日本デジタルゲーム学会
Digital Games Research Association JAPAN

デジタルゲーム学研究
Journal of Digital Games Research

Improved Game Artificial Intelligence through Interactive Environments

Paul DEWAAL[i] Andrea KUTICS[i] Satoshi TOYOSAWA[ii] and Akihiko NAKAGAWA[i, iii]

[i]Natural Science Division, International Christian University 10-2-3, Mitaka-shi, Toyko, 181-8585 Japan
[ii]Welfare and Information Division, Tokuyama University Shunan-shi, Yamaguchi, 745-8566 Japan
[iii]School of Information Systems, University of Electro-Communications 1-5-1, Chofu-shi, Tokyo, 182-8585 Japan

E-mail: g126723e@yamata.icu.ac.jp, matz@icu.ac.jp, toyosawa@tokuyama-u.ac.jp, ranaka@nt.icu.ac.jp

Abstract Game AI has become standardized to use behavior trees and scripts, but both have problems due to indirect communication between the game environment and the games objects. Our design uses an environment-driven approach which directly links the environment and objects. The evaluation experiment showed that the proposed method was perceived as having a higher level of character behavior as compared to a standard architecture. Likewise, when comparing the CPU-usage of calls, the method showed an overall decrease in CPU time.

Keywords Video Games, Artificial Intelligence, Behavior, Behavior Trees, Scripting, Interactive Environment

1. Introduction

Artificial intelligence (AI) is a rapidly growing field in the game industry[1]. In recent years the focus of game development has shifted away from graphical advancements. The focus has moved towards the improvement of artificial intelligence and character behavior[2]. Thus spurring the interest and development of bigger and better AIs in the game industry.

Game AI has come a long way since early video games and has branched into many subfields such as pathfinding[3] and animation[4]. Our focus for the Interactive Environment and another cornerstone of game AI is behavioral AI. Behavioral AI is developing systems and architectures to mimic intelligent thoughts, actions, decisions, and emotions for the agents in games. There have been many books[5, 6, 7] on the subject throughout the years and it is a highly discussed topic in developer circles[8, 9]. Though one of the newer focuses, it majorly impacts the gameplay experience. After all, if the game character on the screen looks just like a human, it will be expected to act like one as well.

In the broadest sense, the implementation of behavioral AI can be divided between two extremes: putting the agents on a predetermined path, or making them autonomous[6]. By predetermining the agent's behavior, the developer has to script the agent and tell it exactly what it wants it to do, when, and where. This method allows for agents to mimic human behavior very closely and appear very human and cinematic. But this process must be repeated for every new agent and every new behavior. However, it can allow for the developers make the agent behave to exact specifications (like a movie script) at the cost of forcing the agent to follow a very specific set of rules. This limited behavior is a side effect of forcing the agent to do a predetermined order of scripted behaviors in the game.

The alternate approach is autonomy[6]. Instead of very specific rules dictating exactly how the agent is to behave, behavior is approached from a broad point of view. Instead of handcrafting specific behaviors for the agents they try to come up with rules broad enough to react to a large range of player actions. Since they run autonomously, adding agents becomes a simple task of replicating them in the game world. The problem is, without specific and unique scripts and behaviors; continued interactions with the agents will uncover repetitive, generic actions, limiting the scope of their behavior's effect.

Finding a balance between the two extremes is the goal of many AI developers. This poses the question: how exclusively mutual are they? The common perception is you can only have one at the detriment to the other; and to some extent that's true. Certainly it's hard to say a cut scene (an

in-game movie) leads to no loss in autonomy. But, with increasingly popular techniques such as “interactive cut scenes” (where the environment and agents rely heavily on scripted behavior but the player is still able to affect the situation in a limited way) developers are attempting to meld both together. Usually if the developer wants to make something cinematic he has to script the specific actions for the environment, the agents, and possibly the player.

2. Related Work

A core goal of many AI architectures is autonomous behavior. These architectures are focused on allowing an agent in a game to run by itself without any help or influence from other classes and objects in the game. Some of these architectures achieve this by basing the agent on a series of states. Each state in the agent can be used to respond to a different circumstance and the AI system decides what state would be best to be in according to what’s happening in the game. Common game states might include an ‘attack’ state for when the agent is being aggressive or a ‘flee’ state for when the agent is trying to escape and hide in the game. The first of these architectures, and a staple of the AI industry, are finite state machines[10]. From these finite state machines, a new industry standard has emerged: behavior trees.

2.1 Behavior Trees

Trees can come in a number of different forms (such as behavior trees, decision trees, event-driven trees, etc) but their function remains the same. Behavior trees take the concept of state machines and add structure to the states and transitions, partitioning them into branches of a tree graph. This formulates a system that is not only easy to visualize, but also easy to manipulate using the multitude of tree traversal algorithms available.

Behavior trees are a way of grouping similar behaviors and actions together under a parent node to form “sub-trees” to create high level behaviors[11, 12].

These behavior trees often have their states and nodes linked to blackboards[13, 14]. A blackboard stores global variables the agents need or want to watch. Blackboards may store commonly needed variables like the location of the player’s character in the game. As these variables change they can trigger the behavior trees to change their state, individual nodes, or their behaviors. While monitoring these changes and reacting to them can give a real sense of thought and planning to the agents in a game, they also come at the cost of needing constant checking.

2.2 Scripts

While all the previous architectures are used for autonomous behavior in agents, different scripting systems are also used to manually direct agents in the game. A script is a piece of code written in a scripting language that calls functions and executes tasks in the game code that would otherwise have to be executed one after another by one of the developers. One of the most popular scripting languages used in games today is Lua[15]; with a number of different game engines supporting it. Although scripts can be used for a number of different tasks in game development, this paper focuses on scripts for game AI.

AI scripts are often used to give specific behaviors to an agent that would otherwise not work in an autonomous system. This can be for a number of different reasons, such as the behavior being too specific to warrant repeated use (such as a script for an agent’s death) or the behavior is too complex to code into an autonomous system

Another group of, and the largest scripts used for games are cut scenes. Games use cut scenes and cinematics to direct the game’s narrative and draw in the player. In these cases, every single object, agent, and even the player in the game is stripped of their control and is run exactly as the developers’ script.

3. Our New Design: The Interactive Environment

We propose a new game AI architecture that allows the agents and objects in the environment to send, receive and manipulate data between themselves within the game.

Normally objects in a game are nothing more than an art asset: the doors, walls, trees, etc. With the Interactive Environment (IE) we turn these 'art assets' into objects able to communicate with the agents in the game. Using this system there are a number of weaknesses that can be overcome from both autonomous and scripted AI systems. Autonomous systems have very little perception of the outside game environment, and the little perception they do have is based upon predetermined variables that are constantly checked.

Scripted systems, on the other hand, are very context specific, non-scalable, and also are usually embedded into predetermined variables. By allowing the agents to communicate (send/receive data) with the environment we overcome these weaknesses and create a new middleware game AI system. By showing this improvement in game AI through an interactive environment we bring further interest and more focus on using the environment in AI; rather than the more traditional method of focusing solely on the agents and their inner systems or scripts. Using the environment and empowering the objects inside it we attain new AI behaviors and a new architecture.

For this new AI architecture we designed a three class system. We designed a single class, the Interactive Environment class, to be in charge of holding and helping with communication between the other two classes, the IEEvent and IEObject classes.

3.1 Interactive Environment Class

The interactive environment class is the central class storing all IE objects in the game, and handles all events that are broadcasted (sent to the IE) while the game is running. The first variable that is looked at is the 'toID' variable. Should it be set the IE only checks the range between the event and the IEObject tied to that specific ID and if it is within range sends the event. Otherwise, an event is sent and processed using three variables: the size variable, range variable and the type variable. The range variable limits the receivable IEObjects to only those within the set range.

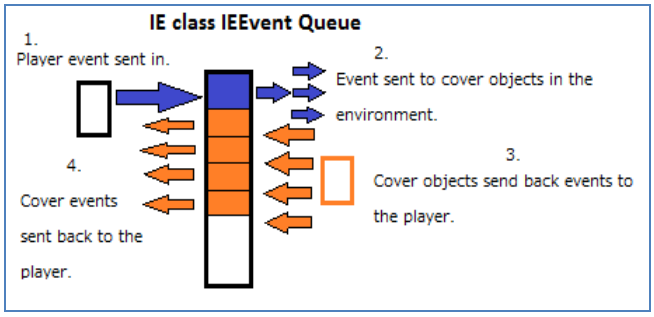

Figure 1: Order of event passing in the IE.

```
public void sendEvent(){
    IEEvent e= blackBoard.get(0);
    System.out.println("Sending Event"+e.
    toString());
    if(e.toID != 0){
        for(int x=0; x < objects.size(); x
        ++){
            if((objects.get(x).ID == e.toID
            ) && (withinDis(objects.get(x).
            location,e.location, e.range))){
                objects.get(x).sendEvent(e);
                x=objects.size();
    }}}
    else if(e.type == "Player"){
    else if(e.size == 0){
        for(int x=0; x < objects.size(); x
        ++){
            if(objects.get(x).getClass().
            getName() == e.type){
                if(withinDis(objects.get(x
                ).location,e.location, e.
                range)){
                    objects.get(x).
                    sendEvent(e);
    }}}}
    else{
    blackBoard.remove(0);
}
```

Figure 2: Code for processing events in the IE class.

The type variable tells the IE which types of IEObjects the event should be sent to. The size variable tells how many to send it to. If the size is set to '0' the event is sent to all IEObjects that fall under the type. Otherwise the event is sent to the first X (set by size) number of events, in range, that fall under the type. This can provide for both group-wide "goal-sharing" as well as unique data exchange.

```
public void typeHelper(String type){
    if(type == "Chair"){move();}
    else if(type == "Waiter"){paused=true;sinceP=System.
    currentTimeMillis();pause=toProcess;}
    else if(type == "OrderGive"){recieveOrder();}
}
public void move(){
    moveTo= (Point)toProcess.getData();
    moving=true;
    tree.state=4;
}
public void order(){
    chats.add(new Chat("I'd like a hamburger and fries."
    ,location));
    Order o= new Order(ID,location,"Hamburger and Fries"
    ,toProcess.ID);
    blackBoard.add(new OEvent(ID,1,1000,"Order",
    toProcess.ID,location,o));
    pause=null;
}
public void recieveOrder(){
    tree.state=3;
}
```

Figure 3: Code for processing events in the Customer class.

Figure 1 shows how an event would be sent using the IE. The process starts with the player sending out the black-outlined event to the IE. The actual passing of the events to the IE is done by an array list shared by the IE and all its IEObjects. The sent event then gets put in the IE's event array. When processing the events the IE implements a First-in-first-out (FIFO) approach. It updates every loop, and processes one event per loop. The code use by the IE to process events is shown in Figure 2.

3.2 IEObject class

The IE object class is the superclass of all objects that can send and receive events. Meaning both the agents in the game and actual game objects are children of this class. The object class stores a few important variables to be used: the shared IEEvent array list from the IE, an ID variable to store the objects unique ID, an array list to store events it needs to process, and a queued event that is run after a certain condition is met (e.g. reaching a certain location). Furthermore, by extending the other object classes (the player(s), agents, environmental objects, etc) from the IEObject class, it allows the game to hold all of these different objects in an IEObject array in the same place for use by the IE class.

The object class works using an update method which checks to see if any events have been received. If there are any in its queue it retrieves the oldest one (first in first out) and sends it to the typeHelper method (shown in Figure 3). From the typeHelper, the event's type is used to determine what function should be called, while many of the functions also change or determine the state the customer will be.

3.3 IEEvent Class

The event class has a number of identifiers saved to it. It stores: who the event is from, who (if anyone) it is going to, the size of how many times it should be sent, a range variable to decide how far the event is to be sent out, a variable to store the specific type of event it is, the location of the object sending the event, and then the data specific to the event.

The event class can be extended and by setting the 'type' variable in the event the developer can make new events for different IEObjects. Below are figures explaining more specifically how these types and IEEvents were handled in our IE game and is further discussed in chapter 4.1.

3.3.1 Flowchart for an Event Broadcast

Figure 4 shows an IEObject sending IEEvent12 to the IE. In Figure 5 the IE is processing IEEvent12. The first thing the IE does is look at the ToID field. If it is not null the IE will then scan its IEObject array for an object that shares that specific ID. Once found it will then send that event to that IEObject and the process will be complete. Since it is blank, the IE then looks at the type and size. The type tells the IE to look for Waiter IEObjects (because it's 'type' is waiter). The size of 1 tells the IE to only send this event once, to the first Waiter it can find in range. Thus the IE comes to the first Waiter in its array, Waiter1. It checks the range and if within range will send IEEvent12 to Waiter1 (Figure 6). Otherwise it continues checking all Waiters until one is found in range or it reaches the end of the array (at which point the event is discarded).

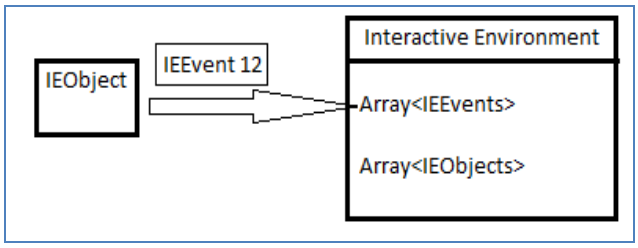

Figure 4: IEObject sends IEEvent12 to the IE; stored in the IEs event array.

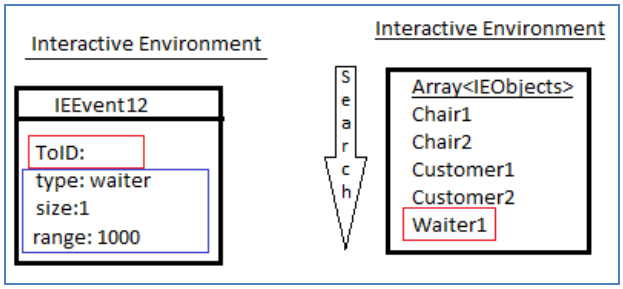

Figure 5: IE processing IEEvent12.

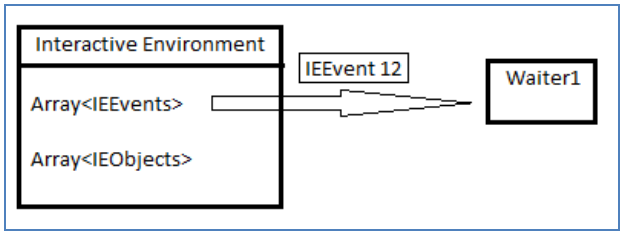

Figure 6: IEEvent12 passes the range check and is sent.

4. Evaluation Method

Game AI is focused on creating believable human behavior in their agents and on creating a system that is easy to understand and can scale to fit the size needed by the game. Therefore we decided to evaluate based on these two common goals of game AI; believability and the simplicity of the code.

To determine the believability of the games we made and compared two games. One game used the more common approach of state machines and scripting to model the standard of today's game AI architecture. A new architecture that could not beat or tie the current one would not be of use to game developers. The other game used our new Interactive Environment AI. The games were rated on their believability in a number of fields.

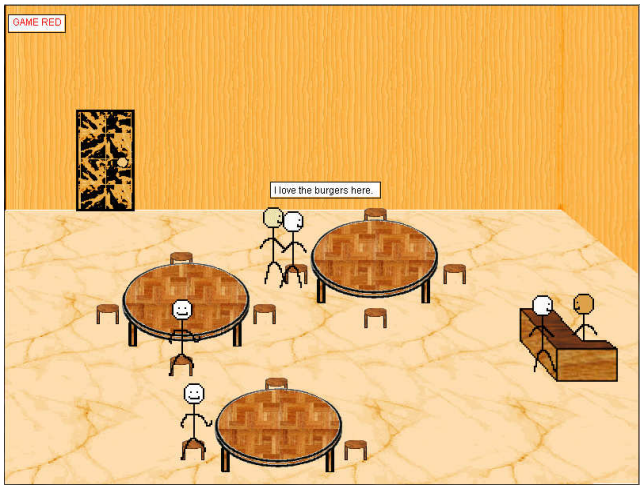

Figure 1: Screenshot of the game in action.

The major points of contention were in the behavior and reactions between the agents in the game. Participants were asked to rate their believability overall, believability within their interactions, and within their patterns.

Another self evaluation was given on the ease and abilities of the IE code compared to the more traditional methods of state machines and scripting. This code was compared between the systems based on the complexity of adding in new behaviors to each system and running the current behaviors. Complexity was judged based on the number of calls and checks that must be made to monitor and switch between the behaviors.

4.1 The Evaluated Games

The games were a single room environment portraying a bar and restaurant. There were tables in the game environments and chairs for the agents to sit in. The agents were the customers, the waiter, the barkeep, and the player. The waiter interacted with the customers and player to get their orders and relayed them to the barkeep. The barkeep interacted with the waiter by taking and sending out completed orders. The barkeep also gave out drinks and talk to the player when interacted with. The customers could call on the waiter to place and order, and then once it was received they would eat the order and either order again or leave the bar. Customers, the waiter, and the barkeep talked

to the player if interacted with.

The game was played from a top-down isometric view (shown in Figure 7), and the view was locked-in and could not be changed. The player could use a number of different keys to interact with the game. The ‘w’, ‘a’, ‘s’, ‘d’ keys were used for movement and the ‘f’ and ‘c’ keys were used to interact with other agents in the game. The ‘f’ key would interact with and send a chat event to the nearest agent while the ‘c’ key would call the waiter to the player to place an order.

4.2 The Participants

The participants were chosen from males, age 18-23. All with experience playing games. They completed the survey with what genres they play as well as how often they play per week. We chose all male participants to further keep the study group as non-biased as possible.

4.3 The Environment

The room will be climate-controlled to a comfortable temperature and they did not have outside distractions disturbing them. The games were played on a laptop, a Toshiba Satellite P750-BT4G22. The specs are as follows:

a) Processor: Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz (2 CPUs)
b) Memory: 6144MB RAM
c) Display Mode: 1366 x 768 (32 bit) (60Hz)
d) Display Size: 15.6 inch widescreen LCD screen
e) NVIDIA Geforce GT 540 1GB video card

The laptop was able to run both games with no performance issues. The participants were seated about 3x the height of the display away from the laptop. The brightness was set to around 80%.

4.4 Evaluation Survey

There are two games that were played by the evaluators. One game used scripts and state machines to run their agents behavior and the other game used the Interactive Environment system to run the agents. Both games ran off the same main classes. In each game the screen was drawn in the class named DrawingBoard (shared by both games) and handled user input using the same class, Interface. Both games had no sound and shared the same controls.

4.4.1 The Interactive Game Environment

The interactive environment game used the IE system of passing IEEvents between the objects (chairs) and agents (Customers, waiter, barkeep, and player) to change agent behaviors. Chairs would send events to custoemrs to tell them where to sit. Customers would send events to the waiter to place orders and the waiter would send events back to return orders.

There was also an agent (called the barkeep) that would take orders from the waiter and “cook” them before sending them back to the waiter to deliver. He also received events from the player for drink orders.

4.4.2 Standard Game

The other game for the evaluation used a mixture of state machines and scripts to direct its agent’s behavior. A script at the start seated the customers and from there state-machines dictated their behavior with simple “blackboard” check methods for responding to input from the player and from the customers wanting to place an order.

4.4.3 Survey

The evaluators spent 15 minutes with each game and then afterwards were given a survey to complete based on their opinions of the game. The survey was given in a 5-point Likert scale. The questions were based on the topic of “Rate the below items on how realistic and human-like you thought their behavior was” with the scale as: Not believable at all, slightly believable, believable, nearly real, and completely real.

The start of the survey asked questions regarding their age, favorite genres, amount of time spent gaming, and their preferred platform for gaming. The behavior questions using the Likert scale referred to the individual behaviors of the agents in the games as well as the behaviors among the

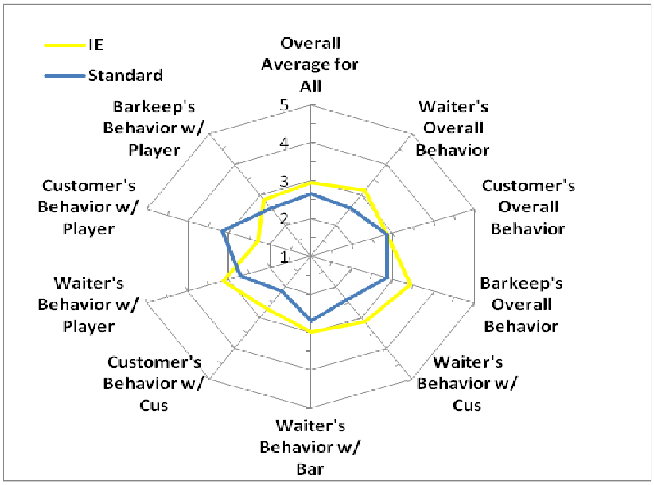

Figure 8: The average scores of survey questions.

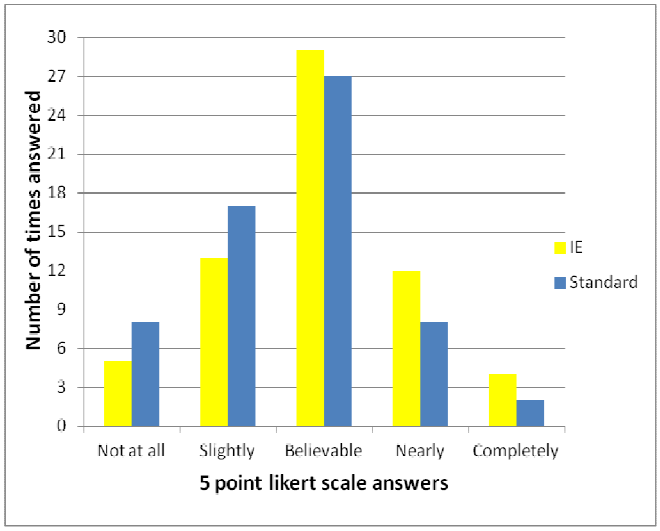

Figure 9: Chart of the survey answer dispersion.

agents. See the appendix for a copy of the survey.

4.5 Code Evaluation

The evaluation of the code is based on the comparison of the efficiency of the behaviors or states in the programs. This comparison is based on the number and type of calls used to achieve these behaviors. For a behavior that checks every cycle to decide if it needs to be checked, it was given an "every turn" rating; if it is activated by an action, then the sequence of events to activate it was counted and was given in the other ratings.

4.6 Procedure

The evaluators were told they would be playing two 15-minute games and then taking a survey afterwards. The order of the games was chosen randomly. There was a 5 minute resting period between the two games. The games themselves have no way to lose or complete the game; so there is no risk of the evaluator getting to an ending state. At the end of the second game session they were handed the form and asked to fill it out. The total time was around 40 minutes for each individual.

5. Results

5.1 Survey Evaluation

Seven male participants took the survey. The participants had an average age of 21. They averaged 3.3 hours of game playing every week.

While the average individual scores between the games showed differences, the overall average scores for both games were remarkably similar as shown in Figure 8. An average of all surveys was taken for each question in the survey and the results were also plotted.

Figure 9 shows the dispersion of the answers. As shown, the results of answers from both games showed a very similar dispersion as most participants rated most questions from the games as believable.

5.2 Code Evaluation

The only difference between the games in terms of code was the AI implementation; therefore the remaining code shall be omitted from the evaluations. The evaluations were separated based on the behaviors and states in the games. Therefore there was a separate state and behavior in the Standard game and/or IE game for a waiter taking an order and for delivering that same order. We evaluated 5 different behaviors (Waiter taking order, Waiter delivering order, Customer placing order, Customer receiving order, and Waiter dropping order) and summed their scores below in Figure 10.

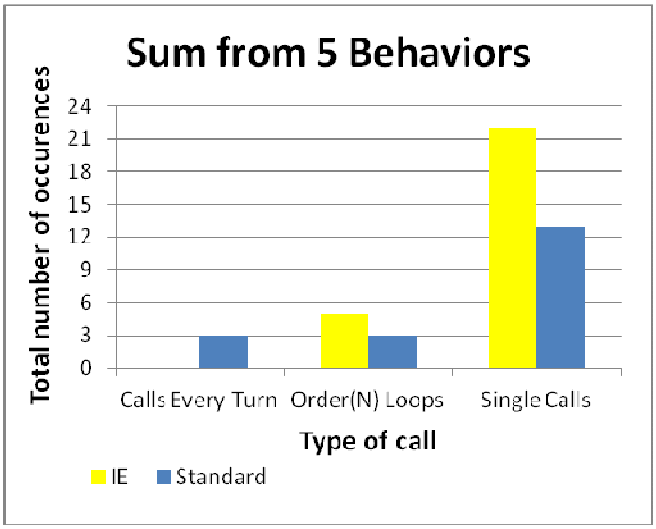

Figure 10: Chart of Code Results.

An example of our evaluation is given below in section 5.2.1. Some behaviors, such as entering and leaving the game, were fundamentally the same and thus were not evaluated.

5.2.1 Customer Placing Order

The behavior of the customer placing an order shows more improvement for the IE over previous evaluations. As shown in table 3, the number of lines needed to run the behavior in the Standard game uses 2 more calls that must be made every turn. The IE, on the other hand, does have a higher number of individual checks and calls.

The Standard gets its 2 every-turn calls from the inability of the customers to directly call the waiters. The Standard must check every round for customers needing to order. Since a waiter can only handle one customer at a time the other customers needing to order must spend every round sending out requests to have their orders taken until they are serviced. The second reoccurring call comes from inside their ordering state as they must check every turn if the waiter is close enough for them to give their order. The loop comes from once again having to loop through all the customers to get its order request sent out.

The IE also has its usual one for loop that is required to send/receive events to start the behavior and another to check the range for the event. Once inside though it requires

Table 1: Table showing the number and types of calls for the 'Customer Placing Order' behavior.

Calls by Type	IE	Standard
Every Turn	0	2
Order(N)[loops]: Run Once	1	1
Boolean Checks or Calls	6	3

a number of calls and checks to place its order. It must first send out the event to the waiter, after which it waits for the waiter to arrive and receives another event in response. From there is must make a call to create an order to be placed in the event it will send back out to the waiter. It then makes a final check changes its behavior to a waiting state.

6. Discussion

6.1 Survey Findings

As we discussed in the preliminary findings, the results were quite similar. From a player's point of view, they did not pay enough attention to note the small differences that might come up by using two different AI systems. In some cases, they even perceived a difference when there was none there.

In the IE for instance, the participants ranked the customer's behavior with the player significantly lower than in the Standard. The code and mechanics though, in both games, was exactly the same. The customers gave the exact same responses in each game, and were interacted with in the exact same way (standing near them and hitting the 'f' key). There was likewise no difference in the delay between responses in either game and they were drawn to the screen in the exact same way. Likewise, the barkeep's overall behavior was also graded differently though the code and implementation was equal.

But, there were areas where they were graded similarly by the participants; even if in some cases the code was different. The waiter's behavior with the bar and the player were graded nearly identical. Interestingly enough, the waiter's

behavior with the bar was the same in both games; but when dealing with the player each game's waiter had quite different approaches. In the Standard, the waiter gave preference to the player when they called for an order. On the other hand, the IE's waiter queued up each response for an order and then took them in that ordering (meaning if the player didn't call "first" they would have to wait their turn).

This suggests that as long as the behaviors given are equal (both games have the waiter come to the player to take their order, and both games have the waiter chat to the player and the player answer back with his order) that the actual mechanical differences working the background (giving preference to the player or not) aren't as easily noticed.

Throughout all the results, it gives a clear picture of equality. Given each game had the same behaviors and interactions available to the player it was shown the games thus give an equal level of believability. While small differences in the execution of the behaviors can be present between different systems, most went unnoticed and did not make a large impact on the final evaluation.

It was also shown that the overall experience when playing a game, can color the perception of different areas regardless of the actual differences that exist. In the case of both games, they had areas that were graded with a clear difference but were in reality coded and behaved the exact same way. This gives the appearance of the overall game experience and player mindset from other behaviors and interactions helping influence their thoughts and feelings on other parts of the game.

6.2 Code Findings

The coding findings showed a good pattern of the differences between the IE and the Standard game. In the case of the IE game there were no calls that needed constant checking; they were erased by the inherent architecture of the IE system. The Standard on the other hand consistently had at least 1 value that would need constant checking to cover for its inability to communicate and interact outside of its own agent system. Considering the small-scale and simplicity of the games it isn't hard to see the number of repeat checking getting larger with the game size in the FSM game. The IE, on the other hand, would continue to keep its need for constant checking to a near minimum.

It does come at a small cost though, as the IE consistently uses more calls and checks to run its behaviors. As a side effect of letting behaviors be triggered by events, it always had to at minimum run through the list of objects to find the one to send it to and then have the object check to discern what type of event it was once it was received. This is something that grows larger in waste as the size of the object array grew and needs to be taken into consideration. If efficiency was needed there are solutions like sorting the object array by ID or the range of the objects with respect to the player.

Overall though, the cost of constantly checking variables and states will waste more processing time and resources than a few extra checks. Furthermore, as found in the behaviors involving food orders, the IE had the advantage of being able to pass the actual order class inside the event it broadcasted allowing for the order to be changed and manipulated. In the case of the Standard it had to store the order inside itself and lock it away until it received approval from the waiter.

7. Conclusion

The behavior evaluations of the IE showed equal levels of believability to the industry-standard systems of FSMs and scripts. Given equal behaviors and interactions, the games were ranked equally. With further testing in future studies, we expect to find a difference felt by players as new behaviors are introduced into one game, while the other game stays the same. Since new behaviors can be easily coded and introduced into our IE, if the difference felt by players is significant enough; the benefits and uses of an IE could be further shown. As the ground case has been proven equal (i.e. with equal behaviors, different coding practices

achieve the same results in subjective terms) we plan to further expand on showing both the importance of adding new behaviors to improve player perception and showing the more efficient coding method achievable by an IE.

Small scale testing of the code showed the ability of the system which can rid itself of excessive redundant variable checking which you might find in a more traditional FSM or behavior tree system. While larger and more scaled up games have not yet been tested using the Interactive Environment, we see the benefits as showing greater improvement while the testing environment is scaled up.

In modern game development, deciding between having agents being autonomous or simply scripted is a situation every developer faces. The state of current architectures seem to acknowledge this point but rarely accommodates both. The future for making better games which draw in the user and make for more realistic gameplay is something that won't choose sides but instead tries to close the gap between the two. It's no longer good enough to simply keep building up architectures that only work within the confines of themselves. Instead it's better to look toward being able to communicate and share with other systems.

Incorporating a system that allows for communicating between agents and other objects in the environment will open up a number of different paths for allowing both autonomy within script and reducing the waste associated with constant checking in autonomous systems. The goal is to spread out more of the burden of behavior off the agents themselves and allow for a much more interactive environment for behavior to be made in. While many technical and coding issues must be overcome, the first step to making any new game architecture is creating and bringing forth the theory behind it. We anticipate creating a greater awareness of the power and ease of using the environment to create more game AI systems like in our Interactive Environment; and by bringing this awareness, furthering the theory of video game artificial intelligence for future systems and games.

References

[1] Parker, L,. The future of A.I. in Games. Gamespot, November 11th, 2010, <http://www.gamespot.com/features/the-future-of-ai-in-games-6283722> (Jan. 15, 2012).

[2] Woodcock, S., Game AI: The State of the Industry. Gamasutra, August 20th, 1999. <http://www.gamasutra.com/view/feature/3371/game_ai_the_state_of_the_industry.php?> (Oct 20, 2011).

[3] Sturtevant, N. R., Memory-Efficient Abstractions for Pathfinding. (2007). *Proceedings of the 3rd Artificial Intelligence and Interactive Digital Entertainment Conference*, 31-36.

[4] Champandard, A., and Gyrling, C., Animating in a Complex World: Integrating AI and Animation. (2009). *Proceedings of the Game Developers Conference AI Summit.*

[5] Mark, D. (2009). *Behavioral Mathematics for Game AI.* Course Technology PTR.

[6] Rabin, S. (2008). *AI Game Programming Wisdom 4.* Charles River Media.

[7] Russell, S. J., and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Upper Saddle River, N.J. Prentice Hall.

[8] Game Developers Conference. June 12th, 2012, <http://www.gdconf.com/> (May 24, 2012).

[9] Game AI Conference. June 12th, 2012 <http://gameaiconf.com/> (May 24, 2012).

[10] Lee, D., and Yannakakis, M. (1996). Principles and Methods of Testing Finite State Machines-A Survey. *Proceedings of the IEE. Volume 84, number 8*, 1090-1123.

[11] Colvin, R., and Hayes, I. J. (2010). *A Semantics for Behavior Trees.* The University of Queensland.

[12] Cerpa, D. H., Champandard, A. J., and Dawe, M. (2010). Behavior Trees: Three Ways of Cultivating Strong AI. *Presented at Game Developers Conference AI Summit.*

[13] Riedl, M. O. (2005). Towards Integrating AI Story Controllers and Game Engines: Reconciling World State Representations. *Proceedings of the IJCAI Workshop on Reasoning, Representation and Learning in Computer Games.*

[14] Isla, D. and Blumberg, B. (2002). Blackboard architectures, *AI Game Programming Wisdom.* Charles River Media, Inc, 333-344.

[15] DeLoura, M. The Engine Survey: General Results. Satori, March 5th, 2009. <http://www.satori.org/2009/03/the-engine-survey-general-results> (Feb. 9th 2012).