Creating Non-Player Characters in a First-Person Shooter Game Using Learning by Observation

Vivian Andreeva, Jordan Beland, Sabrina Gaudreau, Michael W. Floyd, and Babak Esfandiari

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, Canada

Abstract. Video games can require complex behaviours from a player and often involve performing a variety of different tasks. A key part of the player's gaming experience can involve managing how these tasks are performed and when they are performed. One option for task management is to train autonomous agents to perform individual tasks and deploy them as necessary. In this paper we present a case-based learning by observation approach that allows a player to dynamically create embodied teammate agents during a game, demonstrate a task, and deploy the agents to perform that task in the future. We provide an initial proof of concept in a first-person shooter game by training agents to perform several item-gathering tasks. The agent was able to learn a simple item-gathering behaviour from a single demonstration but had difficulty collecting the items when environment exploration was necessary.

Keywords: learning by observation, autonomous agents, first-person shooter games

1 Introduction

A video game player may have a variety of related tasks that they are attempting to complete concurrently. For example, the player might be exploring the virtual environment while at the same time fighting enemies, managing resources, and collecting items. From a high-level perspective, the game involves managing how these tasks are performed (e.g., aggressively attacking an enemy or attacking from a distance) and when they are performed (e.g., attacking a nearby enemy or continuing to search for items). These individual tasks can be performed by the player or, alternatively, they can be delegated to autonomous agents that can be deployed by the player as necessary.

One option would be to have a variety of preprogrammed agents that the player is able to use. However, this requires prior knowledge about what tasks will be assigned to the agents and how the tasks should be performed. Even if the player does not want to perform a task themselves they may still have a preference as to how the task should be performed. This paper will examine how a player can dynamically create and train an agent by demonstrating how the task should be performed. The agent, which will serve as an embodied teammate of the player, observes the player performing the task, records those observations as cases, and uses case-based reasoning to imitate the player's behaviour. Over time, the player can create and train a team of agents, each of which is represented by a unique virtual character and is responsible for one particular task or subtask.

The remainder of this paper will detail our preliminary attempt at allowing a player to dynamically create and train teammate agents. Section 2 will examine related work in learning by observation and learning by demonstration. Section 3 will describe how an agent is created, trained and deployed. A preliminary proof of concept in a first-person shooter game is described in Section 4 followed by conclusions and areas of future work in Section 5.

2 Related Work

When an agent is created, it uses *learning by observation* to record and imitate the player's behavior. Case-based reasoning has been a popular approach to learning by observation with applications in real-time strategy games [1], poker [2], Tetris [3], simulated soccer [4], and first-person shooter games [5]. In addition to the domains, these various approaches differer in the complexity of case solutions (e.g., plans [1] or atomic actions [3]), the source of the observations (e.g., a single expert [4] or multiple experts [2]), or the properties of the expert being observed (e.g., purely reactive [3], non-deterministic [5], or state-based [6, 7].

Outside of case-based reasoning, other learning techniques have been used for learning by observation. These include using neural networks in a first-person shooter game [8], a Kalman smoother to learn aerial robotic manoeuvres [9], and mixed-initiative control in robotic soccer [10]. Lead-through learning [11] is similar to learning by observation in that robots are trained to perform fixed sequences of actions by having a human manually control the robot's movements.

The primary difference between our work and these other learning by observation systems is that they look to train an agent that will replace a human whereas we look to create an agent that will join the human as a teammate. A teammate agent will not need to learn all of the human's behaviour but can instead focus on one or more subtasks. This can potentially simplify the learning process by constraining the problem space that the agent will be responsible for. Training teammates using learning by observation has been examined previously [12] but the teammate agents were more complex agents, like a real human player, rather than the simpler assistive agents we use.

Our work also has similarity to learning interface agents that learn by observation. In these systems, the agent learns to perform assistive tasks like e-mail sorting [13], meeting scheduling [14], and note-taking [15]. Like our work, these agents are constrained to assist the user with fixed tasks but, unlike our work, the agents are only designed to learn a single, predefined task.

3 Teammate Agent

The learning by observation process used by a teammate agent has three important components: how observations are stored in cases, how cases are collected, and how the agent uses cases to perform the behaviour.

3.1 Case Structure

The learning by observation process is based on the idea that the teammate agent will behave similarly to the player (i.e., performing similar actions) in similar situations (i.e., receiving similar sensory inputs). As the player interacts in the environment, by performing actions in response to sensory inputs, the teammate agent can record these interactions (Figure 1). A case C_t, observed at time t, contains the sensory input S_t that the player received (the *problem*) and the action A_t that was performed (the *solution*):

$$C_t = \langle S_t, A_t \rangle$$

As the agent observes the player, it can automatically record cases (one for each input-action pair) and store them in its case base.

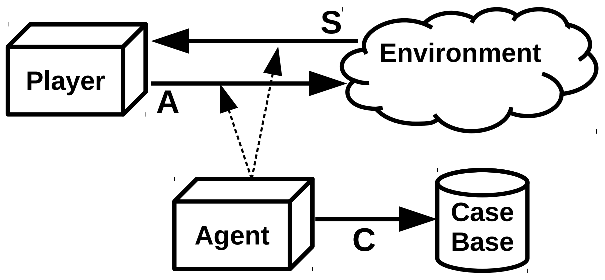

Fig. 1. Agent observing a player interacting with the environment

3.2 Case Acquisition

The observation process, where the teammate agent will observe the player and record cases, is controlled by the player. When the player is going to begin performing a specific task, a new teammate agent can be created with an empty case base (or an existing agent and its case base can be loaded). For example, the player may press a *record* button prior to retrieving an item in the environment so that a teammate agent will be created and observe this behaviour. Similarly, the player is also able to press a *pause* button to tell the agent to stop observing. This would occur if the player needed to perform another task (e.g., battling an enemy) that was unrelated to what the agent should be learning.

Two methods are used to guide the agent in adding cases to its case base: *store all* and *store unsolved*. In the store all approach, once the player indicates that the agent should observe (pressing the record button), all observed cases will be added to the case base. The agent will continue adding cases until a limit has been reached (e.g., it will only store a fixed number of cases) or the player indicates it should stop (pressing the pause button). Using this approach, if the agent observed N state-action pairs it will add N cases to its case base.

The second approach, store unsolved, takes into account that the agent may have constraints on the case base size or search time. A newly observed case C_i is only added to the case base CB if the case-based reasoning cycle does not return the associated action A_i when the case's sensory input S_i is used to query the case base ($A_{retrieved} = searchCBR(S_i, CB)$ and $A_{retrieved} \neq A_i$[1]).

3.3 Deployment

The teammate agent can be deployed in the environment once it has observed the player and added cases to its case base. The player decides when the agent should be deployed and initiates the deployment by pressing a *play* button. When the teammate is deployed, it will become an embodied agent in the environment. This means in will be visible to other agents, can receive its own sensory inputs, and perform actions.

As the teammate agent receives its own sensory inputs, it attempts to select an action that the player would have performed given the same sensory input. This is achieved by taking the current sensory input $S_{current}$ and searching the case base for an action $A_{current}$ ($A_{current} = searchCBR(S_{current}, CB)$). In practice, this often involves returning the action A_{max} of the case C_{max} with the most similar sensory input ($\forall C_j \in CB, sim(S_{current}, S_j) \leq sim(S_{current}, S_{max})$). The teammate agent can then perform that action (or possibly perform no action if no cases were sufficiently similar). By performing this search, the agent is constantly attempting to perform actions that it thinks the player would have performed based on observed evidence (the cases in the case base).

The agent will continue operating in the environment until the player terminates it using a *stop* button. This can occur if the player determines the agent is no longer necessary (e.g., the task it performs has been completed) or the agent is performing poorly and must be trained further. If retraining is necessary, the player can either empty the agent's case base (completely retraining) or add to the case base with further observation sessions (additional training).

In our system, agents only learn from observing the player demonstrating a task. During deployment, an agent will not retain new cases based on its experience since there is no guarantee that the actions it performed are the same as the actions the player would have performed. This ensures that an agent does not inadvertently add erroneous cases to its case base. However, if there are multiple agents trained to perform the same task, it would be possible for them to transfer cases between their case bases.

[1] Alternatively, a similarity metric can be used to compare actions to see if the search returns similar action.

4 Proof of Concept

An initial proof of concept was performed in a first-person shooter game called Scared. This section will describe the game, the initial learning by observation scenarios we examined, and our results.

4.1 Scared

Scared [16] is a first-person shooter game where a player moves around a 3D environment composed of rooms and corridors that are connected by doors (Figure 2). The game is simplified by only having a single type of enemy and a single type of weapon that can be used.

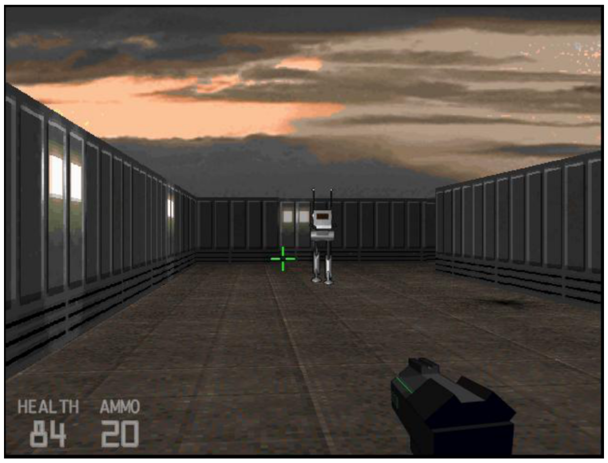

Fig. 2. Screenshot of the Scared first-person shooter game

The player has four primary actions that can be performed: moving the player (in the x and y directions), turning the player (either left or right), aiming the player's weapon (in the x, y and z directions), and firing the weapon. The sensory inputs the player receives are related to what objects are currently visible in its field of vision. These objects include walls, doors, enemies, ammunition packs (used to replenish the weapon's ammunition), health packs (used to replenish the player's health) and keys (used to open locked doors). If there are k objects currently visible to the player, the player's currently sensory input S_t will contain an entry for each visible object:

$$S_t = \{o_1, o_2, \ldots, o_k\}$$

Each visible object o will contain its object type *type*, distance from the player *dist*, and direction relative to the player *dir*:

$$o = \langle type, dist, dir \rangle$$

Since the player has partial observability due to a limited field of vision, the number of objects in the sensory input can vary over time as items move in and out of view or are removed from the game (e.g., an item is picked up or an enemy is killed). The similarity metric used to compare these type of sensory inputs (that have a variable number of visible objects) matches objects based on their object type so that they are only compared if they are the same type (e.g., ammunition packs are only compared with other ammunition packs) [4]. The sensory inputs are internally partitioned into lists containing objects of the same type ($list_{type_1} \cup list_{type_2} \cup \cdots \cup list_{type_N} = S_t, list_{type_1} \cap list_{type_2} \cap \cdots \cap list_{type_N} = \emptyset, \forall o \in list_{type_i} o.type = type_i$). If there are multiple objects of a given type (e.g., three visible enemies in one sensory input and two visible enemies in the second sensory input), they are matched based on a secondary feature. In our implementation, objects of the same type in a sensory input are ordered based on their distance relative to the player ($list_{type_i} = \{o_{i_1}, o_{i_2}, \ldots, o_{i_M}\}, o_{i_1}.dist \leq o_{i_2}.dist \leq \cdots \leq o_{i_M}.dist$) and matched with objects in the other sensory input that occur at the same position in their respective list (e.g., the closest enemy in the first sensory input $o^1_{enemy_1}$ would be matched to the closest enemy in the second sensory input $o^2_{enemy_1}$, the second closest in the first $o^1_{enemy_2}$ is matched to the second closest in the other $o^2_{enemy_2}$). This simple object matching approach is used to quickly match objects while still being computationally efficient enough to be used in real-time situations. Once objects are matched, they can have their positional similarity calculated (measuring how similar their position is relative to the player) and the global similarity computes the average of all matched object similarities (with weights applied depending on the relative importance of each type of object). If there were any unmatched objects, which would occur if there were more objects of a given type in one sensory input than the other, they are not included in the similarity calculation[2].

4.2 Scenarios

We will examine two simple scenarios where the player is training a teammate agent. The player is able to control the observation process (*record*, *pause*, *play*, *stop*) by pressing the associated keys on a keyboard. In the first scenario, which we will call the *collector* scenario, the agent observes the player collecting an ammunition pack that is currently within the player's field of vision. This involves the player performing actions that move it through the environment (moving forwards, backwards, left, right, turning left and turning right). For this scenario, the agent initially had an empty case base when the player started the observation session and collected 86 cases during observation. The second scenario, called the *explorer* scenario, the ammunition pack is not currently visible so the player must move throughout the environment until one is visible and

[2] Alternatively, a penalty value can be applied to the similarity when there are any unmatched objects.

then collect it. In this scenario, after starting with an empty case base 101 cases were collected during observation. For both of the scenarios, these case bases represent a single demonstration of the task and took approximately one minute to demonstrate.

After the case bases were collected, simulated annealing was used to determine the optimum weights for each type of object. This was performed because some objects, like ammunition packs, will be more important when attempting to replicate the observed behaviour. Each of the two agents determined their own optimum weighting and used those weights during deployment.

4.3 Results

To test the ability of the teammate agents to perform the learnt behaviours, each of the agents was deployed in the game alongside the player. Figure 3 shows a teammate agent visible to the player. The agents were placed into scenarios that were similar to those that they were trained in (either with a visible ammunition pack or an ammunition pack that needed to be searched for) and were able to act autonomously. The success of each agent was measured by whether it was successfully able to achieve its goal of collecting the ammunition pack. This process was repeated 100 times for each agent and the results are shown in Table 1.

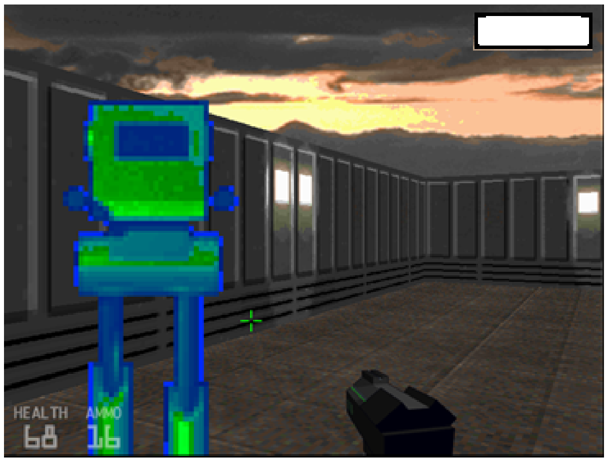

Fig. 3. Screenshot showing the teammate agent in the environment

The agent trained to perform the simple collection task was generally able to perform the task successfully. However, the agent trained on the explorer task was never able to successfully locate the ammunition pack. The explorer agent was able to successfully move throughout the environment but was not able to learn what area it had already explored and that it should move to other rooms.

Table 1. Results for the two learning by observation scenarios

	Successful Ammunition Pickups	**F1 value**
Collector	74%	1.00
Explorer	0%	0.70

This is because, although it had observed the player moving between rooms, it did not store any internal information related to how much of a room it had already searched.

The agents were also given test inputs from testing case bases to evaluate their performance. Since each case in the testing case bases has both the sensory input and action, the sensory input can be provided as input to the teammate agents and compared to the actions the agents attempt to perform. The results from the training cases were used to calculate the F1 score[3] ($\frac{2 \times precision \times recall}{precision + recall}$, with values ranging from the lowest value of 0.0 to the optimal value of 1.0). The explorer agent, while unable to successfully retrieve the ammunition packs, did perform reasonably well on the test cases. However, the errors that it did make (particularly related to not exploring different areas) resulted in its inability to achieve its goals. Similarly, we see that the collector agent achieved a perfect F1 score but still did not collect the ammunition packs in all trials. This is because the various trials were different enough from both the training and testing cases so the agent was unable to solve some of the novel input problems.

The case bases used by the agents were created by using the *store all* case acquisition technique. While the collector agent was adding cases to its case base, a second case base was also created using the *record unsolved* acquisition method. The second case base, which only added cases when the agent could not select the correct action with the current case base, had 36 cases. When the previously described evaluations were run using this case base, there was no noticeable difference in the number of ammunition packs collected or the F1 score compared to when the larger case base was used.

It should be noted that these results are from an agent that was trained with a single demonstration from the player. Had the agents been able to observe several demonstrations of the same task they could potentially have been able to handle a wider range of scenarios. Additionally, if the agents had been performing poorly during the game the player would have been able to provide additional information or correct erroneous behaviour.

[3] The F1 score was used since the actions in the case bases are highly imbalanced (some actions are far more rare than others). The F1 score penalizes values that are often selected incorrectly so simply selecting the most common action (rather than the correct one) will not result in a high F1 score.

5 Conclusions and Future Work

This paper has presented an approach for creating, training, and deploying teammate agents in a gaming environment. Each agent observes the player performing a simple task and then is responsible for performing that task in the future. This allows the player to manage tasks by controlling when they are performed and how they are performed. The agents use learning by observation to learn the tasks by storing the observations, which contain the actions the player performed in response to sensory inputs, as cases in a case base.

An initial proof of concept was presented in the first-person shooter game Scared[4]. We examined the ability of the agents to observe and learn both simple item-gathering and more complex exploratory item-gathering behaviour. The agent was able to successfully locate and collect ammunition packs but was less successful when it needed to search the environment for the items. This was a result of the agent repeatedly exploring the same location and not venturing out into new areas of the environment.

Our initial experiments were limited in scope and future work will look to perform a more complex evaluation of our system. The player only ever had a single teammate agent deployed in the environment so we have not explored using multiple agents simultaneously. We plan to compare the player's in-game performance when it performs all tasks itself to when some of the tasks are delegated to teammate agents. When some tasks are performed by the agents we would expect the player to have more focus for its tasks and be able to more efficiently achieve its goals.

We also plan to compare single-agent learning by observation to multi-agent learning by observation. When a single agent attempts to learn the player's entire behaviour, it faces a complex learning task because it needs to be able to solve all problems the player may face. Instead, if the player's behaviour can be broken down into simpler sub-behaviours, each agent will only be responsible for a simple learning task. Behaviours that were too complex for a single agent might be learnable by a team of agents. For example, in the Scared game it might be possible to train three different agents, with each agent learning a single simple task. One agent could be trained to explore rooms, one to move to different rooms, and one to collect ammunition packs. These agents could work together to perform the exploratory ammunition collection task that, as we have shown, was difficult for a single agent to learn. For example, the agent responsible for searching the room could deploy the agent that collects ammunition packs if one was found (or deploy the agent that moves to different rooms if the current room was completely searched).

References

1. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: Case-based planning and execution for real-time strategy games. In: 7th International Conference on Case-Based

[4] Our implementation of this is open source and publicly available for download: `https://code.google.com/p/scared-case-based-reasoning/`

Reasoning, Springer (2007) 164–178
2. Rubin, J., Watson, I.: On combining decisions from multiple expert imitators for performance. In: 22nd International Joint Conference on Artificial Intelligence, AAAI Press (2011) 344–349
3. Romdhane, H., Lamontagne, L.: Forgetting reinforced cases. In: 9th European Conference on Case-Based Reasoning, Springer (2008) 474–486
4. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating RoboCup players. In: 21st International Florida Artificial Intelligence Research Society Conference, AAAI Press (2008) 251–256
5. Gillespie, K., Karneeb, J., Lee-Urban, S., Muñoz-Avila, H.: Imitating inscrutable enemies: Learning from stochastic policy observation, retrieval and reuse. In: 18th International Conference on Case-Based Reasoning, Springer (2010) 126–140
6. Floyd, M.W., Esfandiari, B.: Learning state-based behaviour using temporally related cases. In: Proceedings of the 16th United Kingdom Workshop on Case-Based Reasoning. (2011) 34–45
7. Ontañón, S., Floyd, M.W.: A comparison of case acquisition strategies for learning from observations of state-based experts. In: 26th International Florida Artificial Intelligence Research Society Conference, AAAI Press (2013) 387–392
8. Thurau, C., Bauckhage, C.: Combining self organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In: 4th International Conference on Intelligent Games and Simulation, EUROSIS (2003) 119–126
9. Coates, A., Abbeel, P., Ng, A.Y.: Learning for control from multiple demonstrations. In: 25th International Conference on Machine Learning, ACM Press (2008) 144–151
10. Grollman, D.H., Jenkins, O.C.: Dogged learning for robots. In: 24th IEEE International Conference on Robotics and Automation, IEEE Press (2007) 2483–2488
11. Deisenroth, M.P., Krishnan, K.K.: On-line programming. In Nof, S.Y., ed.: Handbook of Industrial Robotics. John Wiley and Sons (1999) 337–352
12. Silva, M., McCroskey, S., Rubin, J., Youngblood, M., Ram, A.: Learning from demonstration to be a good team member in a role playing game. In: 26th International Florida Artificial Intelligence Research Society Conference, AAAI Press (2013) 393–398
13. Maes, P., Kozierok, R.: Learning interface agents. In: 11th National Conference on Artificial Intelligence, AAAI Press (1993) 459–465
14. Horvitz, E.: Principles of mixed-initiative user interfaces. In: 18th Conference on Human Factors in Computing Systems, ACM Press (1999) 159–166
15. Schlimmer, J.C., Hermens, L.A.: Software agents: Completing patterns and constructing user interfaces. Journal of Artificial Intelligence Research **1** (1993) 61–89
16. Brackeen, D.: Scared. `http://www.brackeen.com/scared/` (2012) [Online; accessed July 15, 2014].