Gathering and utilising domain knowledge in commercial computer games

Sander Bakkes Pieter Spronck

Universiteit Maastricht / MICC-IKAT
P.O. Box 616, NL-6200 MD Maastricht, The Netherlands
{s.bakkes, p.spronck}@micc.unimaas.nl

Abstract

In practice, adaptive game AI in commercial computer games is seldom implemented, because machine learning techniques require numerous trials to learn effective behaviour. To allow fast adaptation in games, in this paper we describe a means of learning that is inspired by the human capability to solve problems by generalising over limited experiences with the problem domain. We compare three approaches, namely straightforward case-based reasoning, situated case-based reasoning, and a k-nearest neighbour classifier. From the experimental results we conclude that the situated approach performs best, both in representing knowledge and generalising to similar situations.

1 Introduction

Computer-controlled agents in commercial computer games are often called 'Non-Player Characters' (NPCs). NPCs are typically situated in rich and complex virtual environments, which excel in visual realism and realistic physics simulation. In such an environment one would expect the NPCs to behave realistically ('human-like') too. However, the current state of the art in this respect leaves much to be desired [6]. One aspect of human-like behaviour of NPCs, namely the ability to adapt to changing circumstances, has been explored with some success in recent research [3, 5, 11]. This is called 'adaptive game AI'.

When implemented successfully, adaptive game AI is able to (1) fix errors in programmed game AI, and (2) seek counter-tactics to human gameplay. It is imperative that adaptive game AI is based on domain-specific knowledge [7, 10]. Since a game's engine will be regularly updated, especially in the popular Massive Multiplayer Online Games (MMOGs), domain knowledge on what sort of behaviour is successful in the current environment should be automatically gathered. Human beings are able to deduce successful behaviour effectively from observations [9]. Our goal is to endow NPCs with a similar capability.

Within a typical commercial computer-game environment most machine-learning techniques are unsuitable for our goal, since they require numerous trials, make numerous mistakes before obtaining successful behaviour, or are computationally intensive [10]. Humans are capable of reasoning reliably on a preferred course of action with only a few observations on the problem domain. This paper investigates to what extent NPCs can gather domain knowledge from a few observations, and immediately (i.e., without trials and without resource-intensive learning) utilise that knowledge to evoke effective behaviour.

The outline of the paper is as follows. The environment we use for our research is discussed in Section 2. In Section 3 we discuss how we gather domain knowledge in a case-base. Three approaches for utilising this domain knowledge are described in Section 4. Section 5 describes an experiment to test the performance of the three approaches and provides the experimental results. Section 6 concludes and looks at future work.

2 Simulated environment

The research discussed in this paper is focussed on designing approaches for online gathering and utilising domain knowledge in commercial computer-game environments. For our investigation we have created a simulator environment representing a simple game. The game is an obstacle course, in which an NPC has to travel from the bottom of a grid to the top of the grid. The grid is 12 cells wide and 100 cells high. Each cell of the grid can contain an object. The following six object types can be placed on the grid:

1. **NPC.** One NPC is located in the grid. It is the only object that can change its position. Initially the NPC is placed in a random empty location on the bottom row of the grid. To manoeuvre to its goal (i.e., the top row of the grid) it has three actions available: (1) to move to the cell directly to its upper left, (2) to move to the cell directly above it, and (3) to move to the cell directly to its upper right. As input, the NPC can observe all cells of the five rows above it. It has two properties: health and fitness. Initially it is provided with a health value of 100. The NPC 'dies' (i.e., is removed from the grid) when its health reaches zero. The fitness value of the NPC is determined when it either dies or reaches the top row of the grid. It is calculated as $\frac{S}{H-1}$, where S is the number of steps taken by the NPC, and H is the number of cells the grid is high. On the screen, the NPC object is visually represented by a light-blue colour.

2. **Wall.** All the leftmost and rightmost cells of the grid contain wall objects. When an NPC tries to enter a cell in which a wall object is located, it dies. On the screen, the wall object is visually represented by a black colour.

3. **Goal.** All the cells on the top row of the grid (except for those containing a wall object) contain goal objects. When an NPC enters a cell in which a goal object is located, its fitness is set to 1.0 and it is removed from the grid. On the screen, a goal object is visually represented by a grey colour.

4. **Tree.** An arbitrary number of trees can be located in the grid. Trees are treated as walls, i.e., when an NPC tries to enter a cell in which a tree is located, it dies. On the screen, a tree object is visually represented by a green colour.

5. **Turret.** An arbitrary number of turrets can be located in the grid. When an NPC is within reach of a turret (defined as, within a square of seven cells on each side and the turret at its centre), its health is reduced by 10. Furthermore, when the NPC enters a cell where a turret is located, it dies. On the screen, a turret object is visually represented by a brown colour.

6. **Mine.** An arbitrary number of mines can be located in the grid. NPCs can *not* observe mines. NPCs are allowed to move into a cell containing a mine. However, when that happens the mine 'explodes', reducing the NPC's health by 33. On the screen, a mine object is visually represented by a red colour.

An NPC can co-exist with another object in a cell, but of all the other objects each cell can contain at most one. The three grids designed for our experiments are displayed in Figure 1.

3 Gathering domain knowledge in a case-base

Case-Based Reasoning (CBR) is a machine learning paradigm for reasoning over a collection of previously observed situations. In commercial games, the environments NPCs are situated in typically excel in richness and visual complexity. This poses a challenge for a case-based reasoning approach as the environment information that is gathered by the NPC should, for effective and rapid use, be (1) represented in such a way that stored cases can be reused for previously unconsidered situations, and (2) compactly stored in terms of the amount of retrievable cases [1].

In our simulation, we wish to store the observations of an NPC, with the action it took, and an assessment of the success of that action. Regarding the representation we decided to store observations in the form of *labelled state-action pairs*. Each case consists of an abstraction of the world

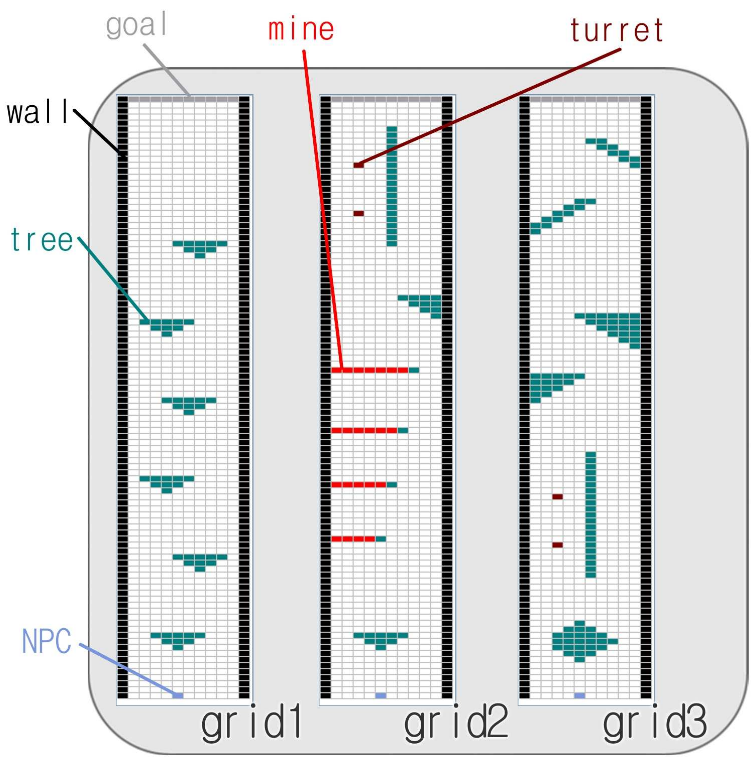


Figure 1: Three grids designed for our experiments. Note that mines are invisible to the NPC.

visible to the NPC, together with the NPC's action. The environment state is abstracted relative to the NPC position (rather than from an absolute viewpoint). This abstraction is illustrated in Figure 2. Additionally, each state-action pair is labelled (and relabelled during a simulation trial) with the average observed fitness the stored action has led to when applied. The labelling process is algorithmically described below.

```
//Labelling algorithm
1. To determine the fitness value, observe the NPC until it reaches a goal cell or
dies.
2. Retrieve NPC–observations for every step performed.
3. Check for each retrieved observation whether it is already stored.
4a. If so, update the fitness value of the retrieved observation by averaging over
both fitness values.
4b. If not, store the particular observation with the action performed and the
fitness obtained by the NPC.
```

Regarding a compact storage of cases, we decided to reduce the possible number of cases stored in the case-base by only storing those state-action pairs that did not directly lead to a death in the next observable state. Accordingly, a labelled case-base arises of which all state-action pairs lead to either a local optimum or to the global optimum.

4 Proposed approaches for utilising domain knowledge

In this section we discuss three approaches we propose for utilising domain knowledge in our simulator. The approaches are based on the *reuse* process-step of the case-based reasoning paradigm, in which solutions from previously stored cases are mapped to a new case.

Our case base contains experiences of NPCs traversing grids. To utilise the case base for a new case, similarity values are calculated between the list of observations of the new case, and the lists of observations of all the cases in the case base. Of the most similar cases, the particular case with the highest fitness value is retrieved, and the action which was performed by the NPC for that case is selected for the new case.

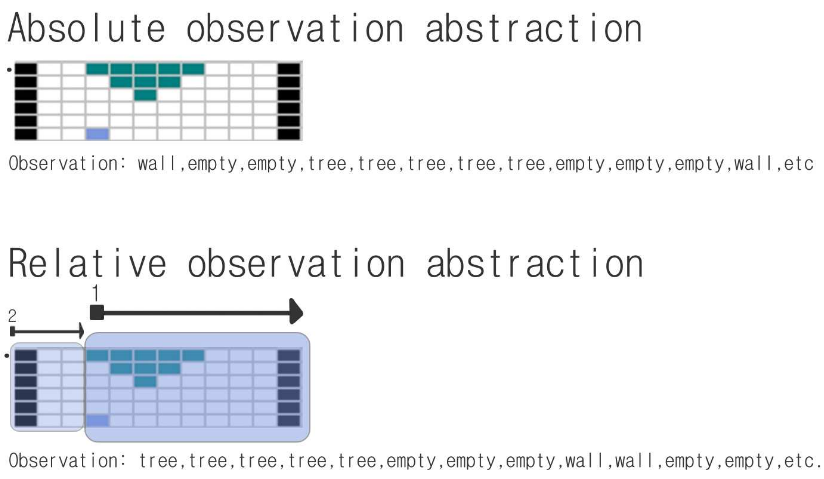

Absolute observation abstraction
Observation: wall,empty,empty,tree,tree,tree,tree,tree,empty,empty,empty,wall,etc
Relative observation abstraction
1
2
Observation: tree,tree,tree,tree,tree,empty,empty,empty,wall,wall,empty,empty,etc.

Figure 2: Example of an absolute and relative observation abstraction. First the top-row cells are observed from left to right, subsequently the rows below are observed analogously and last the gathered information is stored. In our experiment we used the relative observation abstraction.

The three proposed approaches vary in their similarity-calculation algorithm. They are: (1) Straightforward CBR (Subsection 4.1), (2) Situated CBR (Subsection 4.2), and (3) k-Nearest Neighbour Classification (Subsection 4.3). The straightforward CBR approach was chosen as a way of baseline comparison. The situated CBR and the k-nearest neighbour classification approaches where chosen under the presumption of improved performance with regard to the use of relational features [4] and an increase in generalisation power, respectively.

4.1 Straightforward CBR

For straightforward CBR, similarity is calculated by a syntactic matching in which every identical observed feature receives the same weight. The algorithm scales the similarity value to 100% when all observed features are identical.

When similarity values are calculated for each case in the case-base, the matching algorithm subsequently retrieves the case with the highest similarity value within a *similarity-window*. The similarity-window defines the threshold similarity value of cases to be denoted as 'similar'. Should multiple cases be retrieved, the first cases with the highest fitness value is selected for execution. Should no cases be retrieved, the process is iterated by adjusting the similarity-window so that it allows for less-similar cases to be retrieved. This process is illustrated in pseudo-code below.

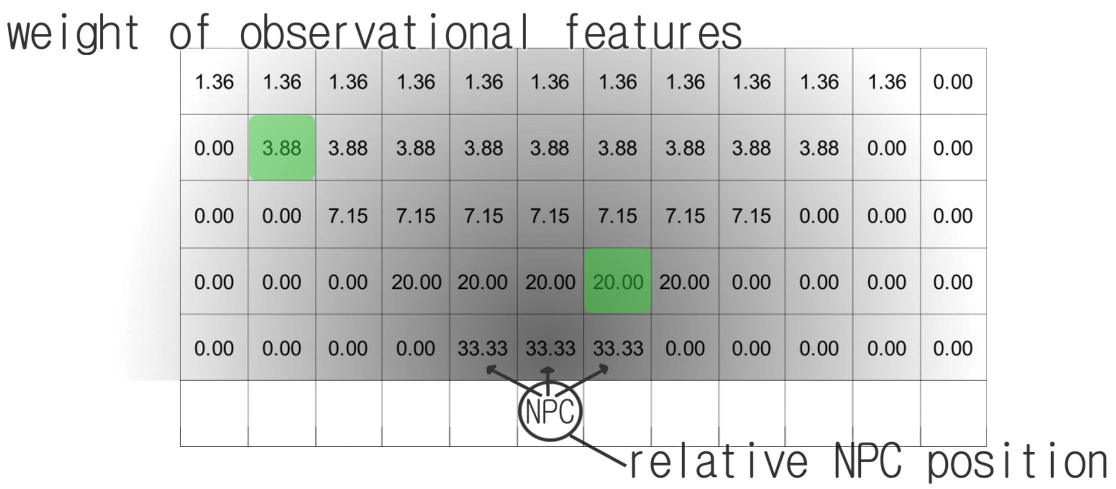
```
function calcSimularity(oldcase, observation);
begin
 similarity := 0;
 for (all_features_in_observation) do
  if (observation.feature = oldcase.feature)
   then similarity := similarity + observation.feature.Weight;
 result := scale_to_100percent(similarity);
end;

function selectMostSimilarObservation();
begin
case_selected := false;
fitness_best := 0;
similarity_window := 100;
repeat
 begin
  for (all_cases_in_the_casebase) do
   begin
    similarity_value := calcSimilarity(oldcase, currentObservation);
    if (similarity >= similarity_window)
     then
```

```
      if (oldcase.fitness > fitness_best)
       then begin case_selected := true; fitness_best := oldcase.fitness; end
    end;
   if not(case_selected) then
    begin similarity_window := similarity_window - 2; fitness_best := 0; end;
  end;
until (case_selected);
end;
```

4.2 Situated CBR

For situated CBR (i.e., CBR where environmental observations are evaluated from the point of the view of the NPC), similarity is calculated by a semantic matching in which features closer to the NPC are weighed heavier than features further from the NPC. For instance, an observational feature directly above an NPC is assigned a larger weight than a feature that is located two rows above the NPC. The weights assigned to observational features are given in Figure 3. For the situation portrayed in the figure, the observational feature on the left (the darker cell) will receive a relatively low weight as it is quite far from the NPC.

The similarity matching and case retrieval algorithm follow the same process as that of the straightforward CBR approach.



1.36	1.36	1.36	1.36	1.36	1.36	1.36	1.36	1.36	1.36	1.36	0.00
0.00	3.88	3.88	3.88	3.88	3.88	3.88	3.88	3.88	3.88	0.00	0.00
0.00	0.00	7.15	7.15	7.15	7.15	7.15	7.15	7.15	0.00	0.00	0.00
0.00	0.00	0.00	20.00	20.00	20.00	20.00	20.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	33.33	33.33	33.33	0.00	0.00	0.00	0.00	0.00

Figure 3: Situated similarity matching is established by weighing each observational feature with respect to the relative NPC position. The portrayed situation displays that part of the grid observed by the NPC. Allowed actions of the NPC are denoted with arrows.

4.3 *k*-Nearest Neighbour Classification

For the *k*-nearest neighbour classification approach, similarity is calculated as a generalisation over stored cases. A straightforward *k*-nearest neighbour algorithm is used to classify a new case based on the majority of the *k*-nearest neighbour category. This process consists of two steps:

1. Given a new case, the *k*-nearest neighbour algorithm retrieves the *k* number of state-action pairs of which the states are syntactically most similar to the new case (i.e., with highest similarity as calculated by the straightforward CBR approach). If state-action pairs exist with identical state-description, but with different observed action, only the state-action pair with the highest fitness is retrieved.

2. Action classification is established by a majority vote among the actions of the retrieved state-action pairs.

5 Experimental Study of the Approach

We performed several experiments in order to asses whether our approaches successfully gather and utilise domain knowledge, as to endow an NPC with effective behaviour in the designed grids.

5.1 Experimental Setup

Our first experiment consists of a comparative study of the three approaches discussed in Section 4. Our goal is to detect whether the approaches are able to induce successful NPC behaviour without feeding prior knowledge to the NPCs whose experiences are stored in the case base. The experimental procedure is as follows. Given a particular grid, we sample data of NPCs attempting to reach the goal objects by choosing only random moves. We performed separate tests with experiences stored of 2, 10 and 50 randomly behaving NPCs. Subsequently, we let an NPC use each of the three approaches, and we determine the fitness achieved. As different random behaviour leads to different observations (and thus different domain knowledge gathered), for each grid and each approach, the results are averaged over 30 repetitions of the test. Regarding the k-nearest neighbour classification algorithm, the k-value was chosen by assessing five different k-values for each trial, and selecting the best performing value for reference. The offered k values were 3, 5, 10, 25 and 50.

Secondly, we tested how each of the three approaches would generalise to different environments (i.e., test whether knowledge learned in one environment can be transferred to another environment). We created a case base for grid3 by running 50 randomly-behaving NPCs. We then used the three approaches to control an NPC that used this case base on both the first two grids. We chose grid3 for gathering knowledge as it is the most difficult grid for an NPC to traverse. For the second experiment, we repeated the process 100 times for each of the grids and each of the approaches.

5.2 Results

In Table 1 an overview of the experimental results is given. They reveal that, as may be expected, all proposed knowledge utilisation approaches endow an NPC with more effective behaviour whenever more gathered domain knowledge is available to the utilisation process. The first experiment revealed that, when training and testing on the same grid, both the straightforward CBR and situated CBR approach significantly outperform the k-nearest neighbour classification approach. The performance of the straightforward CBR and situated CBR approach do not significantly differ in this setting.

Experimental results of the second experiment, however, reveal that when training and testing on different grids for the purpose of generalising over domain knowledge, the performances of the straightforward CBR and situated CBR approach differ. When transferring domain knowledge from grid3 to grid1 and grid2, the situated CBR approach outperforms the straightforward CBR approach with statistical reliabilities of 94% and 79%, respectively [2].

Note that the k-nearest neighbour classification approach is the only approach where the performance does not significantly degrade when transferring domain knowledge. This is an important issue with regard to generalisation. However, despite the stability of the k-nearest neighbour classification approach, it performs worst of all three approaches as for knowledge transfer from grid3 to grid1 and grid2 it is even outperformed by the straightforward CBR approach with statistical reliabilities of 51% and 92%, respectively [2].

6 Conclusions and Future Work

In this paper we discussed a means of gathering knowledge in the domain of commercial computer games. Three approaches were proposed to utilise gathered domain knowledge stored in a case base. These three approaches are: (1) straightforward case-based reasoning, (2) situated case-based reasoning, and (3) k-nearest neighbour classification. From the results of the experiments that test the proposed approaches, we draw the conclusion that the situated case-based reasoning approach performs best in the simulated commercial computer-game environment.

	NPC's observed	*Straightforward CBR*	*Situated CBR*	*kNN*	
Grid1	2	0.23 (0.27)	0.26 (0.31)	0.17 (0.20)	*k=3*
(30 repetitions)	10	0.55 (0.40)	0.60 (0.39)	0.16 (0.17)	*k=5*
	50	0.66 (0.33)	0.70 (0.34)	0.35 (0.37)	*k=3*
Grid2	2	0.25 (0.21)	0.24 (0.21)	0.18 (0.21)	*k=10*
(30 repetitions)	10	0.50 (0.28)	0.48 (0.29)	0.27 (0.24)	*k=3*
	50	0.55 (0.24)	0.55 (0.22)	0.35 (0.26)	*k=3*
Grid3	2	0.18 (0.17)	0.18 (0.17)	0.12 (0.13)	*k=3*
(30 repetitions)	10	0.38 (0.21)	0.41 (0.26)	0.20 (0.21)	*k=10*
	50	0.60 (0.19)	0.61 (0.24)	0.37 (0.26)	*k=25*
Knowledge transfer Grid 3 to grid 1 (100 repetitions)	50	0.33 (0.29)	0.40 (0.34)	0.33 (0.34)	*k=10*
Knowledge transfer Grid 3 to grid 2 (100 repetitions)	50	0.43 (0.25)	0.46 (0.27)	0.38 (0.26)	*k=25*

Table 1: Comparison of the performance of the three approaches. The performance is expressed in terms of the mean fitness and the standard deviation, denoted as 'mean (standard deviation)'

Our findings on the approaches' capabilities to transfer domain knowledge showed that the situated case-based reasoning approach performed best, but the k-nearest neighbour classification approach gave the most stable results. For future work, we aim to extend the situated case-based reasoning approach with a means of stable generalisation, inspired by the (distance-weighted [8]) k-nearest neighbour classification approach. We also intend to extend our research to actual commercial computer games, so to analyse the effect of our approach and to demonstrate its practical applicability.

Acknowledgements

This research was funded by a grant from the Netherlands Organization for Scientific Research (NWO grant No 612.066.406).

References

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning : Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1), March 1994.

[2] Paul R. Cohen. *Emperical Methods for Artificial Intelligence.* The MIT Press, 1995. ISBN 0-262-03225-2.

[3] Pedro Demasi and Adriano J. de O. Cruz. Online coevolution for action games. *International Journal of Intelligent Games and Simulation*, 2(3):80–88, 2002.

[4] Kurt Driessens and Saso Dzeroski. Integrating guidance into relational reinforcement learning. *Mach. Learn.*, 57(3):271–304, 2004.

[5] Thore Graepel, Ralf Herbrich, and Julian Gold. Learning to fight. In Quasim Mehdi, Norman Gough, and David Al-Dabass, editors, *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, pages 193–200. University of Wolverhampton, 2004.

[6] John E. Laird and Michael van Lent. Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2):15–25, 2001.

[7] John Manslow. *Learning and adaptation*, pages 557–566. AI Game Programming Wisdom. Charles River Media, Inc., 2002. Steven Rabin, editor.

[8] Tom M. Mitchell. *Machine Learning.* McGraw-Gill Series in Computer Science, 1997. ISBN 0-07-042807-7.

[9] Brian H. Ross. Some psychological results on case-based reasoning. In Morgan Kaufmann, editor, *Proceedings of the Case-Based Reasoning Workshop*, pages 144–147. DARPA, Pensacola Beach, 1989.

[10] Pieter Spronck. A model for reliable adaptive game intelligence. In David W. Aha, Hector Munoz-Avila, and Michael van Lent, editors, *Proceedings of the IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 95–100. Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, Washington, DC, 2005.

[11] Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Online adaptation of game opponent AI with dynamic scripting. *International Journal of Intelligent Games and Simulation*, 3(1):45–53, 2004.