Using Cyclic Scheduling to Generate Believable Behavior in Games

Richard Zhao and Duane Szafron
Department of Computing Science, University of Alberta, Edmonton, AB, Canada T6G 2E8
{rxzhao, dszafron} @ualberta.ca

Abstract

Video game virtual characters should interact with the player, each other, and the environment. However, the cost of scripting complex behaviors becomes a bottleneck in content creation. Our goal is to help game designers to more easily populate their open world with background characters that exhibit more believable behaviors. We use a cyclic scheduling model that generates dynamic schedules for the daily lives of virtual characters. The scheduler employs a tiered behavior architecture where behavior components are modular and reusable. This research validates the designer usability of an implementation of this model. We present the results of a user study that evaluates the scheduling system versus manual scripting based on three metrics of behavior creation: behavior completeness, behavior correctness and behavior implementation time. The results indicate that the behavior architecture produces more reliable behaviors and improves designer efficiency which will reduce the cost of generating more believable character behaviors.

Introduction

Video game worlds contain many virtual characters that are Non-Player Characters (NPCs). As virtual worlds become larger and more realistic-looking, the importance of socially believable virtual characters also grows. These characters should interact with the player, each other, and the environment. In recent years, the behavior AI of virtual characters has not developed as quickly as other aspect of video games, such as graphics. Complex behaviors can be created manually, but the creation time makes the cost prohibitive for background characters at a large scale. Scripting costs are an important contributing factor to a major bottleneck in content creation, leading to a plethora of games with very simplistic behaviors for almost all virtual characters outside the main plot line. AAA titles such as Dragon Age II (BioWare 2011), Mass Effect 3 (BioWare 2012) and The Elder Scrolls V: Skyrim (Bethesda 2011) have several main characters whose behaviors are quite believable in some instances, due to extensive scripting. However, these same games have secondary characters and extras that lack believability and simply exist to fill up the game world.

Three key changes are necessary to address this problem. First, the behaviors must be more believable as measured by game players. Second, the behaviors must be reliable so that they work as designed. Third, designer efficiency must be increased to reduce production costs.

This paper extends our work on a tiered behavior architecture model (Zhao and Szafron 2014) which attempts to alleviate the content creation bottleneck for cyclic behaviors. Our architecture employs a cyclic scheduling algorithm, which determines the general objectives of the virtual characters and specifies the roles that will satisfy these objectives dynamically at game time. We validated the believability of the generated behaviors against a recent commercial game with a user study (Zhao and Szafron 2014).

The research in this paper further evaluates the design and implementation of this tiered architecture for cyclic behaviors by addressing the behavior reliability and designer efficiency issues. It presents the results of a second user study that compares the behaviors generated by an implementation of our architecture with behaviors created by manual scripting. It uses three metrics: behavior completeness, behavior correctness, and behavior completion time. These metrics are indicators of behavior reliability and designer efficiency, key elements in reducing the cost of generating more believable behaviors, thus adding more depth to the characters.

In addition, as noted by Li et al. (2014), having autonomous agents with their own back stories has benefits outside of the video game environment, as they increase user engagement and can be applied to other non-gaming situations.

Related Work

Researchers have explored many ways to control the behaviors of virtual characters. At the highest level, research has been done on providing complex interactive narrative experiences for players (Riedl and Bulitko 2013). Although this is the ultimate goal, to achieve success it is also necessary to improve the behaviors of individual characters who contribute to the narrative. One approach is to directly control all characters via manually written programming code. Specialized programming languages, such as ABL, have been proposed to help with designing believable agents (Mateas and Stern 2002) and they certainly have been demonstrated to improve the resulting characters. Gomes and Jhala (2013) have demonstrated the use of ABL in the believability of NPC social conflict resolution. However, even with such better scripting languages the production costs are high. Our goal is to evaluate the alternative of eliminating manual scripting in favor of automatic script generation.

Other alternatives to scripting exist. Finite State Machines (FSMs) have been used in commercial games for many years. However, with increasingly complex game environments, FSMs do not scale well and get harder to maintain (Schwab 2009). Hierarchical FSMs and Behavior Trees were introduced to address this problem (Isla 2005) with the game Halo. Further improvements include Data-Oriented Behavior Trees and Event-Driven Behavior Trees (Champandard 2012). More recently, learning Behavior Trees have been proposed, which adapt Behavior Trees with known player traces and modifiers (Tomai and Flores 2014). Layered Statechart-based AI by Dragert et al. (2012) stresses modularity and reusability as key features to successful AI techniques. As we will see, at the lowest level of our proposed architecture, any of these techniques can be used to fulfill the modular role behaviors.

However, at a high level, behavior specification can be viewed as a planning exercise since the designer should be able to generate behaviors from a set of constraints. For example, in the scheduling domain for daily behaviors, specifying a behavior for each hour or even for a specific period of time can be burdensome to a designer who only wants to specify the total time allocated for each behavior. Rather than using transitions (FSMs, Decision Trees or Behavior Trees) for each hour (or period of time), the designer should be able to specify more general constraints. Nevertheless, we would welcome user studies that evaluate the believability and reliability of generated behaviors and designer efficiency for complex cyclic behaviors using FSMs, Decision Trees or Behavior Trees.

Planning techniques have also been used in the context of guiding behaviors for characters. Goal-Oriented Action Planning was used by Orkin (2006). Recently, Coman and Munoz-Avila (2012) used case-based planning techniques to provide a variety of behaviors and demonstrated it in the Wargus real-time strategy game engine. Kadlec et al. (2012) constructed a specific game where a virtual character tried to achieve a goal using planning with an incomplete knowledge of the environment, and the player must help the character succeed. Kelly et al. (2008) used offline planning with Hierarchical Task Networks to generate behaviors. We share their goal of shifting the workload offline, but we embed scheduling in a broader architecture that supports dynamic online assignment of low-level behaviors (roles) to the root nodes (objectives) of the scheduler. They also indicated that the generated scripts require human effort to "compose and debug." The behavior scripts we produce require no human intervention. In addition, our implementation provides a graphical user interface with different constraint types that our user study has shown to be efficient and reliable.

In the scheduling domain, the interval scheduling maximization problem is one of the oldest problems, where the goal is to schedule a largest set of non-overlapping intervals. A greedy polynomial time algorithm exists as the solution (Kleinberg and Tardos 2005). The scheduling problem presented in this paper does not aim to find an optimal solution. It determines whether a solution is possible that fits all requirements, and if possible, it finds a family of solutions. Instead of intervals of fixed start and end times, it has blocks of fixed lengths with variable start and end times to better match requirements for scheduling believable cyclic behaviors of virtual game characters.

Bakkes et al. (2012) described player behavioral modeling strategies, and argued that circumstance-based abstractions of player models are needed. The level of details concept is used in this paper, even though this paper focuses on the behaviors of background NPCs exclusively.

Graphical drag-and-drop interfaces and pattern-based approach to scripting have been explored by many. In the academic environment, CMU's Alice (Pausch et al. 1995), MIT's Scratch (Resnick et al. 2009), and University of Alberta's ScriptEase (Schenk et al. 2013) are but a few examples. Becroft et al. (2011) described a graphical tool to create and edit Behavior Trees. These tools are all aimed at story designers and those who are learning to program. Some commercial game engines have employed visual interfaces as well, such as the Blueprints visual scripting system of the Unreal Engine (Epic Games 2014), formerly known as Kismet. While these tools are able to provide event-based scripting capabilities, they are general purpose tools which are not specialized in generating believable behaviors for virtual characters. It would be difficult to implement a cyclic scheduler using these graphical tools.

Cyclic Scheduling

Our tiered behavior architecture model (Zhao and Szafron 2014) separates the behaviors of a virtual character into leveled tiers, where each subsequent tier contains behaviors at more focused granularity (Figure 1).

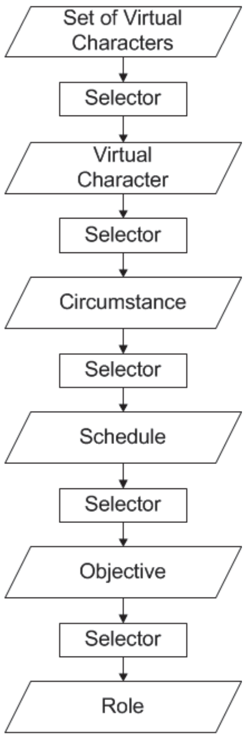


Figure 1. The tiered behavior architecture model, from Zhao and Szafron (2014).

Figure 1 shows circumstances, schedules, objectives, and roles as data layers, which are sets or lists of items, where each item is composed of items from the next layer below. With this architecture, each item is modular and reusable in the sense that they are self-contained units which can be adapted to a different place in the behavior architecture.

When a layer (usually the schedule) is expressed as a list, with time as a natural ordering mechanism, a cyclic scheduler generates the items in this layer. Specifically, a cyclic scheduler determines the objectives of the virtual character in a schedule. A simple schedule includes four common objectives: sleep, eat, work, and social activities, as shown in Figure 2, although more objectives can be created by the designer. Each objective can contain multiple roles that can satisfy the objective. For example, to eat, one can eat at home, eat at a tavern, eat at a friend's place, etc. according to the designer's wishes, and the availability of executable roles during game time.

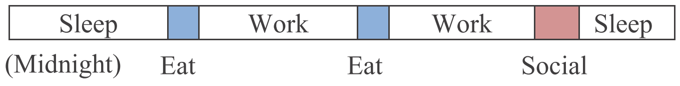
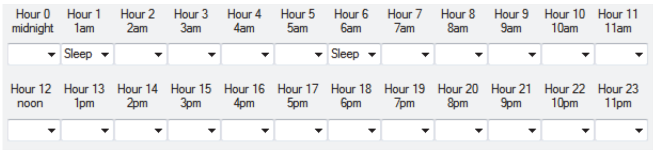


Figure 2. An example of daily schedule of a virtual character in a medieval setting.

In The Elder Scrolls V: Skyrim (Bethesda 2011), a typical daily schedule of a virtual character is usually cyclic on a daily basis. Given the large number of characters in such a game, automatically populating these behaviors would be very helpful. This system allows game designers to directly specify any important aspects of a daily schedule and the system selects the other aspects.

A designer considers the principal mission of a particular virtual character, and how the character can execute this mission. For example, a farmer usually does typical farming tasks, along with sleeping, eating, and socializing. All virtual characters fitting the farmer mission have similar daily routines in Skyrim.

The cyclic scheduler tool includes several key aspects. First, it has a timeline (Figure 3) that allows key constraints at specific hours to be specified. For example, the designer can ensure that the character is asleep at 1am and 6am, using the timeline. The planner may add more sleep times at unconstrained times on the timeline, but the times on the timelines are honored. We have included one-hour increments in our timeline but this can be easily generalized to arbitrary increment sizes.



Hour 0 midnight	Hour 1 1am	Hour 2 2am	Hour 3 3am	Hour 4 4am	Hour 5 5am	Hour 6 6am	Hour 7 7am	Hour 8 8am	Hour 9 9am	Hour 10 10am	Hour 11 11am
	Sleep					Sleep					

Hour 12 noon	Hour 13 1pm	Hour 14 2pm	Hour 15 3pm	Hour 16 4pm	Hour 17 5pm	Hour 18 6pm	Hour 19 7pm	Hour 20 8pm	Hour 21 9pm	Hour 22 10pm	Hour 23 11pm

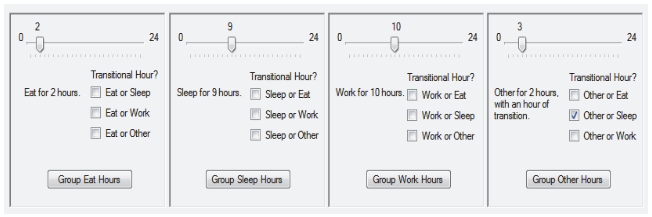
Figure 3. The timeline tool showing some hours filled in by the designer.

Second, the cyclic scheduler tool provides designers with the option to set total hours for each objective (Figure 4), as well as options to group the objective hours into consecutive or non-consecutive blocks. For example, the designer may require a total of 9 consecutive sleep hours. Along with a timeline constraint of sleeping at 1am and 6am, this would require the character to sleep for any 9-hour consecutive block that includes the hours 1am to 6am. Figure 4 shows the hours for the schedule from Figure 2.

The ability to specify the exact location and role of a character at a particular time allows designers to coordinate the schedules of multiple characters without over-specifying the behaviors of these characters at other times. For example, the designer may arrange for a thief and a fence to be in the roles buyer and seller in a specific alley at mid-night and design the rest of their behaviors independently.

Third, a designer can specify transitional hours (Figure 4), where the character has a probability of using the next objective an hour earlier, an hour later, or anytime in between. This provides stochasticity in the schedules so that they do not look identical from day to day.

Fourth, the tool allows a designer to pick one or more roles that can be used to satisfy each objective, as well as the percentage chance each role can be picked, subject to dynamic availability of each role at game time (Figure 5).

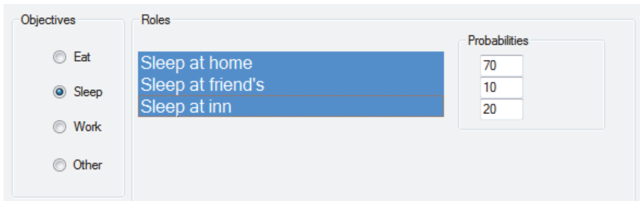

0 2 24
Eat for 2 hours.
Transitional Hour?
☐ Eat or Sleep
☐ Eat or Work
☐ Eat or Other

0 9 24
Sleep for 9 hours.
Transitional Hour?
☐ Sleep or Eat
☐ Sleep or Work
☐ Sleep or Other

0 10 24
Work for 10 hours.
Transitional Hour?
☐ Work or Eat
☐ Work or Sleep
☐ Work or Other

0 3 24
Other for 2 hours, with an hour of transition.
Transitional Hour?
☐ Other or Eat
☑ Other or Sleep
☐ Other or Work

Figure 4. The designer can specify hours for each objective.


Objectives
○ Eat
● Sleep
○ Work
○ Other

Roles

Roles	Probabilities
Sleep at home	70
Sleep at friend's	10
Sleep at inn	20

Figure 5. The designer can specify probabilities for each role.

Finally, the scheduler informs designers if the specified constraints are satisfiable or unsatisfiable, as specified by the cyclic scheduling algorithm in the Appendix. The greedy depth-first search algorithm employed in this case is complete: if a solution exists, the solution must contain every specified objective block in some location. The greedy algorithm iterates through every location for each block until a satisfiable location is found. Therefore the algorithm will find a solution if one exists.

Schedules are generated off-line before the start of the game, and game-time selectors dynamically assign roles to objectives as the game is played, allowing dynamic adaptation. The scheduling component is done off-line to reduce game-time computation. As noted by Wright and Marshall (2010), game AI must be fast. With only a fraction of the processor time allocated to AI each frame, the game-time component of the AI must be computed between frame displays.

The tiered behavior architecture also enables Level-Of-Details AI (Wißner et al. 2010), which allows off-screen and faraway characters to use only the appropriate high tier behavior without going into the details at the lower tiers, thus saving computation time on-line.

User Study Results

Our previous user study (Zhao and Szafron 2014) showed that a cyclic architecture is able to produce behaviors that are more believable than the default behaviors of the commercial game, The Elder Scrolls V: Skyrim. However, unless the architecture can generate reliable behaviors quickly, the scripting bottleneck will remain. In this paper, we present a user study that measures the reliability of generated behaviors and the efficiency of behavior designers using the tiered architecture as compared to manual scripting of behaviors.

We used Skyrim to evaluate behavior believability (Zhao and Szafron 2014). However, we found it more convenient to evaluate designer efficiency and behavior reliability using Neverwinter Nights (NWN) (BioWare 2002), based on the availability of a pool of suitable study participants with previous NWScript experience who could write manual scripts. Therefore, we modified our Skyrim-based implementation to work with the NWN engine and to generate NWN scripting code, but retained the GUI.

There were two groups of participants: one group used a tool that implemented the cyclic architecture. The other group used the manual scripting method for NWN. To ensure a fair comparison, all participants were required to have played the NWN game so that they could test their behaviors quickly using familiar settings and controls. Scripting Group participants were also required to be programmers who had prior experience with NWN scripting. Participants in the Tool Group were not required to have NWN scripting experience. No participants had seen the cyclic architecture tool before the study.

At the start of the study, both groups were given an instruction manual for their respective methods. The Tool Group was given a detailed manual on how to use the architecture tool, while the Scripting Group was given access to the NWN Lexicon, a familiar online manual for the scripting language. Both groups were also given a game file, which contained an identical pre-made town area, populated by four characters. Participants used this game file, and creating the behaviors was their only task.

Lastly, the participants were each given a sample behavior for a sample character. The sample behavior was identical for both groups, but was implemented in the respective method for each group. Participants were asked to first examine the sample behavior to see how it was implemented (either with the tool or with scripting code), and were told they could use the sample as a starting point. The sample character behaved as follows: He sleeps at home from midnight until hour 6. He starts to work at the Market at 7, for 10 hours, then eats at Tavern A at hour 17 for 2 hours, and then sleeps at home from hour 19 until the next day. He repeats the same behaviors for three days.

The actual behaviors that the participants were asked to create were identical for both groups. They were asked to follow instructions to create the behaviors for four characters in order, Adam, Bob, Cathy, and Donna, each for three consecutive days in game. Each subsequent character had increasingly complex behaviors as shown in Table 1. Participants were allocated a maximum of three hours to complete all behaviors, after which the study was stopped regardless of completion. Participants were also asked to record the time that they spent on each character.

A total of 25 participants were recruited in the user study. The Tool Group had 15 participants, where 5 were programmers and 10 were not. A programmer is defined as someone who self-reported having written many computer programs (in any language). The Scripting Group had 10 participants, all experienced NWN script programmers.

	Adam	Bob	Cathy	Donna
Multiple Schedules	Yes	Yes	Yes	Yes
Stochastic Schedules	Yes	Yes	Yes	Yes
Multiple Roles		Yes	Yes	Yes
Blocks of Roles			Yes	Yes
Dynamic Roles				Yes

Table 1. The aspects of the behaviors.

Completeness

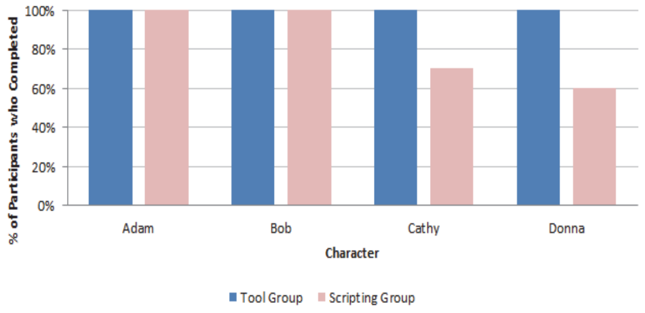


Figure 6. Completeness at 80% or higher: Tool Group vs. Scripting Group.

Figure 6 shows the percentage of participants who completed at least 80% of the requirements of all behaviors for each character. We assigned 100-160 requirement correctness points for each day of each character. We used the same scoring rubric for tool users and scripters. For example, on Day 1, character Adam had the following points assigned: 10 points (Adam starts at home), 20 points (Adam goes to the city gate at hour 7), 20 points (Adam goes to Tavern B at hour 17), 20 points (Adam goes home at hour 18 or 19), 20 points (previous time is random), 10 points (Adam stays home for rest of day). For each of the 20 point destination-time requirements, 10 points were awarded for the correct time, and 10 points for the correct location. We used 80% completeness to avoid penalizing a participant who misinterpreted a requirement. For example, they may pick the wrong location. A participant who made small misinterpretation errors and proceeded to the next character would not be penalized for completeness, only correctness.

In addition, the NWN scripting environment notifies a user if a script does not compile. In this study all scripts of all scripting participants compiled successfully and ran without crashing the game. Therefore a scripter was not penalized for completeness due to undetected scripting errors.

All 15 participants of the Tool Group completed at least 80% of the requirements of each character. From the Scripting Group, all 10 participants completed at least 80% of the requirements of the first two characters, 7 participants completed at least 80% of the requirements of the third character, and 6 participants completed at least 80% of the requirements of the fourth character. Due to the small sample sizes, there are no statistically significant differences between the two groups for the percentage of participants who completed 80% of each character.

Correctness

We used two different rules to measure correctness, one using all characters created by participants, and the other using only the characters completed at the 80% level. The second measure allows minor requirement misinterpretation errors to not adversely affect the interpretation of correctness as a measure of reliability, since a blunder in interpretation is not necessarily a reliability issue. In addition, the second measure excludes characters that were not attempted by a participant so it only measures correctness of the work done.

For each of the characters, the correctness number represents the percentage of the requirements of the behaviors that were correctly created, averaged over the participants. Figure 7 shows the results. The bottom of each bar indicates the minimum correctness value across all participants, and the top of each bar indicates the maximum correctness. The solid line in between represents the average correctness score counting all participants, and the dotted line (for the Scripting Group) represents the average correctness of only those participants who completed at least 80% of the character (ignoring those less than 80% correct). For Adam/Bob there are no dotted lines since everyone completed at least 80%.

For all four characters, correctness was higher with the Tool Group than the Scripting Group. Looking at the overall results (rightmost two bars), the Tool Group had an average correctness rate of 97.01%, while the Scripting Group had a correctness rate of 79.72% counting all characters (solid line), or 91.30% counting only at least 80% completed characters (dotted line). A one-tailed unequal variance T-test indicates this difference is significant at the 95% level for both measures (Table 2).

Looking at each individual character, there is a significant difference between the Tool Group and the Scripting Group for the most complex character Donna as well as Bob (Table 2). For Cathy the difference is significant only when counting all characters.

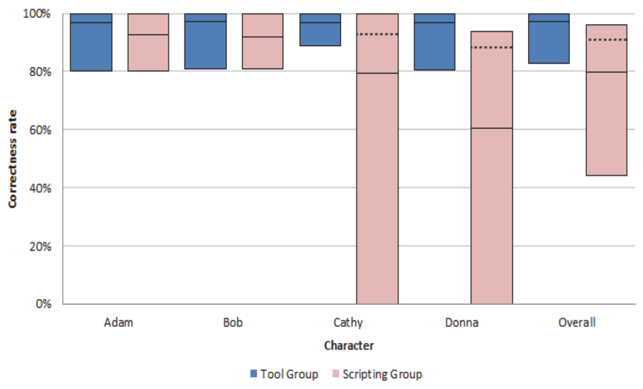


Figure 7. Correctness: Tool Group vs. Scripting Group.

	Tool Group vs. Scripting Group (all)	Tool Group vs. Scripting Group (80% completed)
Adam	0.057	0.057
Bob	0.039	0.039
Cathy	0.045	0.094
Donna	0.011	0.001
Overall	0.006	0.007

Table 2. P-values of T-tests comparing the two groups. P-values less than 0.05 indicate significance at 95% level.

Completion Time

Figure 8 shows the average time needed to implement each character by the two groups. Again, each bar indicates the minimum, average, and maximum time values across participants. Since three participant did not complete Cathy, and four did not complete Donna (all from the Scripting Group), they did not write down the time to complete these characters. To provide a fair comparison, for these participants only, we estimated the time (T) they would have taken to complete each of these characters, using their own time on the previous character, together with the average of other participants' ratio of times between the uncompleted character and the last completed character, using the formulas:

$$E(T_{Cathy}) = T_{Bob} \times (T^*_{Cathy} / T^*_{Bob})$$

$$E(T_{Donna}) = T_{Cathy} \times (T^*_{Donna} / T^*_{Cathy})$$

E() represents the estimated time. T^* represents the average time spent on the character denoted in subscript by all participants who completed that character.

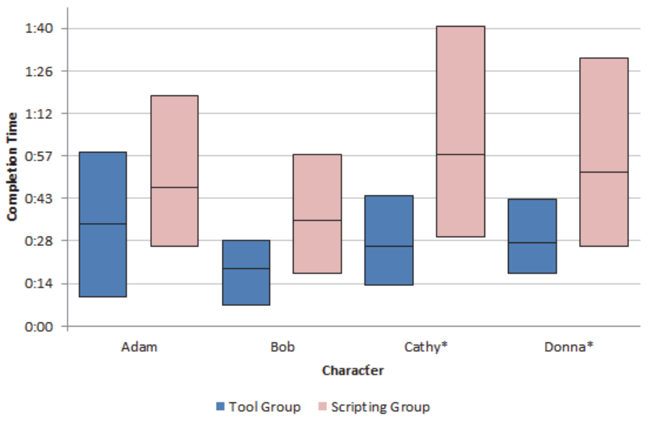


Figure 8. Completion time: Tool Group vs. Scripting Group for all characters. The starred characters include estimates.

	Tool Group	Scripting Group	P-value of T-test
Adam	0:34	0:47	0.032
Bob	0:19	0:35	0.005
Cathy*	0:27	0:57	0.001
Donna*	0:28	0:52	0.002
Total	1:49	3:12	0.001

Table 3. The average time (hh:mm) for each character, in hours and minutes, with the p-value of T-tests comparing the times. Starred characters include estimates.

Once the completion times for the incomplete characters (three Cathys and four Donnas) were estimated, the average time for each character was calculated. The Scripting Group took a longer time to complete than the Tool Group for each character. The results are significant at the 95% level for all characters, and for the total completion time (Table 3).

It is perhaps not surprising to observe a decrease in completion time from Adam to Bob (statistically significant for both groups). Although Bob has more complex behaviors, the time to implement Adam includes a steep learning curve for both groups, as participants were getting familiar with the tasks and solution environment. With Adam done, participants were able to use what they learned creating Adam's behavior to help them with Bob. Note that these completion times do not include the time participants were asked to examine a sample character before they started working on Adam.

Efficiency

Albert and Tullis (2013) define efficiency in the context of usability testing as "the ratio of the task completion rate to the mean time per task." A larger ratio implies that more participants were able to successfully complete a task per unit time. As expected, the Tool Group has a significantly higher efficiency than the Scripting Group (Table 4). Each entry in Table 4 represents the percentage of all character behaviors for all four characters that were successfully completed per minute, averaged over all participants in a group. On average, to complete all behaviors for all characters someone using the tool would require (100% behaviors) / (0.91% behaviors/minute) = 109.89 minutes. The first line excludes learning time spent examining the sample character, and the second line includes this time.

	Tool Group	Scripting Group	P-value of T-test
Excluding sample time	0.91	0.46	0.000
Including sample time	0.78	0.43	0.000

Table 4. The percent efficiency (completion/time).

Discussion

While it was necessary to implement a specific scheduling tool to conduct a user study, the architecture is general. A scheduling tool may need to be tailored to the game being designed. For example, in a stealth game, the scheduling of guards/targets needs to be more fine-grained. A scheduling tool could be constructed so that designers specify the duration and granularity of a schedule before being presented with a scheduling template graphical interface. Instead of presenting the objectives Eat, Sleep, Work, and Social, stealth game objectives could be used. For example, patrol an area, guard a set of portals (doors), check the security (lock state) of portals, etc. Each of these objectives can be satisfied by multiple roles. For example, patrolling an area can be done using random waypoints, a fixed path or patrolling subareas with frequencies based on current threat levels. One example of dynamic roles is a blocked patrol path where the guard may switch to random waypoints. Another example is when a designer only wishes to switch to threat-level patrols when the threats in some subareas go above a threshold.

Conclusion

In this paper, we showed that a tiered behavior architecture utilizing cyclic scheduling lets game designers populate behaviors for virtual characters more reliably and with higher efficiency than with manual scripting. Although in our user study we used a specific implementation of a scheduling tool, our goal was to introduce a general behavior architecture with powerful scheduling capabilities across a wide range of story-based games. One could argue that a usability study should be conducted that compares the experience of a group of designers using the high-level tool to the same group of designers using manual scripting. However, story-based game development has evolved so that the roles of designers and scripters (often referred to as technical designers) are usually fulfilled by different individuals. Therefore, a heads-up comparison of the two techniques by the same study participants would be of little use. Therefore, we instead looked at objective measures of reliability and efficiency for each approach as the key factors in comparing the two techniques. According to Albert and Tullis (2013), "in almost every situation, the faster a participant can complete a task, the better the experience." Therefore our efficiency measure provides an indirect measure of a positive designer experience.

Acknowledgements

This research was supported by GRAND Network of Centres of Excellence and NSERC. We thank the anonymous reviewers for their feedback. We also thank members of the BELIEVE research group at the University of Alberta and our colleagues in the GRAND network across Canada for fruitful collaborations.

Appendix: The Cyclic Scheduling Algorithm

```
schedule = []
itemList = sort(itemList) // longest obj block first
CyclicSchedule(itemList)

bool CyclicSchedule(itemList)
   item = pop(itemList)
   for each location L in schedule
      if satisfiable(L, item)
         schedule[L] = item
         returnValue = CyclicSchedule(itemList)
         if (returnValue == true)
            return true
         else
            schedule[L] = ""
   return false

bool satisfiable(location L, obj block Item)
   if (length of obj block > available obj blocks at L)
      return false
   else if (non-consecutive obj block requirement is
violated)
      return false
   else if (current number of assigned obj hours >
specified number of obj hours)
      return false
   else
      return true
```

References

Albert, W. and Tullis, T. 2013. Measuring the User Experience, Second Edition: Collecting, Analyzing, and Presenting Usability Metrics. Waltham, MA: Elsevier Inc.

Bakkes, S., Spronck, P., and van Lankveld, G. 2012. Player Behavioral Modeling for Video Games. Entertainment Computing, 3(3):71-79.

Becroft, D., Bassett, J., Mejía, A., Rich, C., Sidner, C. 2011. AIPaint: A Sketch-Based Behavior Tree Authoring Tool. In Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-11).

Bethesda Softworks LLC. 2011. The Elder Scrolls V: Skyrim. http://www.elderscrolls.com/skyrim.

BioWare. 2002. Neverwinter Nights. Infogrames/Atari. http://www.bioware.com/en/games/#game-neverwinter-nights.

BioWare. 2011. Dragon Age II. Electronic Arts. http://dragonage.bioware.com/da2.

BioWare. 2012. Mass Effect 3. Electronic Arts. http://masseffect.bioware.com/.

Champandard, A. 2012. Understanding the Second-Generation of Behavior Trees. http://aigamedev.com/insider/tutorial/second-generation-bt/.

Coman, A., Munoz-Avila, H. 2012. Plan-Based Character Diversity. In Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12).

Dragert, C., Kienzle, J., Verbrugge, C. 2012. Statechart-based Game AI in Practice. In Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12).

Epic Games. 2014. Blueprints Visual Scripting. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html.

Gomes, P. and Jhala, A. 2013. AI Authoring for Virtual Characters in Conflict. In Proceedings on the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-13).

Isla, D. 2005. Handling complexity in the Halo 2 AI. In Proceedings of the GDC 2005. Gamasutra.

Kadlec, R., Tóth, C., Cerny, M., Barták, R., Brom, C. 2012. Planning is the Game: Action Planning as a Design Tool and Game Mechanism. In Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12).

Kelly, J.P., Botea, A., Koenig, S. 2008. Offline Planning with Hierarchical Task Networks in Video Games. In Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-08).

Kleinberg, J., Tardos, E. 2005. Algorithm Design. Addison-Wesley.

Li, B., Thakkar, M., Wang, Y., Riedl, M. 2014. Data-Driven Alibi Story Telling for Social Believability. In Proceedings of the FDG 2014 Social Believability in Games Workshop.

Mateas, M., Stern, A. 2002. A Behavior Language for Story-based Believable Agents. Intelligent Systems, IEEE, 17(4):39-47.

Orkin, J. 2006. Three States and a Plan: The AI of F.E.A.R. Game Developers Conference (GDC-2006).

Pausch, R., Burnette, T, Capeheart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., White, J. 1995. Alice: Rapid Prototyping System for Virtual Reality. IEEE Computer Graphics and Applications.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai Y. 2009. Scratch: Programming for All. Communications of the ACM, 52(11):60-67.

Riedl, M. O., and Bulitko, V. 2013. Interactive narrative: An intelligent systems approach. AI Magazine 34(1).

Schenk, K., Lari, A., Church, M., Graves, E., Duncan J., Miller, R., Desai, N., Zhao, R., Szafron, D., Carbonaro, M., and Schaeffer, J. 2013. ScriptEase II: Platform Independent Story Creation Using High-Level Patterns. In Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-13).

Schwab, B. 2009. AI game engine programming. Boston, Mass.: Course Technology.

Seif El-Nasr, M., Drachen, A., and Canossa, A. 2013. Game Analytics: Maximizing the Value of Player Data. Springer.

Tomai, E., Flores, R. 2014. Adapting In-Game Agent Behavior by Observation of Players Using Learning Behavior Trees. In Proceedings of the 9th International Conference on the Foundations of Digital Games (FDG 2014).

Wißner, M., Kistler, F., and Andre, E. 2010. Level of Detail AI for Virtual Characters in Games and Simulation. In Proceedings of the Third International Conference on Motion in Games, 206-217.

Wright, I., Marshall, J. 2010. More AI in Less Processor Time: 'Egocentric' AI. Gamasutra. http://www.gamasutra.com/view/feature/131567/more_ai_in_less_processor_time_.php.

Zhao, R., Szafron D. 2014. Virtual Character Behavior Architecture using Cyclic Scheduling. In Proceedings of the 9th International Conference on the Foundations of Digital Games (FDG 2014).