Evaluating Sketch-based Retrieval Speed-up for Behaviour Design in Soccerbots [*†]

Gonzalo Flórez-Puga and **Belén Díaz-Agudo** and **Pedro González-Calero**
Complutense University of Madrid. Madrid. Spain
{gflorez, belend, pedro}@fdi.ucm.es

Abstract

Sketch-based retrieval is a technique that supports the design of behaviour for game characters by reusing previously designed behaviours. Most techniques for specifying behaviour for game characters use some kind of graph-based formalism to represent such behaviour. Through graph-matching techniques, sketch-based retrieval allows to use any intermediate graph generated along the design process, a *sketch* of the final behaviour, as a query to retrieve similar behaviours from a library of complete behaviours. In this paper we describe the design and results from an experiment designed to measure to what extent having a library of reusable behaviours accessed through sketch-based retrieval can speed-up the behaviour design process in the Soccerbots game.

Introduction

We present a new approach to the authoring of behaviours for non-player characters (NPCs) in videogames based on retrieval and reuse from a collection of reusable behaviours. The motivation behind our approach is that typically in a large game we can find simple behaviours that are replicated within different complex behaviours. For instance, in a soccer game, `Defend` could be a complex behaviour that is composed of two simpler behaviours like `Go to the ball` and `Clear`; meanwhile `Attack` could be made up of `Go to the ball`, `Dribbling` and `Shoot`. However, the actual process of AAA[1] games development, where a group of game designers and programmers collaborate over a long period of time to iteratively design a large number of complex behaviours (for instance, Halo 2 had an average of 60 different behaviours arranged in 4 layers (Isla 2005)), does not currently rely on behaviour reuse. Without supporting tools and technology, reuse is not an option, and game designers tend to develop new behaviours from scratch, resulting in variations of similar behaviours coexisting in the same game, ignoring the benefits of reuse in terms of quality and scalability.

To help designers in the task of building and reusing behaviours we have developed *eCo*[2], a visual editor capable of storing, indexing, retrieving and reusing previously designed behaviours. Although in this paper we exemplify the approach with behaviours represented as hierarchical finite state machines (HFSMs) (Millington 2006), the editor can deal with other formalisms typically employed for designing behaviour in videogames, such as finite state machines (FSMs) (Bourg and Seemann 2004), and behaviour trees (BTs) (Flórez-Puga et al. 2011).

One of the most notable features of our editor is its capability for sketch-based retrieval. In the image retrieval domain, sketch-based retrieval consists in finding a complex image using an approximate representation of it (a sketch) as a query (Eitz et al. 2012). We can translate that idea to the behaviour domain, where a sketch is a partial representation of a behaviour (for instance, a FSM that is missing some edges or where the behaviour of a node has not been specified). In sketch-based retrieval of behaviours we search in a repository for behaviours, represented as graphs, that are similar to the one the user is drawing, making suggestions about how to complete it. There are several CBR systems that use approximate graph retrieval in domains as disparate as workflow retrieval (Bergmann and Gil 2011), textual CBR (Cunningham et al. 2004), image analysis (Eshera and Fu 1984), or architectural design (Weber et al. 2010). The novelty of our work resides in using the different intermediate states of the behaviour construction process as queries to retrieve a relevant case.

The question we address in this paper is to what extent sketch-based retrieval can speed-up the process of designing a new behaviour when the target behaviour is already in the library of reusable behaviours. In essence we measure at what point in the design process our algorithm would retrieve the completed target behaviour and how much effort is saved in terms of editing steps.

The rest of the paper runs as follows. Next section describes the main ideas of sketch-based retrieval and its implementation in the *eCo* behaviour editor. Next we describe the experiment setup before going into the section describing the results. Finally, we draw some conclusions.

[*]Supported by Spanish Ministry of Economy and Competitiveness under grant TIN2009-13692-C03-03.

[†]Funded by Complutense University of Madrid.

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]High budget, high quality videogames, expected to sell well.

[2]eCo: `http://gaia.fdi.ucm.es/research/eco`

eCo Behaviour Editor

eCo is a visual editor that helps game designers in the task of developing behaviours for NPCs in games. In particular, the version presented in this paper allows creating HSFMs that implement behaviours for Soccerbots robots, but the editor can be configured to be used with other games. For a more detailed description of *eCo* we refer to (Flórez-Puga et al. 2013).

Soccerbots[3] is a well-known simulation environment that simulates the dynamics and dimensions of a regulation RoboCup[4] small size robot league game. Two teams of five robots compete on a soccer field by pushing and kicking a ball into the opponent's goal. To execute the matches we rely on SBTournament[5] (Jiménez-Díaz et al. 2011), an enhanced environment to run Soccerbots matches. SBTournament offers different interfaces that allow to configure and run automatically multiple matches between two sets of teams. Besides, it generates a very useful complete log with statistics regarding the matches played.

Aside from assistance for configuring and launching large sets of matches, SBTournament provides users with a set of sensors and actuators, which are an enhanced superset of those provided by SoccerBots. Actuators are the most simple actions that a robot can execute, while sensors are the pieces of information that a robot can gather from the game world. For example, actuators in SBTournament allow users to kick the ball or set the desired heading and speed for a robot. Likewise, sensors provide information about the ball position or the position of the opponent's goal. The editor uses sensors and actuators to build the HFSMs that our robots will execute. On one hand, sensors are used to build the conditions for the edges of the HFSMs. On the other hand, actuators are used to build the *basic behaviours*, i.e. the basic building blocks for the robot's behaviour. Basic behaviours are the simplest actions that can be executed in a node of a robot's HFSM. These basic behaviours generally consist of a sequence of calls to different actuators.

eCo allows users to "draw" HFSMs to specify the behaviour of SBTournament's robots and teams. As the user draws, the partially completed HFSM is used as a sketch to retrieve previously created behaviours. The users can reuse pieces of the retrieved behaviours to complete the one being edited. Once a HFSM is finished it can be exported and executed in SBTournament. Finished behaviours are added to a library of created behaviours in order to be reused later.

Figure 1 shows the editor's graphic interface that allows users to design individual players and teams. The area in the middle is a canvas where users can draw the HFSMs that represent the robots' behaviours. Under the drawing canvas there is a text editor where users can write code to create and/or modify the basic behaviours.

The most important feature of the editor is its capability to retrieve behaviours stored in a library using two kinds of searches: *attribute-based* retrieval and *sketch-based* retrieval. For attribute-based retrieval we make the behaviours in the library play several matches versus a set of preselected *trainer* behaviours. From each match we gather some statistics that describe the gameplay of the behaviour (for instance, the number of goals scored, the covered distance, the average distance to each goal, etc.). To issue a query, the user assigns values to a subset of the statistics. The behaviours with the most similar statistics are retrieved from the library. Sketch-based retrieval allows designers to instead retrieve behaviours using as a query a partial graphical representation of the desired behaviours as we describe next.

Sketch-based Retrieval

In sketch-based retrieval designers can use a sketch of the desired behaviour as a query. A sketch is a partial or unfinished representation of the behaviour. The sketch is typically a behaviour that is being drawn by the designer but it is still not finished (that is, an intermediate step in the process of creating a behaviour). This allows the editor to automatically suggest different ways to finish the behaviour.

This approach requires an appropriate similarity function. For complete behaviours like the ones in the library, we can make them play and gather statistics about their gameplay to see if they are similar because they behave similarly. But in the case of a sketch, that is not possible, because the behaviour is not finished yet. Instead, we have to rely on another similarity metric that allows us to compare behaviours and predict which of them behave similarly. In particular we use the graph edit distance to compare the underlying graphs of the sketch and the cases in the library. The problem with the, so to speak, "standard" graph edit distance is that its cost is exponential on the number of nodes of the graph (Bunke and Messmer 1994). For this reason we have used the heuristic proposed in (Riesen and Bunke 2009), adapting it to HFSMs. As we have shown in past studies (Flórez-Puga et al. 2013) using this heuristic we obtain a result set that is almost indistinguishable from the original similarity function, but at a rather reduced cost in time.

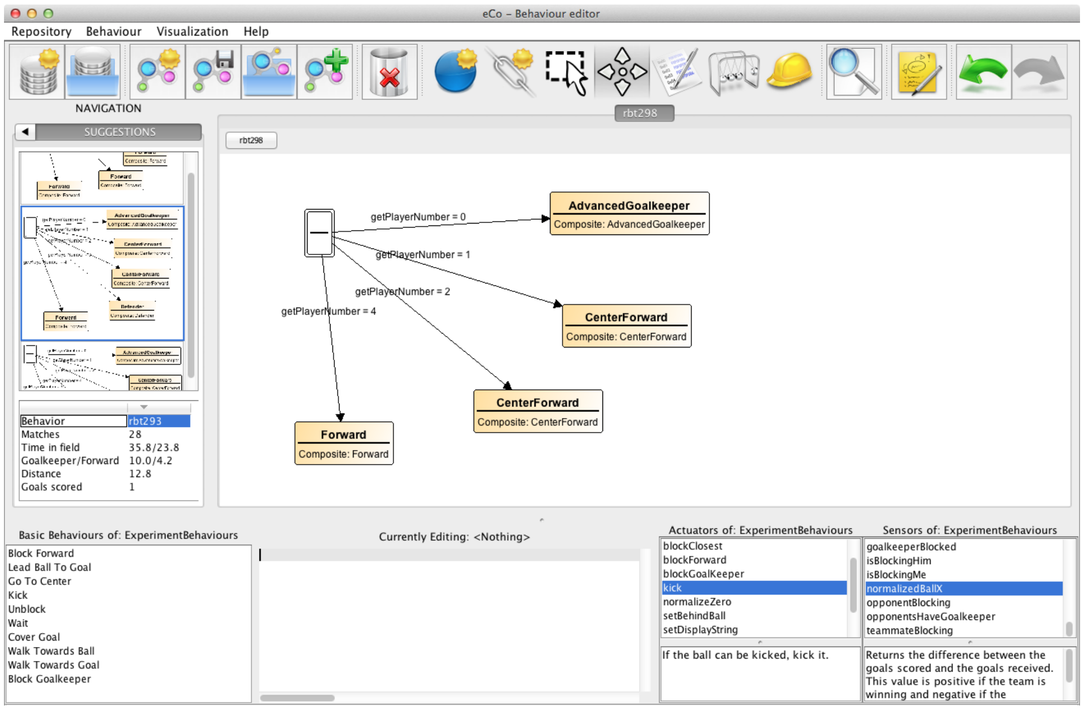
The suggestion feature in the editor works as follows: while the designer is drawing the behaviour, the editor uses the current (probably unfinished) behaviour as a query. If the designer takes a new step, like adding a node or changing the label on an edge, a new query is issued with the changed sketch. The top results of the query are shown in the suggestions panel, which is at the left side of the drawing canvas in Figure 1. The designer can use any of the results instead of the current sketch, or can combine them with the sketch being edited. When the user selects a behaviour suggested from the library, the editor shows some statistics about it in the table below. The statistics are gathered by making the teams play versus a predefined set of *trainer teams*.

The adaptation process is not automatized, but the system offers some assistance for manual adaptation. Information regarding the gameplay of the teams suggested can be employed by the users to adjust the team being built. For instance, if the user wants to develop a team that has a defensive gameplay she could compare her team with the teams suggested. She could then find a more defensive team (with

[3]SoccerBots: `http://www.cs.cmu.edu/~trb/TeamBots/Domains/SoccerBots`

[4]Robocup: `http://www.robocup.org/`

[5]SBTournament: `http://gaia.fdi.ucm.es/research/sbtournament`



Behavior	rbt293
Matches	28
Time in field	35.8/23.8
Goalkeeper/Forward	10.0/4.2
Distance	12.8
Goals scored	1

Basic Behaviours of: ExperimentBehaviours

Block Forward
Lead Ball To Goal
Go To Center
Kick
Unblock
Wait
Cover Goal
Walk Towards Ball
Walk Towards Goal
Block Goalkeeper

Currently Editing: <Nothing>

Actuators of: ExperimentBehaviours

blockClosest
blockForward
blockGoalKeeper
kick
normalizeZero
setBehindBall
setDisplayString

If the ball can be kicked, kick it.

Sensors of: ExperimentBehaviours

goalkeeperBlocked
isBlockingHim
isBlockingMe
normalizedBallX
opponentBlocking
opponentsHaveGoalkeeper
teammateBlocking

Returns the difference between the goals scored and the goals received. This value is positive if the team is winning and negative if the

Figure 1: Capture of eCo

few goals against or matches lost) but still similar to hers, and use it as a model to modify its configuration.

Experiment setup

As we have seen, sketch-based retrieval can help game AI designers to create behaviours for NPCs by providing candidate behaviours to be reused. This way, development time of new behaviours can be reduced. In this section we describe an experiment we have conducted to measure the savings in development time or, more precisely, in development steps.

The idea behind the experiment is to compare the number of steps taken by a user to create the same behaviour with and without the sketch-based retrieval feature. The experiment was conducted with 43 students from the Knowledge Based Systems course of the Complutense University, divided in 16 groups, during two sessions of two hours. In the first session we taught the users how to build the behaviours using the editor: we asked them to follow a tutorial that taught them how to build an example goalkeeper behaviour and then a team using different simple behaviours.

Before the second session they had one week to design several roles (e.g. goalkeepers, forwards, defenders, etc.) for a Soccerbots team using “pen and paper”. For the second session we asked them to implement those roles in the editor and build a team combining them. For this second session they weren’t allowed to use the retrieval features of the editor.

Once they had finished, we collected the behaviours they built. In total we collected 95 behaviours with an amount of nodes that ranged from 2 in the simpler behaviours (e.g. “Go to my goal” or “Kick ball”) to more than 20 for the team behaviours.

Together with the behaviours, we collected an execution trace generated by the editor, that contained all the editing steps that the users had followed. We consider an editing step any operation that introduces a change in the behaviour being edited: adding or deleting a node or edge, editing the label associated to a node or an edge or changing the initial node of a behaviour. We don’t consider editing steps, for instance, the creation of new basic behaviours or adding a behaviour to the library. Unsurprisingly, the quantity of steps is related to the number of nodes of the behaviour. In the behaviours we collected we found that the number of steps ranges from around 10 for the smaller behaviours (with 2 or 3 nodes) to more than 300 for the teams. Using this trace we were able to rebuild the original behaviours.

For each behaviour implemented by the users, B_i, we used its trace to obtain a set of intermediate steps, which are incomplete versions of the behaviour B_i. We called this set the intermediate behaviours $I_i = \{I_{i,0}, \ldots, I_{i,si}\}$, where si is the total amount of steps taken to obtain B_i. Hence, the set I_i ranges from the empty behaviour $I_{i,0}$ (a behaviour without any nodes or edges) to the final behaviour that was implemented $I_{i,si} = B_i$. Each $I_{i,j}$ is the intermediate behaviour obtained after applying step j.

To run the experiment we also needed a case base. Our case base is composed of all the final behaviours from the users, plus a set of 205 behaviours that were created by randomly composing different roles we already had from past experiments (Flórez-Puga et al. 2013), which makes a total of 300 behaviours in our case base. The size of the added behaviours ranges from 14 to 35 nodes.

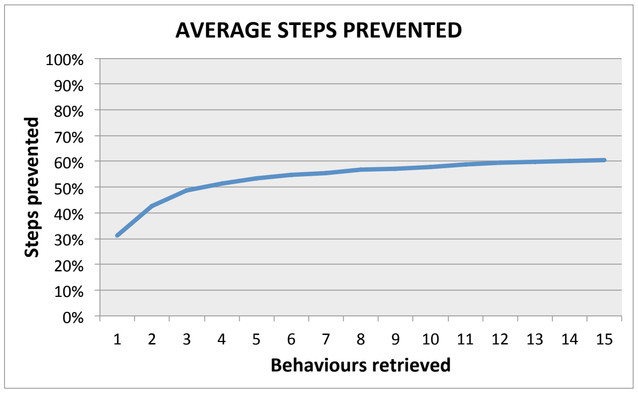


Figure 2: Average steps saved when using sketch-based retrieval

To determine the number of steps saved by the sketch-based retrieval for each B_i, we have used as a query each of the intermediate behaviours $I_{i,j}$, with $0 \leq j \leq si$, retrieving from the case base the k-nearest neighbours. In this way we are mimicking the behaviour of the editor when using the sketch-based retrieval feature. As we will see in the following section, the number of behaviours retrieved, k, has a great influence on the final results.

Then, we checked if the retrieved list contains the final behaviour B_i. If B_i belongs to the list of behaviours retrieved by query $I_{i,j}$ it means that using sketch-based retrieval the user can obtain the desired behaviour at step j, saving the remaining steps until si.

Results

In the Figure 2 we show the average number of steps saved for each value of k up to 15 behaviours retrieved. We can see that, when we increase the value of k, the number of steps saved is also increased. Although we have registered better values of saved steps for values of k higher than 15, it is not practical for the users to have a big list of retrieved behaviours and, in any case, they are most likely to analyse only the first few of them.

We observe that when we only show the most similar behaviour to the user (that is, when $k = 1$), the user can save one third of the total amount of steps. If we show more behaviours to the user, the number of saved steps rises to around 50% for $k = 3$. From there on the improvement is more gradual, reaching the 60% when $k = 11$. This observation indicates that, on average, users can save a great number of editing steps (up to 50%) when showing them only a few results from the query.

The standard deviation σ for each value of k remains almost constant around 20% for all the values shown. This indicates that for most of the cases, the number of saved steps are in a $\pm 20\%$ range from the average.

To narrow this range and get a better idea of the overall results we also have studied the frequencies of the results for different values of k, as shown in the Figure 3. Each pie chart represents the results of retrieval for a different value of k. Each section in the chart represents the proportion of retrieved behaviours for which we save at most the percentage of steps indicated. Table 1 shows the specific values of frequency. Each column in the table represents a value for k, while each row is a range of saved steps. The value in each cell shows the number of behaviours for which we save a number of steps in the corresponding range.

We note that for $k = 1$ there is a saving of more than 50% of the steps needed to create the behaviour in 16 out of the 98 behaviours studied. This value grows to 34 when $k = 2$ and to 44 (almost half of the total) when $k = 3$. This upward trend is steady for bigger values of k, but with a gentler increase. We also can see that the section labelled with 0% is present only when $k = 1$. This means that, when the number of retrieved behaviours is 2 or more, there is no case in the case base for which we don't save any steps.

Another factor to take into account is from what step the results retrieved are reliable. If a query is issued after too few steps, the sketch is less likely to summarize the structure of the desired behaviour and, hence, the retrieved behaviours won't be what the user expects. For that reason, before issuing any query it is advisable that the user takes some editing steps to reach a more detailed sketch. We have observed that the number of steps needed before obtaining the desired behaviour depends on the final size of the behaviour and on the number of elements retrieved in each query.

Table 2 shows the number of editing steps the user needed to take to retrieve the desired behaviour using sketch-based retrieval. We divided the set of behaviours in three groups according to their size: *small* behaviours with a total of 2 or 3 nodes (41 of them), *medium*, with 4 to 7 nodes (also 41) and *large* behaviours, with 8 and 9 nodes (of which we have 5). The remaining 8 behaviours are too scattered to be grouped. The second column shows the average number of editing steps that the users needed to create the behaviours in that group. This gives an upper bound of the number of steps needed in the worst case (that is, without using sketch-based retrieval). The remaining columns show the average number of steps needed to obtain the desired behaviour using sketch-based retrieval for different values of k.

We can see again that when we increase the number of behaviours shown to the user (k) the number of steps needed

	Retrieved behaviours (k)					
Spared steps	1	2	3	5	10	15
0 %	7	0	0	0	0	0
(0, 10] %	8	3	1	0	0	0
(10, 20] %	21	11	6	3	2	2
(20, 30] %	15	18	17	14	10	8
(30, 40] %	12	17	14	11	9	9
(40, 50] %	16	12	13	15	14	13
(50, 60] %	9	13	11	12	14	11
(60, 70] %	4	11	18	20	21	23
(70, 80] %	0	5	7	9	8	8
(80, 90] %	3	5	8	9	12	12
(90, 100] %	0	0	0	0	5	9

Table 1: Frequencies for different ranges of saved steps

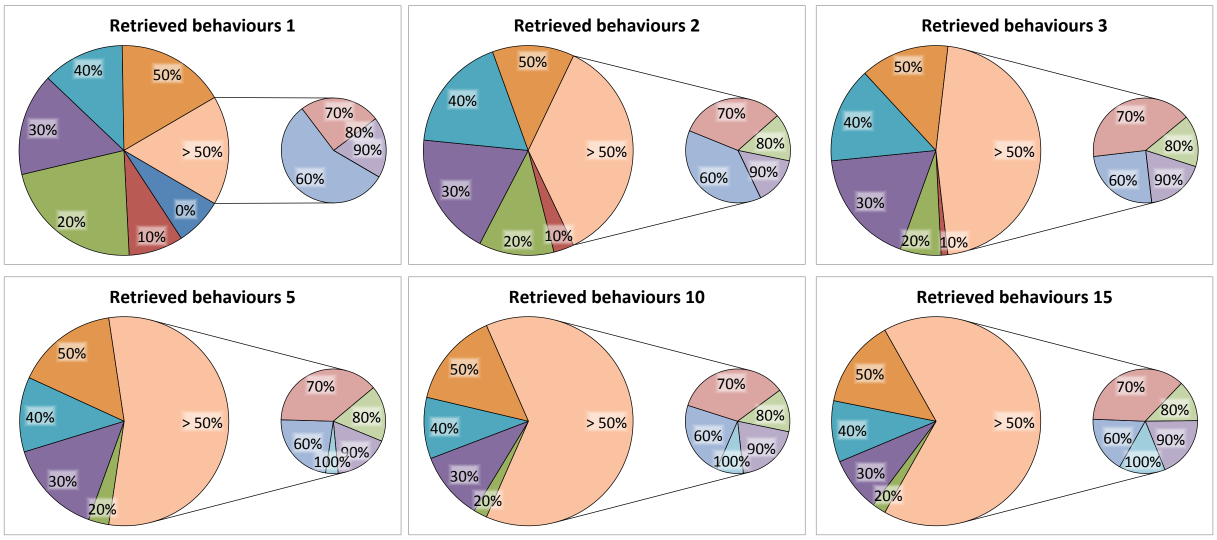


Figure 3: Steps saved for different values of k

Size	**Total steps**	**Steps needed for k**					
		1	2	3	5	10	15
Small	18	10.9	8.5	7.2	6.1	5.1	4.2
Medium	56	40.4	33.7	31.6	29.4	28.0	27.4
Large	80	58.4	54.2	50.2	38.4	33.2	32.8

Table 2: Number of editing steps needed on average to obtain the final behaviour in a query

to obtain the behaviour decreases, fast for the first values of k but in a smoother manner for values over 5. We can also see that, for bigger behaviours the user needs to take more steps to obtain the expected result.

In Figure 4 we show the evolution of the position of the desired behaviour in the results list for three different behaviours of different characteristics. Although they are three particular behaviours, we have chosen them in such a way that they are representative of each of the classes described before. The first case corresponds to a *small* behaviour, of only 2 nodes, that was completed by the user in 27 steps. The second one is *medium* sized, has 5 nodes and took the user 40 steps. The last one is *large*, has 8 nodes and took 56 steps. The horizontal axis of the graphs represents the number of steps taken by the user and the vertical axis the position of the desired behaviour in the results for that step. We have placed vertical dotted lines dividing the number of steps at 25%, 50% and 75%. The desirable result is a graph that goes down fast (meaning that the desired behaviour is found after a few steps) and then stays stable at a low position (this way, although the user has missed the behaviour in the first positions of the results list, he can retrieve it again in a later query). That is the case of the first example. We can see that, although at the very first steps it is retrieved in a high position, the position goes down to 3 at step 11 and is retrieved the first after step 12, staying there for the remaining steps. This means that using sketch-based retrieval, the appropriate behaviour is retrieved using the 40% of the steps that were needed to create it in the first place.

The second example behaves similarly. In this case we see that the result needs more steps to stabilize (16 steps to reach position 3 and 24 to reach the first position), but if we attend to the percentage of steps, we are also around 40%. In the results shown in the third graph we can see that it takes still more steps to find the behaviour and also to stabilize. The percentage of steps needed in this case has also grown up to 55%.

Analysing the graphs of these and other similar behaviours we can conclude that the number of steps the user needs to take before issuing a query grows along with the size of the behaviour but decreases when we increment the number of behaviours retrieved in each query (k). In general terms, the user needs to take around 40% of the total number of steps to be sure that the system retrieves the adequate behaviour. This percentage is greater for the biggest behaviours in our collection.

Regarding execution times, as we mentioned earlier, we used a heuristic similarity function to avoid the exponential cost inherent to structural similarity functions for graphs. Using this heuristic, each query was resolved in an average time of 0.117 ms. Therefore it is possible for eCo to automatically issue a new query after each editing step without imposing any delays for the user interaction, at least for the size of the cases bases that we have tried.

Conclusions and Future Work

In this paper we have presented the evaluation of a novel approach to the creation of intelligent behaviours that is based on reuse. We have called this approach *sketch-based retrieval*. In sketch-based retrieval designers use a sketch of the desired behaviour to retrieve from a library previously created behaviours that are similar to it. The similar behaviours are shown to the user who can then select a com-

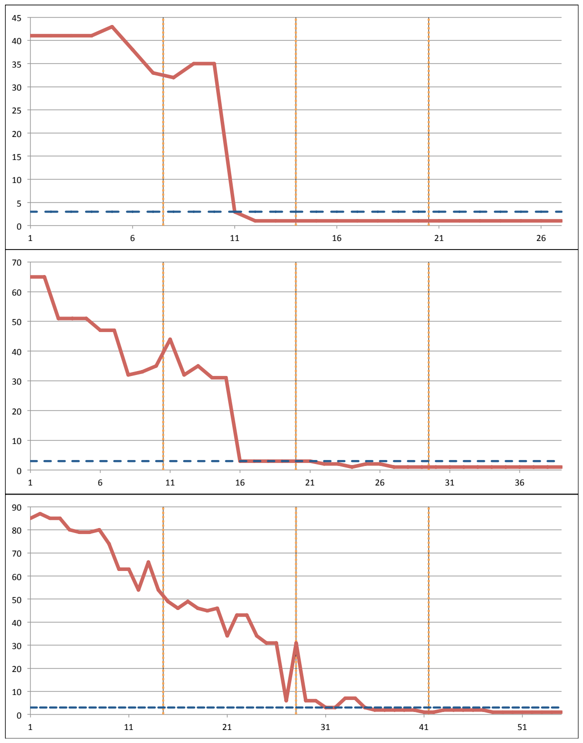


Figure 4: Evolution of the position of the target behaviour in the results list for three example queries

plete behaviour or a fragment to complete the one used as a query. It is interesting to note that the querying process doesn't require user intervention.

Using the retrieved behaviours to finish the one being developed designers can save time in the edition of behaviours while, at the same time, reduce the possible errors, because they are using behaviours or pieces of behaviour that have been previously tested. We have demonstrated experimentally that using sketch-based retrieval actually reduces the number of editing steps the designer has to take to obtain the desired behaviour, hence reducing the development time. For this experiment we assumed that the target behaviour is already in the library.

The amount of steps saved is dependant on the number of behaviours retrieved (k): when we increase the number of behaviours retrieved the number of steps saved also grows.

But retrieving too many behaviours is not useful for the user, because he would have to search in a long list for the behaviours he is interested in. We propose to use a value of $k = 3$. In our experiment we have shown that using this value we save at least 50% in 47 behaviours out of the total of 98 cases evaluated. We have also found that for this value of k we obtain good results for small behaviours (2 or 3 nodes) after the user has taken around 7 steps. For bigger behaviours (between 4 and 7 nodes) we needed around 30 steps to get the result. For behaviours of 8 or 9 nodes, our approach needed around 50 steps to return the relevant result within the first 3. In the examples of Figure 4 we have drawn a dashed horizontal line in $k = 3$. We can see that for the small behaviour, we obtain a good result after step 11, in the second example, the medium sized, after step 16 and in the last one after step 31.

In view of the promising results obtained, we plan, as future work, to drive an evaluation experiment with game designers. The idea of the experiment is to let game designers use eCo to create several behaviours using also the sketch-based retrieval tools and measure how many of the retrieved behaviours are used, in which step and how many adaptation steps are done by the designers after the retrieval. To complete the experiment we also plan to conduct a satisfaction survey of the users.

References

Bergmann, R., and Gil, Y. 2011. Retrieval of semantic workfows with knowledge intensive similarity measures. In *ICCBR 2011*, volume 6880 of *LNCS*, 17—31. Springer.

Bourg, D. M., and Seemann, G. 2004. *AI for Game Developers*. O'Reilly Media, Inc.

Bunke, H., and Messmer, B. T. 1994. Similarity measures for structured representations. In *EWCBR '93*, 106–118. London, UK: Springer-Verlag.

Cunningham, C. M.; Weber, R.; Proctor, J. M.; Fowler, C.; and Murphy, M. 2004. Investigating graphs in textual case-based reasoning. In *ECCBR 2004*, 573–586. Springer.

Eitz, M.; Richter, R.; Boubekeur, T.; Hildebrand, K.; and Alexa, M. 2012. Sketch-based shape retrieval. *ACM Trans. Graph.* 31(4):31:1–31:10.

Eshera, M., and Fu, K.-S. 1984. A graph distance measure for image analysis. *Systems, Man and Cybernetics, IEEE Transactions on* SMC-14(3):398 –408.

Flórez-Puga, G.; Llansó, D.; Gómez-Martín, M. A.; Gómez-Martín, P. P.; Díaz-Agudo, B.; and González-Calero, P. A. 2011. Empowering designers with libraries of self-validated query-enabled behaviour trees. In *Artificial Intelligence for Computer Games*. Springer. 55–82.

Flórez-Puga, G.; González-Calero, P. A.; Jiménez-Díaz, G.; and Díaz-Agudo, B. 2013. Supporting sketch-based retrieval from a library of reusable behaviours. *Expert Systems with Applications* 40(2):531–542.

Isla, D. 2005. Handling Complexity in the Halo 2 AI. In *Game Developers Conference*.

Jiménez-Díaz, G.; Menéndez, H. D.; Camacho, D.; and González-Calero, P. A. 2011. Predicting performance in team games. the automatic coach. In *ICAART 2011*, 401 – 406. Rome, Italy: SciTePress.

Millington, I. 2006. *Artificial intelligence for games*. The Morgan Kaufmann series in interactive 3D technology. Morgan Kaufmann Publishers Inc.

Riesen, K., and Bunke, H. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.* 27(7):950–959.

Weber, M.; Langenhan, C.; Roth-Berghofer, T.; Liwicki, M.; Dengel, A.; and Petzold, F. 2010. a.scatch: semantic structure for architectural floor plan retrieval. In *ICCBR '10*, LNCS, 510–524. Berlin, Heidelberg: Springer-Verlag.