Pickman's Machine: A Reasoning Architecture

Baki Cakici

Master's Thesis
Department of Computer and Systems Sciences (DSV)
Royal Institute of Technology (KTH)

Academic Supervisor: Magnus Boman
April 2008

This thesis corresponds to 20 weeks of full-time work.

Abstract

A two-layered architecture for reasoning that uses narratives to guide its behavior is presented. The narratives are interpreted by reactive and deliberative reasoning layers to generate responses to external events, in accordance with internal desires created by the deliberative layer. A C^{++} implementation of the proposed Pickman architecture is described and tested in five scenarios in which three different Pickman versions are tasked with healing a small population suffering from a virtual plague. It is shown that a Pickman implementation using a static desire generation mechanism is efficient, but unreliable. The dynamic version, on the other hand, is not as efficient, but it performs successfully in all scenarios.

Acknowledgments

I would like to thank:

> Magnus, who inspired me to think deeper;
> Hanna, who kept me sane;
> and my mother, who taught me to ask questions.

Contents

Chapter 1

Introduction

Often, situations encountered in daily life are seen as trivial to the point that no one remembers that they were ever problems to begin with. Picture opening a door; the direction in which the door opens, the position of the doorknob or the strength required to manipulate it must each be considered before attempting to solve the mystery that is a closed door. How does the human mind handle such problems with no apparent effort? The elusive trait is often called common sense, the collection of "things we expect other people to know and regard as obvious" (Minsky 2006, p.164). Since this type of knowledge is mostly implicit, its representation presents an additional challenge.

Push Singh proposed narratives as a method of capturing commonsense knowledge in "EM-ONE: An Architecture for Reflective Commonsense Thinking" (2005). The present thesis explores a system of reasoning inspired by his work as well as by Marvin Minsky's theories of the mind.

1.1 Overview

A layered model of reasoning for an actor in a virtual plague domain is presented. A virtual plague is defined as the spread of a behavior-affecting property among the inhabitants of the simulation (Boman & Johansson 2007). The primary actor, called Pickman, uses a reasoning mechanism based on pre-generated narratives to guide its behavior within the domain. This mechanism is referred to as Pickman's Machine. The Pickman architecture is based on a subset of Marvin Minsky's six-layered model of mental activities, the Emotion Machine (2006). In addition, the architecture borrows elements from Push Singh's EM-ONE (2005), the first implementation of the Emotion Machine.

After the description of the theoretical background, a C++ implementation of the proposed architecture is introduced. The implementation is tested in five scenarios where Pickman is tasked with healing a small population suffering from a virtual plague.

1.2 Delimitation

The Emotion Machine architecture is designed to organize extremely complex agent societies into six layers of reasoning. Pickman's Machine follows the same layered structure but is not agent-based. Instead, it implements the "If−>Then" rules of the Emotion Machine's reactive layers using the narrative model of EM-ONE. In addition, the two reactive layers of the Emotion Machine (instinctive and learned reactions) are merged to form a single layer, identical to the EM-ONE architecture.

While Pickman's Machine draws from both the Emotion Machine and its first implementation, EM-ONE, it does not attempt to match either in depth or breadth. It is better viewed as an exploration of Minsky's theories of the mind, motivated by the desire to build a minimal reasoning architecture.

1.3 Hypothesis

In the virtual plague domain, Pickman is tasked with halting the epidemic by healing all infected agents. Efficiency in this domain is defined in terms of infection duration. An actor is considered to be more efficient if the entire population is healed faster.

Two different Pickman versions (see Chapter 4 for details) are compared using a random-only version as the baseline. It is conjectured that a dynamic Pickman using the deliberative layer to create desires is more efficient than a static reactive-only Pickman in the proposed virtual plague domain.

1.4 Methodology

The process of healing a small population was simulated using five different scenarios. In each scenario, the performances of three different Pickman versions (dynamic, static and random) were compared. Every scenario/version combination (three Pickman versions and five scenarios, 15 unique pairs in total) was repeated 1000 times. 1000 repetitions were chosen to obtain accurate results for the fourth scenario where the static version failed on approximately 90 percent of the runs. The scenario was tested with 100, 1000 and 10000 runs. 1000 was chosen as the final value because the results obtained using 100 runs were not precise enough and the results obtained from 10000 runs were not significantly different from the results of 1000 runs. For a single run, the maximum number of cycles was also defined as 1000. For each run, the following parameters were measured:

1. Simulation length in cycles

2. Remaining infected actors
3. Number of realm-to-realm moves made by Pickman
4. Number of actors healed by Pickman

The selection of these specific parameters was based on how Pickman interacted with the external world. When viewed at a lower level, Pickman could choose to either heal, move or skip a cycle. *Remaining infected actors* was used to monitor if Pickman had eradicated the plague completely. *Number of realm-to-realm moves* was used to calculate Pickman's activity level during a simulation. When combined with *simulation duration* and *number of actors healed*, this factor was useful in understanding how long Pickman had remained idle in each simulation. Another metric that was derived directly from the values above was cycles per heal. This value provided an idea of how much time Pickman spent searching for an infected actor on average.

Programming an implementation of Pickman's Machine was instrumental in ensuring that all components of the system were fully specified at the theory level. The implementation was revised several times before reaching the final version used in the experiments. After obtaining the results, the code was reviewed again to ensure that the experiments did not contain any programming errors.

1.5 Disposition

Chapter 2 describes earlier works in reasoning architectures by Push Singh and Marvin Minsky. Chapter 3 examines the components of the Pickman architecture in detail. Chapter 4 presents the virtual plague domain and the experiments performed using different Pickman versions. Chapter 5 summarizes the work and suggests possible extensions.

Chapter 2

Background

A definition of artificial intelligence and an introduction to earlier research is attempted in this chapter. This is not without its challenges; examples from various sources illustrate the diversity of opinion within the field:

> "Artificial intelligence is the design and study of computer programs that behave intelligently." (Dean et al. 1995, p.1)
>
> "Artificial intelligence may be defined as the branch of computer science that is concerned with the automation of intelligent behavior." (Luger & Stubblefield 1993, p.1)
>
> "Any problem for which no algorithmic solution is known is a problem in Artificial Intelligence." (Lauriere 1990, p.3)
>
> "Artificial intelligence is the study of techniques for solving exponentially hard problems in polynomial time by exploiting the knowledge about the problem domain." (Rich 1983, p.37)

While this collection of definitions is not exhaustive, it is a representative sample. All definitions above imply that AI involves computers and most mention the relevance of solving problems in one way or another. Given the focus on computers and problem solving, it is not surprising to find research on reasoning systems from the very early years of the field. Singh (2005, p.114-126) examines many of these early models and Davis & Morgenstern (2004) provide a comprehensive overview of more recent systems.

The rest of this chapter focuses on Marvin Minsky and Push Singh's work on reasoning systems. As mentioned previously, their research is the primary inspiration for Pickman's Machine. The motivation behind the present thesis resembles what Singh has expressed for EM-ONE:

> When Minsky's book *The Society of Mind* was published, implementations of the theory did not follow shortly thereafter. I hope

that this thesis leads to many more implementations of Minsky's ideas, by giving workers in AI one example of how the theories in *The Emotion Machine* can be brought to life. (Singh 2005, p.146)

2.1 Push Singh and Reasoning Architectures

2.1.1 Failure-Directed Reformulation

Push Singh's work in reasoning architectures centers on commonsense systems. In his master's thesis (Singh 1998), he examines the role of multiple representations in a problem-solving system and proposes a reasoning mechanism that *reevaluates* failures to solve problems. He provides an implementation in which an actor attempts to carry a foldable box through a door that is not wide enough. The primary actor executing the task represents the box internally using several different methods. He argues that a failure recovery system can be created by supplying the actor with multiple methods to represent the problem.

In a system with multiple representations, each representation can be useful in reaching a potential solution. When solving a problem, the system can choose the representation that simplifies the problem the most. When the representations are organized into *difference networks*, a better alternative can be chosen in case of failure by following a *difference pointer*. Singh calls this method of organizing representations based on goals *failure-directed reformulation*. In a difference network, a cup and a chair can be classified by their difference in weight when the problem requires a heavy object and by their height when the problem involves building a tower. He states that "robustness and flexibility comes from having many ways to approach problems" (Singh 1998, p.56).

Singh concludes his thesis with a defense of microworlds by illustrating that several microworlds can be combined and used as sources of analogies to approach more complicated problems. Microworlds enable the problem solver to refactor large problems into several smaller ones with known solutions (Singh 1998, p.51).

2.1.2 EM-ONE

In EM-ONE (2005), Singh describes an architecture for commonsense thinking. EM-ONE reasons by applying *mental critics* to its library of narratives about physical, social and mental activities. Within the EM-ONE architecture, knowledge is represented by narratives. These narratives are authored using a frame-based description language (Minsky 1975). Singh argues (2005, p.28) that representation using narratives is superior to representing knowledge as a collection of abstract rules, because narratives connect knowledge to purpose, help us control inference, are easy to acquire and can contextualize knowledge. The mental critics of EM-ONE are arranged into three layers and are guided by separate meta-critics.

As an implementation example, Singh provides a scenario in which two actors using the EM-ONE architecture work together in *Roboverse* (Singh 2005, p.83), an artificial life environment. One of the actors in the simulation desires to build a table. By using its critics network and the narrative collection, it asks the second actor for help. Working together, they manage to attach sticks to a board and build a table. While the domain is simple compared to the real world, possible variations in scenarios are different enough from the actors' built-in scenarios to make mistakes possible. Encountering these mistakes while following narratives causes the actors to reflect upon their choices and revise their plans.

2.2 Marvin Minsky's Emotion Machine

In *The Emotion Machine* (2006), Marvin Minsky offers a theory of the human mind. Due to the size and scope of his work, the Pickman model should be viewed as validating a small subset of Minsky's cognitive architecture. The reader is encouraged to consult *The Emotion Machine* for a thorough discussion of many issues surrounding reasoning mechanisms that are not explored in the following chapters.

The Emotion Machine employs a layered model of reasoning. It is composed of six layers stacked on top of each other. Starting from the simplest, these are:

1. Instinctive Reactions
2. Learned Reactions
3. Deliberative Thinking
4. Reflective Thinking
5. Self-Reflective Thinking
6. Self-Conscious Emotions

The layers are ordered in increasing complexity. The reactive layers produce actions as direct results of events that happen in the external world. The deliberative and reflective layers involve previous observations and reactions to "happenings inside the brain" (Minsky 2006, p.130) in the action creation process. The uppermost two self-related layers act upon previously created plans and ideals.

The Emotion Machine's layers contain critics that guide the behavior of an actor. Pickman, on the other hand, uses only narratives organized into two different layers. This is where Pickman's Machine diverges most from Minsky and Singh's models, as discussed further in the next chapter.

Chapter 3

The Pickman Architecture

The Pickman architecture is a minimalist approach to reasoning that consists of three components: the memory, the narrative collection and the reasoning layers. The memory is divided into two parts: short-term and long-term. The short-term memory contains observations from the current cycle. At the end of every cycle the observations are removed from short-term memory and stored in long-term memory. The narrative collection is also divided into two groups: reactives and deliberatives. These contain scripts used by the reasoning layers during the decision-making process. The reasoning layers draw from the narrative collection, the memory and the desire list to produce new actions.

3.1 A Pickman Cycle

When Pickman is running, it executes a certain set of commands repeatedly. One iteration of this command set is called the Pickman cycle. During these cycles Pickman observes its surroundings, activates the reactive layer to decide on an action that is consistent with the current desire and executes that action. If the current desire list is empty, an action cannot be generated. When this occurs, the deliberative layer is activated to generate new desires. At the end of every cycle, Pickman's observations and the executed action are committed to memory for future use by the deliberative layer. The strict ordering of the Pickman cycle enables the system to function without a meta-layer.

Observe populates Pickman's internal model of the world using the current state of the realm. The name of the realm and its contents are added to the observations array. This array stores all actors who are currently in the same realm as Pickman and their health states.

React searches for narratives that can satisfy the first entry in the desire list. After all such narratives are found, the first narrative with a PRE frame that matches the current observations is executed. It should be noted that the ordering of the narratives influences this choice, as discussed in the "Selecting Narratives" section. If no suitable narrative is found, *react* moves to the next desire and repeats the process. If the desire list is empty or if none of the narratives can fulfill the current desires, *deliberate* is called.

Deliberate searches for a narrative whose PRE frame matches a long-term memory entry. If a matching entry is found, then the POST frame of that narrative is compared with the current observations. If a suitable match is found in the observations, the narrative's ACT frame is added to the desire list. If no matching narratives are found and the desire list is empty, the *keep-alive counter* is incremented. This counter tracks the number of cycles Pickman has spent without a desire. If a react action was not performed during the current cycle, *deliberate* invokes the reactive layer.

Execute parses the ACT frame and performs the specified action. If there is no action to be executed, it compares the value of the *keep-alive counter* to the maximum memory size. Pickman terminates when the counter value is greater than the maximum memory size.

3.2 Narratives

Narrative arrays contain collections of events that could potentially occur in a domain. The collections for the reactive and the deliberative layers are called *reactives* and *deliberatives.*

All narratives are written in a frame-based language where every frame is represented by a series of binary predicates (Minsky 1975), (Singh 2005). A narrative is composed of three frames in the form of a Hoare triple (Hoare 1969):

PRE holds the domain state description to be matched,

ACT specifies an action that can be performed by the actor,

POST describes the final state of the domain after the action is taken.

At the beginning of every reactive or deliberative layer execution call, Pickman searches the corresponding narrative collection for narratives containing a PRE that is currently true. For reactive narratives, if such a PRE is found, its outcome is compared to Pickman's current desire. If there is a matching outcome, the actions listed in ACT are performed. If no narrative contains a desirable outcome, the deliberative layer is invoked for suggestions on how to proceed.

3.2.1 Examples

Example narratives from the current implementation are provided below. It should be noted that if a narrative is reactive, its ACT frame specifies an action that can be executed by Pickman. If the narrative is deliberative, it specifies a desire that can be added to Pickman's desire list. A simpler way of stating this is: Reactive generates actions, deliberative generates desires.

Reactive Heal actor:

```
narratives[k].pre = {"isNotHealthy", "actor"}
narratives[k].post = {"isHealthy", "actor"}
narratives[k].act = {"HEAL", "actor"}
```

Reactive Move to another realm:

```
narratives[k].pre = {"pickmanNotAt", "realm"}
narratives[k].post = {"pickmanAt", "realm"}
narratives[k].act = {"MOVE_TO", "realm"}
```

Deliberative Heal previously healed actor:

```
narratives[k].pre = {"healed", "actor"}
narratives[k].post = {"isNotHealthy", "actor"}
narratives[k].act = {"isHealthy", "actor"}
```

Deliberative Move away if the realm is empty:

```
narratives[k].pre = {"pickmanAt", "realm"}
narratives[k].post = {"isEmpty", "realm"}
narratives[k].act = {"pickmanNotAt", "realm"}
```

3.2.2 The Narrative Potential

Narratives organized as Hoare triples can be used to describe a wide variety of situations. Most of these are not relevant to the current implementation and are not included in Pickman's narrative collection. However, some examples are given below to illustrate the flexibility of the representation:

Reactive Avoid infected actor:

```
narratives[k].pre = {"isNotHealthy", "actor"}
narratives[k].post = {"isHealthy", "pickman"}
narratives[k].act = {"MOVE_AWAY", "realm"}
```

Reactive Search for an empty realm:

```
narratives[k].pre = {"isNotEmpty", "realm1"}
narratives[k].post = {"isEmpty", "realm2"}
narratives[k].act = {"MOVE_AWAY", "realm1"}
```

Deliberative Move to an occupied realm that used to be empty:

```
narratives[k].pre = {"isEmpty", "realm"}
narratives[k].post = {"isNotEmpty", "realm"}
narratives[k].act = {"pickmanAt, "realm"}
```

3.2.3 Selecting Narratives

Pickman's behavior depends entirely on the narrative that is followed during a cycle. Consequently, the narrative selection method can strongly influence Pickman's behavior pattern. To minimize the impact of the selection strategy on the overall behavior, the selection process always picks the first match it encounters. The narratives are listed in the same order they appear in the source code and the ordering does not change while Pickman is running. This simple implementation aims to move the focus away from the efficiency of the selection strategy to highlight the interaction between the reactive and the deliberative layers.

3.2.4 Specialized Narratives

The categorization of every narrative as either reactive or deliberative is mutually exclusive and its current implementation uses two separate layers. However, a meta-level categorization of the supplied narratives is also possible.

The narratives can be categorized as general purpose or domain-specific based on their relevance in a particular domain. Domain narratives can then be applied to problems within specific domains while general purpose narratives can be used to reason about problems regardless of their domains. The border between these two categories depends on the definition of the current domain. Narratives of a certain domain can certainly be applicable in another domain with similar characteristics. The division merely highlights the fact that some narratives are utilized more often. This property does not need to be built into the system explicitly and it provides an insight that higher reasoning layers can build upon.

3.3 Implicit Inference

Pickman starts with no initial knowledge of the domain except the narrative collection. Therefore, it has to infer everything using these narratives and its own observations. Primitives that include negations (such as *isNotHealthy* or *isNotAt*) are created using a mechanism called implicit inference. These can be viewed as very simple narratives with no *ACT* component. Alternatively, they can be considered If−>Then relations that are stored in suitable narrative frames. They provide a simple way for Pickman to derive knowledge from observations without invoking the reactive cycle. Pickman assumes that if an observation is true, its negation exists and is false. While this significantly simplifies the process for many observations, it requires narratives to be authored with care. All observable primitives must also have valid negations; this is the case for pairs such as *isHealthy* and *isNotHealthy* or *isAt* and *isNotAt*.

The importance of implicit inference can be demonstrated with an example. When Pickman observes an actor that has the property "isNotHealthy", it can infer that the actor's health state can be changed to "isHealthy". However, Pickman does not have any information on what form the bearer of the "isNotHealthy" property should have. Therefore, if a room with the "isNotHealthy" property were encountered, Pickman would try to apply the same healing technique to the room and fail miserably.

There are two ways of preventing such actions. The first is by choosing property names that are unique and not replicated in different object types in the world. However, this approach may not be possible if the system contains millions of object types. An example of a system that deals with this issue and many other relevant problems is Cyc (Lenat et al. 1990), a vast ontology and database of commonsense knowledge.

Another way of preventing invalid inferences is to design *ACT* statements that target specific object types. This would make it impossible to apply the *HEAL* action to a room by design. In both cases, a human-centric way of viewing these changes is to assume that Pickman implicitly infers a room is not a suitable target for healing (even though Pickman has no information about the chances of such an act succeeding or failing).

A more complex implementation of Pickman's Machine could use explicit inferences where every inference is stored in narrative form. While this would complicate the translation process from narrative to action, it would make Pickman much more flexible in different domains.

3.4 Memory

Pickman stores its observations and the actions it has taken during the past cycles in the memory. The memory is divided into two parts: short-term and long-term. The short-term memory contains the observations from the current cycle. At the end of the cycle, the action taken by Pickman during that cycle is appended to this collection and the resulting list is transferred to the long-term memory.

A single observation is represented as a vector of strings. The short-term memory stores these in a vector of string vectors. The long-term memory collects the short-term memory vectors in a static array, making it an array of vectors of vectors of strings.

In the current version, the memory is implemented as a FIFO queue. This means that when the memory limit is reached, the oldest cycle entry is removed from the beginning and the most recent observation is added to the end.

After the short-term memory has been populated, its contents have the following form. In the example, Pickman is in a realm called Air with three other actors:

Vector Element	Value
pickman.observations["one"]	isHealthy
pickman.observations["three"]	isHealthy
pickman.observations["four"]	isNotHealthy
pickman.observations["pickmanAt"]	Air

3.5 Reasoning Layers

The Pickman architecture includes two reasoning layers; reactive and deliberative. Each layer contains its own narrative collection. The reactive layer is used to generate responses to events based on current observations. The responses aim to fulfill one or more of Pickman's desires stored in the desire list. When a reactive narrative is chosen, the contents of its ACT frame are executed. The deliberative layer is used to generate new desires for Pickman. It is activated when the reactive layer cannot find a narrative to satisfy any of the current desires. It can also be activated if the desire list is empty. When a deliberative narrative is executed by the deliberative layer, its ACT frame is added to the list of desires.

The main difference between the reasoning layers is that the reactive layer uses only current observations when choosing an appropriate narrative while the deliberative layer consults both current and previous observations (the memory) before modifying the desire.

3.5.1 The Reflective Layer

In both the Emotion Machine and EM-ONE the reflective layer is responsible for evaluating and modifying the decision process of the deliberative layer. However, this requires a complex interaction between the narratives and the deliberative layer. If the narratives can be modified by the deliberative layer, the changes have to be evaluated by an external mechanism that can prevent critical errors. This is the role typically fulfilled by the reflective layer. Since the Pickman architecture does not reach this level of complexity, adding a reflective layer at this stage would be redundant.

The scheduling of reflection (or of any other meta-process) presents an interesting problem. How can a system decide when to reflect? As the focus of problem solving moves from external events to internal processes, the need for a control mechanism arises. This mechanism needs to weigh the potential gain from self-improvement against the resources that will be consumed in the process and the time that could be spent solving external problems. Even when such a mechanism is implemented, the question is not fully answered. How can a system evaluate the results of its reflections?

EM-ONE sidesteps this issue by including a separate meta-layer that directs the behavior of all other layers. In contrast, the Emotion Machine resolves the issue by terminating the cycle of meta-deliberation at the upper self-conscious layers. These layers, capable of running limited simulations of the whole model, are powerful enough to evaluate themselves. In general, such systems are said to be anticipatory. Rosen defines an anticipatory system as "a system containing a predictive model of itself and/or of its environment, which allows it to change state at an instant in accord with the model's predictions pertaining to a latter instant" (Rosen 1985, p.339). Davidsson (1996) provides an implementation example, and further examines issues related to computational anticipatory systems.

3.6 Mental Critics

Problem solving can be defined as the process of reducing the differences between the current situation and the desired outcome. *Mental critics* attempt to solve problems by first recognizing and then removing such differences. In general, critic systems employ procedures that are aimed specifically at detecting and preventing failures.

Critics are a crucial component of both EM-ONE and the Emotion Machine. Pickman's Machine, on the other hand, does not include critics. Since Pickman's narratives are structured as Hoare triples, interpreting them is simpler when compared to an EM-ONE narrative. Instead of using complex narratives and specialized critics, Pickman's Machine implements the reactive and the deliberative layers as general narrative interpreters. This limits the flexibility and power of narratives but greatly simplifies the system.

For example, the reactive EM-ONE critic "Difference between conditions and desires−>Propose action by analogy" summarizes the functionality of Pickman's reactive layer as a whole because the reactive layer calls either *deliberate* or *execute* in every cycle. A meta-critic from EM-ONE, "Unachieved Desire−>React" is also replaced by the last step of the Pickman cycle, where actions proposed by the reactive layer are executed. The functionality of many other EM-ONE critics is not present in Pickman because

Pickman exists in a simpler world. The reactive EM-ONE critic “Explicitly Communicate Intent” which communicates the action to be taken to other actors is a good example. Since Pickman’s world does not include any other cognitive actors there is no need to communicate intent.

Chapter 4

Implementation

A C^{++} implementation of Pickman's Machine and experiments performed with three different versions of the architecture using five different scenarios are presented.

4.1 The Domain

4.1.1 Realms

In the current implementation, the world is composed of four separate realms. Each realm can be reached from the others with a single move action. The contents of a realm can only be observed by those inhabiting it. Initially, each realm contains a single actor. There are no size constraints, every realm is large enough to hold all actors and Pickman.

4.1.2 Actors

The actors are vectors for the virtual plague, implemented as objects that can move between realms at will. Every actor takes one action per cycle. If an actor is infected it automatically infects other actors it encounters. This ability is considered to be a free action; it is triggered whenever an infected actor is in the same realm with one or more healthy actors.

4.1.3 The Plague

The virtual plague is an internal property represented using a boolean value. In three of the scenarios (introduced in the next section), it flags the actors as infected but does not affect their behavior. In the remaining two, it modifies the behavior of the infected actors as well. This second version of the plague is closer to the definition,"a process in which a behavior-affecting property spreads among characters" provided by Boman & Johansson (2007, p.1).

4.2 Experiments

In all experiments, Pickman is tasked with removing the plague from all actors. Three different Pickman versions are tested in five scenarios for a total of 15 experiments. Every experiment is repeated 1000 times with different seeds. Pickman is immune to infections in all experiments. It should be emphasized that the Pickman cycle only includes a single *execute* call, meaning that Pickman must choose either to move or to heal during each cycle.

4.2.1 Pickman Versions

The experiments are performed using three different Pickman versions; dynamic, static and random. The dynamic Pickman uses both the deliberative and the reactive layer. The static Pickman uses only the reactive layer and a fixed desire-generation mechanism. The random Pickman does not have any cognitive capability; it takes a random action each turn regardless of the world state.

The motivation behind selecting these versions for the experiments is to observe the effects of including a deliberative layer on Pickman's behavior. The random Pickman is used as a baseline to measure the performance of the other two versions.

A challenge in building a reactive-only Pickman is creating a mechanism to keep it alive. Normally, Pickman stays alive to fulfill its desires and terminates shortly after all its desires have been fulfilled. Since only the deliberative layer can create desires, an unmodified, reactive-only Pickman terminates very quickly. To prevent an early termination, the static Pickman uses a separate desire generation procedure. This procedure is responsible for generating *MOVE* and *HEAL* desires. A *MOVE* desire targeting a random realm is generated when the current realm is empty while a *HEAL* desire is generated when an infected actor is encountered. The desire generator enables the static Pickman to perform the same actions as the random and the dynamic versions and also prevents an early termination due to a lack of desires.

4.2.2 Scenarios

The three Pickman versions are tested in five scenarios. These are:

1. All actors infected, no actor moves.
2. One actor infected, actors move randomly.
3. All actors infected, actors move randomly.
4. One actor infected, actors move based on infected/healthy behavior.

5. All actors infected, actors move based on infected/healthy behavior.

The first scenario demonstrates Pickman's behavior in a static world. It is designed to highlight the effect of a changing world on behavior. The second and the third scenarios demonstrate how Pickman behaves in a chaotic world because in these cases there is no apparent logic behind the actors' movements.

The last two scenarios are true virtual plague cases which test Pickman's decision-making capabilities due to the fact that there is an underlying pattern for the actors' behavior. In these scenarios, the actors try to avoid each other when healthy by constantly searching for empty realms. When an actor is infected, it tries to find realms with healthy actors and infect them as well. These two scenarios are the most challenging because the infected try to spread the plague actively.

4.2.3 Results

After each run, the following values are stored for analysis:

- Simulation duration (in cycles)
- Remaining infected actors
- Number of realm-to-realm moves made by Pickman
- Number of actors healed by Pickman

As mentioned previously, one metric that can be derived directly from the values above is *cycles per heal*. This value is useful in understanding how much time Pickman spent searching for an infected actor. Table 4.1 displays the *cycles per heal* values of different Pickman versions for every scenario. The corresponding variance values are listed in Table 4.2. *Remaining infected actors* is not reported because it is used only to verify whether the plague has been eradicated. *Number of realm-to-realm moves* is not reported either because it is proportional to *cycles per heal* in almost

Scenario	Random	Static	Dynamic
1	18.29	∞	8.38
2	9.09	3.21	5.75
3	5.63	2.66	3.34
4	14.15	3.12	10.63
5	5.41	2.18	3.16

Table 4.1: Cycles per heal for different Pickman versions. The scenario 2 value for the static version was measured in 324 runs out of 1000. The scenario 4 value for the same version was measured in 96 runs out of 1000.

Scenario	Random	Static	Dynamic
1	175.00	–	4.11
2	1604.98	0.75	138.23
3	30.89	0.40	1.99
4	2623.53	0.77	202.80
5	32.85	0.25	1.93

Table 4.2: Variance values for Table 4.1.

all scenario-version pairs and does not provide additional information about the simulations.

4.2.4 Analysis

The results demonstrate that the dynamic version succeeded in healing the entire population in more scenarios. The static version, in contrast, was more efficient in the small number of runs where it managed to heal the whole population. In the hypothesis, efficiency was defined in terms of infection duration in cases where the entire population was healed faster. Sticking closely to this definition, it is trivial to show that the dynamic version is more efficient overall. However, a more interesting result is hidden in the cases where the static version was more efficient. Since the static version could only react to changes in the world, it failed to fulfill any of its desires in some scenarios. This led to an infinite loop in which Pickman did not perform any action but failed to terminate because it had unfulfilled desires. The infinite loop was observed in all runs of the first scenario, in 676 runs of the second scenario and in 904 runs of the fifth scenario. Although very unreliable, the static Pickman was actually a better choice for solving the problem in certain situations.

Both dynamic and static versions of Pickman were more efficient than a random approach on each case where the plague was eradicated successfully. The difference between the static and dynamic versions was, predictably, one of flexibility. The dynamic Pickman successfully handled every scenario, while the static version was occasionally stuck in an infinite loop. In scenarios 3 and 5 where it avoided the loop, it was more efficient than the dynamic version. It also performed better in 32% of scenario 2 runs and 10% of scenario 4 runs but failed to terminate in the remaining runs.

One conclusion that can be drawn from these results is that a static desire generator can be used to increase efficiency if the scenario is chosen carefully. A deliberative model, in contrast, is slower, but much more reliable.

Chapter 5

Conclusion

A two-layered architecture for reasoning that uses narratives to guide its behavior was presented. A C^{++} implementation of the proposed architecture was tested in five scenarios where three different Pickman versions were tasked with healing a small population suffering from a virtual plague. It was shown that a Pickman implementation using a static desire generation mechanism was efficient but unreliable. The dynamic version, on the other hand, was not as efficient but it was able to heal the population successfully in all scenarios.

5.1 Possible Extensions

Learning from Failure An interpreter for data collected from previous experiences can enable Pickman to learn from the results of its own actions. One possible implementation is producing new narratives from failures and integrating these into the narrative collection. Such an addition would require failures to be stored in a form accessible to the system itself.

Generalized Narratives As stated in Section 3.2.4 Specialized Narratives, the current Pickman architecture can be extended by organizing narratives based on their relevance to a particular domain where those that are applicable in a large number of domains are classified as general narratives. Pickman can then use these narratives to make decisions in domains where no domain-specific narrative exits.

Frame Size Currently all narrative frames contain two slots. The frame structure can be expanded to include a variable number of frame slots useful for representing more complex situations using narratives. If the number of slots in a given frame are specified within the frame itself, the interpreting mechanism can select a suitable parser for the additional slots. As an example, a reactive narrative with a three-slot PRE frame for following an actor is given below:

Reactive Follow an actor:

```
narratives[k].pre = {"movedTo", "actor", "realm"}
narratives[k].post = {"pickmanAt", "realm"}
narratives[k].act = {"MOVE_TO", "realm"}
```

5.2 Reflections

When designing Pickman's Machine, programming the implementation was a major part of the design process. Details unintentionally obscured in theory were revealed when the concepts were transferred to code. Even though the final result presented here is primarily a theoretical model of reasoning, attempts at programming the architecture were the best tests of consistency for the model during the development process.

As a final remark, it should be mentioned that the main question which drove the present thesis was: "Can we build systems flexible enough to handle the diversity of the real world?" The answer is certainly larger than what a single master thesis can hope to accomplish. Pickman's Machine was an attempt to explore a small part of this problem in detail by creating a simple yet self-sufficient reasoning mechanism.

References

Boman, M. & Johansson, S. J. (2007), Modeling epidemic spread in synthetic populations - virtual plagues in massively multiplayer online games, *in* B. Akira, ed., 'Situated Play', The University of Tokyo, pp. 357–361.

Davidsson, P. (1996), Autonomous Agents & the Concept of Concepts, PhD thesis, Lund University.

Davis, E. & Morgenstern, L. (2004), 'Introduction: Progress in formal commonsense reasoning', *Artificial Intelligence* **153**(1-2), 1–12.

Dean, T., Allen, J. & Aloimonos, Y. (1995), *Artificial Intelligence: Theory and Practice*, The Benjamin/Cummings Publishing Company.

Hoare, C. A. R. (1969), 'An axiomatic basis for computer programming', *Communications of the ACM* **12**(10), 576–583.

Lauriere, J.-L. (1990), *Problem Solving and Artificial Intelligence*, Prentice Hall International.

Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D. & Shepherd, M. (1990), 'Cyc: Toward programs with common sense', *Communications of the ACM* **33**(8), 30–49.

Luger, G. F. & Stubblefield, W. A. (1993), *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, The Benjamin/Cummings Publishing Company.

Minsky, M. (1975), The psychology of computer vision, *in* P. H. Winston, ed., 'A Framework for Representing Knowledge', McGraw-Hill, New York, NY, pp. 211–277.

Minsky, M. (2006), *The Emotion Machine*, Simon and Schuster, New York.

Rich, E. (1983), *Artificial Intelligence*, McGraw-Hill.

Rosen, R. (1985), *Anticipatory Systems: Philosophical, Mathematical and Methodological Foundations*, Pergamon Press.

Singh, P. (1998), Failure-directed reformulation, Master's thesis, MIT.

Singh, P. (2005), EM-ONE: An Architecture for Reflective Commonsense Thinking, PhD thesis, MIT.