Smart Areas
A Modular Approach to Simulation of Daily Life in an Open World Video Game

Martin Cerny[1], Tomas Plch[1], Matej Marko[2], Petr Ondracek[2] and Cyril Brom[1]
[1]*Faculty of Mathematics and Physics, Charles University in Prague, Malostranské náměstí 25, Prague 1, Czech Republic*
[2]*Prague Game Studios, Pernerova 55, Prague, Czech Republic*
{*cerny.m, tomas.plch*}*@gmail.com,* {*matej.marko,petr.ondracek*}*@warhorsestudios.cz, brom@ksvi.mff.cuni.cz*

Keywords: Smart Areas, Industrial Applications of AI, Computer Games, Open Worlds, Non-Player Characters

Abstract: Constructing believable behavior of non-player characters (NPCs) for large open worlds in computer games is a challenging application of AI. One of the greatest obstacles for practical game applications lies in managing the complexity of individual behaviors and in managing their development cycle. We propose the use of "Smart areas" to overcome these obstacles and allow for realistic simulation of NPCs day-to-day life and describe a particular implementation for an upcoming AAA game. For practical applications it is also vital to resolve usability issues and assess the productivity of the technology. We have conducted a qualitative study with 8 subjects that compares the performance of working with Smart Areas to using default AI tools. The study indicates that Smart Areas are not difficult to understand, allow for substantial code reuse, resulting in speedup in modification of existing behaviors, and force good structuring of behavior code.

1 INTRODUCTION

There is a growing number of first-person computer games that describe themselves as featuring "large open worlds". There are no universal criteria a virtual world has to meet to be considered "large" and "open" but one of the most important properties of open worlds is freedom: the constraints the environments enforces on an user's actions should be minimal. In an ideal case, the constraints are similar to the real world. Contemporary game worlds that are considered large feature a landscape of tens to hundreds of square kilometers. Such worlds are then populated with hundreds of non-player characters (NPCs).

The action-selection mechanism for NPCs in such games provides both theoretical and practical challenges for applied AI. Since the user has a large degree of freedom, the behaviors must not only look reasonable to a spectator, but must also maintain *interactive believability*, i.e., NPCs should react in a believable way to user actions. While combat behavior in contemporary games is usually well designed and interactively believable to a large degree, even recent and successful open-world games such as Red Dead Redemption (Rockstar Games, 2010) have resorted to severely limited non-combat NPC behaviors.

Furthermore, there are three difficult challenges for game AI in general. The first one is the severely limited CPU time, the second one is the need for tight control over NPC behaviors to assure that NPCs do not break the main story line or other mechanics and the third one is the need to create behaviors effectively and with relatively low-skilled staff.

All those constraints have forced the game industry to use simple reactive approaches for action selection — most notably FSMs (Fu and Houlette-Stottler, 2004) and behavior trees (Champandard, 2007). While some games have incorporated planning technology (Champandard, 2013), the severely limited resources prevent planning from being applicable for more than a handful of NPCs at one time and reactive approaches are still state of the art for non-combat behaviors in large open world scenarios.

Our task was to devise a technology that would allow designers and scripters to create a world where the NPCs actually "live" — they carry on their daily routine (work, relaxing, etc.), while maintaining interactive believability to a reasonable degree. We have approached this as a software engineering problem. With a naive approach it would be very difficult to create so many individual NPC behaviors and it would be close to impossible to maintain, test and debug the resulting codebase.

In this paper we present a case study of applying the concept of *smart areas* (SAs) taken from crowd simulation research to remedy these issues (Tecchia

et al., 2001). A SA is a specific place within the game world (e.g., a pub) that contains all the information the NPCs need to behave appropriately in this place (e.g., scripts for the innkeeper and for the guests). If necessary, a disembodied agent attached to the SA coordinates various NPCs within (e.g., assigns free seats to the guests, chooses NPCs to engage in a brawl, ...). As such, SA is a standalone object that may be plugged in without modification to existing NPCs.

Although the concept of SAs is simple in principle, actual implementation with real-world applicability is not straightforward. To test our implementation, we have performed a small-scale study. Its goal was to determine, if the concept is suitable for our scenario and if users are able to understand it correctly, to estimate the productivity gain from using SAs on top of our AI system and to uncover usability issues with the accompanying toolchain.

The paper proceeds as follows: first we introduce related work (Section 2) and discuss our implementation (Section 3). Then we present the evaluation we have performed and its results (Sections 4 and 5). The paper is concluded with discussion and aims for future work (Section 6).

2 KEY RELATED WORK

The concept of *smart objects* (Kallmann, 2001) is well-established in the game industry. A smart object is typically a graphical entity in the game world that is accompanied by a character animation (or several animations) that should be used when a character desires to use the object. A typical example of a smart object is a lever on a wall. An NPC that wants to change the state of the lever simply fires a "use smart object" action and the smart object takes care of the necessary details. This way, many different levers and switches may be present in the environment, but the AI only needs one action to use them all properly.

The concept of smart objects has been extended by crowd simulation research to whole areas. In (Tecchia et al., 2001) the environment is overlaid with a grid, where each cell may dictate a behavior for the agents in it. In case of the paper it was movement style. In (Sung et al., 2004) so-called "situation based behavior selection" is presented. The system detects situations in the environment and instructs the agents participating in the situation what should they do. While situation based behavior selection is more general than smart areas, the situations tested in the paper are mostly triggered by entering a location.

However, the crowd simulation approach cannot be directly translated to computer games because crowd simulations generally have low fidelity of individual behaviors.

(Brom et al., 2006) take the idea one step further with so-called "smart materializations". In their work the world is inhabited by agents using the belief-desire-intention architecture and smart materializations are behavior fragments embedded in the environment that provide ways to achieve intentions. For example the character may adopt a "have fun" intention. A pub in the environment would provide a materialization that realizes the "have fun intention" by instructing the agent to go to the pub and adopt subintentions "buy a beer" and "drink a beer".

3 OUR IMPLEMENTATION

Here we describe our implementation of the SA concept for an upcoming AAA RPG game. First we introduce the AI system which formed a base for our code and the design objectives we followed and then we discuss key aspects of our implementation.

3.1 Background

Our implementation is built on top of an existing custom AI system. The basic character decision making is performed by a variant of behavior trees (BTs) (Champandard, 2007). The BT formalism is extended with variables and a custom type system which allows for complex structured types and type inheritance. The AI also makes heavy use of a mature NPC-to-NPC messaging system. An NPC may have multiple *inboxes*, each having its own message type. The message system is tightly integrated with the BTs.

There were several design objectives we have followed when extending the AI system by SAs:

- Gameplay-critical behaviors (quests, combat, ...) may never be disrupted by SAs.
- Primary use-case of SAs is the simulation of routine day-to-day activities (work, sleep, eating, relaxing,...) of NPCs.
- Adding new places where NPCs could perform some of the activities (e.g., a new pub) should be possible without changing any of the NPC code.
- All NPCs should not behave identically within a SA: e.g., in a pub, rich people behave (and are treated) differently than poor people.
- Dynamic situations that are spawned by a certain player action (e.g., murdering someone on the street) and that provide all NPCs nearby with an appropriate behavior for such a situation must be

supported. Adding a new situation should not require a modification to code of any NPC.

- Synchronization of joint behaviors among agents (a pub brawl, a game of cards, ...) must be supported.
- The NPCs retain their own AI which coexists with the SAs. Smooth transition of control between NPC AI and SA must be ensured.

3.2 Behavior Adoption and Drop

To ensure that gameplay-critical behavior remains uninterrupted, we have decided that active NPC cooperation is required to receive a behavior from a SA. We have thus added a new BT node that requests a behavior from SA, if possible (further referred to as *request node*). This leaves the NPC AI in full control over transition to SA-controlled behavior and the SA behavior may be forcefully terminated by activating a different BT branch.

When the request node is updated and it does not hold a behavior already, it asks for one. Both the node and the SA take part in deciding, which behavior is applicable. Consider an example: an NPC asks for a behavior in a pub. Out of all of its behavior templates, the SA filters those with satisfied constraints on the requesting NPC (e.g., a template that involves inviting everybody for drink is available only to NPCs that are marked as rich). The request node then chooses among those templates based on its parameters (e.g., it may prefer "get-drunk" template to "eat" or forbid "play-cards" template). The same behavior may be implemented in several SAs — the NPC may require a "relax" behavior, which would have different implementations on a beach or in a pub.

If an applicable template is found, it is instantiated in a data structure called *behavior tag* which is passed to the request node. The behavior tag contains metadata about the behavior and an instance of a BT that achieves the behavior. The BT is pasted as the only child of the request node and the tree continues execution by evaluating the behavior subtree. If needed, new message inboxes are added to the NPC. If no applicable behavior is found, the request node fails.

When the NPC leaves the SA that has provided the tag, or the associated BT succeeds/fails there are three possibilities: the behavior may be kept by the request node as if nothing happened, it may be kept but marked as *overridable* or it may be dropped. The actual outcome is determined by the tag, by default the behavior is marked as overridable when it succeeds and dropped upon leave/failure. If the behavior is marked as overridable, the request node asks for a behavior upon its update but if no applicable behavior is found it continues executing the current one instead of failing. This allows some behaviors to persist their state between succesive executions or upon leaving the SA.

3.3 Smart Area's Brain

The basic decision making of the SA is passive: for each behavior template, the SA maintains information whether new instances of the template may be requested and the maximum number of instances that may be adopted at the same time. This information is used upon request processing.

Some SAs have an active *brain* — a behavior tree that gets updated regularly and may either modify the passive decision making based on external conditions (e.g., disable "drinking" behavior in a pub if no innkeeper is present) or it may perform some coordination among behavior tag holders inside the area (e.g., instruct a pair of customers to play cards together). The coordination is done by message passing between the SA and the tag holders. Since the NPCs are now controlled by BTs provided by the SA, the SA can make strong assumptions about NPCs responses to its messages.

In many scenarios, the SA needs to perform some action whenever an NPC adopts/drops a behavior (e.g., assign a free seat to a customer in a pub) or when an NPC enters/leaves the area (e.g., innkeeper says goodbye to the leaving guest). To streamline the development in such scenarios and to make the BTs of the SA brain and the behaviors more readable for developers, we have introduced event handlers to the SA brain. An event handler is simply a BT that is executed until completion for each instance of an event. So far, our system uses four events: OnAdopt — an NPC adopts a behavior, OnDrop — an NPC drops a behavior, OnEnter — an NPC enters the area, OnExit — an NPC leaves the area. The latter two fire whether the NPC has requested a behavior or not.

The event handler trees are queued and executed one at a time and may not be interrupted by execution of the main tree. This reduces the parallelism of the main tree and the event trees to a bare minimum and thus facilitates easier debugging. It is up to the scripters to make sure the handler trees execute quickly.

3.4 Smart Area Hierarchy

One of our primary use cases of SAs was the day-to-day behavior of NPCs. Now, once an NPC enters a pub for example, everything works well. But how does the NPC know, where a pub is? As mentioned in

our design objectives, the pub locations should not be hardcoded in the NPC's behavior. Our solution was to introduce parent-child relationship between SAs and make the whole city a SA and make the pubs its children. Now the city (the city designer) knows the locations of all pubs within. The NPC thus requests a "have fun" behavior from the city, the city gives it a BT that consists of a sequence of a move node that moves the NPC to one of the pubs and a request node that requests a "drinking" behavior in the chosen pub.

In a different situation, an NPC that is currently in a pub (without a behavior) may decide it wants to work, but the pub should not be required to know of all work possibilities in the city. It is thus a good idea to ask the city in such a case. For this reason, if the current SA cannot provide any applicable behavior, the request node asks the parent SA.

To implement dynamic situations the system is prepared to support SAs created on the fly (e.g., after a murder on the street an SA that instructs the witnesses to run away is created). The presence of such an SA will trigger a higher priority branch in the NPC's behavior tree which will contain a request node asking for situational behavior.

3.5 The Tools

As is always the case with practical applications, toolchain support is vital. We have modified a BT editor already present in the game to properly visualize the adopted behavior subtrees of the request node during runtime and to support breakpoints and other debugging features properly inside behavior subtrees and the event handler trees. A special SA editor was added to edit behaviors available within a SA and the associated conditions, variables and inboxes.

4 EVALUATION

Our main goal was to investigate what are the advantages/disadvantages of work with SAs compared to plain BTs. We were interested only in cases, where using SAs is natural, i.e., individual behavior subtasks are connected to specific places as in the daily cycle problem. We had four hypotheses: 1) Learning to create daily cycles with SAs is harder than with BTs, 2) creating a behavior from scratch is done faster with BTs, 3) modifying existing behavior is done faster with SAs and 4) SA code is easier to read. Note that the latter three hypotheses are related to a typical development lifecycle.

To investigate these hypotheses we have conducted a small-scale (8 subjects — 6 males, 2 females) pilot qualitative study. We followed the methodology from our previous research (Gemrot et al., 2014) as we are not aware of any other methodology for comparing behavior design tools. An important goal of the study was to find possible usability issues with the workflow we envisioned for SAs and the supporting toolset. While eight is a rather small number of subjects, usability testing research has shown that even small-scale studies do yield significant informations (Turner et al., 2006).

4.1 Experiment Setup

We have used a within-subject design as a between-subject approach would be impractical with so few participants. As all of the experiments had to be performed at one of the company's computers (for the sake of data protection policy) we had tight time constraints on the study.

We have conceived four simple tasks that would fit in the timeframe we had and would emulate the development of NPCs' daily cycles:

1. Create a single NPC with a simple daily cycle (4 behaviors, repeated in a fixed sequence, each with a predefined place to be performed).
2. Create two more NPCs with different daily cycles (4 and 6 behaviors, 1 behavior was newly added, others were the same as in Task 1).
3. Modify the individual behaviors that make up the daily cycle (4 of the behaviors should have been slightly modified).
4. Add a new place to perform one of the existing behaviors and let the NPC choose one of the two places with 50% probability, whenever it wants to perform this particular behavior.

As we are not interested in the individual behavior design per se, the BTs for the individual behaviors were prepared for the subjects and they could just copy and paste them into their solutions.

The study consisted of two parts: in one part, subjects performed the tasks using plain BTs and in the other they performed them with SAs. Half of the subjects used SAs for the first part, others started with plain BTs. To test code readability we have introduced a twist in the second part: the subjects were given a complete solution of Task 1 (created by the researcher) and solved only Tasks 2-4.

For the SA part a two-layered SA structure was required: a parent "city" area was responsible for guiding the NPC to a location of a specific child SA, where the actual behavior should have been performed, see Section 3.4 for details. For those solving Task 1 with SAs, the SAs were already present in the level, two of

them contained empty behaviors to hint the user at the desired structure and the rest did not contain any behaviors at all. The tasks 2-4 were technically identical for both parts, although they had different parameters. The whole experiment took 90-180 minutes.

Two of the subjects had substantial programming experience ($>$ 90 man-months), while two had very limited experience ($<$ 5 man-months). Two of the subjects were employees of the game studio and had a lot of experience working with the game editor and NPC design, the other six were university students and except for two with a minor experience had never seen the editor before and have not designed NPCs before. One of the employees could be considered an "SA expert", the rest of subjects had no prior experience with SAs.

It was measured how long the subjects took to complete the tasks. Qualitative feedback was given in a questionnaire after each part and the researcher assisting with the experiment took notes on the subject's usage of the tools as well as moments they seemed to experience trouble during the experiment.

The solutions the subjects ended up with after Task 4 were analyzed both qualitatively (reading the code by an exprt) and quantitatively (the number of nodes used in all of the BTs together).

5 RESULTS

A summary of the quantitative results is given in Table 1. In Task 1, users using SAs took on average much longer to finish than those using BTs. There are two explanations for this: First, except for one subject, the users were not familiar with the concept of SAs and thus spent more time figuring out what to do. Second, structuring content — as in our two-layered SA setup — is very likely to take more time than creating less structured content — as in plain BTs setup. While neither of the explanations can be ruled out, the data seems to support Hypothesis 1 (SAs are harder to learn) and Hypothesis 2 (creating new behaviors is faster with BTs) since subjects using BTs for Task 1 (none of them familiar with the editor) solved it very quickly.

However, the measured time to learn to work with an unknown technology is definitely acceptable ($<$ 1.5 hour for all subjects). When the users are presented with a working example, the learning time is further reduced: all users took less than 0.5 hour to replicate and modify the SA example in Task 2.

Considering the modification tasks (Tasks 2-4), solving them with BTs was faster than with SAs except for Task 3. This does not support our hypothesis that modifying behaviors with SA is faster, although the difference in Task 3 is statistically significant (p = 0.01, Wilcoxon paired test), while the other differences are not. Moreover both Task 2 — SA and Task 4 — SA involve time users spent learning the new technology. In Task 2, half of the users were using SAs for the first time, averaging over the users already familiar with SAs results in lower average time (9.75 mins) than for Task 2 — BT. In Task 4, the solution required the users to create a new SA, which is not difficult, but was not required in the previous assignments. The total time users spent with Tasks 2 - 4 slightly favors SAs over BTs, and the difference is on the verge of statistical significance ($p \approx 0.05$, Wilcoxon paired test).

Table 1: Summarized results of the experiments. Times for individual tasks are given in minutes with standard deviation in brackets. Task 1 was performed only in Part 1 and thus with only one technology per subject. For full experiment results, e-mail the first author of this paper.

	Behavior Trees		**Smart Areas**	
Task 1	10.00	(3.5)	47.25	(20.3)
Task 2	12.00	(1.5)	13.50	(6.0)
Task 3	12.88	(5.5)	4.88	(1.5)
Task 4	9.38	(3.7)	10.63	(6.8)
Sum 2-4	34.25	(9.4)	29.00	(12.5)
# Nodes	85	(14)	56	(4)

Table 2: Subjective evaluation of the tasks averaged over all subjects. The scale was 0 - 3, where 3 is most easy/tedious

	Assignment easy?	**Recreating tedious?**	**Modifying tedious?**
BT	2.7	1.3	1.1
SA	2.6	0.1	0.3

Users have marked the modification tasks more tedious when working with BTs than when working with SAs (see Table 2). Qualitative data supports this view: users modifying plain BTs complained about repetitiveness of the tasks and made more mistakes and spent more time testing the behaviors.

Generally, the data provide some support that SAs are better when modifications are frequent — which is the case in real development — but the results are not clear and further research is needed.

Except for one, subjects working with SAs produced almost identical code, while the solutions of users working with BTs differed more. We consider this positive: SAs enforce common structure and coding style which is a substantial advantage for large-scale development. Also the number of nodes is smaller for SAs and they were scattered in 14 simple

trees as opposed to 3 large trees for BT approach.

All of the subjects reported on usability issues with the SA editor, which have been addressed in further development. The last two subjects tested did not discover any previously unreported usability issues, which we consider to be a strong indication that only few were left undiscovered.

6 DISCUSSION AND FUTURE WORK

We have presented a case study of implementing smart areas (SAs) to suit the needs of game industry. We have emphasized code readability and reusability and support for development lifecycle as vital for industrial applications.

We have found SAs to be a suitable tool for developing daily-cycles of NPCs and other behaviors that are strongly bound to a particular place and only loosely bound to the NPC performing it. Our data provides support (although admittedly weak) that SAs are a better tool in modification-intensive applications, which is the case in practice. This advantage should outweigh the difficulties involved in more complex structure of the SA code and the necessary learning curve.

We have found that when presented with a working example, even users that have just been introduced to the SA concept can quickly replicate and modify the code. Moreover, the code produced with SAs is much more uniform than the code produced with BTs.

Our implementation of SAs has been approved for real-world deployment within an upcoming AAA RPG game. The SAs allowed our scripters not only to structure the code well, but to create experiences never before seen in a commercial game while retaining maintainability and reusability of code.

On the other hand, we could not experimentally support the hypothesis that SA code is easier to read and only the most basic features of the SA system were actually evaluated. In our setup a BT system with a reasonable support for subtree reuse would likely perform very similarly. The need for an NPC to be physically present in an SA to receive a behavior has also raised some complications, but it was kept as it lets the designer make stronger assumptions about NPC behavior.

Thorough usability testing and right tool support have also been a vital part of the initial success of the technology. We believe that these are lessons to be learned by the broader academic AI community to promote adoption of academic techniques in practice.

As a future work, more advanced usages of the framework should also be tested. Moreover, it would be interesting to evaluate the resulting behavioral fidelity of the world with the focus group method. We also plan to make use of the SA infrastructure to test some of the concepts coming from the interactive storytelling community in an AAA game setting.

ACKNOWLEDGEMENTS

Human data were collected with APA principles in mind. This research is supported by the Czech Science Foundation under the contract P103/10/1287 (GACR), by student grant GA UK No. 559813/2013/A-INF/MFF and partially supported by SVV project number 267 314.

REFERENCES

Brom, C., Lukavský, J., Šerý, O., Poch, T., and Šafrata, P. (2006). Affordances and level-of-detail AI for virtual humans. In *Proceedings of Game Set and Match*, volume 2, pages 134–145.

Champandard, A. (2007). Understanding behavior trees. *AIGameDev.com*. `http://aigamedev.com/open/article/bt-overview/` Last checked 2014-01-05.

Champandard, A. (2013). Planning in games: An overview and lessons learned. *AIGameDev.com*. `http://aigamedev.com/open/review/planning-in-games/` Last checked 2014-01-05.

Fu, D. and Houlette-Stottler, R. (2004). The ultimate guide to FSMs in games. In *AI Game Programming Wisdom II*, pages 283–302. Charles River Media.

Gemrot, J., Černý, M., and Brom, C. (2014). Why you should empirically evaluate your AI tool: From SPOSH to yaPOSH. In *Proceedings of 6th International Conference on Agents and Artificial Intelligence*. In press.

Kallmann, M. (2001). *Object interaction in real-time virtual environments*. PhD thesis, École Polytechnique Fédérale de Lausanne.

Rockstar Games (2010). Red Dead Redemption. `http://www.rockstargames.com/reddeadredemption/`. Last checked: 2014-01-05.

Sung, M., Gleicher, M., and Chenney, S. (2004). Scalable behaviors for crowd simulation. In *Computer Graphics Forum*, volume 23, pages 519–528.

Tecchia, F., Loscos, C., Conroy-Dalton, R., and Chrysanthou, Y. (2001). Agent behaviour simulator (ABS): A platform for urban behaviour development. In *Proceedings of Game Technology 2001*. CD-ROM.

Turner, C. W., Lewis, J. R., and Nielsen, J. (2006). Determining usability test sample size. *International encyclopedia of ergonomics and human factors*, 3:3084–3088.