Acquisition of Cases in Sequential Games using Conditional Entropy

Luc Lamontagne, Francis Rugamba & Guy Mineau

Université Laval, Québec, Canada
{luc.lamontagne, guy.mineau }@ift.ulaval.ca
francis.rugamba.1@ulaval.ca

Abstract. In this paper, we present an experiment we conducted on the acquisition of cases in sequential game environments. We describe an approach where demonstration traces are segmented into cases without supervision. Formation of the traces is performed using unsupervised clustering of game states and is guided by the conditional entropy of the sub-sequences. Application of this approach to a real-time reactive game indicate that the performance of the resulting case bases are comparable to a reactive CBR structure while reducing substantially the number of cases obtained from the acquisition phase.

1 Introduction

Developing game characters is a difficult task due to the increasing complexity of the game environments and scenarios. Game developers have to face challenges to create characters with interesting and intelligent behaviours. A promising direction to reduce programming complexity is to develop video game characters using machine learning techniques.

One paradigm gaining popularity for this purpose is learning by demonstration [2][5][7]. In such an approach, the system observes a teacher performing a task and cumulates information from this demonstration. The teacher can either be a human or some external components. The sequences resulting from the demonstration, usually obtained from an execution trace, indicate the various situations that were tackled by the teacher and prescribe the corresponding actions that were taken. Such demonstrations can be used subsequently for constructing a software artefact that can be reused to control some character behaviours.

In this paper, we concentrate on the problem of constructing cases from traces generated in a sequential environment. Our work can be interpreted in two different ways. The first interpretation would be that to build cases from traces where the action recommendations are of varying length. In this setting, each case would cover an arbitrary number of episodes in the traces. In the second interpretation, we aim to build state-transition diagrams (STD) approximated by a set of cases. Here the main issue would be to choose, based on some characteristics of the traces, a compact set of states. But, from a technical point of view, both interpretations are almost equivalent as the cases are constructed from unsupervised segmentation of the traces.

In our experimentations, we evaluate our approach for Pac-Man, a reactive pursuit game. Pac-Man constitutes an interesting test bed as it offers a real time dynamic environment and it involves sequential decision making.

The paper is organized as follows: the next section presents an overview of the proposed approach. In Section 3 we describe how the approach was applied to the game of Pac-Man. Section 4 explains how segmentation is performed using unsupervised clustering and conditional entropy (an information theoretic measure). Finally, experimental results are presented in Section 5 to illustrate how this scheme compares to a reactive approach for case acquisition. The paper ends with a conclusion and some recommendations for future work.

2 Building Cases from Demonstration Traces

As mentioned previously, learning by demonstration was adopted to acquire traces from game plays. The steps required to implement this approach in a dynamic game environment are the following:

(a) To acquire game traces by observing demonstrations from a teacher indicating how some non-player characters (NPC) should behave during a game;
(b) To construct, from the game traces provided by the teacher, an intermediate structure that can be reused to build a decision policy and to activate the NPCs.
(c) To modify the decision structure in order to optimize either behaviour performance or decision time.

As illustrated in Fig. 1, the traces acquired in step (a) consist of a sequence of episodes where each episode describes the states of the game board and the action taken by the teacher.

A conversion of the trace into an intermediate representation is required to build a decision policy. We make use of cases to make this representation. CBR offers various techniques to reuses episodes in new situations and to modify the structure of the case base to optimize the performance of the system.

From a CBR point of view, each pair <*game state*, *teacher action*> of an episode can be associated to a single case. In this acquisition strategy, a reactive case corresponds to a single episode.

Another strategy is that a case represents multiple consecutive episodes. A monolithic case could represent a full trace [6]. However, as our game traces are long and can contain a few thousands of episodes, we must provide some techniques to break down the traces into sub-sequences. We propose in Section 4 a technique to break down the sequences into cases containing a variable number of episodes.

As efficient time response is required, video game designers often make use of finite state machines (or state-transition diagrams - STD) [3] to compile the decision policy of non player characters (NPC). With slight modifications to a case representation, it is possible to approximate a state machine using a case base. This is the approach we adopt in this work as we structure cases to represent the elements of a state

machine while we exploit the sequence of states observed in the traces to form the case base.

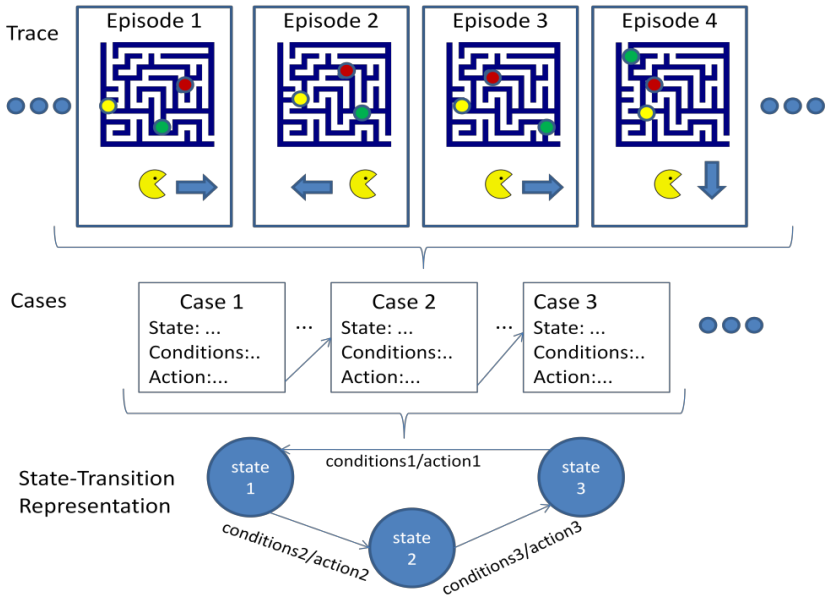

Fig. 1. A trace converted as a sequence of states and actions

A state machine is a directed graph consisting of states related to each other by transitions. We consider Mealy machines where states are associated to symbol(s) observable from the environment. And transitions have two different parts:

a) some conditions that should be satisfied to enable a change of state,

b) an output symbol to be emitted when activating the transition.

To approximate such diagrams with cases, we must map states and transitions to case features. In our experiments, each case is divided into 3 parts as follows:

- We select one or more attributes of the problem description to represent the states of the STD. Hence the set of states correspond to the combination of values for these attributes.
- The remaining <*attributes*, *value*> pairs of the case problem represent the conditions of the transition.
- The action recommended by the teacher (the solution part of the case) is the output symbol of the transition.

Given the current state of the game board, a NPC decision module would consider all the cases associated to this state and would choose the action(s) recommended by the case having conditions most similar to the current game configuration.

To build cases that are not limited a reactive formulation (i.e. having a single action as solution), we propose in Section 4 a scheme where we identify states of the game that are highly correlated. Correlation of the states is evaluated using conditional entropy, an information theoretic measure. Based on these correlations, we merge corresponding cases together. Case merging produces a more compact case base where each case designates a subset of episodes coming from one of the original traces. This scheme could also be considered as a segmentation of the traces into cases of variable length. Before to describe this case acquisition scheme in further detail, we explain in the next section how we applied the general acquisition approach to the game of Pac-Man.

3 Application to a Reactive Game – Pac-Man

To conduct our experiments, we made use of Pac-Man as a test bed. The task for this pursuit game is to move a yellow circled character in a maze to eat dots and fruits while avoiding some ghosts. The structure of the maze does not change during the game (Fig. 2). The actions of the Pac-Man correspond to the four possible moves in the maze: up, down, left and right. The motion of the ghosts is pseudorandom and the behavior rules change from one implementation to another. To conduct our experiments, we modified an implementation of this game developed by Benny Chow [1] to extract traces from game demonstrations.

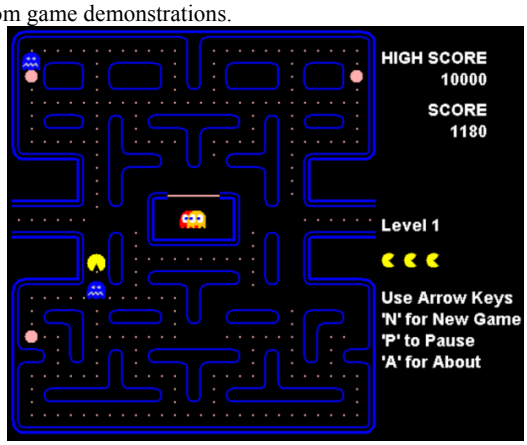

Fig. 2. Configuration of the Pac-Man board

We developed a computer player to make the demonstrations. Given the current location of the Pac-Man, this agent selects a target on the board, i.e. the closest object among the dots, the fruits, the power pellets or the edible ghosts. Then it uses a best-first search algorithm to determine the shortest path to the target object while avoiding the ghosts (if they are in attacking mode). Pac-Man applies the actions to follow this path as long as the target remains unchanged.

3.1 Representation of the state of the game board

To represent traces provided by these demonstrations, we selected some of the attributes found in the literature [4][9] to model the game board:

- *Location*: an identifier designating the current location (the cell) occupied by the Pac-Man.
- *Power*: the state of the power pellet;
- *Wall states*: four (4) attributes indicate if neighbouring cells are accessible (open or closed);
- *Ghosts distance*: some attributes indicating the Manhattan distance between the Pac-Man and the four ghosts. Special variables indicate the distance to the nearest ghost and if this ghost is edible or in attacking mode;
- *Dot distance*`: two variables indicate the distance between the nearest power pellet and fruit.

3.2 Using reactive cases as an approximation of a state machine

From game play traces, we initially form one case for each episode. Doing so, we adopt a reactive approach as each case recommends only one action to cope with a situation. All the episodes of the traces are stored in the case base.

As each case must represent a pair <*state*, *transition*> to approximate a state machine, the cases are structured as follows:

- *State*: the location of the Pac-Man character on the game board.
- *Time*: some index indicating the position of the episode in a trace.
- *Conditions*: some attribute values pairs describing the state of the game board as presented in the previous section
- *Action*: the move applied to Pac-Man.
- *Outcome state*: the location where Pac-Man ended up after applying the action.

Hence the case base represents the various transitions that were observed in the state space during the demonstrations. While this approach seems attractive, the size of the case base represents a limitation from a computational point of view. In our experiments, we generated some case bases containing tens of thousands of cases. This size can represent a serious limitation for applying CBR in real-time games. We investigate in section 4 how to merge cases together to reduce the case base.

3.3 Playing with reactive cases

To perform reasoning with cases structured as state transitions descriptions, we reuse the nearest neighbour from a partition of the case base corresponding to the current state of the game. To do so, the following steps are performed:

– Given the current state of the game board (in our case, the cell containing Pac-Man), fetch all the cases representing transitions originating from this state;

- Compare condition attributes of the cases to their corresponding values on the game board to determine the similarity of each candidate case.
- Apply the action recommended by the case having the most similar conditions.

In our experiments, the similarity of a case corresponds to the proportion of attributes having values corresponding to the game state. Uniform weights were assigned to the attributes.

4 Segmentation of Traces using Conditional Entropy

The scheme presented in the previous section relies on the usage of reactive cases to approximate state machines. This approach results in thousands of cases, each of them recommending a single action. Hence case retrieval must be performed at each frame of the game to determine the actions of the non player character.

We investigated the possibilities to reduce the number of cases in the case base. Analysis of the game traces reveals some interesting aspects that can be exploited. Game traces contain the complete history of states and actions performed during some games. One can try to determine some frequent patterns of states in the traces to determine sub-sequences of episodes that would form cases.

In our application, this would correspond to finding some regularity in the moves of the NPC. Analysis of the sequence of episodes can help to identify positions that are often observed in sequence. We also denote the fact that there is little game play variation in some parts of the game board. This might be explained by the structure of the game board containing long corridors and few intersections. If such-sub-sequences exist, we propose to identify them and to group corresponding episodes as individual cases. We describe in the next paragraph how we used hierarchical clustering and information theory to address this problem.

4.1 Finding correlations in consecutive episodes

To identify sequences of states that are correlated, we evaluate the level of uncertainty associated to specific trajectories. Given that some episodes with state x often lead to episodes with state y, then we can consider that the uncertainty associated to state x is low. In the context of Pac-Man, if one player frequently chose the same direction at one specific intersection, then decision behaviour at this intersection cell is highly predictable (which correspond to low uncertainty for this state). On the other hand, if the sub-sequences of the traces leading to a specific position y are much diversified, then uncertainty on how games can end up at state y is higher. To make these intuitions operational, we use conditional entropy to measure the level of uncertainty related to variations in sequences.

Entropy [8] is an information theoretic measure that estimates the amount of disorder, or uncertainty, related to the distribution of random variables. Given a variable X and a probability distribution indicating how likely the variable can take different values x_i, entropy $H(X)$ is given by

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

where $p(x_i)$ is the frequency of state value x_i in the demonstration traces.

Conditional entropy quantifies the entropy remaining on a variable X given that some random variable Y has been observed. Marginal conditional entropy is the same measure defined for some specific values, and is formulated as follows:

$$H(X = x|Y) = -\sum_{i=1}^{n} p(x|y_i) \log p(x|y_i)$$

By using marginal conditional entropy in our Pac-Man application, we intend to measure if moving from location x to location y during game plays is highly predictable (or unpredictable). Lower conditional entropy values would indicate that players often move from y to x, or equivalently that there is low uncertainty that has x follows y in the demonstration traces. If low entropy values are observed in the traces, we suppose that a case could be formed each time the two states are observed consecutively in the traces without losing too much information or game play performance.

To measure the dependency between two consecutive locations y and x, we measure the residual uncertainty RU which is the reduction in entropy when knowing that location y precedes location x:

$$Residual\ uncertainty(x, y) = RU(x, y) = \frac{H(x|y)}{H(x)}$$

And we merge cells for which the average value of $RU(x,y)$ and $RU(y,x)$ is the lowest.

4.2 Merging Cases using Conditional Entropy

To determine the states, we apply an agglomerative hierarchical clustering algorithm that successively group states together (i.e. merge cells of the game board as new states). Given a threshold value indicating acceptable levels of residual uncertainty, the algorithm determines a set of states as follows:

(a) Form a set containing states for each possible location on the board.
(b) Compute the residual information between each pair of state that can be observed consecutively in the traces.
(c) While some states can still be merged together:
    (i) Find the two states with the lowest residual information;
    (ii) Remove them from the set of states;
    (iii) Form an new state by grouping them together and add it to the set;
    (iv) Evaluate the residual information of the new state with respect to all the neighbouring states;
(d) Once completed, remove from the set all states with residual entropy higher than some threshold values.

Fig. 3 illustrates the first few iterations of the algorithm when applied to Pac-Man where neighboring cells having low entropy values have been merged together to form new states of the game.

Once a set of combined states have been constructed by this algorithm, cases corresponding to the states are merged together. A new combined case contains the following attributes:

- *State*: the locations merged together on the game board;
- *Time*: the time index when the first location in the combined state is first occupied;
- *Conditions*: the state of the game board when the combined state was first entered;
- *Action*: the list of moves applied to the PacMan while staying in the combined state.
- *Outcome state*: the location where PacMan ended up after applying the action.

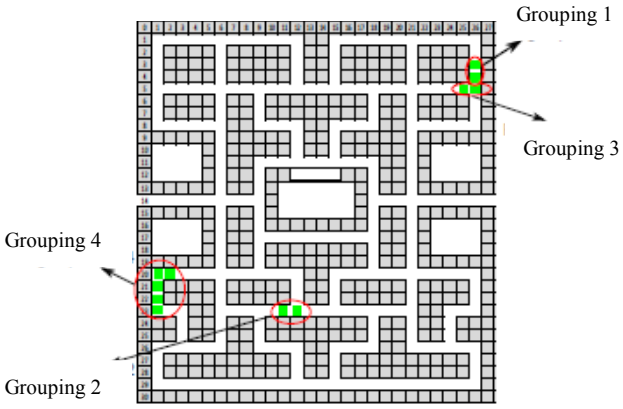

Fig. 3. - Grouping of states in the first few iterations of the algorithm.

To exploit these merged cases as a decision policy, retrieval is performed to select the best case to apply for a specific board configuration. Once a case is selected, all the actions recommended by the case are applied before any further retrieval is executed. In fact, whenever a sequence of actions is triggered, no observation of the game board is made before the complete sequence of actions is executed.

One would expect that execution of combined cases with short list of actions would provide performance similar to reactive cases. However, experiments are needed to determine how cases with longer action lists can influence performance.

5 Experimental results

To conduct our experiments, we built traces acquired during 33 games using the computer player described in Section 3 of this paper. The traces generated from these games contain a total 29 552 episodes. The board configuration in our experiments contains 300 cells that are accessible by Pac-Man. Hence the initial case base used to approximate a state machine contained almost 30 000 reactive cases defined over 300 possible states.

To evaluate the performance that can be obtained using this reactive case base, we played 200 games. An average score of 5684 points was reached over these games.

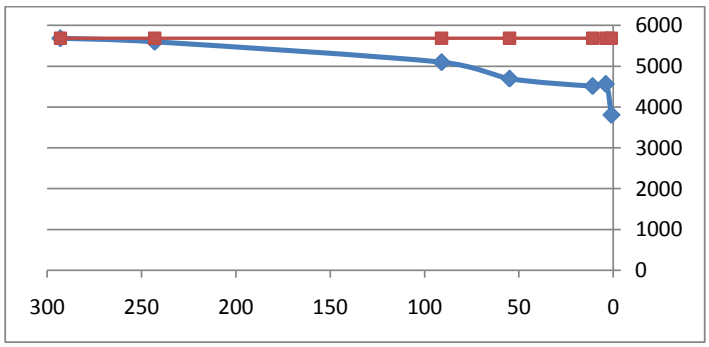

Fig. 4. Average score obtained for different number of states. The square line corresponds to the results of the computer player. And the diamond line corresponds to the case base player.

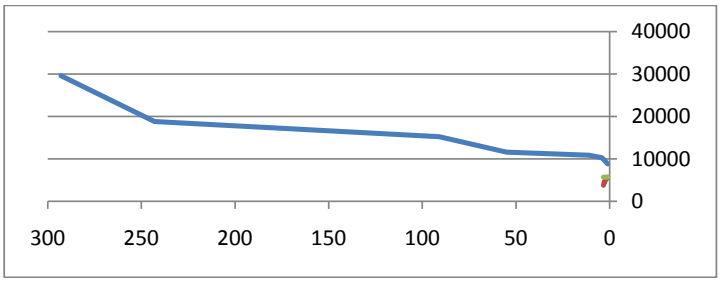

Fig. 5. Average score vs. number of cases resulting from segmentation.

We repeated the same experiments with different case bases generated by merging cases using the scheme described in Section 4. Results presented in Fig. 4 indicate that a reduction in the number of states (the horizontal axis) degrades the performance (average score). As more states are merged together, lists of actions in the cases get longer which results in less reactivity in the behaviour of the game character.

However, we consider that the loss of performance is minor. By compressing the state space to less than 100 states, which represent less than a third of the original states, we observe a reduction of performance of only 10%. Moreover, a reduction of more than 90% of the states results in a degradation of 17.4% in average game score. Hence applying important reductions of the number of states seem to have limited effects on the performance of the corresponding case base being generated.

Fig. 5 indicates the number of cases (vertical axis) corresponding to different set of states (horizontal axis). It is interesting to note that states that were grouped earlier during the clustering process involve more cases that states during later in the process. Hence the proposed case acquisition scheme seems to substantially reduce the size of the case base while limiting performance degradation.

6 Conclusion

In this work, we conducted a study to determine how cases can be built from sequential traces acquired during game demonstrations. We considered an approach where game episodes are converted into cases representing the states of the game, the conditions describing the game environment and actions that lead to subsequent episodes. We proposed an algorithm to group cases together to represent multiple episodes in the traces. And evaluate for this scheme for the game of Pac-Man indicates that dynamic segmentation of traces into cases seem to be a viable approach. As future work, we would like to explore algorithms to select attributes describing states. And we would like to explore how segmentation algorithms, not relying on unsupervised clustering techniques, could be applied to the acquisition of cases from traces.

7 References

1. Chow, B. Executable open source Pac-Man in java. http://www.bennychow.com (01/06/2010).
2. Floyd, M. W., Esfandiari, B., Lam, K.. A case-based reasoning approach to imitating RoboCup players. FLAIRS, pp. 251-256, 2008.
3. Fu, D., Houlette., H. The ultimate guide to FSMs in games. AI Game Programming Wisdom 2, pp. 283-302, 2004.
4. M. Gallagher and A. Ryan. Learning to play Pac-Man: an evolutionary, rule-based approach. In Evolutionary Computation 2003, IEEE, volume 4, pp. 2462-2469, 2003.
5. M. Mehta, S. Ontanon, T. Amundsen, and A. Ram. Authoring behaviors for games using learning from demonstration. In ICCBR Workshop on CBR for Computer Games, 2009.
6. S. Ontañón, Case Acquisition Strategies for Case-Based Reasoning in Real-Time Strategy Games, FLAIRS 2012.
7. Houcine Romdhane and Luc Lamontagne. Forgetting reinforced cases. In Advances in Case-Based Reasoning, LNCS 5239, Springer Berlin / Heidelberg, pp. 474-486. 2008.
8. R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. Methodology and Computing in Applied Probability, 1(2), pp. 127-190, 1999.
9. I. Szita and A. Lorincz. Learning to play using low-complexity rule-based policies: illustrations through MS. Pac-Man. JAIRS, 30(1), pp. 659-684, 2007.