Authoring Behaviors for Games using Learning from Demonstration

Manish Mehta, Santiago Ontañón, Tom Amundsen, and Ashwin Ram

CCL, Cognitive Computing Lab
Georgia Institute of Technology
Atlanta, GA 30332/0280
{mehtama1,santi,amundsen,ashwin}@cc.gatech.edu

Abstract. Behavior authoring for computer games involves writing behaviors in a programming language. This method is cumbersome and requires a lot of programming effort to author the behavior sets. Further this approach restricts the behavior set authoring to people who are experts in programming. This paper will describe our approach to design a system that will allow a user to demonstrate behaviors to the system, which the system will use to learn behavior sets for a game domain. With learning from demonstration, we aim at removing the requirement that the user has to be an expert in programming, and only require him to be an expert in the game. The approach has been integrated in a easy to use visual interface and instantiated for two domains, one a real time strategy game and another an interactive drama.

1 Introduction

State-of-the-art computer games are usually populated with many characters that require intelligent and believable behaviors. However, even though there have been enormous advances in computer graphics, animation and audio for games, most of the games contain very basic artificial intelligence (AI) techniques. In the majority of computer games traditional AI techniques fail to play at a human level because such games have vast search spaces in which the AI has to make decisions in real-time. Such enormous search spaces cause the game developers to spend a large effort in hand coding specific strategies that play at a reasonable level for each new game. Game designers are typically non-AI experts, and thus defining behaviors using a programming language is not an easy task for them. They might have a clear idea in mind of the behavior they want particular characters in the game to exhibit, but the barrier is encoding those ideas into actual code. Ideally, we need an approach that can allow game designers to easily author behavior sets for particular games.

Human learning is often accelerated by observing a task being performed or attempted by someone else. In fact, infants spent a lot of their time repeating the observed behaviors [11]. These capabilities of the human brain are also evident in computer games where players go through a process of training and imitating experienced players. These results have inspired researchers in artificial intelligence to study learning from imitation techniques. However except for a few attempts, there have been very few attempts at their integration in computer games. By observing an expert's actions, new behaviors can quickly be learnt that are likely to be useful; because they are already being used by the expert successfully. In this paper, we present an approach that utilizes this ability to extract behavioral knowledge for computer games from expert demonstrations. Using the architecture presented in this paper the game authors demonstrate the behavior to be learnt (maybe by controlling some game characters manually) instead of having to code the behavior using a programming language and the system learns from that demonstration. In order to achieve that goal, we use case-based reasoning (CBR) techniques, and in particular case-based planning [12]. The idea is to represent each behavior as a plan, and use case-based planning to reuse the behaviors learnt from demonstrations in order to play the game. Our architecture has been instantiated in two domains, one a real time strategy game and the other an interactive drama.

The rest of the paper is organized as follows. We present our architecture in Section 2. We discuss the concrete instantiation of the architecture in real time strategy game WARGUS (an open source clone of the popular game WARCRAFT II) in Section 3 and interactive drama domain in Section 4. The paper closes with related work and conclusions.

2 Learning from Demonstration Architecture

Our main goal is to create a system that allows a game designer to easily author AI behaviors using learning from demonstration, in constrast to having him encoding behaviors in some programming language. In order to achieve that goal, we have designed a learning from demonstration architecture (shown in Figure 1) that consists of four steps:

- *Demonstration*: The human plays the game, demonstrating the particular behavior he wants the system to learn. This process results in a trace, i.e. a log file that contains each action that the expert executed, together with their respective game state and time stamps.
- *Annotation*: The human annotates the trace specifying which goals (selected from a predefined set of goals) he was attempting with each action. In our experiments, annotation is performed using an easy to use GUI. Section 2.2 explains why annotation is desirable.
- *Behavior Learning*: The annotated trace is handed to a behavior learning module, which can automatically extract procedural behaviors from the annotated trace, and store them in a behavior base.
- *Behavior Execution*: Once the behavior base has been populated, the learnt behaviors can be executed in the game using a behavior execution engine. We propose to use a case-based planning [12] behavior execution engine, where each one of the behaviors is represented as a case.

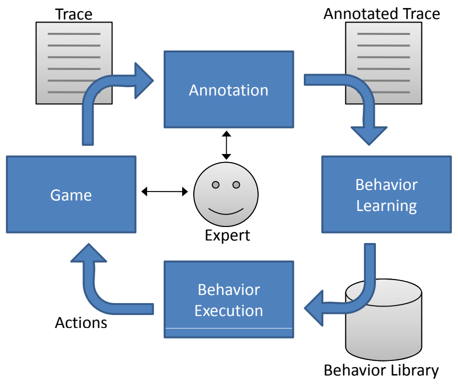


Fig. 1. Our general Learning from Demonstration Architecture, involving 4 steps: demonstration, annotation, behavior learning and finally behavior execution.

2.1 Demonstration

The game domain needs to provide a way to demonstrate behaviors. Depending on the game at hand, this can be done using the normal interface that a player would use to play the game, or through a special interface if required. The main idea is to let the expert use the basic set of primitives that are available within the game world. For example, in our RTS game domain, WARGUS the standard game playing interface can be used. However, in our interactive drama domain, Murder Mystery, a specific interface to control virtual characters inside the game world was developed. This was the case because the default game interface in that game did not generate traces nor allowed us to control the characters at the level of detail we wanted.

The author uses the demonstration interface to play the game. Apart from this interface, a basic mechanism to record the trace is required. In our architecture, a trace is composed of a list of *entries*, where each entry is a triple: time stamp, game state, primitive actions. Representing at a particular time in a particular state, the expert executed some primitive actions.

2.2 Trace Annotation

The next step is to annotate the trace. In this process, the expert specifies which goals was he pursuing for each particular action. This process requires a collection of goals being defined for each game for which the architecture is instantiated. Once a set of goals is defined, the expert can simply associate each of the actions in the game with one or more of the set of available goals.

The intuition behind annotation is that if a set of actions are labeled as achieving the same goal, then the system will put those actions in a single behavior that achieves the specified goal. Thus, annotations can be used in order to group together the actions that were demonstrated into individual behaviors.

We can now see that the set of goals that have to be defined for each game is a set of goals that allows the human to decompose the task of playing the game in subtasks for which behaviors can be learnt. Annotation could be partially automated (as we propose in [9]). However, an automatic process of annotation leaves the expert with less control over the learnt behaviors. An automatic annotation process is desirable if the goal is to build a system that can learn how to play the game autonomously. However, if the goal is to facilitate the task of a human author, annotation provides a simple way in which the author (or expert) can control which behaviors will be learnt. For example, in a given game, the expert might, by accident, achieve some particular goal during the game in a way that he did not want to demonstrate (just as a side effect of some actions). In an automated annotation process, that will result in the system learning an undesired behavior, which for the purposes of the system learning to play the game is desirable, but for the purposes of helping the author defining the behaviors he wants to define is undesirable. For that reason, we believe that annotation is desirable when the goal is to assist a human in behavior authoring.

2.3 Behavior Learning

In order to learn behaviors, the annotated trace is analyzed to determine the temporal relations among the individual goals appearing in the trace. In our framework, we are only interested in knowing if two goals are pursued in sequence, in parallel, or if one is a subgoal of the other. We assume that if the temporal relation between a particular goal g and another goal g' is that g happens *during* g', then g is a subgoal of g'.

From this temporal analysis of goals, procedural descriptions of the behavior of the expert can be extracted. Notice that an expert might assign more than one goal to each action. Thus, the system can learn hierarchical behaviors. Also, once the system has learned behaviors for each one of the goals used by the expert, a global behavior that uses these behaviors as "subroutines" can also be inferred (See [10] for more details).

Each one of the learnt behaviors are stored in a behavior library for future use. Notice that no generalization of the behaviors is attempted at learning time. Since we are proposing to use a case-based reasoning approach (where each behavior is considered to be a case), all generalization is left for problem solving time, i.e. for when the system is playing a game.

2.4 Behavior Execution

Once behaviors have been learned, they are ready to be executed in the game. Thus, a behavior execution engine is required. We propose to use a hierarchical case-based planner to perform this task. Each behavior will be seen as a partial plan to achieve a particular goal, and the hierarchical planner will combine them together to form full plans to achieve the goals of the character or characters the system is controlling.

Cycle	*Player*	*Action*	*Annotation*
8	1	Build(2,“pig-farm”,26,20)	-
137	0	Build(5,“farm”,4,22)	SetupResourceInfrastructure(0,5,2) WinWargus(0)
638	1	Train(4,“peon”)	-
638	1	Build(2,“troll-lumber-mill”,22,20)	-
798	0	Train(3,“peasant”)	SetupResourceInfrastructure(0,5,2) WinWargus(0)
878	1	Train(4,“peon”)	-
878	1	Resource(10,5)	-
897	0	Resource(5,0)	SetupResourceInfrastructure(0,5,2) WinWargus(0)
...	...	...	...

Table 1. Snippet of a real trace generated after playing WARGUS. The game states for each entry in the trace are omitted.

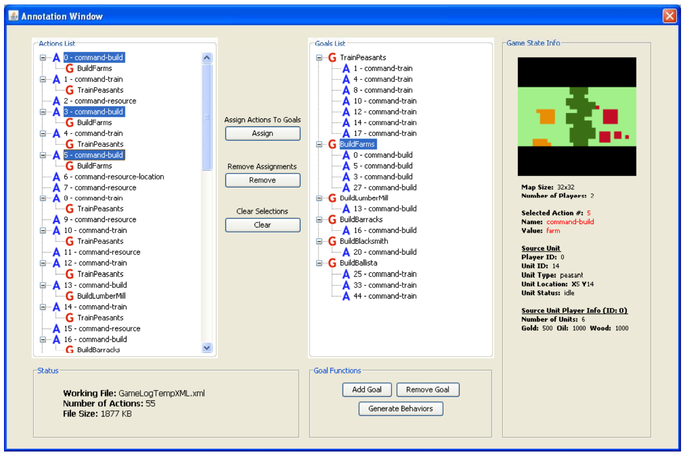
3 First Game Domain: WARGUS

Real-time strategy (RTS) games have several characteristics that make behavior authoring difficult: huge decision and state spaces [1, 2], non determinism, incomplete information, complex durative actions, and real time. WARGUS is a real-time strategy game where each player's goal is to remain alive after destroying the rest of the players. Each player has a series of troops and buildings and gathers resources (gold, wood and oil) in order to produce more troops and buildings. Buildings are required to produce more advanced troops, and troops are required to attack the enemy. In addition, players can also build defensive buildings such as walls and towers. Therefore, WARGUS involves complex reasoning to determine where, when and which buildings and troops to build.

In order to demonstrate a behavior set for WARGUS an expert simply plays a game. As a result of that game, we obtain a game trace. Table 1 shows a fragment of a real trace from playing a game of WARGUS. In the WARGUS domain, each trace entry is limited to a single action. For instance, the first action in the game was executed at cycle 8, where player 1 made his unit number 2 build a “pig-farm” at the (26,20) coordinates.

The next step is to annotate the trace. For the annotation process, the expert uses a simple annotation tool that allows him to specify which goals was he pursuing for each particular action. The annotation tool simply presents the execution trace to the expert (with small screenshots of the state of the game at every trace entry, to help the human remember what he was doing) and he can associate goals to actions. All the goal types defined for the WARGUS domain are available to the expert, and he can fill in the parameters of each goal when annotating. Figure 2 shows a screenshot of such tool.

In our approach, a *goal* $g = name(p_1, ..., p_n)$ consists of a goal name and a set of parameters. For instance, in WARGUS, some of the goal types we defined are: $WinWargus(player)$, representing that the action had the intention of making



Actions List

- A 0 - command-build
  - G BuildFarms
- A 1 - command-train
  - G TrainPeasants
- A 2 - command-resource
- A 3 - command-build
  - G BuildFarms
- A 4 - command-train
  - G TrainPeasants
- A 5 - command-build
  - G BuildFarms
- A 6 - command-resource-location
- A 7 - command-resource
- A 8 - command-train
  - G TrainPeasants
- A 9 - command-resource
- A 10 - command-train
  - G TrainPeasants
- A 11 - command-resource
- A 12 - command-train
  - G TrainPeasants
- A 13 - command-build
  - G BuildLumberMill
- A 14 - command-train
  - G TrainPeasants
- A 15 - command-resource
- A 16 - command-build
  - G BuildBarracks

Goals List

- G TrainPeasants
  - A 1 - command-train
  - A 4 - command-train
  - A 8 - command-train
  - A 10 - command-train
  - A 12 - command-train
  - A 14 - command-train
  - A 17 - command-train
- G BuildFarms
  - A 0 - command-build
  - A 5 - command-build
  - A 3 - command-build
  - A 27 - command-build
- G BuildLumberMill
  - A 13 - command-build
- G BuildBarracks
  - A 16 - command-build
- G BuildBlacksmith
  - A 20 - command-build
- G BuildBallista
  - A 25 - command-train
  - A 33 - command-train
  - A 44 - command-train

Game State Info

Map Size: 32x32
Number of Players: 2

Selected Action #: 5
Name: command-build
Value: farm

Source Unit
Player ID: 0
Unit ID: 14
Unit Type: peasant
Unit Location: X5 Y14
Unit Status: idle

Source Unit Player Info (ID: 0)
Number of Units: 6
Gold: 500 **Oil:** 1000 **Wood:** 1000

Status

Working File: GameLogTempXML.xml
Number of Actions: 55
File Size: 1877 KB

Fig. 2. A screenshot of our WARGUS trace annotation tool.

the player *player* win the game; $KillUnit(unit)$, representing that the action had the intention of killing the unit *unit*; or $SetupResourceInfrastructure(player, peasants, farms)$, indicating that the expert wanted to create a good resource infrastructure for player *player*, that at least included *peasants* number of peasants and *farms* number of farms.

The fourth column of Table 1 shows the annotations that the expert specified for his actions. Since the snippet shown corresponds to the beginning of the game, the expert specified that he was trying to create a resource infrastructure and, of course, he was trying to win the game. The annotated trace is next processed by the *behavior learning* module, which encodes the strategy of the expert in this particular trace in a series of behaviors.

Notice that in our system we don't attempt any kind of generalization of the expert actions. If a particular expert action in the trace is $Build(5, "farm", 4, 22)$, that is exactly the action stored in a snippet. Thus, using the learnt snippets to play a new scenario in WARGUS, it is very likely that the particular values of the parameters in the action are not the most appropriate for the new scenario (for instance, it might be the case that in the new map the coordinates 4,22 correspond to a water location, and thus a farm cannot be built there). In our WARGUS implementation, the behavior execution engine is responsible to adapt those parameters at run time.

Our execution engine in WARGUS is a case-based planner, that uses a set of adaptation rules in order to adapt the parameters of each of the actions in each behavior before executing it. Thus, in our implementation in the WARGUS domain, the game state in which the human demonstrated each action is stored together with the behavior. For details on how adaptation at run time is performed, see [10].

In order to evaluate our techniques in WARGUS, we developed an IDE from where users could launch WARGUS to start a demonstration, annotate demonstrations, manipulate behaviors, and test them on the game [13]. Our results show that users were able to successfully demonstrate behaviors using our system, and that they felt demonstrating behaviors was an easier way to generate scripts, than coding them by hand.

4 Second Game Study: Murder Mystery

In recent years, there has been a growing interest in creating story based interactive systems where the player experiences a story from a first person perspective, interacts with autonomous, believable characters. Interactive drama presents one of the most challenging applications of autonomous characters, requiring characters to simultaneously engage in moment-by-moment personality-rich physical behavior, exhibit conversational competencies, and participate in a dynamically developing story arc. Hand authoring of behavior for believable characters allows designers to craft expressive behavior for characters, but nevertheless leads to excessive authorial burden [6]. Tools are needed to support story authors, who are typically not artificial intelligence experts, to allow them to author behaviors in an easy way.

The interactive drama we are developing is named Murder Mystery (MM).The story set up consists of six characters and is set up in a British mansion at the beginning of the 20th century. The player controls one of the character and is free to interact with the rest of the characters using natural language and also move freely around the house and manipulate some objects. In particular, the drama starts when two of the characters decide to celebrate an engagement party, and invite two friends to a dinner in their newly acquired mansion. The remaining two characters are the butler of the house and the father of the bride. Most of the characters have strong feelings (love or hate) for some of the other characters, and as the story unfolds the player will discover hidden relations between them. The player will take the role of one out of three possible characters and will be able to act freely in the mansion.

In order to demonstrate behaviors, the user observes a character from a third person perspective and is able to control it using a GUI. The GUI consists of a series of buttons and text fields that allow the user to perform the following actions: speak, move forward, move backward, move left, move right, rotate, and play an animation. Such an interface records a similar trace as for our WARGUS domain (an example is shown in Table 2. The context associated with each logged action describes the current game state and consists of information about the map and characters. Each object and player in the map is logged with as much information as possible (since it will help the CBR system to adapt actions at run-time).

In order to carry out the annotation, some of the goals that have been used are:

- $Greet(character)$: representing that the action had the intention of greeting another.

Cycle	*Player*	*Action*	*Annotation*
8	Mary	Walk(”230,400,1920”, ”230,400,1920”, Mary)	-
137	Mary	Speak(Tracy, ”Hi Tracy”)	Greet(Tracy)
378	Mary	Wave ()	Introduce(Tracy)
500	Mary	Speak(Tracy, ”I am Manuel Sharma”)	-
678	Mary	Smile ()	-
800	Mary	Speak(Tracy, ”I am working as a technician”)	-
938	Mary	Speak(Tracy,”Could you pass me a drink?’)	AskforObject(Tracy, drink)
...	...	...	...

Table 2. Snippet of a real trace generated after playing Murder Mystery .

- $AskforObject(character, object)$: representing that the action had the intention of asking for a particular object *object* from a character *character*.
- $Introduce(character)$: the action had the intention of introducing to a particular character
- $Insult(character)$: the action had the intention of insulting a particular character
- $Hurt(character)$: the action had the intention of hurting a particular character.

In the same way as for WARGUS this trace would then be given to the behavior learning module, that will learn behaviors from it. Figure 3 shows an example of a learnt behavior in Murder Mystery. Although an extensive evaluation of our system in the Murder Mystery domain is still part of our future work, initial evaluations suggest that it is easier to author behaviors using our demonstration interface than coding them by hand.

In an analogous way as for our WARGUS domain, in the Murder Mystery, the game state associated with each action is stored, so that the behavior execution engine (a case-based planner) can adapt those actions.

```
Introduce(tracy)
{
  Wave();
  Speak(Tracy, "I am Manuel Sharma");
  Smile();
  Speak(Tracy, "I am working as a technician");
}
```

Fig. 3. Snippet of a behavior learnt after behavior demonstration in Murder Mystery .

5 Related Work

Henry Lieberman describes a system called Tinker, that is able to learn from examples that a programmer demonstrates. Using this framework, a programmer can demonstrate sets of examples, starting with simple examples, and work up to more complicated ones. Using these examples, Tinker learns how to operate on its own. A more recent example is provided by Nakanishi et al. [7], who designed a system that learns biped locomotion by observing humans walking. Nakanishi et al. describe an approach of using dynamical movement primitives as a central pattern generator, which are then used to learn the trajectories for the legs in robot locomotion. Nicolescu [8] describes a modular architecture which allows a robot to learn by generalizing information received from multiple types of demonstrations, and allows the robot to practice under the demonstrator's supervision. This system, albeit in a robotic domain is quite similar to ours, and provides a general way to learn primitive behaviors through demonstration in order to accomplish a given task.

Floyd et. al. present an approach to train a RoboCup soccer-playing agent by observing the behaviour of existing players and determining the spatial configuration of the objects the existing players pay attention to [3]. Kaiser and Dillman [5] presented a general approach to learning from demonstration using sensor-based robots. They describe how skills can be acquired from humans, "learned" in such a way that they can be used to achieve tasks, and refined so that the agent's performance will constantly improve. The system uses action primitives that are very concrete and easy to predict, such as determining what angle to move a robotic arm. In our system, action primitives are parameterized like talking to another character in the game, which can potentially have results that are hard to predict. Finally, Floyd and Estefandiari [4] compare several techniques for learning form demonstration (CBR, decision trees, support vector machines and naive bayes), showing very strong results favoring case-based learners.

6 Conclusions and Future Work

Learning from demonstration is a powerful mechanism to quickly learn behaviors. In this paper, we discuss how the principle of imitation learning can facilitate the programming of computer game characters. Moreover, we demonstrated the approach by reporting two implemented systems based on the same learning from demonstration architecture.

One of the key ideas introduced in this paper is that by the use of annotations in the demonstrations, the author can have control of the behaviors being learnt during the learning from demonstration process. Behavior authoring is ultimately a programming task, and as such is non-trivial when the set of behaviors that need to be authored are complex. However, we have seen that by using case-based planning techniques, concrete behaviors demonstrated in concrete game situations can be reused by the system in a range of other game situations, thus providing an easy way to author general behaviors.

Part of our future work involve trying to reduce the annotation task to a minimum, but that the author still has control over the behavior authoring process. One of the ideas is to implement a mixed initiative approach where the system will automatically annotate a trace, and the author will have the option (it desired) of changing the annotations. We are also working on implementing our approach in more domains to evaluate its strengths and weaknesses. In our initial evaluations we have seen that our approach is good for high level behavior demonstration, where as it is still not very good at low level reactive control.

References

1. David Aha, Matthew Molineaux, and Marc Ponsen. Learning to win: Case-based plan selection in a real-time strategy game. In *ICCBR'2005*, number 3620 in LNCS, pages 5–20. Springer-Verlag, 2005.
2. Michael Buro. Real-time strategy games: A new AI research challenge. In *IJCAI'2003*, pages 1534–1535. Morgan Kaufmann, 2003.
3. Michael W. Floyd, Babak Esfandiari, and Kevin Lam. A case-based reasoning approach to imitating robocup players. In *FLAIRS Conference*, pages 251–256, 2008.
4. Michael W. Floyd and Babak Estefandiari. Comparison of classifiers for use in a learning by demonstration system for a situated agent. In *Workshop on Case-Based Reasoning for Computer Games in ICCBR 2009*, page to appear, 2009.
5. M. Kaiser and R. Dillmann. Building elementary robot skills from human demonstration. In *In International Symposium on Intelligent Robotics Systems*, pages 2700–2705, 1996.
6. B. Magerko, J. Laird, M. Assanie, A. Kerfoot, and D. Stokes. AI characters and directors for interactive computer games. In *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Confercence*, 2004.
7. Jun Nakanish, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion with dynamical movement primitives, 2003.
8. Monica Nicolette Nicolescu. *A framework for learning from demonstration, generalization and practice in human-robot domains.* PhD thesis, Los Angeles, CA, USA, 2003. Adviser-Maja J. Mataric.
9. Santiago Ontañón, Kane Bonnette, Prafulla Mahindrakar, Marco A. Gómez-Martín, Katie Long, Jainarayan Radhakrishnan, Rushabh Shah, and Ashwin Ram. Learning from human demonstrations for real-time case-based planning. In *The IJCAI-09 Workshop on Learning Structural Knowledge From Observations*, 2009.
10. Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games. In *Proceedings of ICCBR-2007*, pages 164–178, 2007.
11. Rajesh P. N. Rao, Aaron P. Shon, and Andrew N. Meltzoff. A bayesian model of imitation in infants and robots. In *In Imitation and Social Learning in Robots, Humans, and Animals.* Cambridge University Press, 2004.
12. L. Spalazzi. A survey on case-based planning. *Artificial Intelligence Review*, 16(1):3–36, 2001.
13. Suhas Virmani, Yatin Kanetkar, Manish Mehta, Santiago Ontañón, and Ashwin Ram. An intelligent ide for behavior authoring in real-time strategy games. In *AIIDE*, 2008.