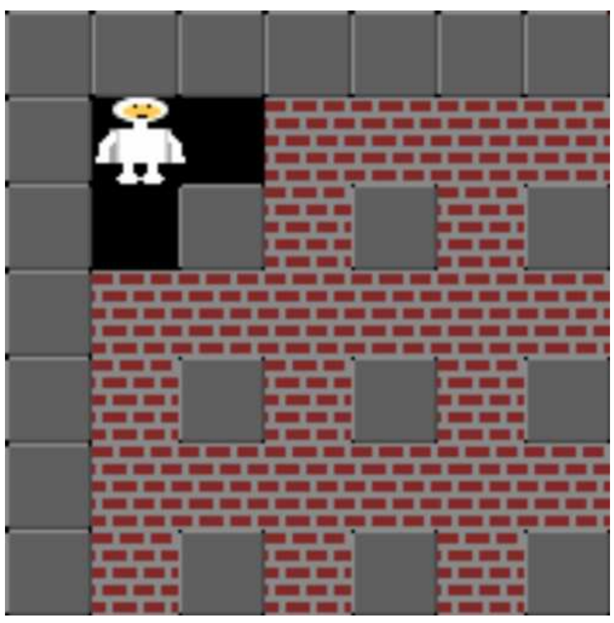
Hierarchical Influence Diagrams



Thomas Ledet

Michael Fogh Kristensen

Summary

The computer gaming industry is a billion dollar endeavor and will probably be one of the steady growth segments of the IT industry over the next decade.

However, the main development in computer games seems to be focusing on graphics and sound. Very few games have agents that reason about their environment and other players in the game, and these agents are often controlled by Finite State Machines that do not allow the agent to reason under uncertainty or adapt itself to the game environment or other players.

This report describes the development of a framework for making decisions in a hierarchical way. The framework is called Hierarchical Influence Diagrams (HID), and is intended to compose an agent in a simple game.

As inspiration for the HID framework, various decision support system languages are examined, including *Bayesian Networks*, *Influence Diagrams*, *Multi-agent Influence Diagrams*, *Network of Influence Diagrams*, *Unconstrained Influence Diagrams* and *Limited Memory Influence Diagrams*.

The game consists of an area divided into squares, some of which are poisonous and some provide more health. Which squares do what is not known by the two or more players that compete to survive the longest. The poisonous squares cannot kill a player, only weaken her. Players can cause damage to each other and thereby potentially provide the fatal blow.

Decisions for an agent in this game can be divided into several smaller decisions that can be structured in a hierarchical tree structure. Strategic decisions on whether to be offensive or defensive can be made on a less frequent basis than the decisions deciding how to realize one such strategy.

When traversing down the HID, a leaf node at the bottom is selected, and this node decides the next *HID decision*, which will then be the next atomic action to execute for the agent.

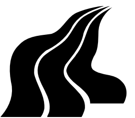
The HID framework offers two choices of HID decision traversing; an event based algorithm and a time slot based algorithm.

The event based algorithm selects the previous HID decision if none of the information it was based upon has changed since the last HID decision. Otherwise it finds the top-most node where information has changed and makes a new traversal beginning from this node.

The time based algorithm reevaluates the previous traversal path starting from a specific node. This specific node is given from a user defined scheduling ratio which specifies the depth location of the next start node.

It is concluded that although the implementation is not finished, the use of Hierarchical Influence Diagrams will decrease the number of computations needed in order to make decisions in some decision scenarios. Also, it is intuitively easier for to solve complex decision problems by decomposing them into less complex problems.

Faculty of Engineering and Science
University of Aalborg



Department of Computer Science

TITLE:
Hierarchical Influence Diagrams

PROJECT PERIOD:
DAT6,
March 2004 - August 2004

PROJECT GROUP:
d635a

GROUP MEMBERS:
Thomas Ledet, ledet@cs.auc.dk

Michael Fogh Kristensen, mfk@cs.auc.dk

SUPERVISOR:
Manfred Jaeger, jaeger@cs.auc.dk

NUMBER OF COPIES: 6

NUMBER OF PAGES: 43

SYNOPSIS:

This project report describes the *Hierarchical Influence Diagrams (HID)* framework for modeling a decision model for an agent in a game. A HID allows the designer to split a complex decision model consisting of a complex influence diagram, into a set of simpler ones and combine these as nodes in a tree structure. By doing so the agent only needs to consider information related to its current decision problem, since information is reduced when selecting an sub node and thereby traversing down the tree.

When traversing down the HID, a leaf node is selected at the bottom, and this node decides the next *HID decision* which is the solution to the agents current decision problem. The HID framework offers the developer two choices of HID decision traversing; the event based algorithm and the time slot based algorithm.

The event based algorithm reevaluates the previous selected HID leaf node if none of the information it was based has changed since the last HID decision. Otherwise it finds the first node where information has changed and makes a new traversal from this point.

The time based algorithm reevaluates the previous traversing path starting from a specific node. This node is given from a user defined scheduling ratio which specifies the depth location of the next start node.

A board-like real-time action game is designed for future test of the HID framework and thereby the test of the algorithms.

Authors

Thomas Ledet

Michael Fogh Kristensen

Contents

1 Introduction **1**

1.1 Project goal 1

1.2 Report Structure 2

2 Problem Domain **3**

2.1 Hierarchical Modeling 3

3 Decision Support Systems **7**

3.1 Bayesian Networks 7

3.2 Influence Diagrams 8

3.3 Discussion 12

4 The Framework **13**

4.1 Hierarchical Influence Diagrams 13

4.2 A single HID decision 14

4.3 Multiple HID Decisions 18

4.4 Discussion 22

5 Application For The Framework **25**

5.1 The FlagBomber Game 25

5.2 Using The Flagbomber game in the Framework 27

6 Implementation **33**

6.1 HID Implementation technical details 33

6.2 HID Implementation 35

6.3 Flagbomber Implementation Overview 36

7 Conclusion **41**

7.1 Future work 42

Bibliography **43**

1 Introduction

Many years ago a whole new industry arose as technology made it possible: computer games. Back then, the computer games were either designed so that no actual opponents were present in the game or the opponents were simple and their "brains" only capable of following a straight line towards the player as it attacked her.

Since then, computers have become more powerful and the computer game industry has become enormous, annually selling millions of games every year. Game development is now an expensive and time consuming process, usually involving large teams of developers. The games have become more complex and sound and graphics have improved immensely.

However, the focus of the developers seems to be on sound and graphics, story lines and general usability of games, but the intelligence of the opponents (agents) in these games has not improved nearly as much.

Very few games have agents that reason about their environment and/or other players in the game, and agents are often controlled by Finite State Automatons. This often makes their behavior predictable or illogical.

1.1 Project goal

The goal of this project is to design a framework for making decisions in a hierarchical way.

We will make an implementation of the framework and describe different scenarios where the strengths of the framework will be applied.

To test the agent, we define the problem setting as a game called Flagbomber.

This problem setting will live up to the principles of Epistimistic Verisimilitude [MG00], where all actors in the game have access to the same amount of information, and have the same possibilities. This is achived by a client-server model, where the server represents the problem setting and gives information to the clients, which represents the actors. The server will not be able to differentiate agents from avatars, which means that no actors have any advantages over others.

1.2 Report Structure

The following chapter introduces the problem domain and the advantages of hierarchical decomposition of an agents decision problem. Chapter 3 investigates the current decision support languages available like Bayesian networks and influence diagrams. In Chapter 4 design details of the Hierarchical Influence Diagram (HID) framework are presented. Chapter 5 present a design for the Flagbomber game, an application that can be used for future test of the HID design.

2 Problem Domain

Consider a game consisting of an arena divided into squares, some of which are poisonous and some provide more health. Which do what is not known by the two or more players, who compete to survive the longest in the game. The poisonous squares cannot by themselves kill a player, only weaken her. However, the players can attack each others, providing the fatal blow, and the last one to be alive wins the game.

The game provides a problem domain, where an agent needs to make real-time decisions under uncertainty. These decisions are based on information about opponents, the world and the agent itself. Needless to say this can constitute a huge amount of information that needs to be decided upon, and since it needs to be done on a real-time basis, this can result in relatively high computational requirements.

To solve this problem we look into ways of minimizing the computational efforts by dividing the decision problem into several sub-decision problems. This can be done by hierarchical modeling, where decisions are made on different levels and where the complexity of the decisions are reduced for each level.

2.1 Hierarchical Modeling

To simplify matters, we introduce another example:

Consider at large corporation that makes washing machines. The Board of Directors has one primary goal: it wants the company to earn a lot of money. This can be accomplished in two ways: sell as many washing machines as possible and to earn as much money as possible on each washing machine. Spending money on marketing will get a lot of washing machines sold and spending money on researching new technologies enables the company to produce the washing machines cheaper. Researching also has the advantage, that the products may also get better.

Once the Board of Directors has decided on what to focus on, they do not want to be troubled by how things are done. Once they have decided that researching is the optimal strategy, they pass decisions of what to research and how on to the

company's Chief Engineers.

The Chief Engineers now have to decide on what to research. They may want to add new wash programs to their washing machines or make them more friendly to the ecosystem. Once they have made up their mind, they leave the actual researching to other people that are experts in the particular field that the Chief Engineers want to have researched.

In the game, decisions can also be split up into several decisions, which can be represented in a hierarchical way. On a strategic level an agent can base a long term strategy on her current health and the believed health of his opponents. Such a strategy could be to either be defensive and avoid contact with opponents or be offensive and actively seek out the opponents in order to kill them. How to actually realize these strategies is not really important when considering, which strategy to follow and could be left to other deciders.

The idea is to have task decisions (typically the strategic decisions) on a less frequent basis, and to have sub-task decisions performed on a more regular basis. Finally, sub-tasks are divided into individual steps. These steps constitute the lowest layer in Figure 2.1.

Making a decision can then be decomposed into choosing the best sub-level decider, given the decision of the above layer. This gives us the following set of advantages for both the game and the washing machine corporation example:

- **The possibility of different types of sub-decision models** By using a hierarchical decision model where each layer consist of a set of individual decision models, it makes it possible to use different types of decision models. This is not limited to using different types of decision models among the layers, but also among the decision models inside each layer. This is an advantage if a solution is optimal for a specific sub-decision model and at the same time less preferable for the rest of the sub-decision models.
- **Better understanding of the decision problem** It is intuitively easier for a computer scientist to solve a complex decision problem by decomposing it into a set of simpler problems.
- **Information reduction** The set of variables that is decided upon in each of the decision models may differ. Variables that have already been decided upon in a decision model on a higher decision layer may no longer be important in the current decision model, since these variables have already been taken into account. Moreover the models placed in the same sublayer do not necessary decide upon the same set of variables. Therefore each decision model can be highly specialized regarding to its functionality of the overall decision problem, by only being designed to decide upon the variables

necessary for solving its sub-decision. This may also have the benefit of computational optimization.

- **Varying the update interval between decisions layers** As stated earlier, recursively dividing the decision problem into sub-decision problems placed in different layers as seen in Figure 2.1 makes it possible to vary the frequency of the decision updates from layer to layer. This results in the advantage that computationally demanding long term decisions can be made in the top of the model and updated occasionally, whereas short term decisions can be place in the bottom of the model and be updated on a regular basis.

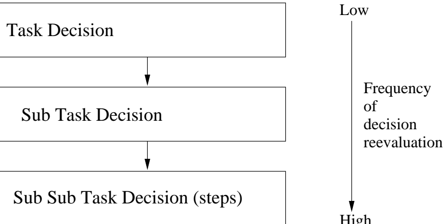


Figure 2.1: The hierarchical layers and the information flow between them. Lower layers have a higher frequency decision update than higher layers. The number of layers is optional, in this example three layers are used.

When making decisions in a hierarchical way, an important issue to consider is "awareness". Is a given layer aware of how the below layer will perform given a decision? Consider the washing machine example: The Board of Directors want to decide whether to spend their money on research or on marketing. We assume that they want to make a somewhat educated decision which leaves them with two options:

1. Asking both the research department and the marketing department how they think they will do, if they get the money.
2. Figure out by themselves what will be the optimal strategy.

In the first situation, each department can only give a useful answer if they evaluate all their possible options, pick the optimal one, and inform the Board of Directors of the utility of this decision. This gives each department a lot of possibilities, which each have to be evaluated. The Board of Directors may then have a somewhat accurate base for their decision, but it is also computationally exhaustive.

In the second situation, the Board of Directors may not need to be experts in research or marketing themselves, but they do need to have some insight into both

areas. They may know, that researching will have a positive influence on the profit, if the customers are not satisfied by the washing machines produced by the company. On the other hand they may also know, that the reason why they are not selling as many washing machines as they would like, is that not a lot of people know that what washing machines produced by the company can do or maybe even that they exists. An example of a Board of Directors decider can be seen in Figure 4.4.

Since recursively evaluating a tree of deciders can be exhaustive, and does not utilize the benefits of a hierarchical model, a decider will be unaware of how its children will solve the decided strategy at any time.

3 Decision Support Systems

As mentioned previously, we need a way to decompose a decision problem scenario into manageable pieces, but we also need to model these pieces and their relationship in a optimal way. In this chapter, different decision support systems are examined with the intension of using these as inspiration for the development of our framework in Chapter 4.

3.1 Bayesian Networks

Bayesian networks create an efficient language for building models of domains with inherent uncertainty [Jen01]. These models make it possible to reason under uncertainty as they integrate a graphical structure that represents the causal relationship between nodes and have a sound Bayesian foundation [BW03].

A Bayesian network consists of a set of variables and a set of directed edges between variables, representing causal dependencies. Each variable has a finite set of mutually exclusive states and together with the directed edges, the variables form a Directed Acyclic Graph (DAG). Additionally each variable has a table of potentials given its parent.

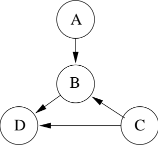


Figure 3.1: An example Bayesian network

An example Bayesian network can be seen in Figure 3.1. The variable B would have attached a potential table $P(B|A, C)$.

3.2 Influence Diagrams

Decision scenarios that consist of the same sequence of decision-observation options are called *symmetric*. A symmetric decision scenario can be represented as a chain of variables. This chain can be represented by a Bayesian network extended with decision nodes (usually represented by squares) and utility nodes (usually represented by diamonds). In this chain, the order of the decisions and the set of observations between two decisions are important. To represent these, the graph of solving the chain is extended with *precedence links* and *information links*. The resulting graph is called an *Influence Diagram* [Jen01].

Information links indicate that the state of the parent must be known (observed) prior to making the decision. They are illustrated as directed links going into decision nodes. Precedence links specify the order in which decisions are made, and are represented by directed links between decision nodes. These help ensure that the order of the decisions are not violated.

Utility nodes have no states and no children. They indicate the utility or "usefulness" of a given network configuration. This is done by having a numeric value assigned to each combined state configuration over all nodes linking to the utility node. If for example the nodes C and D have three states each, the utility node U would then have 3*3 = 9 utility configurations.

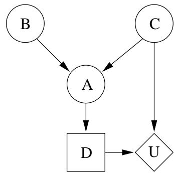


Figure 3.2: An example Influence Diagram

A single decision example influence diagram can be seen in Figure 3.2.

As mentioned, influence diagrams can be used to make multiple decisions as can be seen in Figure 3.3.

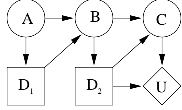


Figure 3.3: An example Influence Diagram with two decision nodes.

In the example in Figure 3.3 two decision nodes exist. An information link exists

from A to D_1, which means that A must be known in order to decide D_1. D_1 has a direct impact on B, and since an information link exists from B to D_2 this means that D_1 must be decided before D_2.

In the following we briefly describe some extensions for influence diagrams that will serve as inspiration for our hierarchical modeling.

3.2.1 Multi-agent influence diagrams (MAIDs)

An extension to influence diagrams for making decisions in multi-agent situations has been developed by Daphne Koller and Brian Milch. This framework is called *multi-agent influence diagrams* (MAIDs) [KM01] and extends the formalisms of Bayesian networks and influence diagrams to represent decision problems involving multiple agents. To do this, every decision and utility variable is associated with a particular agent.

Just as Bayesian networks make explicit the dependencies between probabilistic variables, MAIDs make explicit the dependencies between *decision* variables. This allows for the definition of qualitative notion of *strategic relevance*: a decision variable D strategically relies on another decision variable D' when, to optimize the decision rule at D, the decision making agent needs to take into consideration the decision rule at D'.

The notion of strategic relevance can be expressed in a structure called a *relevance graph* - a directed graph that indicates when one decision variable in the MAID relies on another.

An example of strategic relevance is shown in Figure 3.4, where a MAID for a two-agent situation is shown. Here an agent needs to make a decision $D3$, which is based upon the strategic relevance of its own prior decision $D1$ and another agents prior decision $D2$. Note that in this example the decision $D2$ is based upon strategic relevance of decision $D1$, which is not observable by the agent. In this case the A variable is observable, and is therefore connected (instead of $D1$) to the $D2$ decision node with a relevance graph.

3.2.2 Network of Influence Diagrams (NIDs)

Ya'akov Gal and Avi Pfeffer have developed a framework they call *Network of influence diagrams (NIDs)* [GP03]. NIDs provide a framework for computing optimal decisions for agents that operate in an environment characterized by uncertainty, not only over states of knowledge but also over game mechanics and others' decision process.

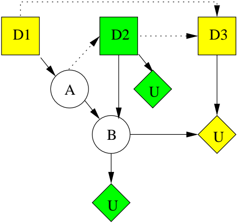


Figure 3.4: An example of a two-agent MAID. Each color node belongs to only one of the agent. Relevance graph are shown as dotted lines. The figure is taken from [KM01].

In this framework, there is an explicit model of the real-world game being played, as well as additional mental models of agents playing the game. It is based on *multi-agent influence diagrams*.

A *Network of Influence Diagrams* is a rooted directed acyclic graph, in which each node is a MAID. Each of these nodes is called *blocks* and the root of the graph is called the *top-level model* and represents the real world from the modeler's point of view. Edges in the graph from block U to block V are labeled i, **D**, where **D** is a set of decision variables in U belonging to agent i.

In Figure 3.5 a small NID example is shown where an analyst has uncertainty over which of the models an agent is using to make a decision. Three possible mental models is possible: The real world Ir model, or one of the agents different models: $I1$ or $I2$ which resembles the fact that the agents view of the world may not be as close to reality as believed.

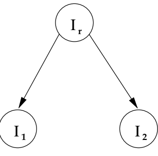


Figure 3.5: An example of a NID. Ir is the real world model, and $I1$, $I2$ are possible belif models of the agent.

3.2.3 Unconstrained Influence Diagrams

A limitation of influence diagrams is that the order of decisions must be explicitly modeled into the network. This makes sense for many decision situations, but what if we want to make a decider, where the order of some or all of the decisions does not matter? This could be modeled by using a decision tree, but as decision

trees grow exponentially with the number of variables it is not always an attractive representation language.

Jensen and Vomlelová has created an extension to influence diagrams to cope with decision scenarios where the order of decisions and observations is not determined. This extension is called Unconstrained Influence Diagrams (UID) [JV02].

A special kind of chance variables is introduced to specify which variables are observable. These new variables are called observable chance nodes and are represented by double circled chance nodes as seen in Figure 3.6.

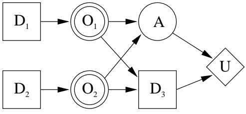


Figure 3.6: An example of an unconstrained influence diagram

Observable chance variables are considered non-existent until all ancestral decisions have been made. This gives us a partial temporal order.

Figure 3.6 is an example unconstrained influence diagram. The order in which D_1 and D_2 are decided does not matter. However, observation nodes are considered non-existent until all ancestral decisions have been made. Furthermore, since we have information links from O_1 and O_2 to D_3, both D_1 and D_2 must be decided before D_3 can be decided. This gives us two possible temporal orderings as seen in Figure 3.7.

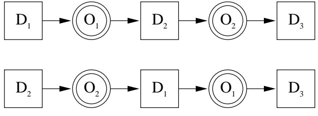


Figure 3.7: Temporal order of Figure 3.6

3.2.4 LIMIDs

The major complexity problem for influence diagrams is that the relevant past for a policy may be intractably large [Jen01]. One way of addressing this problem is to restrict memory by using history variables or information blocking. Another way is to override the no-forgetting assumption when interpreting an influence diagram by specifying explicitly what is remembered when taking a decision. This can be done by using a limited memory influence diagram (LIMID). A LIMID is an influence diagram with direct representation of memory which allows for

forgetting the information n time slices ahead. An example of such a diagram is given in Figure 3.8.

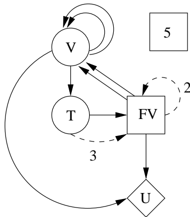


Figure 3.8: An example of a LIMID with five time slices. The dotted lines represents forgetting links indicating that the information is forgotten n time slices ahead. FV is only remembered in the next time slice. T will be remembered two time slices ahead. The figure is taken from [Jen01].

3.3 Discussion

In this chapter we have briefly introduced some languages for making decisions under uncertainty. We will base the design of our framework in Chapter 4 on some of the ideas comprising these languages.

The idea of using influence diagrams as nodes in a network may prove useful as we design our framework. We will base the deciders in this network on influence diagrams as they allows us to represent a decision-observation scenario in compact model, and allows for definition of usefulness of each decision configuration.

The order in which decisions are taken should not be predefined by the structure of the network.

4 The Framework

4.1 Hierarchical Influence Diagrams

In Chapter 2 we argued that decisions often can be split up into sub-decisions, which each can be solved by specialized deciders.

Based on this idea we present Hierarchical Influence Diagrams (HID).

Definition 4.1 ***Hierarchical Influence Diagram***
A Hierarchical Influence Diagram (HID) is a directed acyclic tree of influence diagrams.

Each influence diagram consist of an acyclic graph over chance nodes, utility nodes, and a decision node. A description of influence diagrams can be found in Chapter 3. An influence diagram in a HID contains one and only one decision node.

A decision node is a finite set of mutually exclusive states. In every influence diagram, a decision node exists, which states represents a child influence diagram (a link). If an influence diagram has no parents, it is called the root of the HID.

Chance nodes that share the same name have the same states, and thereby represents the same information.

Definition 4.2 ***Decision***
Each state in a decision node is called a decision. A decision represents a link to a child influence diagram.

An exception to this definition is the HID decision.

Definition 4.3 ***HID Decision***
A HID decision is a decision made by a decision node, which belongs to a leaf influence diagram.

An influence diagram i is a child influence diagram if there exists another influence diagram j, which has a decision node with a state linking to i. This other influence diagram j is called the parent of i. Following a link is called an *invocation*.

The set of children for an influence diagram j is denoted $children(j)$ and its parent is denoted $parent(j)$.

An influence diagram in a HID may be a child and a parent at the same time.

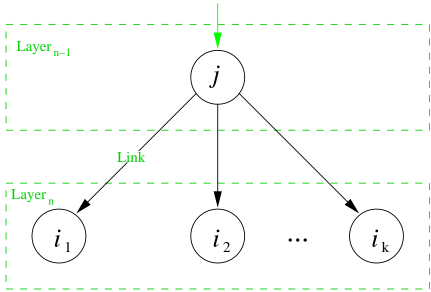


Figure 4.1: Example of a parent, its children, and links between them.

In Figure 4.1 an example of parents and children is shown. j is the parent of i_1, because there is a link between them. j has an incoming link and is therefore both a parent and a child.

In order for a decider to be evaluated in a HID, it is required that its parent has been evaluated. Since the root of the HID is the only decider with no parents, we must always begin our decision path at the root. However, in order to fully benefit from the hierarchical structure, we must have some way of making decisions without starting from the root at all times. This is discussed in more detail in Section 4.3.

4.2 A single HID decision

We will now consider the process of deciding a single HID decision, using the washing machine example from Chapter 2. An overview of this example can be seen in Figure 4.2 where the HID for the Washing Machine Example is given.

At some point in time, the company wants to figure out what its next company action (HID decision) should be. The algorithm *recursivedescend* gives the pseudo code for recursively evaluating a HID.

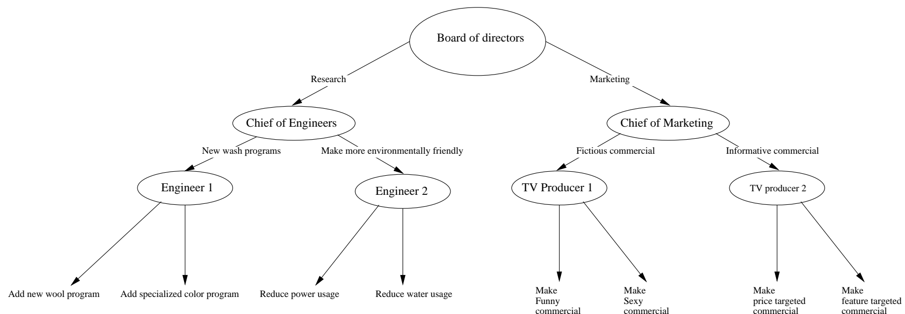


Figure 4.2: The HID for the washing machine example.

Algorithm *decide*($influencediagram$)
1. **for** $\forall$ $chancenode \in influencediagram$
2. **do** *instantiate*($chancenode$) **if** $\exists$ *evidence*($chancenode$)
3. $decision$ $\leftarrow$*evaluate*($influencediagram$)
4. **return** $decision$

Algorithm *recursivedescend*($influencediagram$)
1. $decision$ $\leftarrow$*decide*($influencediagram$)
2. **if** $decision$ is an influence diagram
3. **then return** *recursivedescend*($decision$)
4. **else return** $decision$

The algorithm is called with the root-node of the HID. This influence diagram is then instantiated with all available evidence and evaluated by the algorithm *decide*. The helper functions *instantiate* and *evidence* provide access to information about the world in which we are making decisions.

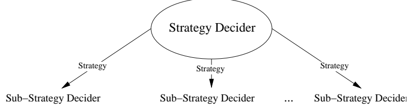


Figure 4.3: Overview of a strategy decider.

The decision made by the root influence diagram represents a link to a child influence diagram as illustrated in Figure 4.3. This link is followed by calling Algorithm *recursivedescend* with the child as parameter. This continues recursively until we reach a leaf influence diagram. Once a leaf influence diagram has been evaluated,

we return with its decision, which is our HID decision. The chain of influence diagrams visited and evaluated in order to get a HID decision is called a *decision path*.

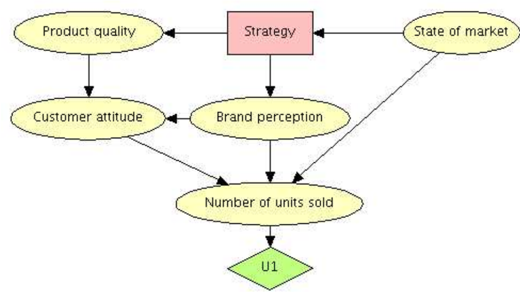


Figure 4.4: Board of Directors influence diagram.

The root-node of the washing machine HID is the Board of Directors influence diagram, which can be seen in Figure 4.4.

Five chance nodes are located in this influence diagram; The *Product quality* node with the states High, Medium and Low. The *State of market* node with the states Prosperous, Normal and Depression. The *Customer attitude* node with the states Happy, Content and Unsatisfied. The *Brand perception* node with the states Good, Medium and Poor. The *Number of units sold* node with the states Many, Some and Few.

In the decision variable has two states: *research* and *marketing*. Each of these states link to a sub-strategy.

In this example, the Board of Directors decides to put their money on research.

Figure 4.5: Chief of Engineers influence diagram.

The link from the Board of Directors influence diagram to the Chief of Engineers influence diagram is then followed. This means that we call Algorithm *recursivedescend* on the Chief of Engineers influence diagram, which is illustrated in Figure 4.5.

Three chance nodes are located in this influence diagram; The *Enviromental standard* node with the states Better, Same and Worse. The *Number of units sold* node with the states Many, Some and Few. The *Customer wash program satisfaction* with the states High, Medium and Low.

The decision node contains the states *New wash programs* and *Make more environmentally friendly*.

The Chief of Engineers influence diagram now has to decide what to research. In this case it decides to research *New wash programs*. We then follow the *New wash programs* link to the Engineer1 influence diagram, which means that the algorithm *recursivedescend* is called on the Engineer1 influence diagram, seen in Figure 4.6.

Two chance nodes are located in this influence diagram; The *Wool programs* node with the states Many, Some and Few. The *Color programs* node with the states Many, Some and Few.

The decision node contains the states *Add specialized color program* and *Add new wool program*.

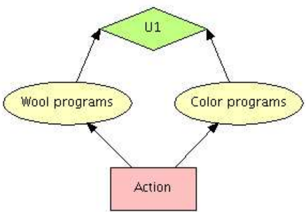


Figure 4.6: Engineer 1 influence diagram.

The Engineer1 influence diagram is a leaf in the HID. This means that, once instantiated and evaluated, this influence diagram will return a HID decision, which will then be our next atomic action.

In this case the *Add new wool program* action is decided, and this yields the decision path shown in Figure 4.7.

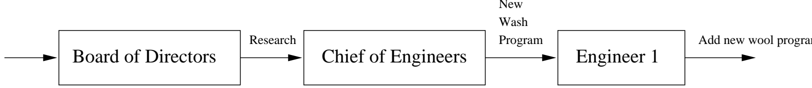


Figure 4.7: The decision path for the Single Decision Washing Machine example.

4.3 Multiple HID Decisions

The previous example considers finding a single HID decision at a single point in time. But what if we need to continue making HID decisions over some period of time? What if our knowledge of the world changes?

Making more than one decision over time simply means that we run the Algorithm *recursivedescend* again and again - once for each needed action.

However, this approach makes us start at the root influence diagram and go through a complete path for each action. This is not very efficient and does not take advantage of the hierarchical nature of HIDs.

4.3.1 Events

A solution is to base the invocation of influence diagrams on events. The concept behind this is that a previous decision of an influence diagram higher in the tree is only valid, if the information it has based its decision on has not changed. If none of the information used to make this particular decision has changed, there is no reason to reevaluate its decision.

If information changes, we will leave our current decider and go to the influence diagram highest in the tree, which bases its decision on this information. We now need to update the chance nodes that model this information, propagate evidence and reevaluate the decision.

As long as none of the information used for deciding the ancestors changes, we will continue with the same decision from the leaf decider at the end of the decision path.

Definition 4.4 ***Event Variable***
An Event Variable is a chance node in an influence diagram, which is included in latest HID decision path.

Definition 4.5 ***Event***
An event is a change of state in a Event Variable.

The generation of events is handled externally. Events occur randomly, and therefore they cannot be foreseen.

First an algorithm is needed to find the upmost location of the event variable in the tree. This algorithm assumes that a HID decision path already exist, which means that the *recursivedescent* algorithm has been executed at least once.

The *findinfluencediagram* algorithm is called with the leaf influence diagram that decided the last HID decision, and the event.

Algorithm *findinfluencediagram*($influencediagram, event$)
1. $upmost \leftarrow influencediagram$
2. **while** $influencediagram$ has parent
3. **do if** $event$ exist in $influencediagram$
4. **then** $upmost \leftarrow influencediagram$
5. $influencediagram \leftarrow parentinfluencediagram$
6. **return** $influencediagram$

An *event* algorithm is now constructed to handle all the decision traversing of the HID. This algorithm makes use of the previous *findinfluencediagram*, and the *recursivedescend* algorithm described in Section 4.2. Furthermore the latest HID leaf node evaluated is always known.

Algorithm *event*(HID)
1. $eventorigin \leftarrow$NULL
2. $event \leftarrow$NULL
3. recursivedescend(root influencediagram)
4. **while** true
5. **do if** $event$!= NULL
6. **then**
7. $eventorigin \leftarrow$*findeventorigin*($influencediagram, event$)
8. $event \leftarrow$NULL
9. *recursivedecent*($eventorigin$)
10. $eventorigin \leftarrow$NULL
11. **else** *recursivedecent*($lastHIDleafinfluencediagram$)

4.3.2 The washing machine example revisited

In the washing machine example in the single decision subsection, the result of running the *recursivedescend* algorithm was the decision path depicted in Figure 4.7. But what happens if we extend the example to decide multiple HID decisions and instead of repeating the *recursivedescend* algorithm use the *event* algorithm. The first HID decision path in both cases is obviously the same.

After the Engineer1 has decided the first atomic action (HID decision), he repeats this decision as the next atomic action. Just after Engineer1 have repeated this decision the third time, the government introduces a new set of rules regarding

standard electric appliances. This rule set is a result of the governments membership in the EU and comes as a surprise for the washing machine company. Despite being a surprise, the washing machine company is the only company that produces washing machines that satisfy this rule-set completely.

As a consequence of this, customers seem to like their washing machines better. This new information may render a decision higher in the hierarchy obsolete. Customers like the fact, that their washing machines satisfy the new rule set, and this means, that the state of the *Customer attitude* chance node in Board of Directors should be changed to *high*.

Since the Board of Directors uses this information for making their decision, chances are that this may change their decision.

In the light of this new information the Board of Director decides that *marketing* to improve the *brand perception* is the best strategy for the company instead of researching. The Chief of marketing is therefore called, and he decides that an *informative commercial* is the best choice. He decides that TV producer2, who has experience with informative commercials should do an advertisement.

TV producer2 is modeled in Figure 4.8, which contains four chance nodes; The *Price targeted commercial being broadcasted* node with the states Few, Fair and Many. The *Feature commercial being broadcasted* node with the states Few, Fair and Many. The *Targetgroup interest* node with the states Features and Price. The *Brand perception* node with the states Good, Medium and Poor.

The decision node has two states: *Make feature targeted commercial* and *Make price commercial*.

TV producer2 decides that *Make feature targeted commercial* is the right choice for the company's next action, in order to make all potential washing machine buyers aware, that their washing machines satisfy the new rule set. TV producer2 repeats this action decision four more times. After this it is realized that *feature commercial being broad casted* node shown in Figure 4.8 has changed to high, because of many of these commercials are now being produced by the competitors.

Therefore TV producer2 then reevaluates the action decision, due to the changed information only has an impact on his action decision, and not on the previous decisions. This new information result in that TV producer2 now decides the *Make price commercial*, and he repeats this decision until another event variable changes.

This decision scenario is summarized in Table 4.1, where the different decision paths, and the influence diagrams where the event variable is located are shown.

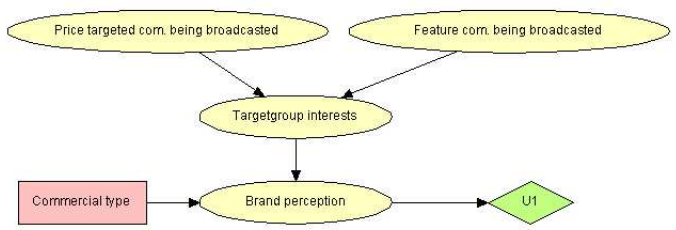


Figure 4.8: The influence diagram for the TV producer2.

Event ID	startlayer	decision history
	1	BOD $\rightarrow COE \rightarrow E1 \rightarrow ANWP$
	3	E1 $\rightarrow ANWP$
	3	E1 $\rightarrow ANWP$
	3	E1 $\rightarrow ANWP$
BOD	1	BOD $\rightarrow COM \rightarrow TV2 \rightarrow MFTC$
	3	TV2 $\rightarrow MFTC$
	3	TV2 $\rightarrow MFTC$
	3	TV2 $\rightarrow MFTC$
	3	TV2 $\rightarrow MFTC$
TV2	3	TV2 $\rightarrow MPTC$
	3	TV2 $\rightarrow MPTC$
	3	TV2 $\rightarrow MPTC$
	3	TV2 $\rightarrow MPTC$

Table 4.1: A decision path example of running the washing decider HID with the event algorithm. The letters used are abbreviations for the HID nodes in the washing decider example shown in Figure 4.2. The Event ID column states in which node the event variable is located.

4.3.3 Time Slotting

An alternative way to make multiple decisions is to run the $recursivedescend$ algorithm with a node located at different layers of the HID at different time slots.

In order for this approach to work optimally, layers must be explicitly defined when designing the HID and all branches must have the same depth.

To make multiple decisions, we may for instance start in layer n for five time slots before starting a single time slot in layer $n-1$. When this is repeated four times and the layer $n-1$ has been started from five times, and layer n has been started from 25 times, the layer $n-2$ is the next starting point at the next time slot.

Layer n	Time slots
1	1
2	2
3	3

Table 4.2: The time slot setup for the multiple decision example.

To do this we need to remember the last decision path starting point. $current()$ returns which influence diagram is active in each layer.

The function $decision()$ holds a cache of the last decision of all influence diagrams.

Algorithm *timeslot3l*
1. $timeslot$ = current time slot
2. n = number of layers in HID
3. *recurcivedecend*$(rootofHID)$
4. **while** $true$
5. **do for** $l \leftarrow n$ **to** 1
6. **do if** $timeslot$ mod (layertimeslot(n)) $= 0$ **then**
7. *recurcivedecend*$(current(l))$
8. $l \leftarrow l - 1$
9. **else** do *recurcivedecend*$(current(l))$
10.
11.

We return to the washing machine example. Engineer1 has just decided the next atomic action to be *Add new wool program*. Now the layers of the HID is decided using Time slotting with the assigned number of time slots shown in Table 4.2.

This could result in table 4.3 wich is an example of a single pass of the washing machine HID using the time slotting algorithms with the assigned time slots shown in table 4.2.

4.4 Discussion

When considering only HID decisions in a multiple decisions context, hierarchical influence diagrams bear some similarities with unconstrained influence diagrams. As described in Chapter 3, UIDs are influence diagrams, where the order of some or all decisions is not dictated by the network structure. The order of HID decisions in a HID is not given by the network structure, which means that we cannot model

Time slot	startlayer	decision history
1	1	$\rightarrow BOD \rightarrow COE \rightarrow E1 \rightarrow ANWP$
2	3	$\rightarrow E1 \rightarrow ANWP$
3	3	$\rightarrow E1 \rightarrow ASCP$
4	3	$\rightarrow E1 \rightarrow ANWP$
5	2	$\rightarrow COE \rightarrow E1 \rightarrow ASCP$
6	3	$\rightarrow E1 \rightarrow ASCP$
7	3	$\rightarrow E1 \rightarrow ASCP$
8	3	$\rightarrow E1 \rightarrow ANWP$
9	2	$\rightarrow COE \rightarrow E2 \rightarrow RWU$
10	3	$\rightarrow E2 \rightarrow RWU$
11	3	$\rightarrow E2 \rightarrow RWU$
12	3	$\rightarrow E2 \rightarrow RPU$
13	1	$\rightarrow BOD \rightarrow COM \rightarrow TVP1 \rightarrow MFTC$

Table 4.3: A decision path example of running the washing decider HID with the time slotting algorithm with the time slot settings from table 4.2. The letters used are abbreviations for the HID nodes in the washing decider example shown in Figure 4.2.

that a HID decision should be made before another. UIDs provide a mechanism for doing this, however, we believe that this feature of HIDs does not impose any limitations in the contexts examined in this report.

If we consider HIDs as a whole, it is obvious, that a well defined order of internal decisions exist, as we cannot get a HID decision, without following a strict path from the root-node to a leaf-node.

Modeling time (taking past or future configurations into account) is possible in HIDs. However, time must be explicitly modeled into the influence diagrams as shown in Figure 5.3. Care must also be taken not to violate the structural requirement in HIDs.

5 Application For The Framework

To make future test of the HID framework possible, the focus is now turned toward designing a decision model for an agent in a game.

The influence diagrams in this chapter is provided as illustrative examples of how the game can be modeled.

5.1 The FlagBomber Game

The Flagbomber game is based upon ideas taken from the game Bomberman [Ent]. The game is drastically reduced in order to simplify our example.

5.1.1 The Problem Domain

The problem domain consist of an arena and a number of actors competing therein. Everything is seen in a top down view, and is performed in real time. The term real time relates to an unlimited series of fixed time slots. The duration of a time slot is S seconds where exactly one decision can be performed as an atomic action. Relating to the HID framework constructed earlier, an HID decision therefore corresponds to deciding upon an atomic action.

The Arena

The world consist of $X * Y$ squares, where each individual square can be either empty or occupied by one or more actors or/and an object. An example of this is shown in Figure 5.1.

The Objects

To make the problem domain non-trivial, the squares are randomly covered with objects. An object is either an invisible item, or a wall. A wall is both movement blocking and un-removable.

The Actors

Two types of actors exist in the problem domain: An agent which is a computer controlled player and an avatar which is human controlled. The game is designed under the assumption that Epistimic Verisimilitude yields the best entertainment to a human player, and therefore all actors have the same set of possibilities or restrictions and access to the same amount of information at any time.

An actor moves from one square to another in a single time step and consequently only occupies one square at a given time. Actor movement is limited to *North*, *South*, *East* and *West*, given that there is no wall occupying the target square. A square can be occupied by more actors at the same time, allowing the actors to run past each other.

Every actor has a vision of $V * V$ squares of the arena. Inside this vision zone the actor can see the other actor if present, and the walls occupying the individual squares. An actor has a "remembered" vision regarding the location of seen walls and the actor outside the current vision zone. However the location of the opponent actor may have been changed without the actor having any knowledge of this. Each actor has T health points at initialization time. Death of an actor is represented by -1 health points.

The Items

The set of invisible items contains the spawning point of each of the actors, and the health modifier items. A spawning point is the square where an actor enters the arena when the game starts. The health modifier items, are items each assigned to a square. If an actor enters a square containing a health modifier, the actors health points are either increased or decreased by one, depending whether if the health modifier item is a poisonous item or a health item. A health modifier item is removed from the arena once triggered. Furthermore a health modifier item can only affect the actors health points to within the range $0 - T$, and can therefore never kill the actor nor give the actor a higher number of health point than upon game initialization.

Actor Goals

The goal of the game is to survive the longest time possible. An actor can extend this time to unlimited if she succeeds in killing the opponent. As an logical side effect of this, an actor with low level of health points, could flee from an opponent rather than being killed.

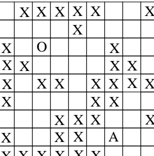


Figure 5.1: The problem domain consist of a set of squares which sum up the world. These squares can be occupied by objects **X**, opponents **O** and the actor **A** itself.

5.2 Using The Flagbomber game in the Framework

First the degree of decomposition of the decision problem of the agent needs to be specified, in other words what depth is needed in the HID that sums up the agents behavior. For the agent in the flagbomber game the depth of three have been chosen since it seems intuitive to model the agents behavior in terms of strategies, sub strategies and atomic actions. The following sections are the result of analyzing the game in regards to using it in a HID framework where the decision tree has the depth of three.

5.2.1 The Agents Set of Strategies

To find the Strategies of the adaptive agent of the Flagbomber game, the goals of the game is analyzed. It becomes clear that a intuitive way of describing the strategies would be in terms of being defensive or offensive.

- **Defensive** The purpose of the Defensive strategy is to support the goal of the game by staying out of reach of the opponent until she either becomes stronger or the threat of being attacked is reduced.
- **Offensive** The purpose of the Offensive strategy is to support the goal of the game by attacking the opponent. If the agent is in low risk of being killed, she will try to get as close to the opponent and hit him.

5.2.2 The Agents Set Of Sub Strategies

To find the Sub Strategies the Strategies are analyzed and intuitively divided into sub strategies. An example is the defensive strategy which is divided into the Flee and the Hide sub strategies. Since the Offensive strategy only exibits one way of attacking the opponent, no sub strategies are found analyzing this strategy. The following set of Sub Strategies is found:

- **Flee** If the agent is in extreme danger of being killed, she will try to get as far away from the opponent as possible. This sub strategy is considered successful is the opponent is not in the agents vision zone after f actions.
- **Hide** If the agent is in moderate danger of being killed, she will try to hide himself behind a wall, so that despite the opponent can see the agent, the opponent cannot move directly toward the agent without being blocked. This sub strategy is considered successful if the direct path from the agent to the opponent is blocked within h actions.

5.2.3 The Agents Atomic Actions

When selecting the agent's set of possible atomic actions, it is done with the previously mentioned idea of Epistimic Verisimilitude. The result of this is that every agent has the same set of possible atomic actions. The following list sums up the atomic actions of the actors.

- **Move** A Move action results in the agent moving from one square to an adjacent square. As described earlier, the following set of move actions are possible: *North*, *South*, *East* and *West*. A move action is only valid if the target square is not occupied by a wall.
- **Hit** A Hit action results in the opponent having her health points reduced by one. If the result is a health point status of -1, the opponent dies and is removed from the game. A Hit action is only valid if the opponent is occupying an adjacent square which is reachable by a move action, or if they occupies the same square.
- **Wait** A wait action, is the absence of the other listed actions in the given atom time space of the game. Therefore it is not a selectable action for the agent, but a default action if no atomic actions have been decided upon within the specific time slot.

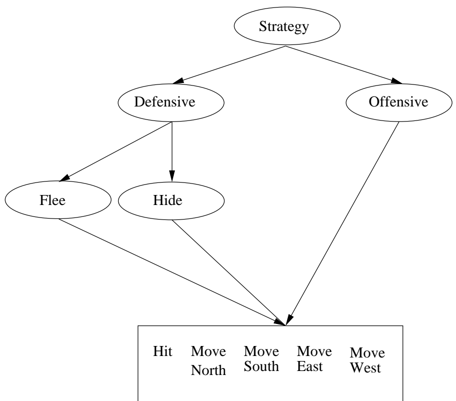


Figure 5.2: The result of using the Flagbomber game in the developed framework. A HID, where each leaf represents the actions which the agent can be seen performing in the game.

5.2.4 Constructing The Strategy Decider

Based upon the HID framework from Chapter 4 the strategy decider for the Flagbomber game is now constructed. First, the decision node of the influence diagram is constructed, where the *offensive* and the *defensive* strategy are states. Second, the utility nodes and chance nodes needed to decide among these states are connected. In this influence diagram we model time, and as a result of this we have four nodes representing two chance nodes before and after a decision is taken. Each chance node pair has the same chance node states but may differ in probability. The chance node pair for *healthpoints* has the states 0, 1, 2, 3. The chance node pair for *Distance opponent* has the states Far, Medium and Near. The resulting influence diagram is depicted in Figure 5.3.

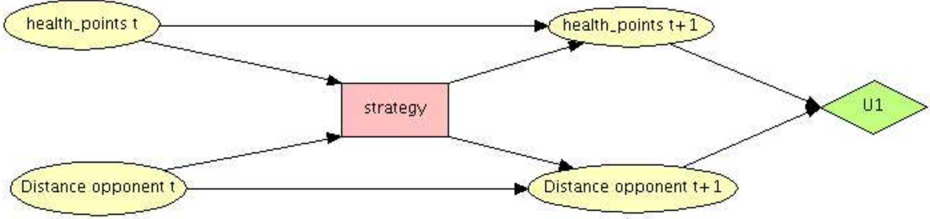


Figure 5.3: The Strategy Decider for the Flagbomber game. The decision node has the following two states: Offensive and Defensive.

5.2.5 Constructing The Sub Strategy Deciders

For each state in the decision node in the strategy decider influence diagram, a sub-strategy decider is constructed. In each influence diagram the decision node is constructed so that each state is a sub strategy. In Figure 5.4 an example of such is given in the form of the defensive sub strategy decider.

In this influence diagram time is also modeled. Here the same *health point* change node pair exists as in Figure 5.3. Furthermore is the *Hidden* chance node pair with the states True and false, stating whether the agent is hidden before and after the decision. The decision node has the two states Flee and Hide.

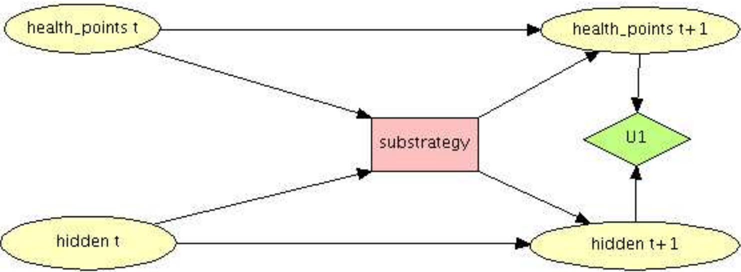


Figure 5.4: The Sub Strategy Decider for the 'Defensive' strategy. The decision node has the following two states: Flee and Hide.

5.2.6 Constructing The Action Deciders

For each sub strategy, an Action decider model is constructed. In the decision node of each of these models the only valid states are the atomic actions described earlier. An example of an action decider model is the decider for the 'Hide' sub strategy depicted in Figure 5.5.

In this influence diagram time is modeled. This is done by observing if the agent is hidden before and after a decision. The decision node has each of the four moving action and the wait action as states. The states of the adjacent squares combined with the decision have an influence on whether the agent is hidden after the decision. Each of the *destination status nodes* has the following states Hidden, Blocked and Unblocked.

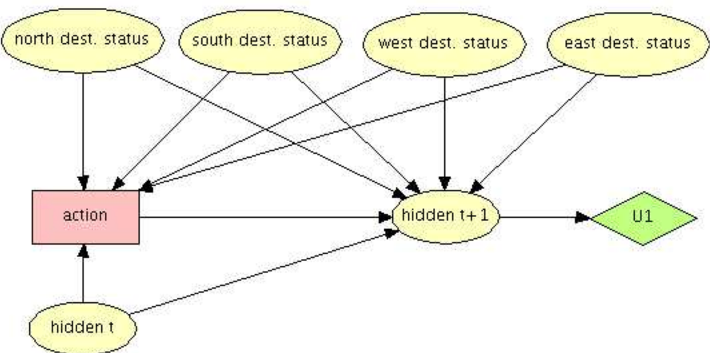


Figure 5.5: The Action Decider for the 'Hide' sub strategy. The decision node has the following states: Move North, Move South, Move East, Move West and Wait.

5.2.7 The Pieces Of The Agent

When all the influence diagrams for the Flagbomber game has been constructed it is time to have a look at what more is needed to get the agent ready for implementation.

First, we need a way of specifying the structure of the HID. This could be done relatively simple by a configuration file describing the tree structure of the HID. In Chapter 6 we give an example configuration file for the washing machine example and this can be used as reference.

Second, the choice of HID algorithm needs to be made. When selecting the appropriate algorithm, the general frequency of chance node change in the influence diagrams that sums up the HID should give the developer a hint to what algorithm is preferable to her decision scenery. A relatively small number of chance node change would probably result in the event based algorithm being superior, whereas the time slot based algorithm might perform poorly.

Besides this a structure for updating the chance nodes is needed. This structure needs to be able to receive information and update the influence diagrams of the HID with this. A mapping of this information to the individual chance nodes in the influence diagrams of HID needs to be constructed. Because of the fact that multiple chance node can have the same name and thereby the same states, this mapping is not limited to being 1:1.

The information structure should also be constructed to monitor the current decision path of the HID if the event algorithm mentioned previously is chosen. If so, the structure must be able to generate an event when a chance node belonging to

decision path (an event variable) is changed.

5.2.8 Summary

Based upon the HID framework, the idea for the Flagbomber game has been formulated, analyzed and thereby divided into strategies and then again into sub strategies which decides the atomic actions of the agent. Each of these strategies and sub strategies has then been made into an influence diagram and connected into one single HID depicted in Figure 5.2, where the parent - child relationship of the influence diagrams becomes clear. These things combined make it possible to make a future implementation of the agent and thereby the game.

6 Implementation

This chapter describes our implementation of the HID framework and the Flagbomber game. It describes the technologies used, and specifies the architecture of the client-server model used.

The code can be downloaded here:
`http://www.cs.aau.dk/~ledet/hid/hidcode.tar.bz2`

6.1 HID Implementation technical details

We implement the Flagbomber game in Perl[1] as this programming language gives a lot of freedom in regards to data structures and because of the POE framework, which we have previous experience with. Hugin [A/S03] is used for evaluating and designing influence diagrams.

6.1.1 POE

Perl Object Environment[2] (POE) is a framework for creating multitasking programs in Perl.

POE is event-based - everything happens for a reason. It consists of a “kernel”, which manages a number of sessions, which all wait for an event to take place.

The kernel functions as a scheduler, always keeping an eye out for waiting data or alarms. If, for instance, data has been received from the network, the kernel calls the appropriate session registered for this event, which runs the appropriate function for dealing with network data.

Furthermore, POE has built-in components for making both networking server and clients. It handles all message queuing and makes it fairly easy to make message

[1] http://www.perl.org
[2] http://poe.perl.org

handlers.

The POE framework is perfect for tasks like game servers, where everything is a result of a request from a client or of some timed event.

6.1.2 HUGIN

Since the HID framework consists of influence diagrams, we have chosen to leave all direct handling of these to Hugin[A/S03]. The following is a description from the Hugin Expert website.

> The Hugin Researcher package contains a flexible, user friendly and comprehensive graphical user interface and our advanced Hugin Decision Engine for application development. The user interface contains a graphical editor, a compiler and a runtime system for construction, maintenance and usage of knowledge bases based on Bayesian network technology.
>
> The Hugin Decision Engine (HDE) encapsulates all functionality related to handling and using knowledge bases in a programming environment. The HDE is delivered with application program interfaces (API's) for four major programming languages C, C++, Java and an ActiveX-server for e.g. Visual Basic. [A/S03]

As described in the above text, programming interfaces for Hugin has been developed for several programming languages. We have previous experience with both Java and C++, which make these languages obvious choices for our implementation. However, the C++ interface seems to be practically impossible to get working on other platforms (linker problems) than those explicitly specified in the Hugin documentation, and no such platforms were readily available to us.

This leaves us with the Java interface, which is fairly easy to set up and works without problems. However, our game is implemented in Perl - mainly because of the POE framework, which means that we need to have a way of accessing Java from Perl.

Fortunately a library for doing exactly this can be found in CPAN[3]. It is called Inline::Java and enables us to write Java code directly inside Perl programs. This means that we can write our Hugin functions in Java and then access them from Perl as if they were written in Perl.

[3] http://www.cpan.org - Comprehensive Perl Archive Network

6.2 HID Implementation

The HID implementation consists of two classes: the HID class and the Decider class, both of which are briefly described in the following.

6.2.1 The HID Class

The HID class is written in Perl and is the heart of the implementation. It maintains the data structure of the HID.

It is instantiated with a configuration file, in which the user can specify the HID structure, which influence diagrams are associated with each HID node and which type of invocation (event based or time slot based) the user wants.

The configuration file consists of a nested hash structure in Perl syntax (it is executed by the HID class constructor) and the constructor then takes appropriate actions needed to load the actual influence diagrams.

An example configuration file for the washing machine example can be seen in Figure 6.1.

Each HID-node is represented as a hash with key-value pairs: The name of the node, a file name for the associated influence diagram in Hugin format, and an optional alias for the node.

The aliases provide us with a way to call a node by another name. This enables us to give the nodes more descriptive names. For instance, the Board of Directors node decides to either *research* or do *marketing*. However, we may want to refer to the “Researcher” as Chief of Engineers to make designing the HID more intuitive.

All methods for traversing a HID are found in the HID class. An example of such as method is the $recursivedescend()$ method described in Section 4.2.

6.2.2 The Decider Class

The Decider class is written in Java and communicates with the Hugin decision engine.

Its constructor takes a filename, which is then opened and loaded into memory by Hugin. Methods for setting evidence, propagating evidence and evaluating, and returning decisions from, and extracting any additional information about the influence diagrams are parts of this class.

```
$hid = {
    0 => {
'Board of Directors' => {
    'file'   => 'boardofdirectorsid',
    'parent' => undef,
    'alias'  => 'root',
},
    },

    1 => {
'Chief of Research' =>  {
    'file'   => 'chiefofengineeersid',
    'parent' => 'Board of Directors',
    'alias'  => 'Research',
},
'Chief of Marketing' =>  {
    'file'   => 'chiefofmarketingid',
    'parent' => 'Board of Directors',
    'alias'  => 'Marketing',
},
    },

    2 => {
'Engineer1' => {
    'file'   => 'engineer1id',
    'parent' => 'Chief of Research',
    'alias'  => 'Add new wool program',
},
'Engineer2' => {
    'file'   => 'engineer2id',
    'parent' => 'Chief of Research',
    'alias'  => 'Add specialized color program',
},

'TV Producer 1' => {
    'file'   => 'tvproducer1id',
    'parent' => 'Chief of Marketing',
    'alias'  => 'Fictious Commercial',
},
'TV Producer 2' => {
    'file'   => 'tvproducer2id',
    'parent' => 'Chief of Marketing',
    'alias'  => 'Informative Commercial',
},
    },
};

$invocation = 'event';
```

Figure 6.1: An example configuration file

6.3 Flagbomber Implementation Overview

The implementation of the Flagbomber game consists of tree parts: a server and two types of clients, as seen in Figure 6.2. One client is an interface between the Flagbomber game and the HID framework and thereby the agent, the other is an avatar/interface for the human player.

6.3.1 Server

The server maintains the game state. All actor positions, actor health states, etc. are known to the server at all times.

Every action performed by an actor is sent to the server, which then confirms the

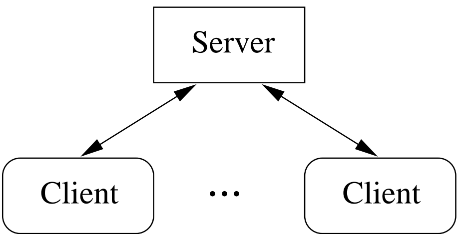


Figure 6.2: The Flagbomber game is implemented as a central server with two or more connected clients.

action if successful and updates its internal representation of the game arena.

If an event occurs that as an impact on one or more actors, each actor will be informed of the event by the server.

In Figure 6.3 and overview of the server can be seen.

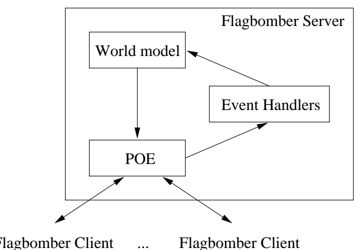


Figure 6.3: An overview of the Flagbomber server

The server listens for connections on the network. Once enough clients have connected and identified themselves, one of them can start the game by sending a *start game* command. The server then stops listening for incoming connections, starts the game, and sends information about the game to the clients. An example of this can be seen in Figure 6.4. The server then sends all information about the clients vision zone to the client.

Whenever a client moves around, it informs the server of its intended movement. The server then checks for possible collisions and if the movement is possible, it

```
client > action=login name=clientname
server > action=login status=ok playerid=1 level=standard
client > action=start
server > action=start status=ok
```

Figure 6.4: The login sequence

updates its internal representation of the world, and informs the client of its new position, its updated vision zone, and its health state if changed.

6.3.2 The Avatar Client

The avatar client is an interface for the human player. It is implemented in Perl using POE and Simple DirectMedia Layer[4]

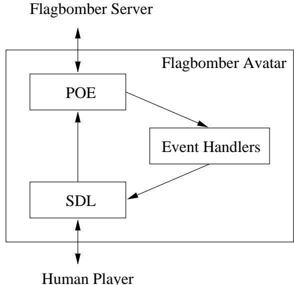


Figure 6.5: An overview of the Avatar Client

The Simple DirectMedia Layer (SDL) is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer through a high-level interface. It handles all the graphics and keyboard input. Keyboard input is sent on to POE, where they are translated to game commands and sent to the server.

The avatar client internals can be seen in Figure 6.5.

The game is played using the arrow keys. Input is currently handled by SDL, which also handles all graphics. However, there seems to be synchronization issues with this approach, where key events are not necessarily detected and sent to POE. This problem is probably linked to problems with refreshing the framebuffer (flipping between buffers), which is currently handled by throwing timed events. This approach has the penalty of being very hard on resources and should be rewritten in any future versions. It does, however, provide us with a somewhat working avatar client for our game.

[4]http://sdl.perl.org

6.3.3 Adaptive Agent Client

The (adaptive) agent functions as an interface layer between the game and the HID framework.

It is based on the avatar client, although the SDL layer has been removed and replaced by the HID framework. All information is fed to the HID, which in turn returns atomic actions, simulating keyboard input from the arrow keys.

The World Observations

The agent has a database, which holds all information it has received about the world from the server. From this database, information suitable to feed to the HID can be extracted or calculated.

An implementation overview of the adaptive agent can be seen in Figure 6.6.

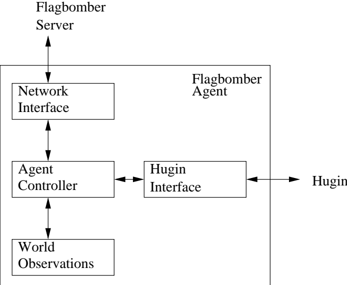


Figure 6.6: Overview of the agent implementation

As of this writing, the agent client remains unimplemented.

7 Conclusion

The purpose of this project was to develop a framework for making decisions in a hierarchical manner. As inspiration for the framework a number of decision modeling languages have been examined. The idea of making a network of influence diagrams is taken from Network of Influence Diagrams. Inspired by Unconstrained Influence Diagrams, the order of decisions and observations are not dictated by the structure of network.

The resulting decision framework is called Hierarchical Influence Diagrams. The structure and semantics of the framework have been described and an example decision scenario of a company that produces washing machines has been given.

An implementation of the framework has been made, making it possible to evaluate a HID.

A game intended as a testbed for the framework has been developed and partly implemented. However, since it is only partly implemented, we have not been able to do any tests on the framework, showing that it is possible to make an agent for the game using a HID.

Creating a game of this type turned out to be far more time-consuming than expected. The client-server nature of the game meant that a communication protocol had to be developed, although POE made networking a lot easier. The game and HID framework was originally intended to be programmed in C++, but getting the Hugin C++ interface to work was not trivial. Instead the implementation was done in Perl and Java. This meant that we had to find a way to get the two languages to work together as one, which turned out to be easier than expected. However, the natures of SDL and POE turned out not to be directly compatible, which gave us problems with reading keyboard input and updating the screen at the same time.

Although the hierarchical decomposition of decisions made it easier to model the deciders for the game, this still turned out to be a complex task, even for a simple Flagbomber game. This may be one of the reasons why most games rely on Finite State Automatons for controlling their agents. More research should be made in the area of using decision support systems in games before game developers will consider using these powerful tools.

However, the Hierarchical Influence Diagrams language is a small step along the way to making it easier to model types of decision scenarios that can be decomposed into a hierarchical structure. Also it can have the benefit that the number of

computations needed to evaluate these can be decreased. Real-life decisions taken by people often follow a hierarchical structure, which seems to support the idea of Hierarchical Influence Diagrams.

7.1 Future work

- **Make the agent adaptive.**
  A way to make the agent in the Flagbomber game more interesting would be to enable it to adapt itself to a other actors. However, this requires that we find a way to reward the actors atomic decisions.
- **Describe the rest of the deciders for the game.**
  As described, modeling the decisions for the game can be very extensive. However, this is necessary in order for the agent to work completely
- **Finish implementation of the framework and compare its performance to other types of decision modeling languages in the game scenario.**
  Doing a comparison between Hierarchical Influence Diagrams and other decision support system languages could be interesting and could give us an idea of the efficiency of HIDs.
- **Test and compare the two invocation methods.**
  The two invocation methods described in the report each seem to have both advantages and disadvantages. Letting them compete against each other in different test environments could give us a better understanding of their strenghts and weaknesses.

Bibliography

[A/S03] Hugin Expert A/S. *HUGIN API Reference Manual, version 6.1*. Hugin Expert A/S, 2003.

[BW03] David Ball and Gordon Wyeth. Classifying an opponent's behaviour in robot soccer. 2003.

[Ent] Hudson Entertainment. *Hudson Entertainment Website*. Hudson Entertainment.

[GP03] Ya'akov Gal and Avi Pfeffer. A language for modeling agents' decision making processes in games. *AAMAS'03, July 14-18, 2003, Melbourne, Australia*, 2003.

[Jen01] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, 2001.

[JV02] Finn V. Jensen and Marta Vomlelova. Unconstrained influence diagrams. 2002.

[KM01] Daphne Koller and Brian Milch. Multi-agent influence diagrams for representing and solving games. 2001.

[MG00] Claus B. Madsen and Erik Granum. Aspects of interactive autonomy and perception. Technical report, Aalborg University, 2000.