The Machine Intelligence Hex Project

Stephan K. Chalup*, Drew Mellor, and Fran Rosamond
School of Electrical Engineering and Computer Science
The University of Newcastle, Australia

Technical Report

21. June 2005

Abstract

Hex is a challenging strategy board game for two players. To enhance students' progress in acquiring understanding and practical experience with complex machine intelligence and programming concepts we developed the Machine Intelligence Hex (MIHex) project. The associated undergraduate student assignment is about designing and implementing Hex players and evaluating them in an automated tournament of all programs developed by the class. This article surveys educational aspects of the MIHex project. Additionally, fundamental techniques for game programming as well as specific concepts for Hex board evaluation are reviewed. The MIHex game server and possibilities of tournament organisation are described. We summarise and discuss our experiences from running the MIHex project assignment over four consecutive years. The impact on student motivation and learning benefits are evaluated using questionnaires and interviews.

*Corresponding author: Dr. Stephan Chalup, School of Electrical Engineering and Computer Science, The University of Newcastle, Callaghan, 2308, Australia, phone: +61 2 492 16080, fax: +49 89 1488 235410, email: chalup@cs.newcastle.edu.au

Table of Contents

1 Introduction

Machine intelligence for games has a long tradition. Computer programs that play strategy board games like Chess belong to the first major achievements mentioned in the history of artificial intelligence (AI) and computer science (Shannon, 1950; Turing et al., 1953). Game programs can demonstrate the state of the art in AI because the performance of the programs can directly be compared with that of humans. Therefore, the design of programs for strategy games has become a topic of central importance for AI education which is often addressed by courses on machine intelligence (Russell and Norvig, 1995, 2003).

Over the years artificial players for many non-trivial games have been developed. Some of them became famous because the computer program finally was able to challenge and beat the best human players (cf. Epstein et al., 1993; Levinson, 1996). The Checkers program CHINOOK (Schaeffer and Lake, 1996; Schaeffer, 1997; Fogel, 2002) was the first official man-machine world champion in any game in 1994. In the meantime programs have been developed which are able to beat the best human players for several other games as well, including Othello (Buro, 1994, 1999, 2002), Chess (Baxter et al., 2000; Hsu, 2002; Campbell et al., 2002; Newborn, 2003), and Backgammon (Tesauro, 1994, 2002; Pollack and Blair, 1998). However, there still exist strategy board games where humans outperform computer programs. These include, for example, the Asian game of Go (Burmeister and Wiles, 1995; Bouzy and Cazenave, 2001) and the game of Hex (Browne, 2000, 2004). Both games have a similar character; that is, they are pattern oriented and if a game tree is employed it has an extremely high branching factor.

The educational project presented in this article uses the game of Hex for an undergraduate programming challenge. It was part of a course on machine intelligence which was an elective within the degree programmes of Bachelor of Computer Science and Bachelor of Software Engineering at the University of Newcastle, Australia.

Our aim was to design an exciting assignment which would motivate the class and allow the students to gain practical experience on how 'real' AI works and what it takes to get it running. The assignment comes at the beginning of the third year after the students have completed their basic education in programming, data structures and algorithms. The Hex assignment is one of the first major programming challenges for the students where non-trivial concepts are involved. It is an example of *project based learning* (Heckendorn, 2002; Moursund, 2003) and offers an opportunity for students to develop and evaluate their skills individually or as members of a team.[1]

An additional motivational factor was that the programs were evaluated in a final class tournament and the programs that ranked the highest in the competition received the highest marks.

Considerable work was involved initially in setting up the assignment. Funding and resources had to be found to prepare the software, the labs and the

[1]The assignment was set as a team assignment in the first year, as optional team or individual assignment in the second and third year, and as an assignment where students work individually in the fourth year we were running the project.

associated web pages. A tutor was employed to maintain the server and run the tournaments over one week at the end of the semester. In hindsight, after four years of running the MIHex assignment, we evaluate our efforts: Was it worth it? What was the impact the assignment had on the course? How did students cope with the assignment and what did they gain from it? Can the assignment be regarded as a significant learning experience (Fink, 2003) for the students?

This article provides some background information on the game of Hex in section 2 followed by an explanation of the MIHex server in section 3. An overview about the MIHex project assignment from an educational perspective is discussed in section 4. The history and concepts of game programming in general are addressed in section 5. In order to help other lecturers run a similar assignment, Hex-specific methods are covered in section 6. Finally, educational outcomes and perspectives are discussed in section 7.

2 The Game of Hex

Hex is a two-person zero-sum board game with perfect information (Browne, 2000; Even and Tarjan, 1976; Gale, 1979; Gardner, 1959; Lagarias and Sleator, 1999; Reisch, 1981). It was invented by two mathematicians, in 1942 by Piet Hein and independently again in 1948 by John Nash (Milnor, 2002). Hex first gained popularity through an article by Gardner (1959). Each player is assigned two opposite edges of the rhombus-shaped Hex board (see Figure 1). Typical board sizes are between 7×7 and 22×22. There are only two types of pieces – *black* pieces for the first player and *white* pieces for the second player. Pieces are put alternately in the board's hexagonal shaped cells. The winner is the first player to connect his or her two edges of the board by a connected chain of his or her pieces (see Figure 1). A detailed description and history of the game can be found in the books of Browne (2000, 2004).

Hex is a challenging task for machine intelligence for several reasons:

- The rules of Hex are simple but a lot of experience and talent is required to become a good player.
- In a typical Hex position on a (11×11)–board, almost three times as many moves are possible than in Chess. Therefore standard game tree search takes a long time and cannot search very deeply.
- Standard board evaluation based on counting material, as used in Chess, is not applicable in Hex. Therefore, a Hex specific evaluation function or other techniques are required.
- Hex is a pattern oriented game similar to the game of Go. Both games are hard for computers to play. Research on Hex can be regarded as a step towards finding a good artificial player for the game of Go.

In Hex there is no draw. According to Gale (1979), this fact, which is also known as the Hex Theorem, is equivalent to the well-known Brower Fixed-Point Theorem. A general winning strategy is only known for small boards

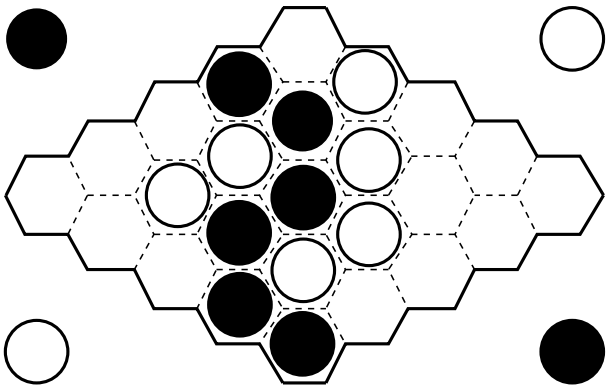


Figure 1: A (5×5)–board with a winning chain for *black*.

up to size 9×9 (Yang et al., 2003; Yang, 2004; Hayward et al, 2003). By using a *strategy stealing argument* it can be shown that a winning strategy must exist for the first player.[2] To weaken a potential advantage of the first player, the *swap rule* can be applied. It allows the second player to take the first move of the first player. If the swap rule is allowed, the first player would usually not start with a strong move because it would then be very likely that the second player applies the swap rule. Hex with swap-rule is called *competitive Hex* and is a second player win.

There are a number of theoretical results on Hex. For example, it has been shown that the problem of determining which player has a winning strategy is PSPACE complete (cf. Reisch, 1981; Even and Tarjan, 1976). On boards of equal size the state space complexity and decision complexity can be compared with that of Go. On a size n board the branching factor of the game tree is $O(n^2)$. For more information on theoretical results, see for example (Browne, 2000; Gale, 1979; Gardner, 1959; Lagarias and Sleator, 1999).

3 The MIHex System

MIHex is a tool that automates routine aspects of the development and evaluation of Hex playing programs. The principal design objective was to allow students or researchers to concentrate on the AI of game programming. A second objective was to restrict the design of the Hex programs as little as possible. The MIHex software is planned to be released at the sourceforge repository,[3] however it will not be supported after release.

The architecture of the MIHex system is shown in Figure 2. It consists of a central server written in the programming language Java, an arbitrary number of client modules each implementing a particular Hex strategy, and

[2] It was noted by John Nash in 1949 (cf. Berlecamp et al., 2001): if there were a winning strategy for the second player then the first player could make an arbitrary move and then apply (steal) the strategy of the second player. An additional move is never a disadvantage in Hex. Therefore both players would have a winning strategy which would be a contradiction to the Hex theorem.

[3] `http://sourceforge.net/projects/mihex/`

signals for server-client communication. Below we discuss each of these three components in turn.

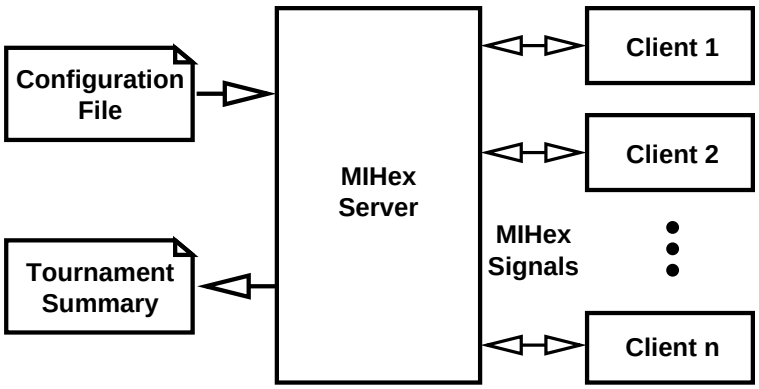


Figure 2: The MIHex architecture.

3.1 Server

The MIHex server is primarily responsible for refereeing, tournament coordination, and producing tournament summaries. Refereeing involves coordinating a match of Hex between two client modules by prompting them with the appropriate signals. The clients must respond before a configurable timeout period expires otherwise they are disqualified and lose the match. Responding with an invalid move will also result in disqualification.

Tournament coordination involves playing a suite of matches between pairings of clients registered with the server. Holding a tournament allows different strategies and implementations to be compared and evaluated. Currently the tournament summary (see appendix A) includes the overall rankings for each player, the win-loss breakdown for each pairing, and the move history for each game played. Configurable options for the server include the size of the board (4–14), the number of games to play per pairing, and the type of game (standard Hex or Misère Hex (Lagarias and Sleator, 1999)). Additionally, games can be displayed as they progress, either graphically or in a text based console.

3.2 Clients

The clients are stand-alone modules that encapsulate the strategy of Hex playing programs. A small library of two software classes are provided to facilitate client implementation, a `Board` class and a `Cell` class. The classes take care of routine Hex operations like maintaining the current Hex position, returning the set of adjacent cells for a given cell, and so on. Additionally, two types of sample clients, a random client and a proxy client, are included with MIHex to aid evaluation and debugging during the client development process. The random client, as its name suggests, makes random moves with a uniform distribution. It is useful because it generates moves quickly and can be used as the strategic low water mark against which to

Signal	*Description*
	server to client
`reset`	Notifies the client that a new game is about to start, client should do any necessary initialisation
`open`	Request the client to make an opening move
`swap`	Notifies the client that their opponent has swapped the first move, and requests the client's next move
`move`	Notifies the client of the co-ordinates of opponent's move, and requests the client's next move
`win`	Game over, the client has won
`loss`	Game over, the client has lost
`finish`	The tournament has finished, the client should clean up and exit
	client to server
`swap`	Notify the server that the client has made the swap move
`move`	Notify the server of the co-ordinates of client's move
`quit`	Resign the game

Table 1: The MIHex signals.

benchmark clients. The proxy client obtains moves from a human player via a graphical or text based interface. It is valuable when it is necessary to know how a client responds to a specific situation, or alternatively it can be used just as a way of playing a game against a MIHex client module.

3.3 Signals

Communication between the server and the clients is via a set of MIHex signals (see Table 1). The signals are based on a socket layer so that client implementation is language independent, providing the implementation language has a socket handling library. For example, a client implementing a rule based approach can be written in Prolog; alternatively a client using a game tree search approach might be coded in C++ or Java.

4 MIHex as Undergraduate Student Assignment

Over four years (2001-2004) the students of our undergraduate Machine Intelligence course COMP3330 were given the assignment to implement a Hex client player for the MIHex system. Each year the assignment was released during the first half of semester one[4] and had to be completed at the end of the one semester course. After submitting their player implementation in the final week students had access to a measure of their players' performance in the form of a ranking in a round robin tournament of all artificial players developed by the class.

[4]In Australia the academic year has two fourteen-week semesters. Semester one typically starts towards the end of February and runs until the exams start in mid June.

In order to obtain feedback on educational aspects as well as to evaluate student understandings of the Hex project, we asked the students to complete before and after questionnaires. In their final project reports students were asked to provide an explanation of which concepts motivated their agent design, advantages/disadvantages for the Hex tournaments, and a description of the algorithms used. Formal interviews were conducted in 2004 with several students and with the lab demonstrator before and after the project. Additional feedback came from informal conversations with students and staff and the university wide student evaluation of courses and teaching in 2001-2004. Typically over 90% of the students in COMP3330/6380 were male. Our evaluation treated all students equally and did not take minority groups separately into account.

The basic assignment has remained the same over four years: implement a player. However, support for students did change. During the first two years, students had to implement and code much of the software themselves. In years three and four, students were provided with libraries of classes to represent the Hex board, Hex cells, clients to send messages to the server, and so on. These client stubs and library of pre-built classes get a lot of the nuts and bolts out of the way and allow the students to focus on the AI. The result was that players were much better in 2003 and 2004. More advanced strategies and more complex algorithms were implemented as the students could devote more of their time to developing a good evaluation function. Having access to a library gave some students a false sense of security; they should be cautioned that there is still a considerable amount of work they need to do.

Year	*2001*	*2002*	*2003*	*2004*
Class size	40	65	41	25
Weighting (marks)	30%	16%	20%	15%
Prize offered	no	yes	no	no
Teamwork allowed	yes	yes	yes	no
Number of Hex tutorials	≥ 3	3	3	2
Client stubs provided	no	no	yes	yes

Table 2: Overview about the MIHex project during 2001-2004.

There was additional supportive motivation in 2002 where we provided prizes for outstanding players. We planned inter-university competitions with Hex teams from other universities. There was a good amount of enthusiasm and the reputation of the Hex assignment spread among the students.

The support changed in other ways as well. During the first three years, there was considerable tutorial assistance both during dedicated labs and in the availability of tutors to answer questions. The first three years had a sequence of at least three tutorials per term dedicated to the Hex project: one at the beginning to introduce the project, another near the middle and another near the end. The second two tutorials assisted with problems that

came up once students had started their work. During year four, tutorials were cut back due to financial considerations, and there were only two tutorials. There were also fewer lab demonstrators available to answer questions.

In year four, we decided to make Hex an individual project rather than a team project. Previously, one person on a team would write the report while the others did the programming. These are both very important but different skills. With teams, students enjoyed developing a collective vision, and Hex inspires this. The individual project is more difficult in that one person must come up with both the ideas and the presentation. However, on the final evaluation most students reported that they appreciated the "...actual experience of getting something like this coded and working". Almost everyone said they learned new programming skills. Students responded to: What did you like most about the Hex assignment?, with statements such as, "Coding it. Testing out things and seeing them work. Two-distance seemed impossible at first, but bit by bit it worked." and "Challenge of pretty difficult programming." and "When it didn't crash when attempting to run it and when it actually started playing half decently."

In 2001 the Hex assignment was the main practical assignment of the course and therefore it had a weighting of 30% of the final course mark (Table 2). In the following years we decreased the weighting because there was at least one other large practical assignment to complete as well.

Hex is a self-contained project: software engineering skills are necessary, and time management is important. The project is not completely focused on an algorithm: students compete with others. When asked if assignments like this should be part of future courses in machine intelligence, student response was overwhelmingly positive.

5 Basic Concepts of Strategy Board Game Programming

A number of fundamental concepts, techniques, and tasks are common for many game programs. These include the evaluation function, game tree search, and the organisation of game tournaments.

5.1 Evaluation Function

One of the most important tasks in designing an artificial player for games like Hex is to develop a good evaluation function for a board position. An evaluation function can be represented by a mapping from the space $\mathcal{S}$ of board positions to the interval $[-1, 1]$ of real numbers.

$$J : \mathcal{S} \longrightarrow [-1, 1], s \mapsto J(s) = \text{value of position } s$$

The value of position s, $J(s)$, is an estimate of the likelihood that s leads to a win. $J(s)$ should be +1 if s is a winning position or $J(s)$ = -1 if s is a loosing position. The other values $J(s)$ are all within the open interval (-1, 1) where higher values are assigned to better positions and lower values to

weaker positions. For a draw, (which does not apply for Hex (see Section 2)) $J(s) = 0$.

A standard approach for designing an evaluation function is to define a list of board features and take a linear combination of them. Typical features for Chess could take the amount of material which is left on the board into account (Russell and Norvig, 2003). However, there are many possible ways to define features and for each game these features can be very different. Often a human expert player can provide the best advice in this matter.

5.2 Searching Game Trees

The success of most computer Chess programs is built on brute force search. The program *Deep Thought*, for example, was able to search 750,000 positions per second and in 1988 was the first Chess machine to beat a grandmaster in tournament play (Hsu et al., 1990; Campbell et al., 2002). In 1997, *Deep Blue II* (a successor of *Deep Thought*) was able to beat the human World Champion Gary Kasparov under normal tournament conditions. *Deep Blue II* has a large searching capacity and a special Chess chip for the move generator, the evaluation function, and search control (Hsu, 1999; Newborn, 2003). Hardware and software search are combined and over 500 processors perform a massive parallel game tree search (Hsu, 2002; Campbell et al., 2002).

An important paradigm, not only for Chess, but for game programming in general, consists of methods that look ahead, evaluating as many sequences of moves as possible until thinking time expires. Several of these search algorithms have been developed over the past years such as Minimax (Shannon, 1950), (α-β)–Pruning (Knuth and Moore, 1975), NegaScout, MDT(f) (Plaat, 1996) and others; see for example (Plaat, 1996; Russell and Norvig, 2003) for an overview.

In games like Tic-Tac-Toe a game tree search can in reasonable time provide a complete overview of all possible ways to play and, if possible, how to win (Russell and Norvig, 2003). However, in Hex where the branching factor of the game tree is very high (see Table 3), the game tree cannot completely be expanded within an acceptable amount of time.

Game	*typical branching factor*
Draughts (Checkers)	3-8
Othello	12
Chess	35-40
10 × 10 Hex	80
14 × 14 Hex	160
19 × 19 Go	300
Backgammon	400

Table 3: Typical branching factors for some well known games.

The game of Go is one of the oldest board games. It has simple rules but considerable training and experience is required to become a good player. Go is hard for traditional computer game playing techniques and therefore still

one of the major challenges in machine intelligence research. The challenge of artificial Go is multi-fold. One reason is the large branching factor. On a (19×19)–board there are typically around 300 moves available from a midgame position. Other reasons are that Go is pattern oriented and that a player must be able to follow multiple interacting goals. Almost the same reasons apply to Hex.

Backgammon is a non-deterministic board game for two players. In contrast to games like Chess, Go and Hex, Backgammon uses dice rolls and therefore involves a random component. Fifteen white and fifteen black pieces are moved on a board of 24 locations. Due to the huge number of possible backgammon positions, a lookup table cannot be used. After a typical dice roll there might be 20 different ways of playing. Therefore the game tree has an effective branching factor of about 400 (far beyond that of Chess or Checkers).

In Chess, Go, Backgammon, Hex, and so on, the aim of the search is to produce a number which is related to the likelihood of a win given the present board position.

A board position s is *terminal* if it can be decided who has won. The minimax strategy determines the values of non-terminal board positions or determines new terminal positions. The algorithm assumes alternate optimum play of both sides and searches the branches of the tree to the terminal nodes, or in complex tasks only up to a given depth (for example n-moves; in other words, *n-ply search*). In the latter case the values of the pseudo terminal positions are estimated via the evaluation function J.

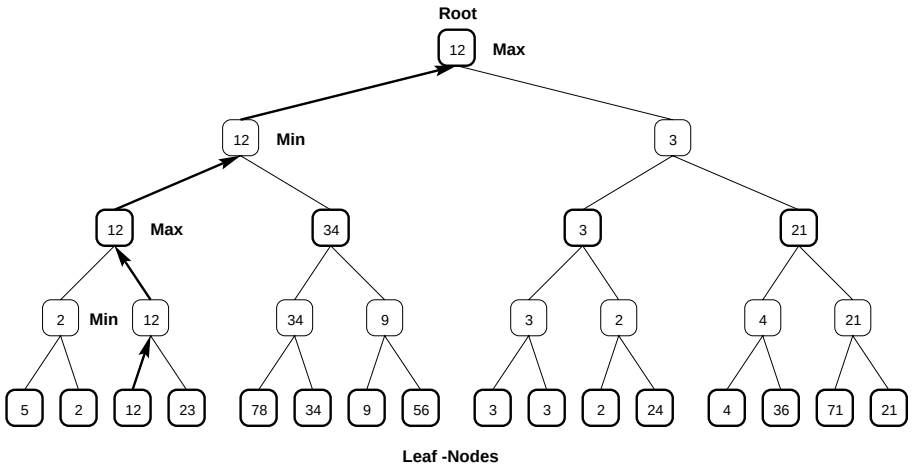


Figure 3: A minimax game tree of degree 2 and depth 4 (2-ply). Example values have been assigned to the leave nodes and have been propagated back from the leaves to the root at the top following the minimax algorithm.

Figure 3 shows an example minimax game tree of degree 2 and depth 4: starting at the root node, two virtual players Max and Min move alternately so that a game tree of several possible options of play is generated. Values are assigned to the leaf nodes by a static evaluation function. These values are propagated back through the tree to determine which is the best move for Max at the root node.

```
BestScore alphabeta(node, α, β)
  if(node = leafNode)
    return eval(node)
  else if(node=maxNode)
    score ← −1.0
    c ← firstChild(node)
    while(c is not a leafNode and score < β)
        score ← max(score, alphabeta(c))
        α ← min(α, score)
        c ← nextSibling(c)
    end while
  end if
  else if(node = minNode)
    score ← +1.0
    c ← firstChild(node)
    while(c is not a leafNode and score > α)
        score ← min(score, alphabeta(c))
        β ← min(β, score)
        c ← nextSibling(c)
    end while
  end if
return score
```

Table 4: (α-β)–function with recursive calls.

There are several variations and improvements of the minimax algorithm. One of them is the alpha-beta algorithm (Knuth and Moore, 1975). It can help to gain speed by pruning the game tree. Pseudocode for minimax with alpha-beta pruning is given in Table 4. Further improvements such as the MDT(f) algorithm can be found in (Plaat, 1996).

5.3 Tournament Play

In tournament play several players can compete against each other. Pairings of players are determined and a winner or a ranking list of the players is produced. Different ranking systems and tournaments have been developed for different games. For Hex, a generally accepted standard tournament and ranking system has not yet been decided. However, several tournament systems which have been used for other games can also be employed directly or with minor modifications to Hex.

In a *Round Robin* tournament, all players play against each other once. Therefore it is very time-consuming. However, it can be used as a first stage for a tournament with a large entry where the players are divided up into groups of equal size. Then the best two players of each group move on to the next stage where a new tournament is held.

A more time efficient alternative to a round robin tournament is the *Swiss pairing system*. It is common for Chess tournaments and can as well be

applied to Hex. The official FIDE (World Chess Federation) Swiss Rules Based on Rating are available on the Internet (World Chess Federation, 1992).

For games where no draw is possible a *Knock-out* tournament can be used. The initial pairings are allotted randomly and the winners progress to the next round, while the losers are eliminated from the tournament. This continues until there is only one undefeated player left—the winner. However, this type of tournament does not provide enough information to form a reliable ranking list.

In a *random play* tournament parings are selected at random. It simulates a server where humans choose to play other humans arbitrarily. The matches can be stopped at any time in a non-ordered fashion.

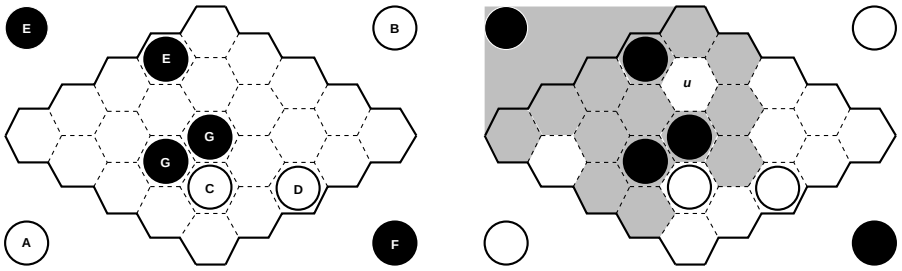
5.4 Programs that Learn

Ultimately, players which employ learning methods should be able to beat systems which simply rely on computational power and exhaustive search algorithms and which cannot improve their performance. Already Samuel (1959) designed an artificial player for Checkers which was trained in self-play. It used the reward at the end of the game to improve its performance. One of the best backgammon players in the world is the program TD-Gammon by Tesauro (1992, 1994, 2002). In TD-Gammon 3.1, a multi-layer neural network with 160 hidden units was trained using a gradient-descent form of the TD(λ) algorithm (Sutton and Barto, 1998). The program played about 6,000,000 games against itself before it stopped improving. Search was relatively shallow, only about two or three plies depending on the version of the program. The Chess program KnightCap uses a special form of temporal difference learning called TDLeaf(λ) (Baxter et al., 1998, 2000). KnightCap was trained while playing on the Free Internet Chess Server (`FICS.onenet.net`). Starting from an already reasonably good set of parameters, the program improved significantly within a few hundred games and reached a high level of play.

So far, no learning players have been developed in the MIHex student assignment. However, separate research work as well as student projects at Newcastle have addressed implementations of simple learning artificial players for Hex.

6 Designing an Artificial Player for Hex

In this section we consider how to construct an evaluation function for Hex suitable for use in a minimax search. The evaluation functions presented below are based around measuring the relative connectivity of a Hex position, which is a very important strategic concept in Hex. We do not claim that these evaluation functions are the only possibilities, nor that the minimax search approach is the only sensible paradigm for Hex, but the material covered does reflect the dominant approach to designing Hex programs reported in the literature. We also address the issue of the large branching factor of Hex, which poses a challenging problem to game tree searches, and



(i) (ii)

Figure 4: A Hex position illustrating the concepts of chains and neighbourhood. In (i) the seven chains in the position have been labelled A–G, and in (ii) the cells in the neighbourhood of u with respect to the player *black* have been shaded.

focus on one technique that offers a partial solution based on recognising fixed patterns in the Hex position. Finally, we include a brief description of two well known Hex programs, Queenbee and Hexy, that have inspired the approaches described below, but first let us start by defining some concepts and terminology relevant to measuring connectivity in Hex.

6.1 Concepts and Terminology

In the following, $s \in \mathcal{S}$ denotes a Hex position specifying the placement of pieces on the board at some stage during a match, and $p, \bar{p} \in \{black, white\}$ denote the two players. It will be convenient to consider each edge of the board as a cell occupied with a piece belonging to the player that owns the edge; we will abuse terminology a little and refer to such a cell as an *edge piece* and denote the two edge pieces belonging to player p as $e1_p$ and $e2_p$. Two cells are *adjacent* if they touch on the Hex board (an edge piece is adjacent to all the cells lying on the edge it represents). A *chain* is a maximal set of connected pieces of the same colour (chains may contain edge pieces). The *neighbourhood* of a cell u consists of the set of cells that are neighbours to u, where two cells are neighbours with respect to player p if they are adjacent or they are connected by a chain belonging to p. The neighbourhood of u with respect to one of the players is denoted by $N(u)$, where it is obvious from the context which player is intended. The concepts of chains and neighbourhood are illustrated in Figure 4, which also contains a Hex position that will be used as a running example throughout the rest of this section.

A Hex position can be represented by a graph containing a set of vertices representing the cells (including the edge pieces), and a set of edges representing the pairs of adjacent cells. Each edge in the graph can be associated with a weight, which indicates the connectivity between its adjacent cells. Such a graph is useful for computing distances and other connectivity measures between arbitrary cells relative to a particular player, although it is often beneficial to remove the vertices corresponding to the cells occupied by the opponent and any edges connected to them. We let $G(s, p)$ denote the graph corresponding to position s relative to player p (with $\bar{p}$'s vertices

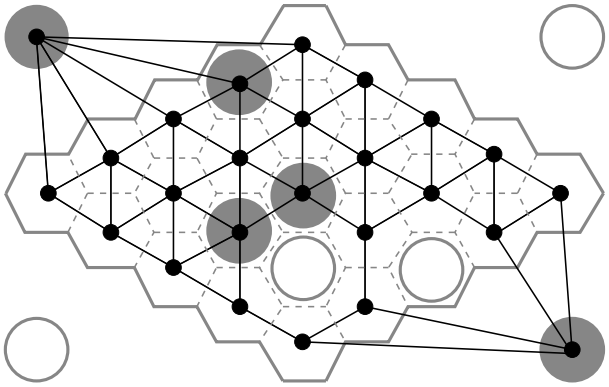


Figure 5: A Hex position s overlaid with the graph $G(s, black)$.

and edges are pruned out). Figure 5 illustrates *black*'s graph for the position in our running example.

6.2 Evaluating a Position

The object of Hex is to connect your two edges of the board, so it is not surprising that understanding connectivity in Hex is crucial to playing the game well. An artificial player too will need some way of assessing how well-connected the opposite sides of the board are. Below we present some metrics for computing connectivity in Hex, but first let us consider how to use such a measure to construct an evaluation function.

Assume we know a function $\kappa : \mathcal{S} \times \{black, white\} \to \mathbb{R}$ that maps an arbitrary position s and a player p to $\kappa(s, p)$, the connectivity between p's edge pieces. We will assume that when p's edges are already connected $\kappa(s, p) = 0$, and that as the connectivity between p's edges falls away $\kappa(s, p)$ increases. Now let us define an evaluation function for Hex as the difference in connectivity between the two players:

$$J(s, p) = \kappa(s, \bar{p}) - \kappa(s, p).$$

This evaluation function has the following useful properties:

1. $J(s, p)$ is positive when the connectivity in position s is greater for p than $\bar{p}$, and is negative when the connectivity for p is less than $\bar{p}$.
2. $J(s, p)$ increases as the connectivity in position s for player p increases, and it decreases as the connectivity for p decreases.

Hence $J(s, p)$ gives an estimate of the strategic value of position s for player p which is suitable for use in a minimax search. Next we consider how the connectivity of an arbitrary position in Hex can be measured.

6.2.1 Counting the Number of Pieces to Build a Connection

The simplest connectivity metric that we will consider is to count the least number of pieces required to connect p's edge pieces, which we will denote

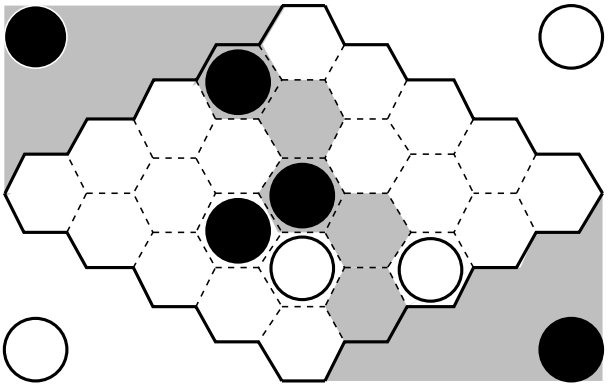


Figure 6: One of the shortest paths connecting *black*'s edges.

by κ_1. It is straightforward to compute $\kappa_1(s,p)$ from the graph $G(s,p)$ by calculating the shortest path between the edge pieces, with Dijkstra's algorithm[5] for example, and then counting the number of unoccupied cells lying on the shortest path. The following scheme can be used to weight the edges for Dijkstra's algorithm. Every cell u is assigned a weight $w_p(u)$ relative to player p as follows:

$$w_p(u) = \begin{cases} 1 & \text{if } u \text{ is unoccupied,} \\ 0 & \text{if } u \text{ contains a piece belonging to } p. \end{cases}$$

Then for each pair of adjacent cells (u,v) a weight $w_p(u,v) = w_p(u) + w_p(v)$ is associated with the corresponding edge. Figure 6 shows one shortest path connecting *black*'s edges, although others exist. Let $V(s,p)$ be the set of vertices that correspond to the unoccupied cells lying on the shortest path, then the least number of pieces required to connect p's edges is given by:

$$\kappa_1(s,p) = \mid V(s,p) \mid .$$

6.2.2 Taking Threats into Account

The problem with counting the number of empty cells lying on the shortest path is that it does not consider alternative paths. If the opponent blocks the shortest path on their next move then the metric says nothing about how well-connected the rest of the board is. Here we present an alternative metric, κ_2, that implicitly takes threats into account.

Consider the problem of measuring the distance $d(u,v)$ between two unoccupied cells u and v relative to a particular player. The κ_1 metric uses the conventional distance, which can be defined recursively as follows. If u and v are neighbours then $d(u,v) = 1$, otherwise $d(u,v) = 1 + d(u',v)$, where u' is an unoccupied neighbour of u that has the minimum distance to v, that is:

$$u' = \arg \min_{w \in N(u)} d(w,v).$$

[5] A description of Dijkstra's algorithm can be found in any undergraduate textbook covering graph algorithms, such as the algorithmics textbook by Cormen et al. (2001).

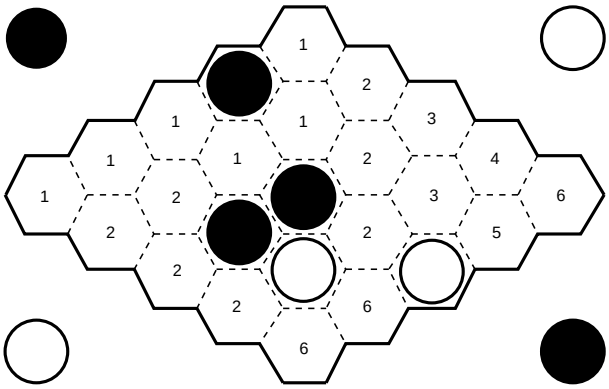


Figure 7: Two-distances to the upper left edge.

Unfortunately, the conventional distance provides a poor estimate of the value of a position because in the next turn the adversary can place a piece in any of the cells lying on the shortest path—unless, by chance, there happens to be two or more shortest paths.

A better metric for Hex would increase the value of a position if it has multiple paths of short length between u and v. Such a metric can be achieved using a distance measure called the two-distance (van Rijswijck, 2000). The two-distance is defined as follows: as above, if u and v are neighbours then $d(u, v) = 1$, but otherwise $d(u, v) = 1 + d(u'', v)$, where u'' is a neighbour of u with the minimum distance to v *after first removing* u' *from* $N(u)$. Now any potential blocking moves by the opponent are accounted for because a second well connected alternative, u'', is considered at each step of the distance calculation. In effect, the metric provides distance information about *multiple paths* between u and v.

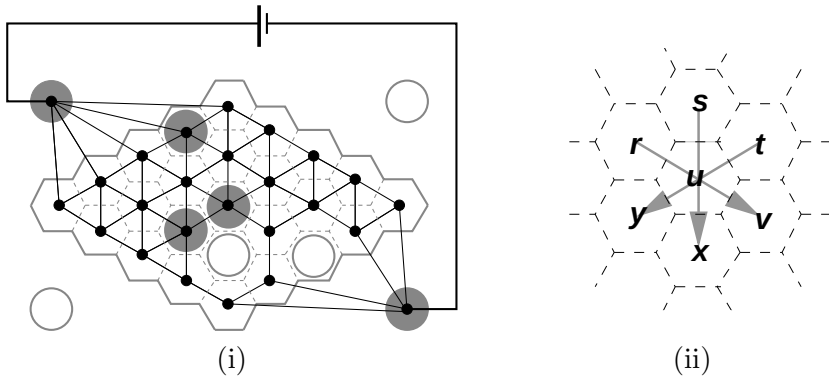
More formally, given an arbitrary position s, and a function $N(u)$ that returns the set of cells and the edge pieces in the neighbourhood of u with respect to player p, then there is a general distance metric defined as:

$$d_z(u, v) = \begin{cases} 0 & \text{if } v = u, \\ 1 & \text{if } v \in N(u), \\ \min\{k : c_k(u, v) \geq z\} & \text{if the set is non-empty,} \\ +\infty & \text{otherwise} \end{cases}$$

where u and v are cells and the function $c_k(u, v)$ returns the number of cells in the neighbourhood of u whose distance to v is less than k:

$$c_k(u, v) = |\{w \in N(u) : d_z(w, v) < k\}|.$$

The conventional distance metric corresponds to $z = 1$ and the two-distance corresponds to $z = 2$. Figure 7 shows the two-distances between each unoccupied cell to the upper left edge piece for the position in our running example. Notice that the cell with two-distance 6 in the bottom corner of figure 7 has a neighbourhood which includes the bottom-right black edge piece as well as the cell with two-distance 5 further up on the right. Fi-



(i) (ii)

Figure 8: (i) *Black*'s resistor circuit. (ii) Direction of electrical current passing through cells.

nally, an estimate of the connectivity between p's edge pieces is given by the two-distance between $e1_p$ and $e2_p$:

$$\kappa_2(s, p) = d_2(e1_p, e2_p).$$

6.2.3 Considering all Paths in Parallel

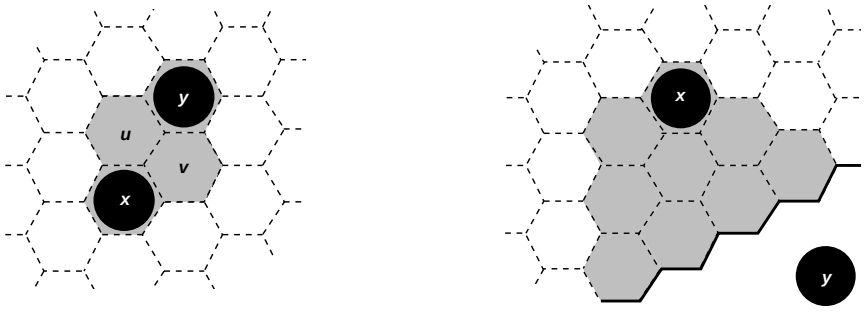
The two-distance metric reduces the possibility of being blocked by the opponent because it considers partial connectivity information in addition to the path length. We now present a metric, κ_3, that takes this concept to its logical extreme—modeling a Hex position as an electrical resistance circuit—which accounts for *all* alternative paths as well as their lengths.

In this approach, the graph G for a given Hex position is interpreted as an electrical circuit where vertices are nodes, edges are resistors, and edge weights are resistance values (the same weightings can be used as described in section 6.2.1). Thus, a connectivity value for the position is obtained by applying an electrical voltage V across the edge pieces (see Figure 8) and calculating the equivalent resistance, $R_p(s)$, of the resulting circuit. The circuit is a complicated one, but $R_p(s)$ can be calculated by application of Kirchoff's Current Law (KCL); below we briefly outline the procedure, however anyone considering implementing this approach would be advised to consult a relevant book on circuit simulation, such as that by Litovski and Zwolinski (1997).

The KCL states that the total current entering a node must be equal to the total current leaving the node, thus an equation can be written for the current passing through a node. For example, the KCL equation for cell u in Figure 8(ii) is:

$$\begin{aligned} \frac{V(r) - V(u)}{R(r,u)} + \frac{V(s) - V(u)}{R(s,u)} + \frac{V(t) - V(u)}{R(t,u)} + \frac{V(u) - V(v)}{R(u,v)} & \\ + \frac{V(u) - V(x)}{R(u,x)} + \frac{V(u) - V(y)}{R(u,y)} &= 0, \end{aligned}$$

where $V(i)$ is the voltage at node i, and $R(i, j)$ is the resistance for connec-



(i) (ii)

Figure 9: (i) A two-bridge and (ii) an edge template.

tion (i, j). A set of KCL equations are written out for each node (given the usual assumption that the direction of current through the resistors is consistent across the equations), and solved simultaneously to yield the voltages at each node. Since all the individual voltages and resistances are known, the total current leaving the circuit can be calculated, which in turn yields a value for the equivalent resistance $R_p(s)$ of the whole circuit. Finally, the value of $R_p(s)$ provides the connectivity measure for the position:

$$\kappa_3(s, p) = R_p(s).$$

6.3 Overcoming the Large Branching Factor of Hex

The branching factor of Hex is large relative to other games (see Table 3) resulting in a massive game tree, which is infeasible to search for more than a few moves ahead. Techniques to combat explosion in the size of the game tree typically include various pruning operations, such as (α-β)–pruning and beam search, and transposition tables, which cache the value of a position for lookup should it occur again through another line of play. However, here we focus on an alternative method that may be more effective for the special case of Hex—the use of connection templates.

A connection template is a recognisable pattern of cells that ensures a certain amount of connectivity between two cells called *terminal points*. Figure 9 shows two examples of connection templates, which consist of a collection of shaded cells with terminal points labeled x and y. Consider the two-bridge pattern: if *white* plays at u then *black* responds with v to complete the connection between x and y; alternatively if *white*'s play is v then *black* can respond with u; hence the outcome of a two-bridge is predictable two moves ahead. Likewise the outcome of the edge template shown is also predictable, as are connection templates in general. The predictable outcome of a connection template permits deep analysis of a position, effectively allowing a program to look ahead without incurring the cost of an explicit game tree search. A presentation of the various different connection templates that can exist in Hex is beyond the scope of this article, instead we show how to include connection templates into the above connectivity metrics. The interested reader is referred to Browne (2000) for a comprehensive treatment

of connection templates.

It is starightforward to modify the metrics given above to include connection templates. For the two metrics κ_1 and κ_3 (counting the least number of pieces and calculating the total resistance of the Hex circuit) extra edges are added to $G(s, p)$ that connect the terminal points of p's connection templates detected for position s. Since a connection template does not guarantee a connection between terminal points with absolute certainty, the corresponding edges are weighted a little higher than edges connecting adjacent pieces of the same colour.[6] In the case of κ_1 the metric is redefined as the sum of weights on the shortest path, which will take into account that, other factors being equal, a path containing more connection templates is weaker than one with less.

For κ_2, the two-distance metric, the concept of neighbourhood can be extended to include cells that are connected via a connection template. Following this idea the neighbour function, $N(u)$, is redefined so that firstly, a pair of terminal points are considered to be neighbours and secondly, all neighbours of one terminal point are considered to be neighbours of the other terminal point. However as already mentioned, connection templates do not guarantee a connection and it should be pointed out that this straightforward approach fails to recognise that connection templates represent a slightly weaker connection. A more sophisticated modification would distinguish between cells that are neighbours in the normal sense and cells that are neighbours because they are connected via the terminal points of a connection template, and assign perhaps a slightly higher two-distance value in the latter case.

6.4 Queenbee and Hexy

The first Hex "program" was built by Shannon (1953), which consisted of an analog machine that modelled a Hex position as an electrical resistance circuit. Despite this early start Hex was then largely ignored as a subject for game programming, until recently when two strong players were developed. The Queenbee program by van Rijswijck (2000) was the first Hex program to play at a level above human novices and introduced the two-distance metric. Subsequently, Anshelevich (2001) wrote Hexy, which employed Shannon's electrical resistance circuit idea and attained the level of good human players—making it probably the strongest Hex program available at that time.

It is interesting to compare how Queenbee and Hexy cope with the large branching factor of Hex. Queenbee's approach is to apply powerful techniques for deep search that were originally developed for games like Chess. Thus, Queenbee performs a massive game tree search using iterative deepening (α-β) search enhanced with Minimal Window/Principal Variation Search and transposition tables. Non-uniform search extensions allow some lines of play to be investigated more thoroughly than others. In contrast to Queenbee, Hexy does not rely on advanced game tree pruning techniques. Instead

[6] In the case of two overlapping templates, an intrusion by the opponent into the cells of the overlapping area can threaten both templates simultaneously with the result that the connection between one set of terminal points is lost.

it discovers virtual connections, which are a generalisation of connection templates that have an arbitrary structure—which makes their detection considerably more difficult. Most of Hexy's resources are spent computing virtual connections rather than searching the game tree, and its author reports that it regularly foresees patterns in a position 20 or more moves deep.

7 Educational Outcomes and Discussion

The Hex assignment supports education on general search procedures, game tree search, adversarial search including the standard (α-β)–search, as well as deeper, high performance pruning, and other refinements to reduce search cost. Beyond general AI concepts the MIHex project requires that students look into Hex specific concepts and confronts them with a significant software engineering task.

7.1 The Challenge to Understand Hex Specific Concepts and Learning to Restrict a Problem

In the previous section we saw that metrics giving the connectivity between a player's two edge pieces can be used to construct an evaluation function for Hex. The metrics we considered lie on a continuum of sophistication. The simplest, counting the number of pieces to make a connection, only measures the length of connecting paths. The second metric, based on two-distances, measures the length of paths containing cells that have a certain minimum level of connectivity. Finally, the last and most sophisticated modelled Hex positions as an electrical resistance circuit, measuring the total number of alternative connections in addition to their lengths. The strategic significance of having many potential paths to form a connection is that they constitute multiple threats, making blocking difficult. The price is additional complexity, both in terms of algorithm complexity and time complexity. We also considered a partial solution to the limited lookahead imposed by the large branching factor of Hex: the identification and incorporation of connection templates into the connectivity metrics. These patterns allow distant connections to be predicted, effectively increasing the lookahead without the cost of searching the game tree.

As is typical for AI, students must learn to restrict a problem. Some small part of a larger problem is examined in a restricted environment. A large part of what students learn in the Hex assignment involves how to break up a problem and take care of the different components in a structured computational manner all the while maintaining consistency among the pieces. There are two different components in the Hex project: the search and evaluation functions. These are not independent. The two functions interact and relate to each other. Students have to understand how to break down the problem, but also how to vary the parameters within each of the components and how this affects the interaction between them. In other words, the first step is to break the problem into components. The second step is to fine-tune them. Students who recognize this are very good students. For example, a slow evaluation function may mean a quick search but it may not be a better

trade-off. Students are required to read and understand research papers (of varying degrees of difficulty) in order to make decisions between logic based, search tree or other approaches. They research the Queenbee Hex player and Hexy's basis in electrical circuits, as well as different ways to implement the evaluation function: for example, electrical circuit (Anshelevich, 2001) and 2-distance (van Rijswijck, 2000).

7.2 Report on Student Approaches and Experiences

There are two main aspects to the project: the AI aspect, and the software engineering component of decomposing the player, breaking into class structure, and documentation.

7.2.1 Intricacies in Using AI Methods

Search and evaluation are key paradigms for a Hex player. For an agent to play Hex effectively, it needs an accurate way of determining the relative strengths of different board positions. On larger boards such as 14×14, it becomes infeasible to search through all possible combinations. There are known patterns in Hex such as the two-bridge, and even three- or four-bridge virtual connections-patterns which humans can spot quite easily, but which may take a program significant searching without some sort of pattern recognition.

Once the evaluation function is determined, the searching algorithm is the next phase. Students were surprised at how long a game tree search takes. Many students used an approach based on the minimax algorithm, such as the alpha-beta search variation, and then added other features to improve performance such as search extensions, reductions or Dijkstra's Shortest Path Algorithm to evaluate the utility of a board. There were variations on assigning values to leaf nodes and iterative deepening was used to vary the budget in the time constraints for each move. Students tried a variety of tuning approaches. The time taken to evaluate the utility of a board was a major overhead when deciding on a move to play.

Students developed short cuts such as ignoring cells that were occupied by the opponent, or automatically taking certain moves if a 2-bridge was endangered by the opponent. Students recognized that a winning agent only takes into account a certain subset of available moves, greatly reducing the time to search the game-space. Students experienced the difference between various AI techniques, and they realized in the end the costs of inefficient implementations.

7.2.2 Becoming a Software Engineer?

As one student wrote in his report, "The idea behind the winning player was straightforward. The implementation was very difficult." Students cannot get away without learning programming; the agent has to work. The implementation is a real challenge and goes beyond the sorts of simple tasks that students had learned before. Students experience object oriented tech-

nology and client servers. The project brings in everything they have learned to date. In fact, students must learn to limit the ideas that they have. A student may have twenty great ideas but only time to implement a few. One student said, "What I like most about the Hex assignment is seeing how my agent evolved during development and experimenting with various parameters and noting the changes in behaviour/strategy by my agent." Another wrote, "It helped me review my knowledge of algorithms for game tree searching, and for development of evaluation functions in particular." Another student wrote, "Issues of efficiency come up to deal with. I learnt a lot about Java." Students saw the realities of efficiency and the practicalities of implementing with say, Boolean arrays or adjacency matrices. One student said in an interview, "I had to have a strategic plan or do research to make one. I couldn't just hack code." Representation is a critical part of AI. All the students appreciated that the project was hands-on and that they could see a graphical representation of how the algorithms/code performed.

7.2.3 Experiencing the Challenge

Perhaps surprisingly, students expressed appreciation for the challenge of the assignment. One student summarised the feelings of most when he said, "No one really likes a hard assignment; but at the end of the game, only really hard assignments make you learn. For this project, you have to get books and actually read. I go to all the lectures but it is easy for me to forget what I hear. If you want to produce computer science professionals, then get them to do the work. Give people an easy way out and they will always take the easy way."

7.2.4 Changes in the Assignment

New elements were introduced into the assignment every term, partly to keep the project fresh for the staff and instructor. The new feature in year four was to develop a losing player, a form of Misère Hex (Lagarias and Sleator, 1999). The Losing Player assignment had never been given before, and its level of difficulty was not known. Students employed a variety of losing strategies: do not play in the main diagonal until necessary; play winning moves only when there are no other moves left; negate the strategies of the winning player, and so forth. Students were asked in the final questionnaire if they minded having an assignment that had not been tested explicitly. Aside from a small concern about marking, the students thought it was adventurous. They said it was more fun doing something that even the lecturer hadn't done.

7.2.5 Competition and Deadlines

In addition to the technical aspects of the assignment, students also experience a competitive element through a tournament in which the student agents compete against each other. Students did not watch the tournament (it would take too long). However, we felt the competition brought an element of real-world realism to the assignment.

Response to the competition was generally positive. Students had concerns: they didn't quite grasp the concepts and would not be able to create their "dream player", or other students had more background and skills, or they had less time to put into the project than others. During personal interviews with two students, one described the above disadvantages while the other countered with: "Oh, that's just like life. There will always be deadlines. Some may find it hard. The only way to learn is if you try." The second student was basically summarizing the attitude of the entire class which was that even though they were concerned that they would not perform well in the tournament for various reasons, it was realistic to be judged against a large group of opponents and "you can see where you stand".

Only about five percent of the students in the course were female, which matches the low female enrollment in computer science generally at the University of Newcastle. Perhaps because these were third year students, they expressed no concern about a tournament competition.

7.3 Student Motivation

Because Hex is a game, we wondered if students had enrolled in the course for reasons of computer games. In our initial questionnaire, we asked the students if they were taking the course primarily because of its connections to computer games or possibly connections to biology and brain science. Typically only a small group of students clearly expressed interest in biology and brain science. In 2004 about two-thirds of the students said that interest in computer games was at least a secondary factor in choosing to enroll. One student wrote, "Yes, this is me. Always loved computer games and it is a boyhood dream of mine to be a game developer." Another said, "I can understand and learn algorithms when computer games are used in course projects. That is important for me." Other students simply wrote, "Exactly!" or "True." Another student wrote, "This is closer to the truth but I don't have any interest in making computer games. I am more interested in robotics or intelligent consumer electronics." About half the students (and all of the female students) said something like, "Computer games are an interesting area where knowledge gained in this course could be used, but games are not my priority. I enrolled mainly for techniques on implementing AI."

We also asked if students would prefer a 3D-Shooter type game rather than a Hex/Chess type game for implementing an AI agent. Responses ranged from a resounding "No way!" and "NO! To run you have to learn how to walk first!" to more ambivalent responses such as "Possibly a 2D shooter would be a good hybrid." and "I think the Hex project is suitable because the concepts are easy to learn but the underlying problems are complex." Students were clear that they wanted to learn about artificial intelligence and that they trusted the instructor to choose the most appropriate type of game or situation for them. Students who enjoyed and were experienced with shooter games also recognized their complexity. They wanted a project that was closer to a pure AI type game and also a simpler platform from which to learn.

As students documented on the questionnaire, they enrolled in the course

because they were interested in the philosophy, current issues and developments, theory and practice of AI. They wanted to know the great names involved: such as Turing, McCarthy, Brooks, and Nash. They were interested in AI as an area that is not completely solved—there are ongoing challenges to the human mind.

We wanted to know if the thought processes used during the Hex project transferred outside of school. For example, people who are very good at Chess can predict several moves in advance. Did working on Hex increase students' ability to anticipate and make predictions in their everyday lives? We enquired on the initial questionnaire and the students responded that they did not think a brief university assignment in one course would make much difference in their overall lives. At the end of the project however, more than one student approached the staff to discuss the idea and gave it merit. This is an area for additional investigation.

7.4 MIHex as Significant Learning Experience

In summary, the outcome of our student evaluations indicated that several factors contributed to the success of the MIHex assignment. The main individual reason was that it provided the right level of challenge and complexity at the right time for the students within their degree programme. Additional motivation (for example, prizes for the winners of the competition) is helpful but not essential. Even in the fourth year when not much motivation, support, or care was provided to the students for the MIHex assignment, even the two students who had most trouble completing the assignment reported that they profited a lot and that it was a very worthwhile experience. One wrote: "... I do know that I did work very hard on my agent. It's almost heart breaking that I missed the tournament ... Maybe I will do a little more work when the holidays come around, and show u my results next semester...". The students felt that this project was exactly the right opportunity to apply for the first time the knowledge and understanding they had accumulated in their studies so far, and see and experience visually how it works.

Another explanation for the success of the MIHex project assignment is that it satisfies a set of features which according to Fink (2003) characterise a *significant learning experience*: For the students it was the first time to see real AI at work; that is, they had the opportunity to connect what they had learned so far to a new dimension of experience of what computers can do. The human dimension comes into play when students play against the machine, and further, when they compete with each other by comparing their programs in the class tournament. The aesthetics of Hex and the fun of play as well as the aforementioned visual experience of the working algorithms contribute to making the students care about their players and their project. Since the project requires a significant amount of work and runs over several weeks, it demands time and project management skills as well as the ability to extract suitable methods from the lecture, readings, and other material to apply to their Hex player. That is, the students are challenged to learn how to learn if they want to be successful in the final tournament.

8 Conclusion

The present paper gave a comprehensive overview of educationally relevant aspects of the MIHex project. It surveyed techniques for designing artificial players for Hex and covered some standard machine intelligence methods for games as well as techniques specific to Hex. Techniques used by two of the currently best Hex playing programs, *Hexy* and *Queenbee*, were addressed and several different concepts for board evaluation and feature design were discussed. The review of these techniques might contribute to design future game programming assignments efficiently and effectively from a technical and an educational point of view.

Two classes of board evaluation methods, electrical resistor circuits, and potentials based on two-distance were discussed. It is not clear which of the evaluation methods is best. Also it is not yet clear whether a combined approach of all presented methods would bring some advantage and how the different features interact.

In the last section we discussed the outcomes of the MIHex project for the students of a course in machine intelligence. We observed that students were very motivated when working on the Hex project. The project offered them fast access to the frontier of research in Machine Intelligence.

Acknowledgements

The authors are grateful to all friends, supporters, and participants of the MIHex project at the University of Newcastle, in particular to Michael Quinlan, Nathan Lovell, Neil Wright, and all students and tutors of the courses COMP3330/6380 Machine Intelligence in 2001-2004. Thanks also to all colleagues who were involved, in particular, to Daniel Le Berre for his interest in the project in 2001 and to Frederic Maire at QUT for many inspiring games and suggestions. We are grateful to the Discipline of Computer Science and Software Engineering at the University of Newcastle for supporting the MIHex project and course development of COMP3330/6380 Machine Intelligence.

Literature

Anshelevich, V. V. (2002). A hierarchical approach to computer Hex. *Artificial Intelligence*, 134:101–120.

Baxter, J., Tridgell, A., and Weaver, L. (1998). TDLeaf(λ): Combining temporal difference learning with game-tree search. In Downs, T., Frean, M., and Gallagher, M., editors (1998). *Proceedings of the Ninth Australian Conference on Neural Networks*, Brisbane. University of Queensland.

Baxter, J., Tridgell, A., and Weaver, L. (2000). Learning to play Chess using temporal differences. *Machine Learning* 40(3):243–263.

Berlecamp, E. R., Conway, J. H., and Guy, R. K. (2001). *Winning Ways for your Mathematical Plays*. A. K. Peters, Wellesley, MA, second edition.

Bouzy, B. and Cazenave, T. (2001). Computer Go: An AI-oriented survey. *Artificial Intelligence* 132(1):39–103.

Browne, C. (2000). *Hex Strategy, Making the Right Connections*. Natick, MA, A. K. Peters.

Browne, C. (2004). *Connection Games: Variations on a Theme*. Wellesley, MA, A. K. Peters, pages 68–77.

Burmeister, J. and Wiles, J. (1995). The challenge of Go as a domain: A comparison between Go and Chess. In *Proceedings of the Third Australian and New Zealand Conference on Intelligent Information Systems, Perth, November 1995*, pages 181–186. IEEE Western Australia Section.

Buro, M. (1994). *Techniken fuer die Bewertung von Spielsituationen anhand von Beispielen*. PhD thesis, Fachbereich 17 - Mathematik-Informatik der Universität-GH-Paderborn.

Buro, M. (1999). From simple features to sophisticated evaluation functions. In van den Herik, H. J. and Iida, H., editors, *Computers and Games: First International Conference, CG'98, Tsukuba, Japan, November 1998. Proceedings*, LNCS 1558, pages 126–145. Springer-Verlag.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence* 134:85–99.

Campbell, M. S., Hoane Jr., A. J., and Hsu, F. (2002). Deep blue. *Artificial Intelligence* 134(1-2):57–83.

Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. (2001). *Introduction to Algorithms*. MIT Press, New York: McGraw-Hill.

Downs, T., Frean, M., and Gallagher, M., editors (1998). *Proceedings of the Ninth Australian Conference on Neural Networks*, Brisbane. University of Queensland.

Epstein, S., Levinson, R., Campbell, M., Flann, N., Korf, R., Pell, B., Russell, S., Tadepalli, P., Tesauro, G., and Utgoff, P. (1993). Games: Planning & Learning. Papers from the 1993 *AAAI Fall Symposium, October 22-24,*

Raleigh, North Carolina. Technical Report FS-93-02, The American Association for Artificial Intelligence. AAAI Press.

Even, S. and Tarjan, R. E. (1976). A combinatorial problem which is complete in polynomial space. *J. Assoc. Comput. Mach.* 23(4):710–719.

Fink, L. D. (2003). *Creating Significant Learning Experiences. An Integrated Approach to Designing College Courses*. Jossey-Bass.

Fogel, D. B. (2002). *Blondie24: Playing at the Edge of AI*. Morgan Kaufman Publishers.

Gale, D. (1979). The game of Hex and the Brouwer fixed-point theorem. *The American Mathematical Monthly* 86(10):818–827.

Gardner, M. (1959). The Game of Hex. Ch. 8 in *Hexaflexagons and Other Mathematical Diversions: The First Scientific American Book of Puzzles and Games*. Simon and Schuster, New York, pages 73–83.

Hamilton, H. (Ed.) (2000). *AI'00: Advances in Artificial Intelligence, 13th Biennial Canadian Society for Computational Studies of Intelligence (CSCSI) Conference*. Springer Verlag, New York.

Hayward, R., Björnsson, Y., Johanson, M., Kan, N., Po, J., and van Rijswijck, J. (2003). Solving 7x7 Hex: Virtual Connections and Game-State Reduction. In: van den Herik, H. and Iida, H. (eds.) *International Federation for Information Processing*. vol. 263, Kluver Academic Publishers, Boston, pages 261-278.

Heckendorn, R. B. (2002). Building a Beowulf: Leveraging research and department needs for student enrichment via project based learning. *Computer Science Education* 12(4):255–273.

Hsu, F. (1999). IBM's Deep blue Chess grandmaster chips. *IEEE Micro* 19(2):70–81.

Hsu, F. (2002). *Behind Deep Blue*. Princeton University Press.

Hsu, F., Anantharman, T., Campbell, M., and Nowatzyk, A. (1990). A grandmaster Chess machine. *Scientific American* 263(4):44–50.

Knuth, D. E. and Moore, R. E. (1975). An analysis of alpha beta pruning. *Artificial Intelligence* 6(4):293–326.

Lagarias, J. and Sleator, D. (1999). Who wins misère Hex ? In Berlekamp, E. and Rodgers, T., editors, *The Mathemagician and Pied Puzzler: A Collection in Tribute to Martin Gardner*, pages 237–240. A. K. Peters, Natick, MA.

Levinson, R. A. (1996). General game-playing and reinforcement learning. *Computational Intelligence, Special Issue on Games: Structure and Learning* 12(1):155–176.

Litovski, V. and Zwolinski, M. (1997). *VLSI Circuit Simulation and Optimization*. Kluwer Academic Publishers.

Milnor, J. (2002). The Game of Hex. In Kuhn, H. W. and Nasar, S. (eds.) *The Essential John Nash*. Princeton University Press, Princeton, NJ, pages 29–33.

Moursund, D. G. (2003). *Project-Based Learning Using Information Technology*. International Society for Technology in Education, Eugene, OR.

Newborn, M. (2003). *Deep Blue, An Artificial Intelligence Milestone*. Springer-Verlag, New York, Inc.

Nowakowski, R. J. (ed.) (1996). *Games of No Chance*, volume 29 of *Mathematical Sciences Research Institute MSRI Publications*. Cambridge University Press, Cambridge, MA.

Plaat, A. (1996). *Research, RE: Search & Re-Seach*. PhD thesis, Erasmus Universiteit Rotterdam.

Pollack, J. B. and Blair, A. D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning* 32:225–240.

Reisch, S. (1981). Hex is PSPACE-vollstaendig. *Acta Informatica* 15:167–191.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall International, Inc., second edition.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall International, Inc.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3:210–229.

Schaeffer, J. (1997). *One Jump Ahead: Challenging Human Supremancy in Checkers*. Springer-Verlag, New York Inc.

Schaeffer, J. and Lake, R. (1996). *Solving the game of Checkers*, pages 119–133. Volume 29 of Nowakowski (1996).

Shannon, C. (1953). Computers and automata. In *Proceedings of the Institute of Radio Engineers*, Volume 41, pages 1234–1241.

Shannon, C. E. (1950). Programming a computer for playing Chess. *Philosophical Magazine* 41(4):256–275.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning* 8:257–278.

Tesauro, G. (1994). TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* 6:215–219.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence* 134:181–199.

Turing, A. M., Strachey, C., Bates, M. A., and Bowden, B. V. (1953). *Digital Computers Applied to Games*, pages 286–310.

van Rijswijck, J. (2000). *Are Bees better than Fruitflies ? – Experiments with a Hex Playing Program*, pages 13–25. In Hamilton (2000).

World Chess Federation (1992). FIDE swiss rules. `http://www.fide.com/official/handbook.asp?level=c04`.

Yang, J. (2004). `http://www.ee.umanitoba.ca/~jingyang/`. Homepage retrieved December 2004.

Yang, J., Liao, S., and Pawlak, M. (2003). New winning and losing positions for 7×7 Hex. In J. Schaeffer, M. Müller, and Y. Björnsson (eds.) *3rd International Conference on Computers and Games, July 25 - 27, 2002, Edmonton, Canada, Revised Papers*. Lecture Notes in Computer Science, LNCS 2883, New York: Springer-Verlag, pages 230-248.

A MIHex Server output

Below is a portion of a MIHex tournament summary file (the move histories have been omitted).

```
NEW EXPERIMENT's specification:
Date            = Wed Jun 25 16:44:53 EST 2003
Tournament type = Round Robin
boardSize       = 11
nGames          = 10
nPlayers        = 5

Individual player scores
------------------------
Player Name = Player1
vs Player2: Wins/Losses = 3/7
vs Player3: Wins/Losses = 0/10
vs Player4: Wins/Losses = 10/0
vs RandomPlayer: Wins/Losses = 10/0
Total Wins/Losses = 23/17

Player Name = Player2
vs Player1: Wins/Losses = 7/3
vs Player3: Wins/Losses = 0/10
vs Player4: Wins/Losses = 10/0
vs RandomPlayer: Wins/Losses = 10/0
Total Wins/Losses = 27/13

Player Name = Player3
vs Player1: Wins/Losses = 10/0
vs Player2: Wins/Losses = 10/0
vs Player4: Wins/Losses = 10/0
vs RandomPlayer: Wins/Losses = 10/0
Total Wins/Losses = 40/0

Player Name = Player4
vs Player1: Wins/Losses = 0/10
vs Player2: Wins/Losses = 0/10
vs Player3: Wins/Losses = 0/10
vs RandomPlayer: Wins/Losses = 10/0
Total Wins/Losses = 10/30

Player Name = RandomPlayer
vs Player1: Wins/Losses = 0/10
vs Player2: Wins/Losses = 0/10
vs Player3: Wins/Losses = 0/10
vs Player4: Wins/Losses = 0/10
Total Wins/Losses = 0/40

Player rankings
-----------------------------------------------------------------
Rank      Player Name           Wins/Losses           Win Ratio(%)
 1.       Player3               40/0                  100.0
 2.       Player2               27/13                 67.5
 3.       Player1               23/17                 57.5
 4.       Player4               10/30                 25.0
 5.       RandomPlayer          0/40                  0.0

Time needed: 761760 millisecs TOURNAMENT COMPLETED
```