

# **FFI RAPPORT**

## **LEARNING TO PLAY OPERATION LUCID FROM HUMAN EXPERT GAMES**

KRÅKENES Tony, HALCK Ole Martin

**FFI/RAPPORT-2002/04041**



FFISYS/806/161

Approved  
Kjeller 8 September 2003

Jan Erik Torp  
Director of Research

**LEARNING TO PLAY OPERATION LUCID  
FROM HUMAN EXPERT GAMES**

KRÅKENES Tony, HALCK Ole Martin

FFI/RAPPORT-2002/04041

**FORSVARETS FORSKNINGSINSTITUTT**  
**Norwegian Defence Research Establishment**  
P O Box 25, NO-2027 Kjeller, Norway



P O BOX 25  
 NO-2027 KJELLER, NORWAY  
**REPORT DOCUMENTATION PAGE**

**SECURITY CLASSIFICATION OF THIS PAGE**  
 (when data entered)

1) PUBL/REPORT NUMBER FFI/RAPPORT-2002/04041 1a) PROJECT REFERENCE FFISYS/806/161	2) SECURITY CLASSIFICATION UNCLASSIFIED 2a) DECLASSIFICATION/DOWNGRADING SCHEDULE -	3) NUMBER OF PAGES 31		
4) TITLE LEARNING TO PLAY OPERATION LUCID FROM HUMAN EXPERT GAMES				
5) NAMES OF AUTHOR(S) IN FULL (surname first) KRÅKENES Tony, HALCK Ole Martin				
6) DISTRIBUTION STATEMENT Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)				
7) INDEXING TERMS IN ENGLISH: <table style="width: 100%; border: none;"> <tr> <td style="width: 50%; vertical-align: top;">           a) <u>Artificial intelligence</u>            b) <u>Machine learning</u>            c) <u>Neural networks</u>            d) <u>Combat modelling</u>            e) _____         </td> <td style="width: 50%; vertical-align: top;">           IN NORWEGIAN:            a) <u>Kunstig intelligens</u>            b) <u>Maskinl�ring</u>            c) <u>Nevrale nettverk</u>            d) <u>Stridsmodellering</u>            e) _____         </td> </tr> </table>			a) <u>Artificial intelligence</u> b) <u>Machine learning</u> c) <u>Neural networks</u> d) <u>Combat modelling</u> e) _____	IN NORWEGIAN: a) <u>Kunstig intelligens</u> b) <u>Maskinl�ring</u> c) <u>Nevrale nettverk</u> d) <u>Stridsmodellering</u> e) _____
a) <u>Artificial intelligence</u> b) <u>Machine learning</u> c) <u>Neural networks</u> d) <u>Combat modelling</u> e) _____	IN NORWEGIAN: a) <u>Kunstig intelligens</u> b) <u>Maskinl�ring</u> c) <u>Nevrale nettverk</u> d) <u>Stridsmodellering</u> e) _____			
THESAURUS REFERENCE: 8) ABSTRACT <p>When the number of possible moves in each state of a game becomes very high, standard methods for computer game playing are no longer feasible. We present two approaches to learning to play such a game from human expert gamelogs. The first approach uses lazy learning to imitate similar states and actions in a database of expert games. This approach produces rather disappointing results. The second approach deals with the complexity of the action space by collapsing the very large set of allowable actions into a small set of categories according to their semantical intent, while the complexity of the state space is handled by representing the states of collections of units by a few relevant features in a location-independent way. The state-action mappings implicit in the expert games are then learnt using neural networks. Experiments compare this approach to methods that have previously been applied to this domain, with encouraging results.</p>				
9) DATE 8 September 2003	AUTHORIZED BY This page only Jan Erik Torp	POSITION Director of Research		



**CONTENTS**

	<b>Page</b>	
1	INTRODUCTION	7
2	THE GAME OF OPERATION LUCID	8
2.1	Definition and Rules of Operation Lucid	8
2.2	A Combat Modelling Interpretation of the Game	9
2.3	The Complexity of the Problem	9
3	IMITATION LEARNING APPROACHES TO AGENT DESIGN	10
3.1	Lazy Imitation Learning	10
3.2	Agent Design based on Constraint Satisfaction Programming	12
3.3	Agent 1 – Action Imitation	12
3.4	Agent 2 – State Imitation Using CSP	14
3.5	Agent 3 – Combining Action Imitation with CSP	16
3.6	Agent 4 – Imitation on Node Level	17
3.7	Conclusions from the Imitation Learning Approaches	19
4	A SEMANTICAL APPROACH TO AGENT DESIGN	20
4.1	State and Action Representation	20
4.1.1	Moves	20
4.1.2	Game States	21
4.1.3	State–Action Mapping	22
4.2	Implementing the Agent Design	23
4.2.1	Database of Expert Games	24
4.2.2	Neural Network Classifiers for Force Groups	24
4.2.3	Implementing the Acting Module	25
5	EXPERIMENTAL RESULTS	26
5.1	The Red Opponent	26
5.2	Agent Performance	26
6	CONCLUSIONS	28
	Abbreviations	29
	References	30
	Distribution list	31





## LEARNING TO PLAY OPERATION LUCID FROM HUMAN EXPERT GAMES

### 1 INTRODUCTION

This report describes the application of machine learning techniques to the problem of making a software agent that plays a highly complex stochastic game. The game we consider, *Operation Lucid*, belongs to the class of two-person zero-sum perfect-information stochastic games. It has been designed as a simplified military land combat model, with rules representing central concepts such as movement (and uncertainty in movement), logistics, and of course combat itself, including the asymmetry between attacking and defending a location.

This work is part of FFI Project 806 “Machine Learning in Simulation”. Our studies are concerned with the application of artificial intelligence techniques to decision making in combat models, and in this research *Operation Lucid* is being used as an environment that captures the important general properties of such models, while allowing us not to get bogged down in unnecessary detail. The insights and results gained in this way can then be used in the development and improvement of full-scale combat models. Our previous work with this game is documented in (1), (2) and (3).

Game playing problems have been extensively studied in machine learning research. A number of papers describing state-of-the-art developments in this field are collected in (4); this reference also contains a survey of machine learning in games (5). However, in most of the games studied in this body of research, the main challenges are different from those posed by *Operation Lucid*, making several of the standard techniques useless for our problem.

If we regard *Operation Lucid* as a decision-making problem in a combat simulation context, related research is somewhat thinner on the ground. Recent work includes (6), in which a genetic algorithm is applied to a force allocation problem not entirely dissimilar to ours, and (7), which describes how a knowledge-intensive agent is used for evaluating military courses of action.

The report is organized as follows. Section 2 describes our problem domain. In Section 3, we describe a set of imitation learning approaches to dealing with the high complexity of the problem in order to make a game-playing agent. The shortcomings of these approaches inspired a different approach where the semantics of moves were taken into consideration; this is the subject of Section 4. Section 5 presents some experimental results of the agents described in the text, and Section 6 concludes the report.

## 2 THE GAME OF OPERATION LUCID

In this section we present our problem environment – the game of Operation Lucid – and describe some of the properties that make it both an interesting and very challenging problem. A fuller description of the game is given in (1).

### 2.1 Definition and Rules of Operation Lucid

In short, Operation Lucid is a two-person stochastic board game where the two players start off with their units in opposing ends of the board, as shown in Figure 2.1. One player, named Blue, is the attacker. Blue starts the game with fifteen units; his aim is to cross the board, break through – or evade – his opponent’s defence, and move his units off the board into the goal node. The defending player, Red, starts with ten units; his task is to hinder Blue from succeeding. The result of the game is the number of units that Blue manages to get across the board and into the goal node; thus, there is no “winner” or “loser” of a single game. The rest of this section describes the rules of the game.

The game of Operation Lucid is played in 36 turns. At the start of each turn, the right to move units is randomly given to either Blue or Red, with equal probabilities. The side winning this draw is allowed to move each unit to one of the neighbouring nodes (that is, to a node that is connected to the unit’s current node by an edge) or leave it where it is. The side losing the draw naturally does not get to move any units in that turn. The movement of the units is subject to two restrictions:

- When the move is finished, no node can have more than three units of the same colour.
- Pieces cannot be moved from nodes where they are defined as *attackers* (see below).

Whenever Blue and Red units are in the same location at the end of a turn, combat ensues, and one of the units in that node is lost and taken out of the game. A weighted random draw decides which side loses a unit. In a node having combat, the player last entering is defined as the *attacker* of that location, while the other part is defined as the *defender* of the location. The weighted random draw is specified by the probability that Blue wins, that is, that Red loses a unit. This probability is given by the fraction

$$(Blue\ strength)/(Blue\ strength + Red\ strength),$$

where a player’s *strength* in a node equals the number of own units in that node, modified in accordance to two rules. Firstly, the defending player in the node gains one extra point of strength. Secondly, if the Blue player does not have an unbroken path of nodes with only Blue units leading from the combat node to one of Blue’s starting positions (a *supply line*), Blue loses one point of strength.

The game ends when the 36 turns are completed, or when Blue has no units left on the board. The result of the game is the number of Blue units that have reached the goal node.

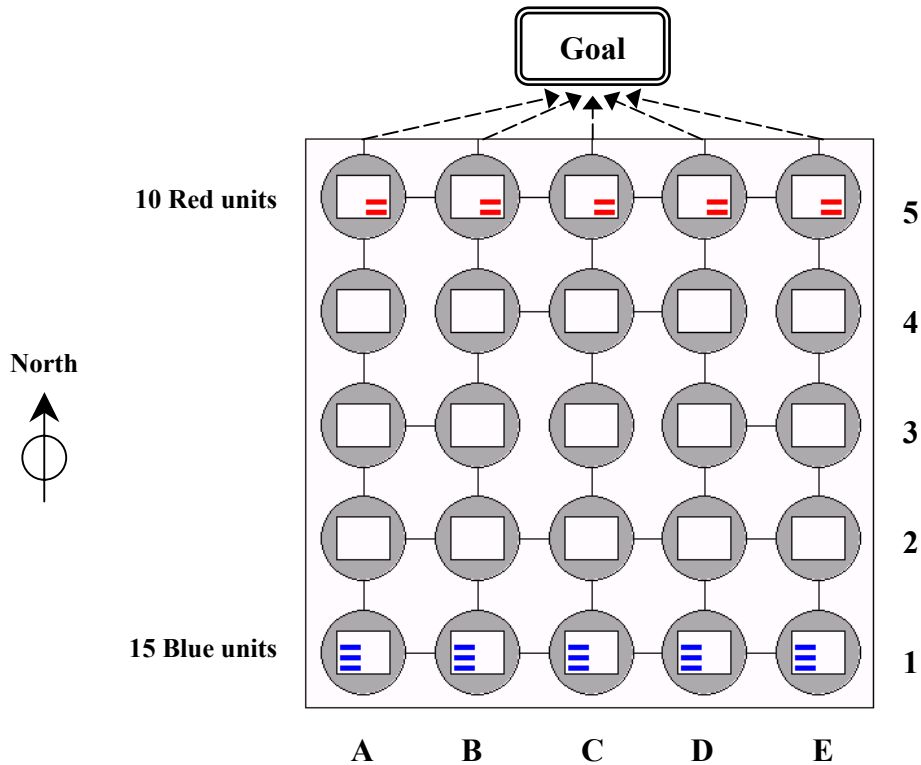


Figure 2.1 The board of the game Operation Lucid, with the units placed in their initial positions.

## 2.2 A Combat Modelling Interpretation of the Game

Operation Lucid was designed to capture important aspects of military land combat modelling; in particular, it represents a scenario where the goal of one side is to break through enemy defence to reach a certain location. Movement of the units on the board naturally represents the movement of force units in a terrain; the stochastic ordering of Blue and Red moves is intended to capture the uncertainty inherent in manoeuvring in possibly unfamiliar territory.

The rules for determining the result of combat naturally take into account the numerical strength of each side in the area of combat. In addition, they represent the advantage of being the defender of a location; this advantage is due to the defender's opportunity to prepare himself and the environs for resisting attacks. The rule regarding Blue supply lines models the effects of logistics; an invading force will need a functioning line of supplies back to its home base to be able to perform well in enemy territory.

## 2.3 The Complexity of the Problem

A seemingly obvious way of playing Operation Lucid is by evaluating (in some way) each legal move in the current game state, and then choosing the one with the best evaluation. This would reduce the problem of constructing a player agent to the problem of evaluating moves in

given game states. This method is generally not feasible, however, as the number of possible moves in each state tends to be huge. A player may allocate one of at most five actions (stand still or move in either of four directions) to each of at most fifteen units, so an upper bound on the number of legal moves is  $5^{15} \approx 3 \cdot 10^{10}$ . If we assume that a computer generates one thousand possible moves each second (a reasonable assumption according to our experience), it might take up to a year to enumerate all legal moves in one state.

In typical game states the number is usually far lower than this – the player may have fewer than fifteen units left, and each of the units may not be free to perform all five actions. Also, a lot of the legal moves are equivalent, as all units of the same side are interchangeable. From the initial position, for instance, the number of possible non-equivalent moves for Blue is 60,112 (disregarding equivalence by symmetry). In intermediate states of the game the number of legal moves increases quickly, and is generally far too large for an exhaustive enumeration. Thus, classical computer game-playing methods, based on enumerating and evaluating all legal moves, are infeasible in this domain.

How can the problem of the game’s high complexity be dealt with efficiently? In the next two sections, we describe our work on designing player agents that learn to play by learning from human expert play. Our efforts so far have mostly been focused on developing Blue agents, and this is the case in the present work as well. Building good Blue agents tends to be a more challenging task than building Red ones, since the nature of the game requires Blue to be the more creative and active side. However, it is usually a minor task to adjust the algorithms to fit a Red player as well.

### **3 IMITATION LEARNING APPROACHES TO AGENT DESIGN**

#### **3.1 Lazy Imitation Learning**

As explained above, the usual way of making computers play games, by generating all possible moves and evaluating which is the best, is not feasible in Operation Lucid. We therefore looked to the way humans play games in order to create good playing agents for the game. In our first main approach to learning from human playing, we took the idea of playing like a human literally, and designed agents that played by lazy imitation learning. The term *lazy* alludes to the fact that the agent does no work in trying to make explicit generalizing rules based on observed instances of good play. Instead, the instances are stored in a database for future retrieval and comparison with new instances. The task of classifying a new instance is thus deferred until the instance actually materializes, and the classification is based on the stored classification of one or more similar instances in the database.

This instance-based approach to learning has the advantage of not having to train some classification function to achieve a generalizing ability. The major downside is that repeated retrievals may in the long run be more time-consuming than training such a function.

The database was constructed by playing a series of expert games and logging all game states and corresponding expert moves from these games. Based on 20 expert games, the database features 632 entries of state-action pairs. The imitation agents use this database by first identifying the stored game state most similar to the one at hand, and then trying to imitate either the action taken in this state, or the state that resulted from this action. Schematically the database would look something like Table 3.1.







<b>Game state</b> <i>aka</i> <b>Reference state</b> <b>(RS)</b>	<b>Action</b> <i>aka</i> <b>Move</b>	<b>Resulting state</b> <i>aka</i> <b>Desired next-state</b> <b>(DNS)</b>
	a1-a2, a1-a2, a1-a2, b1-b2, c1-c2, c1-c2, d1-c1 ...	
	a2-b2, a2-b2, a2-b2, b2-b3, c2-c3, c2-c3...	
	b2-b3, b2-b3, b2-b3, b3-b4, c2-c3, d5-goal, ...	
etc.	etc.	etc.

Table 3.1 Database of game states and corresponding expert moves.

Some notes on terminology follow. The *current state* (CS) is any game state arising during play in which the agent wishes to imitate expert behaviour. The game state in the database most similar to the CS will henceforth be called the *reference state* (RS). The state resulting from being in the RS and taking the tabulated expert action is called the *desired next-state* (DNS). The state resulting from the imitation is called the *resulting next-state* (RNS). Furthermore, a *movelet* is the movement of an individual unit (e.g. a1-a2), while a *move* is synonym to the total action of a player – a move consists ergo of a collection of movelets. An illustration of the relation between these terms is given in Figure 3.1.

We constructed 4 agents of various degree of sophistication using this imitation learning approach; these agents are described in Sections 3.3–3.6. Common for most of these agents is that they use constraint satisfaction programming (CSP) techniques to generate possible moves. CSP, and a previously designed agent based on CSP, is briefly described in Section 3.2.

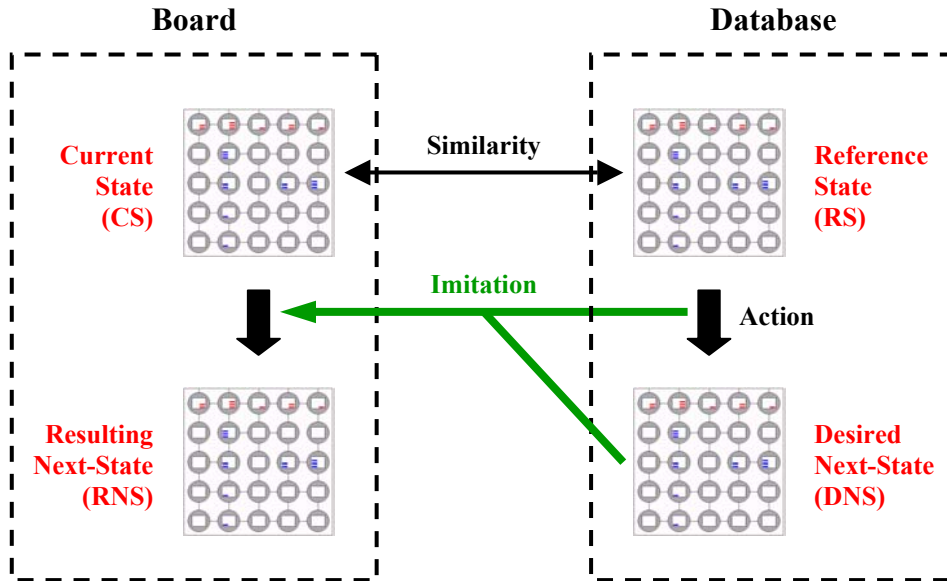


Figure 3.1 Schematic illustration of the imitation learning approach. The reference state (RS) is the state in the database most similar to the current state (CS). The agent imitates the action taken in the RS and/or the state resulting from this action (the desired next-state – DNS). The imitation leads to the resulting next-state (RNS).

### 3.2 Agent Design based on Constraint Satisfaction Programming

In one main approach to agent design we have previously followed, we kept part of the evaluative approach to game-play, but limited the number of moves to be evaluated to a tractable level. This was achieved by imposing constraints on the desired number of units in various areas of the board – these constraints could be seen as a representation of a plan for how to play. In case the constraints did not yield any feasible moves, the constraints were relaxed by reducing the number of desired units in an area. Continuing relaxation in this manner will eventually yield one or more possible moves that satisfy the constraints, and the actual move is then chosen by some evaluation method. This approach in general, and in particular a Blue agent that used self-trained neural networks for evaluating moves, is described further in (3).

In our imitation learning approaches, we used CSP techniques to impose a desired minimum number of units in each node for agents imitating the desired next-state. The constraints were defined by the actual number of units in each node in the DNS. The description of “Agent 2” in Section 3.4 explains in more detail how this constraining approach works.

### 3.3 Agent 1 – Action Imitation

The first agent we made simply imitates unit by unit the action taken in the RS. Should the action specify the movement of  $N$  units from node A to node B, the agent would try to do the same. Should the agent have more than  $N$  units present in node A, he would move only  $N$  units

and leave the remaining untouched. Should the agent on the other hand have fewer than  $N$  (or even none) units in node  $A$ , he would move the units available, and discard the rest of the desired movelets from node  $A$ . Should any node contain more than three units at the end of a move, the perpetrating movelet(s) is simply cancelled. Moreover, units already moved once naturally cannot be moved further, and does not count as movable units in their new node. Movelets are performed (and cancelled if necessary) in the same order as they occur in the database, giving a certain bias to the movement, but this is not considered to be harmful.

The example in Figure 3.2 illustrates this action imitation agent. The purpose of the example is to illustrate the *principle* of imitation, and therefore only a small generic subset of connected nodes is displayed. The action taken in the RS – two units north and one unit east – leads to the DNS. This action serves to forward units as much as possible along the left axis, while keeping the supply line intact. When the same action is applied to the CS, the agent ends up in the RNS. In the example, the three movelets desired are all reproducible in the CS, and they do not result in an illegal state; hence they are all performed.

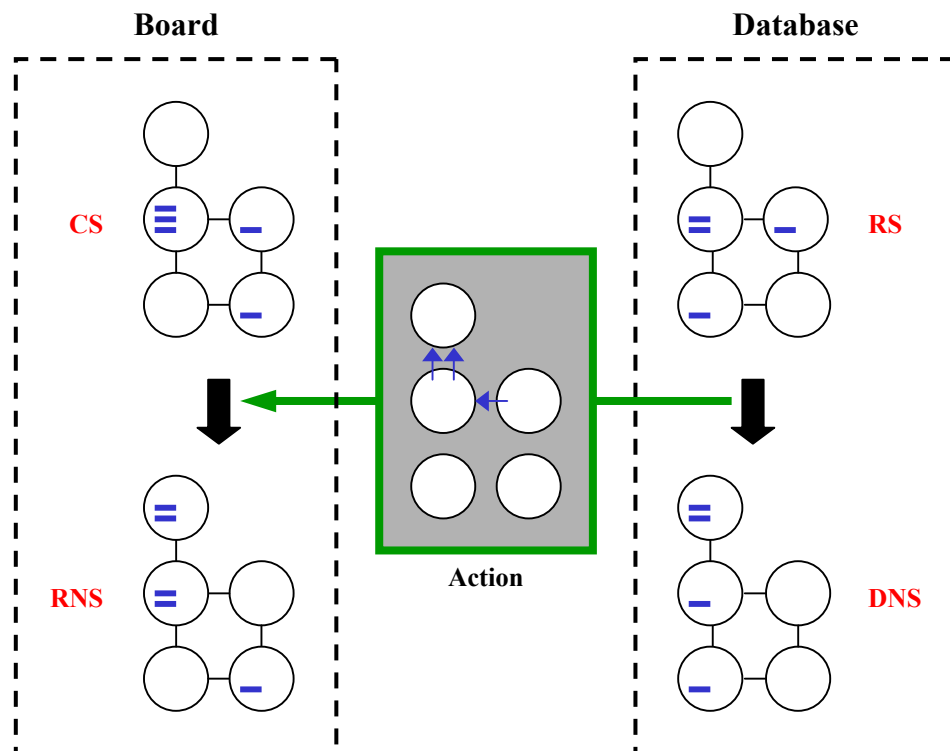


Figure 3.2 Illustration of action imitation. The action taken in the RS is applied to the CS, resulting in the RNS.

This imitation strategy has the obvious drawback of leaving units idle in all the nodes where the RS has fewer units than the CS. This is clearly demonstrated in the example. This leads to an unfortunate scattering of the pieces, ultimately resulting in unconcentrated attacks and broken supply lines. As the game goes along, and close matches in the database become hard to find, this may affect a considerable number of units on the board.

### 3.4 Agent 2 – State Imitation Using CSP

Realizing that the action taken in the RP is only a means for reaching the DNS, it might be better to imitate the DNS directly rather than imitating the actions themselves. This approach would to some extent solve the problem of units left idle, since the movement of a unit into a node no longer has to take place as specified by the action, but can come from any of the neighbouring nodes. Our second agent imitates the DNS by using CSP, where the constraints imposed are the number of units in each node in the DNS. Of course, more often than not, it is impossible to satisfy exactly this strict set of constraints, and constraint relaxation is then performed gradually until a feasible set of constraints is obtained. This set of constraints would then be satisfied by a usually small collection of candidate actions, and a self-trained neural network evaluator chooses the best action among these candidates.

The example in Figure 3.3 illustrates this state imitation agent. This is the same example as before. The constraints are defined by the DNS, and are satisfied by two feasible RNSs. In this case the constraints are immediately satisfied, so no constraint relaxation is necessary.

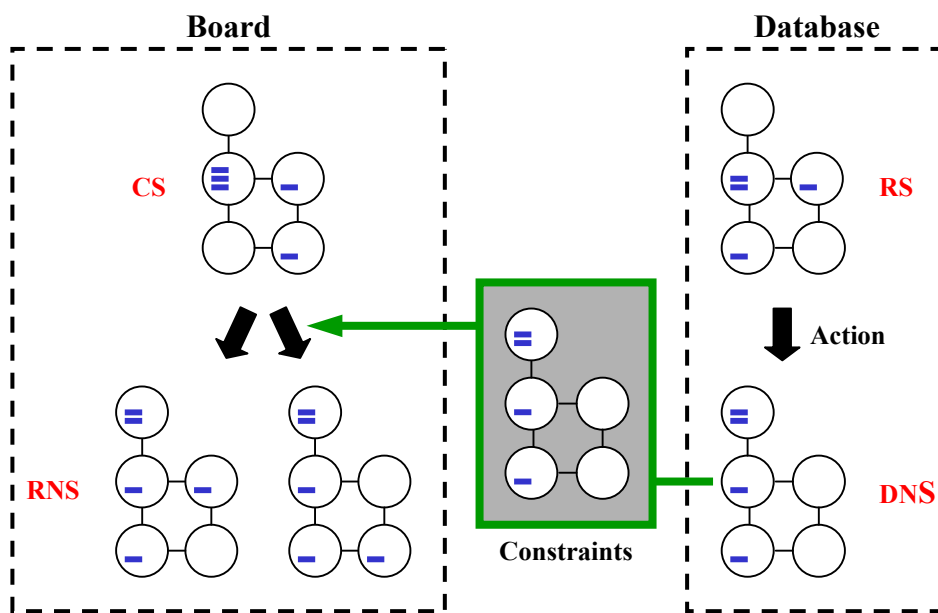


Figure 3.3 Illustration of state imitation. The DNS defines the set of constraints, which are relaxed until they are feasibly satisfied by one or more candidate moves. The final RNS is chosen by an evaluation function.

Constraints are relaxed by reducing the number of required units in nodes. Only *one* unit in *one* node is relaxed at a time; if the constraints are still too strict, a unit in another node is relaxed. This continues through all the nodes, starting over again if necessary, until at least one move can be made that satisfies the current set of constraints. The nodes are visited in a fixed order.



An example to clarify this procedure follows. Taking the board position in Figure 3.4 as our example DNS, the initial set of constraints would be one unit in each of the nodes B1 and B2, two units in node B3, three units in node B4 etc. Continuous relaxation would give the sequence of constraints as shown in Table 3.2. In the table, only the nodes involved in the constraints are shown on the far left.<sup>1</sup> The sequence of columns shows the gradual development in the set of constraints upon relaxation. The entries are the desired minimum number of units in the nodes. Grey slots indicate nodes where a relaxation has just been carried out. Empty slots indicate that the actual node is no longer among the constrained nodes. Note the fixed pattern in which the nodes are traversed.

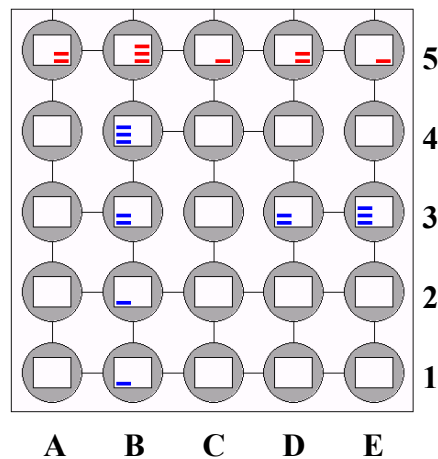


Figure 3.4 Example of a DNS defining the sequence of constraints given in Table 3.2.

Node	Set of constraints												
	1	2	3	4	5	6	7	8	9	10	11	12	13
<b>B1</b>	1												
<b>B2</b>	1	1											
<b>B3</b>	2	2	2	1	1	1	1						
<b>B4</b>	3	3	3	3	2	2	2	2	1	1	1		
<b>D3</b>	2	2	2	2	2	1	1	1	1				
<b>E3</b>	3	3	3	3	3	3	2	2	2	2	1	1	

Table 3.2 Example showing the development in the set of constraints imposed to reach a DNS like the one in Figure 3.4. Only the nodes involved in the constraining are listed. The grey slot in each set of constraints indicates the node whose constraint has just been relaxed.

<sup>1</sup> Recall that a constraint dictates the *minimum* number of units desired in a node, so that a constraint of zero units in a node is really no constraint at all.

Three main problems with this approach were identified. First, the fixed order of constraint relaxation could easily prove to be unfortunate. For example, one can imagine the set of constraints not being met due only to *one* strict constraint occurring late in the set. Several other constraints are thus unnecessarily relaxed waiting for the correct one to be encountered; it might even be the case that several sweeps through the set are required before this obstructing constraint is sufficiently relaxed. All relaxation is in principle unfortunate, since it distorts the agent's comprehension of the DNS. Misconception of the DNS could make the agent neglect performing movelets required by the DNS, thereby making the imitation less precise. Relaxation should therefore be kept at a minimum, and ideally be applied only where needed, i.e. at the bottlenecks in the set of constraints. Identifying these bottlenecks and sorting the constraints accordingly would of course alleviate this problem, but this was not done at this stage of the work.

Second, the fixed order of constraint relaxation (A to E and 1 to 5) introduced a certain bias to the procedure, as nodes in the west and south are relaxed more often than nodes in the east and north. Consequently the agent will consistently be less strict with the manoeuvring in the western and southern areas. This will probably not affect the end result much, but in hindsight this is nevertheless a methodological weakness. Randomizing the relaxation sequence would of course be an easy fix to remove the bias.

The third main drawback is, given a collection of candidate moves meeting the constraints, that the evaluator's choice of move might not be the optimal one. Evaluator imperfectness is a general limitation in problems like these. In our case a self-trained neural network evaluator was used (trained by temporal difference learning, see (2)), and although a fairly good evaluator, it has considerable room for improvement.

### **3.5 Agent 3 – Combining Action Imitation with CSP**

Our third imitation agent combines the main ideas of the first two agents. Like Agent 1, it first performs all the relevant movelets specified by the action taken in the RS. This results in an intermediate board position, and the agent then applies CSP on this position trying to imitate the DNS. Since several units are moved in the first step, and hence cannot be moved further, the CSP problem is reduced to handling only the units not yet moved. This is a far smaller task, and the risk of unnecessary relaxation is drastically reduced, thereby increasing imitation accuracy. Moreover, the movelets already made ensure a flying start towards an optimal imitation. This agent too, however, has the unfortunate tendency of scattering its units over a too wide area, and is not too good at keeping a supply line.

The example in Figure 3.5 illustrates this combination agent. This is the same example as before. The RS action is applied to the CS, and the constraints defined by the DNS are imposed on the intermediate position. No constraint relaxation is necessary in this case, and two candidates for the RNS are evaluated.

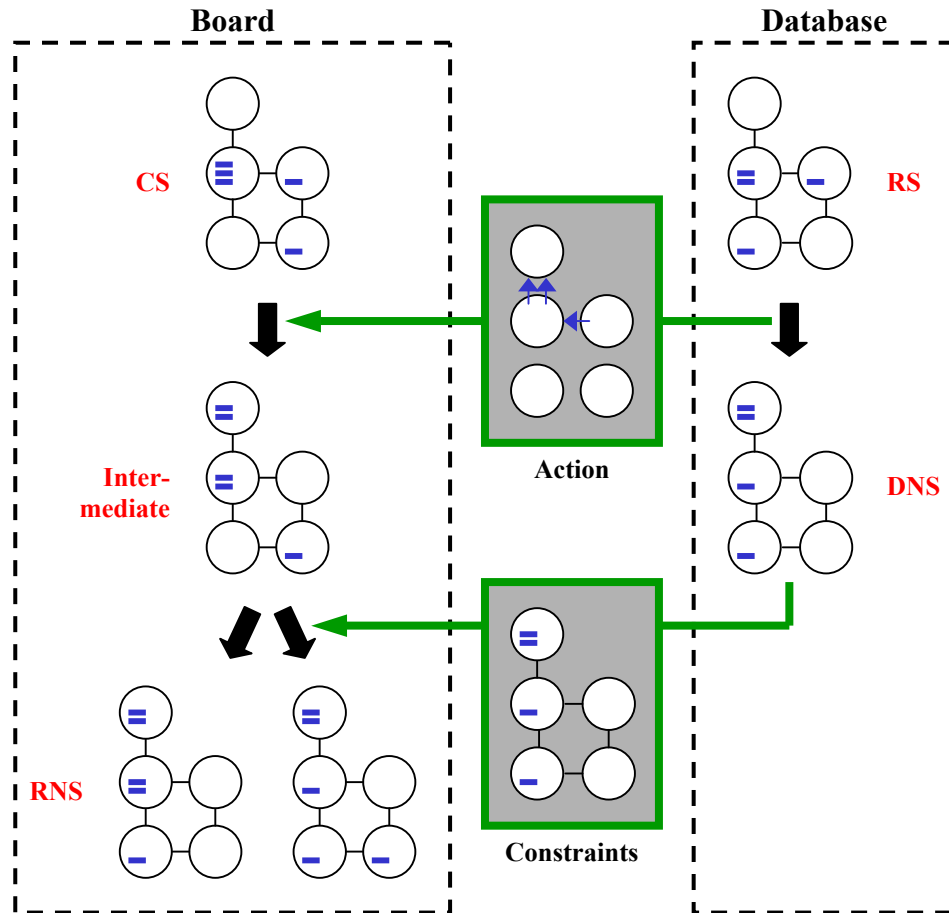


Figure 3.5 Illustration of action and state imitation. The RS action is imitated, resulting in an intermediate position. The DNS defines the set of constraints, and CSP is applied to the intermediate position yielding one or more candidate moves. The final RNS is chosen by an evaluation function.

### 3.6 Agent 4 – Imitation on Node Level

A major problem identified with all the three first approaches, is that units frequently are left untouched instead of participating in the move. To cope with this, we made a fourth agent that has a somewhat aggregated view on unit movement, where actions are seen to be taken on the node level rather than the unit level. This means that if the RS moves two units from node A to Node B, thereby leaving Node A empty, the most important feature of this move is the emptying of Node A, and not the moving of two units exactly. Moreover, if two units are moved out and one is retained, the most significant feature of this move is the retaining of the one unit.

The main rule of movement is that the movement in the CS should proportionally mirror the movement in the RS. This ensures that units acting in unity in the RS would also act in unity in the CS, regardless of differences in number. If the units in the RS split up, the node in the CS would mirror this split by a proportionality vote over where to move its units; ties are broken

by a random draw. One special rule applies: a split resulting in the retaining of only one unit is seen as an act of ensuring supply, and the CS would reflect this by also retaining only one unit. The remaining units in the CS are then moved in proportion to the movement of the remaining units in the RS.

This agent is otherwise equal to Agent 3 described above, using CSP on the remaining units after the (modified) RS action have been taken. The example in Figure 3.6 illustrates this combination agent. This is the same example as before. The (modified) RS action is applied to the CS, and the constraints defined by the DNS are imposed on the intermediate position. No constraint relaxation is necessary in this case, and only one candidate RNS meet the constraints.

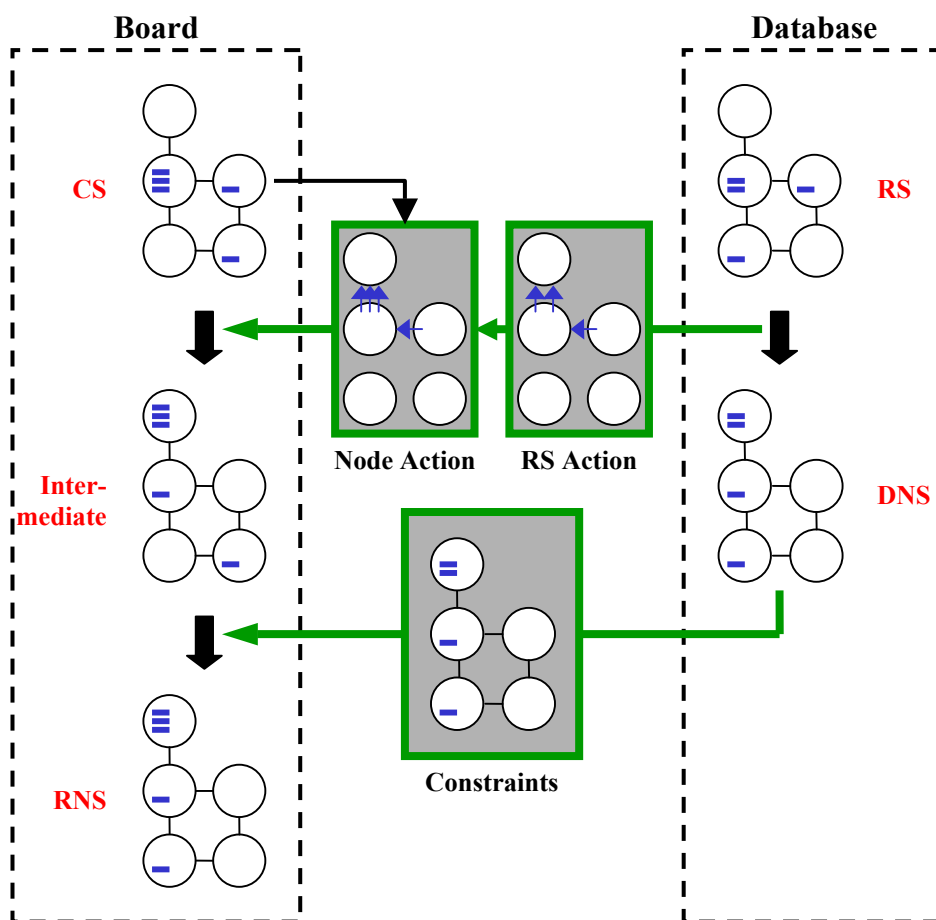


Figure 3.6 *Illustration of node level imitation. The RS action is modified with respect to the actual content of the nodes in the CS. The modified action is imitated, resulting in an intermediate position. The DNS defines the set of constraints, and CSP is applied to the intermediate position yielding one or more candidate moves. The final RNS is chosen by an evaluation function.*

### 3.7 Conclusions from the Imitation Learning Approaches

Although the four agents presented so far show increasing performance, our work with this kind of pure imitation learning produced rather disappointing results. The main reason for this is that the player very quickly finds itself in game states with no very close match in the expert database, so that the move taken in the most similar state (RS) is not sufficiently applicable in the current game state (CS). It was realized that this problem is due to the fact that similarity, both of states and of moves, are seen at what can be called a *syntactical* level. At this level, the actual position and movement of the units in each single node are the basis of the agent's behaviour, without any semantical notion of the role of the units in the game. The result is that this syntactical imitation will often require some units to perform unfeasible moves, while other units are left idle.

One obvious fix to this problem would be to increase the size of the expert database, hoping to cover a greater range of board positions. But this approach requires man-hours, and in our case doubling the database showed only marginal improvement during play at the cost of greatly increased runtime.

Another challenging aspect of the imitation learning approach was to define a suitable distance metric for comparing states in order to choose the most similar one in the lookup table of expert games. The distance metric adopted here is based on the dissimilarity of node content between the CS and the RP. For each node, the number of blue units is counted in both the CS and the RP, and the difference is added to the accumulated distance. The same is done for the red side. In addition, the difference between remaining rounds is also added to the distance, as well as a penalty for differing defender status in case of combat. Iterating through the complete lookup table of expert games, the tabulated state with the smallest distance to the CS is chosen for RP.

This choice of distance metric is a rather simple one, in that it assigns equal weight to all node level differences in the position. If one is able to identify nodes where differences are more crucial to the position than is the case with other nodes, one could imagine increasing the weight of the contribution from these nodes, thereby tuning the distance measure to be more representative for the position. There is of course the challenge of deciding upon which nodes these should be, and how heavy the weighting should be. Similarly there is a tuning possibility in varying the weight of the contribution from the difference in remaining rounds, as well as the weight of the contribution from differing defender status in case of combat.

These possibilities for variation of the distance metric were not explored in great depth, since initial trials failed to produce positive results, and since a new and improved approach to the problem was developed. This approach is the topic of the next section.

## 4 A SEMANTICAL APPROACH TO AGENT DESIGN

### 4.1 State and Action Representation

Humans generally do not test all available moves; rather, we decide on a goal, and form a plan we believe will help us reach this goal. Our experiments with pure imitation learning showed that the semantics of the problem domain would have to be addressed to some degree in order to achieve good and efficient performance. One way of doing this could be by following the methodology used in case-based reasoning (see e.g. (8)), where retrieval of previous cases from a database is combined with symbolic reasoning based on a semantic model of the domain<sup>2</sup>. Case-based reasoning has previously been applied to games such as chess (10) and Othello (11), as well as many real-world problems. The main disadvantages of this approach is that constructing such a domain model is a difficult and time-consuming task, and that frequent case retrievals – as is the case in game playing – may be costly in terms of runtime.

These considerations led us to the conclusion that we required an agent design that imitates the expert moves on a more abstract level than single positions and movements of units, while not depending on a full semantic model of the game. In the following, we describe how this was accomplished in the case of moves and game states respectively, and explain how the agent chooses its moves based on the expert games.

#### 4.1.1 Moves

As with simple imitation learning, this new approach to handling the complexity of the problem is to have an agent capable of mapping directly from game states to moves. In order to make this work well, the action space must be reduced considerably in size. To this end, we collapsed the action space from the syntactical level consisting of all possible combinations of single-unit moves, to a semantical level featuring only a dozen move categories. The move categories are symbols (e.g. *attack*, *outflank*, *breakSupply*) describing the overall character or *intent* of the move. Furthermore, the move categories are not disjunctive – they are defined so that a move may be classified into more than one category.

The move categories are given in Table 4.1, along with a brief description of their meaning. The actions are linked to the notion of *force groups* (FGs), which is explained in the following section.

---

<sup>2</sup> We follow the terminology of Mitchell [9], where the term “case-based” is reserved for a subset of the broader set of “instance-based” methods, namely those using richer instance representations than simple feature vectors.

Category	Description
approach	Move units closer to the goal node
ensureSupply	Keep some units back, with the intent of ensuring a present or future supply line
breakSupply	Break the supply line, i.e. advance units previously withheld for supply purposes
attack	Move units into a node containing only Red units
reinforceCombat	Move additional units into a combat node
continueCombat	Neither exit from nor reinforce existing combat
outflank	Perform an evading manoeuvre, sideways or backwards, aiming at a new exit node
goToGoal	Move units from the northernmost row into the goal node
concentrateForces	Move units within a FG closer, i.e. occupying fewer nodes
splitIntoGroups	Divide a FG into two or more FGs
linkUpGroups	Join a FG to another, creating a larger FG
stayInPosition	Leave all units in the FG unmoved

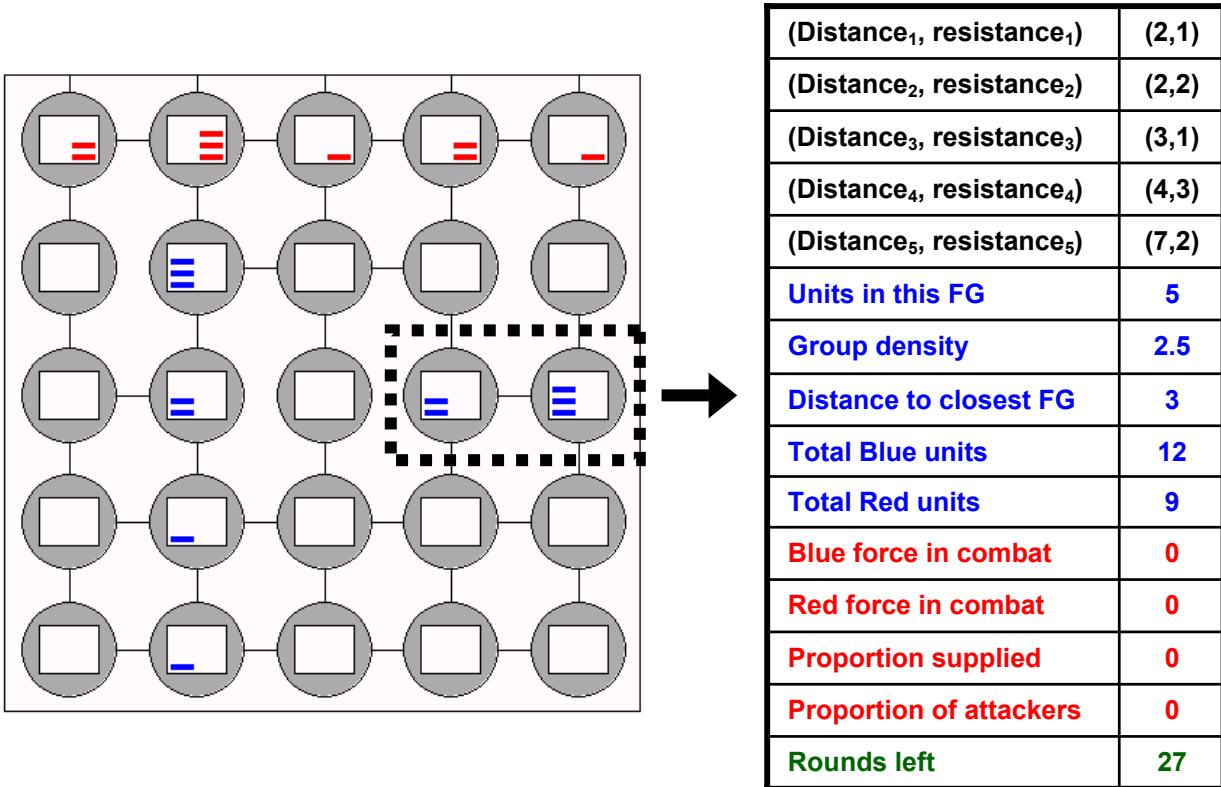
Table 4.1 The twelve move categories for Blue force groups.

#### 4.1.2 Game States

Reducing the action space in the manner described above raises another question: which units are to perform which types of action? Different units in a given expert game may be used for different intentions; this made it desirable to work with sub-collections of units sharing common intentions, rather than with all the units collectively. We call these sub-collections sharing common intentions *force groups* (FGs). Common intentions usually coincide with co-location of units on the board; this led us to define a FG as a collection of units of the same colour that are interconnected with each other but not with other friendly units. The units in a FG act together pursuing the intentions of the group, and should differing intentions occur within a FG, it will break up, forming smaller FGs individually regaining conformity of intentions.

The main advantage of introducing the concept of FGs is that it allows a suitable abstraction of the state space. In all our previous approaches, the game state has been represented by the number of units for each side in each individual node, the number of remaining rounds and which side defended each combat node. We now employ a FG-centric state representation where individual units and nodes, even the board geography itself, are no longer of direct interest. What is of interest is a set of aggregated features like path lengths to the goal node, resistance along these paths, the number of own units in the FG and in total, the number of enemy units, distance to neighbouring FGs (if any), combat strength of the FG (including supply and defender status) and remaining rounds. Each FG in a game state has its individual,

location-independent state perception, on which it bases its actions. This means that FGs located on different parts of the board are considered similar if their environs are similar in terms of the FG-centric state representation. The representation we use is illustrated in more detail by the example in Figure 4.1.



*Figure 4.1 Representation of the game state as seen from the perspective of a force group. The ten first attributes give the distance to each of the five exit nodes, together with the number of Red units on the path to each node. These are sorted in ascending order. The remaining ten attributes describe own and opposing forces, the combat situation, and the number of rounds left.*

### 4.1.3 State–Action Mapping

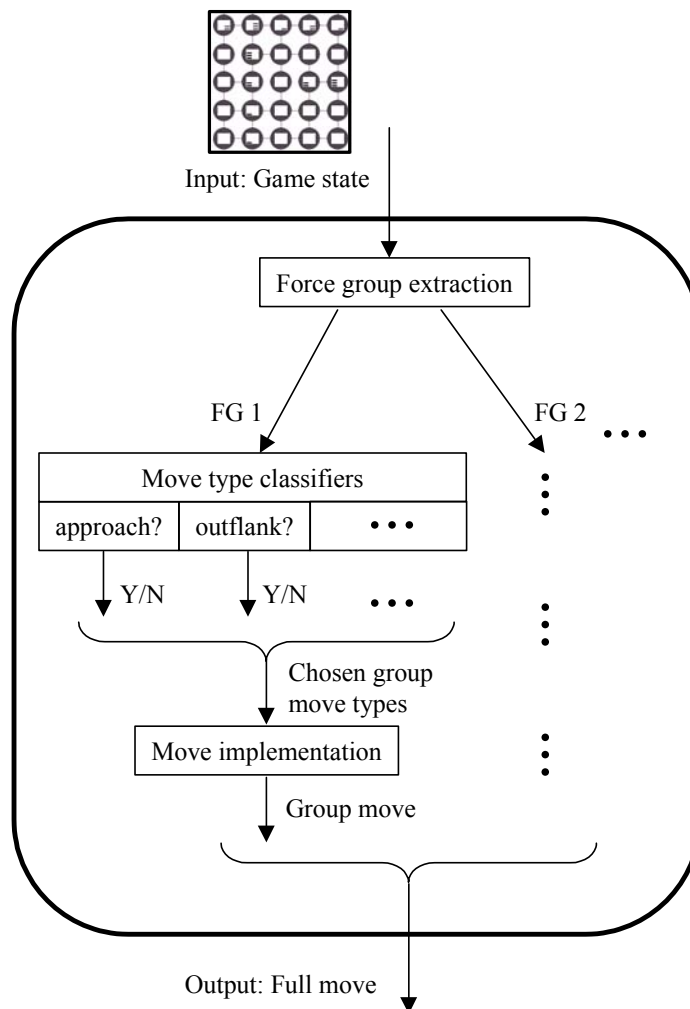
Using the state and move representation just presented, we could have repeated the lookup-table method described in Section 3.1 for mapping observed game states into actions to perform. The problem of finding a good distance metric for comparing FG states in order to choose the most similar one would then still remain. Instead, we chose to abandon the lazy-learning design altogether, and used the expert games to train a set of neural network classifiers for state–action mapping. In this way, the mappings implicit in the expert games may also generalize better to new FG states; an added advantage is that a full scan through the database at each decision point is no longer necessary, so that runtime is decreased. A more detailed look at the making of these classifiers, along with the rest of the player agent, is the subject of the next section.



## 4.2 Implementing the Agent Design

Our goal is to construct game-playing software agents. Such an agent should be able to get a game state as input, and, from this state and the rules of the game, generate a move as output. The move describes where each of the own units should be placed when the turn is finished.

In accordance with the design described in the previous section, our agent selects its move as illustrated in Figure 4.2. Upon receiving the current game state, the agent identifies its FGs and gives each FG a self-centric and simplified perception of the game state. Each FG then uses the move type classifiers in conjunction with this state representation in order to select which class (or classes) of moves is appropriate in its current situation. Finally, the agent should of course be able to translate these pieces of semantical move advice into actual board movement of the units in a suitable way.



*Figure 4.2 Agent design. The agent identifies its force groups (FG), and the collection of move classifiers decide which semantic actions each FG should take. The move implementation module translates the specified actions into actual unit movement.*

The remainder of this section details the steps involved in building the agent. In the following, *game state* or simply *state* refers to the overall game state (i.e. positioning of units, rounds left and attacking sides in the case of combat), while *FG-state* refers to the simplified, egocentric state perception of each FG. Similarly, *move* refers to the collective action of all Blue's units in a turn, while *FG-move* refers to the action taken by a single FG only.

#### 4.2.1 Database of Expert Games

The database of expert games was constructed based on the same series of games that were used in the imitation learning approaches. This time around, the states and expert actions were stored with the new state and action representation, so a new manual classification sweep through the expert games was therefore necessary. On each of Blue's turns in these games, one or more FGs would be present, and the expert categorized his corresponding FG-level moves into one or more of the twelve move categories according to the intention of the move. The total number of Blue FG-states (not necessarily distinct) throughout the game series was 455. Each FG-state was stored along with its corresponding expert semantic action. Schematically the database would look something like Table 4.2.

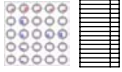
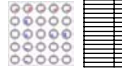
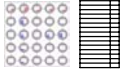
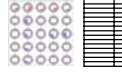
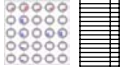
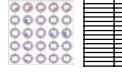
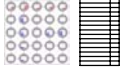
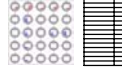
FG-state		
	approach, splitIntoGroups	
	outflank	
	approach, ensureSupply	
	attack, ensureSupply	
etc.	etc.	etc.

Table 4.2 Example of database of FG-states and corresponding expert semantic actions.

#### 4.2.2 Neural Network Classifiers for Force Groups

Having assembled the database of FG-states and corresponding expert semantic FG-level actions, we trained an ensemble of neural networks (NNs) to serve as FG-state classifiers for the agent. One NN was trained for each action category, using the database of FG-states as input data and the presence (0 or 1) of the category in question as target values.

Each network was a standard feedforward NN featuring 20 input nodes, 36 hidden nodes and 1 output node (ranging from 0 to 1), sigmoid activation functions, and weights initially

randomized from  $-0.2$  to  $0.2$ . Training was done by back-propagation. The input vector for each FG-state was scaled so that the magnitude ranges of the components were similar. About  $1/3$  of the data set was initially reserved for validating the training procedure, and the NNs were trained by repeatedly picking random examples from the training data.

The learning of the NNs was evaluated by the proportion of correctly classified examples over the validation data set. The ensemble quickly attained a classification performance of  $0.9$ , and after further training reached about  $0.95$ . We noted that performance on the validation set did not start to decrease, even if training was continued. Taking this as an indication that the data set presented little danger of overfitting – even with the large number of network weights used – we restarted training using all available data. The total classification accuracy on the full training set reached about  $0.99$ ; the individual action-specific NNs showed minor deviations from this average.

As explained above, the trained NNs are used in the game-playing agent. In a given game state, each FG inputs its FG-state into the twelve nets, each of which answers a number between  $0$  and  $1$ . Output close to  $1$  indicates that the FG should perform a move corresponding to the category in question. The output value of  $0.5$  was used as the limit for choosing move categories.

In this procedure, the decision-making task of the player agent can be regarded as delegated to its constituent FGs, which choose and perform actions based on their own perceptions of the game state. Adopting this view, the ensemble of neural networks may be interpreted as a shared overall doctrine for how to act in given situations.

#### 4.2.3 Implementing the Acting Module

With the classification module of our agent properly in place, we turned to the task of actually designing and implementing the acting module. This module receives as input one or more chosen move categories for each FG, and returns the resulting movement of the individual units in the FG.

Due to space restrictions, we are unable to go into details of this module here. Instead, we mention a number of difficulties that arose in the implementation of this part of the agent. In particular, the set of chosen move categories may be inconsistent, in which case not all of the move types may be performed, e.g. if both *breakSupply* and *ensureSupply* are chosen. Another inconsistent set of move types is the empty set – since we have specified *stayInPosition* as a category of its own, and this category was not selected, this is not an order to simply stand still.

In cases where more than one move category is specified, we must decide which units should move according to which categories, or alternatively which categories should be given precedence. At the current stage, we use the simple strategy of using the numerical outputs of the respective neural nets for ranking the categories. Units are moved according to the first category, and if after this some units have not been assigned to an action, moves for the next

category is implemented, and so on. The problem with empty move sets mentioned above was dealt with by defining the *approach* category as a default action; this category was chosen because advancing across the board is a reasonable baseline course of action for Blue – at any rate, it is almost always better than standing still.

## 5 EXPERIMENTAL RESULTS

### 5.1 The Red Opponent

The Red opponent employed in the expert games, named *AxesRed*, is an automatic playing agent adopting two main strategies of play. Firstly, it will never advance from the home row (i.e. the northernmost row) into the field, but stay home and wait for Blue to attack. Secondly, it attempts to position its units within the home row in a manner that proportionally mirrors the perceived threat on each vertical axis of nodes. For instance, if Blue has 13 units left, and 4 of these are located on the B axis (see Figure 2.1 for references to parts of the board), Red will to the best of his ability attempt to position 4/13 of his remaining units on this axis, i.e. in B5. A few simple rules apply to ensure that Red keeps a sound level of play in special cases. Red can of course not have more than 3 units in any node, and should its calculations require more than this, it will ensure that backup units are kept in the neighbourhood. Red will not reinforce in a combat node – this entails losing the defender’s advantage – if this is not more than compensated for by the strength of the extra units. Although *AxesRed* is a rather simple player, it has proved to serve well as a benchmark opponent for evaluating Blue agents.

### 5.2 Agent Performance

We measure the success of the four imitation agents and the semantic agent by the average score (number of units that reach the goal node) against the *AxesRed* opponent. Therefore, we need to have an idea of what constitutes a good result when playing against this particular Red opponent. The result from the expert games is a natural measure of the potential of our agents; after all, it is this expert behaviour we are trying to learn from.

What then is a bad result? This is difficult to say, but we can at least get a notion of a mediocre result by letting some rather naive Blue players challenge the *AxesRed* player. Two such benchmark Blue players have been designed. The first, *SimpleBlue*, employs the simple strategy of moving all its units forward when receiving the turn. This results in a full-breadth simultaneous attack, where three Blue units take on two Red units in each of the northernmost nodes. The second player, *OneAxisBlue*, initially decides upon an axis of attack, and advances as many of its units as possible along this axis for the rest of the game. This results in a focused attack on one of the northernmost nodes. Neither of these two players actively keeps a supply line, although the design of the game ensures that *OneAxisBlue*’s supply line happens to be intact in the first phase of the attack.

Furthermore, it is interesting to compare the performance of the new agents with that of the best Blue agent we have managed to make during our previous work. This agent, called *ConstraintNNBlue*, is the constraint-based agent with NN move evaluation described in Section 3.2. The average results obtained by the four imitation agents and the semantic agent are given in Table 5.1. The table also shows the results for the human expert, the two benchmark agents and *ConstraintNNBlue*. For each automatic agent, 1000 games were played; the human played 20. Standard deviations and 95% confidence intervals are also reported, as well as the approximate average runtime per game.

As we can see, the four imitation agents show increasing performance. Their results are disappointing, however, as the best of them does not even beat the brute benchmark player *OneAxisBlue*. The semantic agent outperforms the two benchmark players and the imitating agents, but still has some way to go to reach the human expert – this latter fact is nothing more than could be expected<sup>3</sup>. Comparing the agent to *ConstraintNNBlue* shows that it fails to set a new record; on the other hand, it is not discouragingly far behind, while being almost an order of magnitude quicker. Moreover, we expect the agent to have a substantial potential for improvement within the limits of the current design; a larger database of expert games and better move implementations are two of the more obvious measures that can be taken.

Player	Score	StdDev	95% C.I.	Time (s/game)
Human	9.47	1.80	8.68 - 10.26	–
SimpleBlue	3.88	1.07	3.81 - 3.94	0.2
OneAxisBlue	5.34	1.60	5.24 - 5.44	0.4
ConstraintNNBlue	6.60	1.57	6.51 - 6.70	90
Imitation Agent 1	1.97	2.05	1.84 - 2.10	9
Imitation Agent 2	4.04	2.10	3.91 - 4.17	319
Imitation Agent 3	4.15	2.12	4.02 - 4.28	16
Imitation Agent 4	4.53	2.17	4.40 - 4.67	15
Semantic agent	6.28	2.13	6.15 - 6.41	12

*Table 5.1 Average score with standard deviation and 95% confidence interval for various Blue agents playing against AxesRed. Approximate runtimes are given. The results are based on 1000 games per player (human: 20 games).*

<sup>3</sup> Indeed, our experience with the game leads us to suspect that the human must have been rather lucky in these 20 games to achieve this score.

## 6 CONCLUSIONS

Operation Lucid is a highly complex stochastic game. The complexity of the game makes it infeasible to use standard game-playing methods to design playing agents. We have presented two approaches to agent design based on learning from a database of human expert games.

The first approach produced four agents of increasing ability that imitated similar expert states and actions using lazy learning. However, the performance of these agents was rather disappointing. The main reason for this is that imitation was done on a syntactical level, i.e. based on the position and movement of individual units on the board. During play, the agents very quickly ended up in states with no close enough match in the database, resulting in unwanted scattering and idling of units.

The shortcomings of the lazy imitation agents inspired a second approach to agent design based on the expert games. This approach handled the game complexity by collapsing the huge action space into a few move categories representing the semantical intentions of moves, and by representing the game states of subsets of the agent's playing units by a few relevant features. The database of human expert games was used to train a collection of neural networks to classify states into semantic actions. This abstracted approach to learning showed encouraging results, and was a remarkable improvement with respect to the imitation agents. The agent did not outperform the best of our previously designed agents. It came very close, however, and was almost an order of magnitude quicker than the record holder.

Both approaches to agent design based on human expert games have generated valuable insight into the techniques used and to the way we think around the game. This is a fuzzy conclusion, but keep in mind that it is not the problem itself that is interesting to us, but rather what the problem can offer in terms of insight that can be used in real simulation models.

**Abbreviations**

C.I.	-	Confidence Interval
CS	-	Current State
CSP	-	Constraint Satisfaction Programming
DNS	-	Desired Next-State
FG	-	Force Group
NN	-	Neural Network
RNS	-	Resulting Next-State
RS	-	Reference State
StdDev	-	Standard Deviation

## References

- (1) Dahl, F.A., Halck, O.M.: Three games designed for the study of human and automated decision making. Definitions and properties of the games Campaign, Operation Lucid and Operation Opaque. FFI/RAPPORT-98/02799, Norwegian Defence Research Establishment (FFI), Kjeller, Norway (1998).
- (2) Sendstad, O.J., Halck, O.M., Dahl, F.A., Braathen, S.: Decision making in simplified land combat models – On design and implementation of software modules playing the games of Operation Lucid and Operation Opaque. FFI/RAPPORT-2000/04403, Norwegian Defence Research Establishment (FFI), Kjeller, Norway (2000).
- (3) Sendstad, O.J., Halck, O.M., Dahl, F.A.: A constraint-based agent design for playing a highly complex game. In: *Proceedings of the 2<sup>nd</sup> International Conference on the Practical Application of Constraint Technologies and Logic Programming (PACLP 2000)*, The Practical Application Company Ltd (2000) 93–109.
- (4) Fürnkranz, J., Kubat, M. (eds.): *Machines That Learn to Play Games*, Nova Science Publishers (2001).
- (5) Fürnkranz, J.: Machine learning in games: A survey. In: Fürnkranz, J., Kubat, M. (eds.): *Machines That Learn to Play Games*, Nova Science Publishers (2001) 11–59.
- (6) Schlabach, J.L., Hayes, C.C., Goldberg, D.E.: FOX-GA: A genetic algorithm for generating and analyzing battlefield courses of action. *Evolutionary Computation* 7 (1999) 45–68.
- (7) Boicu, M., Tecuci, G., Marcu, D., Bowman, M., Shyr, P., Ciucu, F., Levcovici, C.: Disciple-COA: From agent programming to agent teaching. In: Langley, P. (ed.): *Proceedings of the 17<sup>th</sup> International Conference on Machine Learning (ICML-2000)*, Morgan Kaufmann (2000) 73–80.
- (8) Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7 (1994) 39–59.
- (9) Mitchell, T.M.: *Machine Learning*. WCB/McGraw-Hill (1997).
- (10) Kerner, Y.: Learning strategies for explanation patterns: Basic game patterns with application to chess. In: Veloso, M., Aamodt, A. (eds.): *Proceedings of the 1<sup>st</sup> International Conference on Case-Based Reasoning (ICCBR-95)*. Lecture Notes in Artificial Intelligence Vol. 1010, Springer-Verlag (1995) 491–500.
- (11) Callan, J.P., Fawcett, T.E., Rissland, E.L.: CABOT: An adaptive approach to case-based search. In: *Proceedings of the 12<sup>th</sup> International Conference on Artificial Intelligence*, Morgan Kaufmann (1991) 803–809.



## DISTRIBUTION LIST

**FFISYS**      **Dato:** 8 September 2003

RAPPORTTYPE (KRYSS AV)		RAPPORT NR.	REFERANSE	RAPPORTENS DATO	
<input checked="" type="checkbox"/> RAPP	<input type="checkbox"/> NOTAT	<input type="checkbox"/> RR	2002/04041	FFISYS/806/161	8 September 2003
RAPPORTENS BESKYTTELSESGRAD			ANTALL TRYKTE UTSTEDT	ANTALL SIDER	
Unclassified			27	31	
RAPPORTENS TITTEL			FORFATTER(E)		
LEARNING TO PLAY OPERATION LUCID FROM HUMAN EXPERT GAMES			KRÅKENES Tony, HALCK Ole Martin		
FORDELING GODKJENT AV FORSKNINGSSJEF			FORDELING GODKJENT AV AVDELINGSSJEF:		
Jan Erik Torp			Ragnvald H Solstrand		

### EKSTERN FORDELING

### INTERN FORDELING

ANTALL	EKS NR	TIL	ANTALL	EKS NR	TIL
1		Prof A Aamodt IDI/NTNU Sem Sælands vei 7-9 7491 Trondheim	9		FFI-Bibl
1		Prof O Hallingstad UniK Pb 70 2027 Kjeller	1		FFI-ledelse
1		Prof H R Jervell ILF/UiO Pb 1102 Blindern 0317 Oslo	1		FFIE
1		Prof II R A Fjellheim UniK Pb 70 2027 Kjeller	5		FFISYS
1		FIL/FSS v/Seniorforsker B T Bakken  www.ffi.no	1		FFIBM
			1		FFIN
			4		Tony Kråkenes, FFISYS
					<b>Elektronisk fordeling:</b>
					Ragnvald Solstrand (RHS)
					Jan Erik Torp (JET)
					Espen Skjelland (ESd)
					Fredrik A Dahl (FAD)
					Torbjørn Semb Dahl (TDa)
					Tom Arne Sivertsen (TAs)
					FFI-veven