, and that MENACE often lost games because of a poor choice of its third move, so that should be penalised more.

Initial bead provisions were set to 12 for the first matchbox, 8 for the second, 5 for the third and 1 for the fourth, to give the machine plenty of time to explore and avoid the twenty-game limit encountered by the standard MENACE.

A set of ten machines were tested with 200 games each against Danny. The results are presented in Table 5.2, with those from Test C above included for comparison.

There is clearly an improvement to MENACE's performance, likely because of the larger number of beads it is allocated for its opening moves. It is worth noting that only four of the 'standard design' machines managed to win any games at all, while *every* tweaked MENACE won at least one game (the lowest number won was 5, the highest 132).

Table 5.2: The tweaked MENACE's performance against Danny.

		Average			Standard Deviation		
MENACE	Played	Won	Drawn	Lost	Won	Drawn	Lost
Standard	200	53.8	123.4	23.0	68.66	67.7	1.79
Tweaked	200	86.0	59.8	54.2	41.13	32.14	9.53

5.1.3 In Summary

As I developed MENACE it often appeared rather clumsy – it would regularly surprise myself and others by giving the impression of being a competent player before hopelessly losing to a straight line across the bottom of the board.

I feel the tests described in this section show MENACE in a different light. It is effective at finding strategies to cope with the tactics of its opponents and with a little tweaking can often find winning plays too. Of the adjustments suggested I believe those involving rewards based on a past succession of games would find most success. It would be an interesting addition to Web-MENACE.

5.2 Working with Links

Since Links is a very young language I encountered several bugs and mishaps through the development of Web-MENACE. Often these were trivial oversights, like not parsing negative numbers sent to the server, and in all cases the Links team were very quick to assist. Here I give a few comments on particular issues I faced.

Learning Perhaps the biggest obstacle to learning Links was the lack of documentation available. There is a 'Quick help' page on the Links website [5] that proved extremely useful but at times was incomplete or gave outdated information – from reading it I began to use the `l:name` attribute to identify elements of forms, only to be told later that it shouldn't be used since the concept was flawed.

Debugging A particular frustration of Links is the difficulty of debugging programs. The compilation of client-side code into a continuation passing style renders it practically unintelligible, and error messages are often vague or their guidance misplaced. One example I encountered late in development simply read "Variable 'name' does not refer to a declaration." When `name` is a variable used many times in 3,500 lines of code this isn't as helpful as it could be! The Firefox

extension Firebug [1], a tool with excellent facilities for working with Javascript, is an invaluable aid to the Links programmer.

Performance Difficulty getting to grips with the language aside, the single biggest issue I faced was that of speed. When Links runs a program it must parse it, infer types, convert it to an intermediate representation (IR) and then interpret the appropriate part of this IR. Since the client persists it need only ever do this once, but as Links keeps no state on the server (to keep programs scalable) it must do this *every* time there is a context switch. When the program is small this happens fairly quickly, but attempt it with anything large and the program becomes very, very slow. In the case of Web-Menace this was particularly apparent when Menace was playing a move – trips to the server and back took up to eight seconds!

The Links solution is to precompile the program. This caches the IR, so the process does not need to occur on every trip to the server, and made a drastic improvement to Web-Menace's performance, reducing the time for a Menace move by over a factor of twenty.

Precompiling does not solve all speed issues – Links must still compile and type the program when it is first run, and this is no quick process itself, taking in the region of thirty-five seconds to display the individual 'Play and create a Menace' page, and up to three minutes to display the combined program. Such slow loading made testing individual portions of programs rather arduous and sometimes intensely frustrating.

Program Structure Combining each part of Web-Menace – playing, creating, exploring, statistics and history – into one file certainly makes using the program a lot faster once it has loaded, but at the cost of breaking the browser's 'back' button, something many human-computer interaction experts would consider a major flaw. Though the design is my choice I feel my hand was forced by the extremely long loading times that hinder quickly switching between pages.

Finally Once these issues were surmounted and Links' features and quirks grasped, working in Links could really be quite fun. Towards the end of the year I found prototyping new features quick, elegant, and easily integratable with the rest of the Web-Menace program and it should be noted that charts and CSS styles aside, everything presented by the Web-Menace program was written in the language.

6. Conclusions

MENACE is clearly no wizard at noughts-and-crosses – it struggles against intelligent opponents and is often content with draws against those simpler. However to knock it in such a fashion is to miss the point of its conception: Donald Michie created the machine in 1959, a time when many were still convinced computers would never be able to beat humans at chess, or perhaps even play the game, while others were imagining machines would be assisting every facet of life within two decades. Michie's reason for creating MENACE was to provide a simple example of a learning machine to his artificial intelligence-sceptic friends; it was not intended as a revolutionary breakthrough in machine learning.

I believe the construction of Web-MENACE demonstrates these features well. It is an intuitive, visually attractive demonstration of the original MENACE that allows users to play the machine and observe how it copes with varying tactics while explaining the processes used by Michie, giving a clear indication of when MENACE is successful and when it is at the limit of its competence.

Further achievements of the project include the capability for users to create and train their own machines, with provision made for customisation by allowing alternate reward schemes and different initial allocation of beads to matchboxes. These features were used extensively for the tests on MENACE described in Chapter 5 and assisted some of the suggestions for small improvements to the machine, described later in the same chapter.

There is plenty here that could be extended for future work. Some of the suggested improvements that Web-MENACE does not provide for, like reward schemes based on a past history of results, would doubtless have large effect on MENACE's capabilities. The matchbox model could also be extended to other games, for example noughts-and-crosses on a larger scale, or to miniature draughts, and its performance assessed there. For a while an implementation of Martin Gardner's Hexapawn [12], a matchbox learner on a smaller scale, was one of the project objectives but time did not permit its inclusion. I believe it would be an interesting addition to any program attempting to demonstrate the workings of a rudimentary trial-and-error machine.

Using Links proved an interesting experience and I look forward to seeing how its development progresses. Its management of the three tiers of web-programming felt logical and transparent, its provision for concurrency was enjoyable and thoroughly intuitive, and its database tools prove a powerful and accessible tool for the web-programmer, especially when combined with its support for query abstraction. Should support for other basic SQL operators like '`count`' and '`group by`' be included this will be even more apparent.

Bibliography

[1] Firebug. An extension for Firefox to aid web-development. Available from `http://getfirebug.com/`.

[2] Gnuplot. A portable command-line driven interactive data and function plotting utility. `http://www.gnuplot.info/`.

[3] Google Chart API. `http://code.google.com/apis/chart/`.

[4] The Links Homepage. `http://groups.inf.ed.ac.uk/links/`.

[5] *Links Syntax*. Website, retrieved 10/03/2009, `http://groups.inf.ed.ac.uk/links/quick-help.html`.

[6] Maximum URL length is 2,083 characters in Internet Explorer. Microsoft help page, retrieved 14/03/2009 `http://support.microsoft.com/kb/q208427/`.

[7] *Wordpress*. A popular blog engine. `http://www.wordpress.org`.

[8] *The Yahoo! User Iinterface Library (YUI)*. Website, retrieved 10/03/2009, `http://developer.yahoo.com/yui/`.

[9] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. Links: Web programming without tiers. 2006.

[10] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. The essence of form abstraction. In *Sixth Asian Symposium on Programming Languages and Systems*, 2008.

[11] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. An idiom's guide to formlets. Technical report, University of Edinburgh, 2008.

[12] Martin Gardner. *Further Mathematical Diversions*, pages 90–102. Penguin Books, 1969.

[13] James Lighthill. Artificial Intelligence: A General Survey. *Artificial Intelligence: A Paper Symposium*, 1973.

[14] Donald Michie. Trial and Error. *Penguin Science Survey, Part 2*, 1961.

[15] Donald Michie. Experiments on the mechanization of machine learning. Part 1: Characterization of the model and its parameters. *The Computer Journal*, 6:232–236, 1963.

[16] Gordon Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 1965. Available from ftp://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf.

[17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach, Second Edition*, chapter 17, pages 614–618. Prentice Hall, 2003.

[18] BCS Computer Conservation Society. Recollections of Early AI in Britain: 1942-1965. Video, 2002. Available from http://www.aiai.ed.ac.uk/events/ccs2002/.

[19] Adit Software. A MENACE Simulation. Website, retrieved 12/03/2009 http://www.adit.co.uk/html/menace_simulation.html.

[20] The Times. Professor Donald Michie. July 2007. Webpage, retrieved 12/03/2009 http://www.timesonline.co.uk/tol/comment/obituaries/article2061886.ece.

[21] Cheknokov Yuriy. Matchbox Educable Noughts and Crosses Engine (MENACE) in C++, 11 2007. http://www.codeproject.com/KB/cpp/ccross.aspx.