Virtual Character Behavior Architecture using Cyclic Scheduling

Richard Zhao
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
T6G 2E8
rxzhao@ualberta.ca

Duane Szafron
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
T6G 2E8
dszafron@ualberta.ca

ABSTRACT

A story-based video game contains many characters. The majority are virtual characters controlled by artificial intelligence. In recent years, virtual character artificial intelligence has developed slower than other aspects of video games, such as graphics, mainly due to the cost of scripting complex and believable virtual characters. To tackle this bottleneck in content creation, this research proposes a new Tiered Behavior Architecture model for controlling the behaviors of virtual characters. For local scenes, techniques such as Behavior Capture with Hidden Markov Models, which has been evaluated by user studies that validated its success in generating fine-grained behaviors, can be used to fulfill the roles. At a larger scale, a hierarchical cyclic scheduler determines the general circumstances, schedules, and objectives of the virtual characters as well as the roles that will accomplish these objectives. This paper describes experiments and user studies that validate this model.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems – Games; K.8.0 [**Personal Computing**]: General – Games.

Keywords

behavior, artificial intelligence, scheduling.

1. INTRODUCTION

Most of the characters in a story-based video game are AI-controlled non-player characters (NPCs), who interact with the player character (PC), each other, and the environment. These NPCs are referred to as virtual characters in this paper. Over the years, while other areas of gaming technology, such as computer graphics, had large improvements, virtual character behaviors have improved relatively slowly. Creating natural-looking behaviors for virtual characters is not inexpensive. In a typical commercial story-based game, there are hundreds of virtual characters. Games such as the recently released The Elders Scroll V: Skyrim [3] deploy randomly generated virtual characters, and their numbers are only limited by the time a player spends in the game. Since manually scripting each virtual character requires extensive resources, most virtual characters in commercial games have simple and repetitive behaviors. Manual scripting has become a major bottleneck of content creation.

The simplicity of virtual character behaviors stands in staggering contrast with the realism conveyed by advanced graphics. Poor virtual character behaviors distract players from the immersion of the gaming experience. Rather than standing or wandering aimlessly, virtual characters should converse with other virtual characters and interact with game objects in realistic ways. They should also be able to react to unexpected events.

This research proposes a Tiered Behavior Architecture model, with a cyclic scheduler at the high level to determine the general objectives of the virtual characters and the roles that will satisfy these objectives. The low level can use techniques such as Behavior Capture with Hidden Markov Models [15] to generate fine-tuned behaviors. This research explores the possibility to design such a tiered architecture and to implement it in a way that balances the amount of work game designers need to do with the level of control they desire over the virtual characters. This architecture should also limit the amount of in-game computation to acceptable levels in a commercial game.

2. RELATED WORK

There are different methods of creating behaviors for virtual characters. Since manually scripting the behaviors is an expensive process, various specialized programming languages have been proposed to help with the scripting process, such as ABL [11].

Planning has been used in some instances. In the popular life simulation game series The Sims [12], virtual characters have basic motives, such as “hunger” and “social”, which drive their choice of actions. An advertisement is attached to a game object to define how interaction with the object can satisfy these motives. In contrast to story-based games with complex levels of interactions between virtual characters and the PC, the main objective of the virtual characters in The Sims is not to interact with the player, but to live out their lives in the world without specifically telling a story to the player. The Sims Stories is a set of expansions for the Sims, which adds a story mode, but the story mode consists of mainly hand-scripted sequences of events. Applying the behavior system in The Sims to a story-based game would therefore be difficult, as the amount of scripting required would increase substantially.

The story-oriented game, The Elder Scrolls IV: Oblivion [2], claimed to improve virtual character behavior with its Radiant AI system [9]. In Oblivion's pre-release interviews, designers claimed that the game's virtual characters are given goals that they must accomplish in a given day, and they must use their knowledge of

the game world to find ways to accomplish these goals (e.g., to get food, virtual characters can buy, hunt, or steal). However, the final release of the game features a much more restricted version of the behavior system. Some have claimed that the behavior of virtual characters has improved with the next game in the series, The Elder Scrolls V: Skyrim, with virtual characters having a larger set of plausible actions to perform [1]. However, most characters still follow a fixed schedule, which this paper will describe later.

Planning has also been used in first-person shooters. Games such as F.E.A.R. and S.T.A.L.K.E.R.: Shadow of Chernobyl, use goal-oriented action planning (GOAP) for behaviors [13]. Since these virtual characters in first-person shooters all have a relatively narrow combat-related role, the planning system only needs to work in a very specific situation. These planning approaches tend to provide better behaviors than Finite-State Machines (FSMs). Even so, complex behaviors such as team co-operation and flanking are not produced by the planner and have to be manually designed based on the map terrain.

Since state transitions for large Finite-State Machines and Hierarchical FSMs can become very hard to manage, Behavior Trees (BTs) were introduced. They are strictly hierarchical decision trees that allow for reuse and modularization. Champanard [5] presented Data-Oriented BTs and Event-Driven BTs as two different improvements upon traditional BTs. However, we want to support the creation of daily schedules for virtual characters, which are inherently cyclical. Therefore, we created an explicit temporal model, where the main focus is scheduling, since that is how real people organize their activities.

A Monte Carlo Planning approach has been used in RTS games [6]. The planning system was applied at a strategic high level, instead of at the individual action level. The authors claim that the impact of individual actions requires a very deep search to see the consequences of the actions and thus it is not worth the effort or time, while searching at a high level allows the program to envision the consequences of actions much further into the future. A similar idea of using high level "macro-operators" and its enhancements to speed up search is presented in the classical planning domain [6]. With virtual characters in a story-based game, the search space is relatively smaller since individual actions need not be as detailed as RTS units.

Researchers have taken the story-based game Oblivion and implemented a custom offline planning system [10]. The authors claim that real-time planning is computationally expensive and that plans have to be available in real time, while the CPU and memory resources available to game AI modules at runtime are limited. The paper does not give any details supporting this assertion, but from its experimental results, generating plans for 40 virtual characters for 4 in-game hours requires 12.32 seconds. This is probably not acceptable in real time, as in an actual game only a tiny fraction of the CPU resources is given to the AI system, as opposed to the full CPU resources utilized by the experiments.

3. BEHAVIOR ARCHITECTURE MODEL

In the context of a single local scene, Zhao and Szafron [15] proposed a data-driven technique for the creation and generation of behaviors for virtual characters. Behavior Capture is effective at producing characters that behave believably in a scene. With the Behavior Capture system, a designer takes control of a virtual character and produces traces of action sequences. The system then generalizes the traces and uses the trace data to train Hidden Markov-Models (HMMs) to guide the behaviors of characters in game.

While Behavior Capture is a valid control model at a small scale, there are problems with extending this technique to a larger scale. The Behavior Capture system is powerful at tracking a sequence of plausible actions, but it does not take into account the passing of time. Moreover, a daily schedule is usually cyclic, requiring the character to repeat certain actions each day, such as eating and sleeping. Also, at a high level, the only action a virtual character takes is to "go to scene X and play role R". In effect, there is a single possible action that is parameterized by two parameters X and R. This removes the advantage of Behavior Capture, which is the choice of multiple significantly different actions for each virtual character. Identifying the high-level decisions as a fundamentally different problem suggests that alternate approaches may produce better results.

In this paper, we propose a new *Tiered Behavior Architecture* model, as shown in Figure 1. The model divides a behavior controller into two parts, high level and low level. Low-level controllers are specific to each role, and techniques such as Behavior Capture can be used to produce believable behaviors at this role level.

To explain what the high-level controller should be capable of, we use an example. On a grand scale, a day-to-day routine of a virtual character usually consists of accomplishing several objectives: sleep, eat, work, and engage in social activities. The virtual character determines the scene (home, tavern, etc.) and the role, where these objectives will be satisfied. These decisions are affected little by the specific actions that the virtual character has to perform in a given role in a given scene. Therefore, it is natural to divide the behavior model into two levels.

The creation and evaluation of the high-level controller, which determines the daily schedule of a virtual character and takes the virtual character to different locations to assume different roles, is the focus of this paper. It is also important to note that the low level is modular and reusable in that the same high-level model can be used with many different scene-level models.

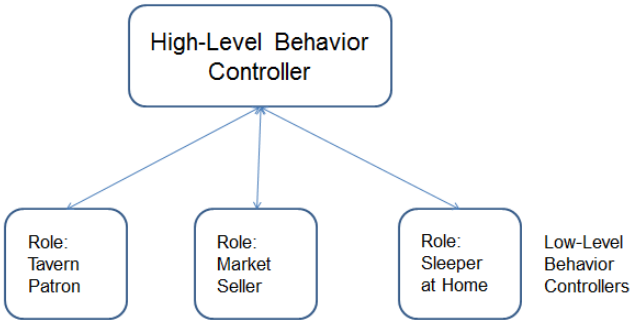

Figure 1. The Behavior Architecture for a virtual character, with the high level producing roles that lead to specific low level scenes. The high level contains a cyclic scheduler.

3.1 High-Level Control

The Tiered Behavior Architecture has a cyclic scheduling system at the high level determining the general objectives of the virtual characters and the roles that will satisfy these objectives.

A good daily schedule is usually cyclic. We want to give game designers a system that allows them to control the components of the schedule that they deem important. The intuition behind this system is that it should allow game designers to directly specify the important aspects of a good daily schedule and for the system to provide suggestions for the unimportant aspects.

To address the issue of the long running time of an online planning system, the cyclic scheduling system has two parts, an off-line scheduler, and an on-line selector. The more static a selection process can be, the less expensive the architecture will be at game time. The goal is to move the most expensive computations off-line.

In the off-line scheduler, game designers specify the duration of objectives (such as "sleep for 8 hours"), any specific important assignments of objectives to specific hours and they include a set of specific roles for each objective ("sleep at home", "sleep at an inn", etc. for "sleep"). Each *role* is an atomic element in the high level of the architecture, and is implemented in the low level by a behavior controller such as one generated by Behavior Capture.

The offline scheduler creates a 24-hour schedule (or however many hours a game world sets in a day) that consists of objectives. This is the most expensive part of behavior scheduling due to the complexity and the range of tigtness-looseness of the constraints that could be provided by the designer. The mapping of a designer's schedule concept to a particular list of objectives generally has many open variables so there are many ways it can be satisfied. If the designer wants to express some constraints such as consecutive blocks, but leave some choices open, then it becomes a planning problem that can take considerable time to run. The planning is also iterative so the designer can change constraints based on seeing generated plans. It is also possible that during this process, the designer could specify contraints that are unsatisfiable (Section 4 has an example). In this case, the architecture can inform the designer off-line. At game time, the selection of a role to an objective in the schedule involves only a filter of a small collection using the current game state (a few condition checks), a probability spin and then a selection from the filtered collection. This makes the in-game process fast.

Selectors are used to map objectives to roles, as well as to map schedules to objectives. We can also group multiple schedules into a circumstance, and use selectors to choose a schedule on-line. Multiple schedules can be useful in instances where a designer would like a virtural character to maintain several different daily routines, such as a weekday routine and a weekend routine. The circumstances can be used to pick different schedules for different acts in the game or to pick different schedules before and after a specific plot event. The Tiered Behavior Architecture is expressed using the hierarchy shown in Figure 2.

The hierarchy is made up of alternating data layers and selectors. Each data layer is a set or list of items, where each item is composed of items from the next layer. Formally, if we use L_0 to L_5 to denote the six layers, then

L_0: {Charcter$_1$, Character$_2$,… ,Character$_n$}
L_1: Character$_i$ = {Circumstance$_{i1}$, Circumstance$_{i2}$,… ,C$_{io}$}
L_2: Circumstance$_{ij}$ = {Schedule$_{ij1}$, Schedule$_{ij2}$, … ,S$_{ijp}$}
L_3: Schedule$_{ijk}$ = [Objective$_{ijk1}$, Objective$_{ijk2}$, … ,O$_{ijkq}$]
L_4: Objective$_{ijkl}$ = {Role$_{ijkl1}$, Role$_{ijkl2}$, … ,R$_{ijklr}$}
L_5: Role$_{ijklm}$= {Role$_{ijklm}$}

While the other layers are sets, a schedule is a list of objectives. Since a schedule is based on time, time serves as a natural ordering mechanism for the objectives. L_5 is represented as a singleton set so that all the layers can be viewed as collections for consistency. We use a selector to choose one item from each layer at any given time. The *selector* is a mapping σ_s from L_s to L_{s+1}.

$$\text{Selector: } \sigma_s (L_s) \rightarrow L_{s+1} \text{ for } 0 \leq s \leq 4$$

As a selector, a designer can pick any mapping that maps a collection to a single item. Examples of selectors include a *time selector*, which picks an item from a list based on a particular in-game time. An *event selector* picks an item based on events happening in the game world. A *probability selector* picks an item based on pre-set probabilities for each item. A *character selector* picks one virtual character from the set of virtual characters being controlled. In addition, each of these simple selectors can have a filter attached to it that filters the layer as part of the selection process, to signify the availability of the items in that layer at a particular instance in the game.

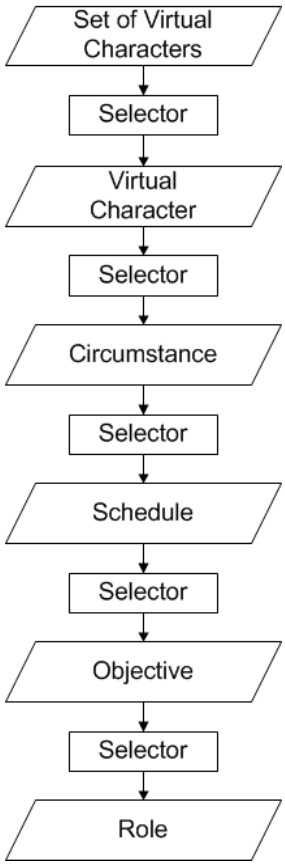

Figure 2. The Tiered Behavior Architecture model. Once a role is selected, generation of behaviors is passed onto low-level behavior controllers.

In our Tiered Behavior Architecture, the items in each data layers are generated offline statically, while the selectors are used at game time to dynamically pick the items. The static component of this mechanism can save considerable game time when the generation of objectives for each schedule is complex.

3.2 Mapping Model to Skyrim

Unfortunately many current story-based commercial games do not have virtual characters that go beyond walking between a set of waypoints. However, the recent games in the Elder Scrolls series provide an improved set of behaviors as they have implemented a Radiant AI system that allows their virtual characters to follow a daily schedule. The newest game in the series, The Elder Scrolls V: Skyrim, claims to have the most improved Radiant AI system.

We examine a typical virtual character in the Skyrim world. The character has a single circumstance, which contains a single schedule. The schedule contains one or more objectives, each with a single role (activity to satisfy that objective). What is interesting about the Skyrim behavior system is that when selecting an objective from a schedule, the selector considers several constraints. Each objective has its associated allowed time interval, as well as filter conditions that must be satisfied. Moreover, each objective has an implicit priority (by the order in which the objectives are listed), and the objective with the highest priority is picked by the selector, when more than one objective satisfied the time interval and filter condition constraints.

As Figure 3 shows, our Tiered Behavior Architecture is able to represent the behaviors in Skyrim. Figure 3 is not the only way of representing this behavior. Since in Skyrim, objectives in the single schedule can have overlapping time intervals, we can separate these objectives into several different schedules, each with no overlapping intervals. In this representation, the single circumstance has multiple schedules, which our Tiered Behavior Architecture can also represent. Moreover, our architecture is able to represent more complex behaviors not found in Skyrim. Consider the following example:

There are two characters, Adam and Eve. At the start of the story, Eve is happily married to Adam, and has two daily routines to follow, a weekday routine, and a weekend routine. On weekdays, Adam works at a market, Eve goes to school in the morning, and goes back home in the afternoon. On weekends, she goes for a walk in the town in the morning, and goes back home after.

However, as the story plays out, Adam passes away. When this happens, Eve is forced to change her routines. On weekdays, Eve takes Adam's job and works as a fish seller at the market. Eve goes to the market at 8am, and works for eight hours. She chooses a tavern to go to after work for two hours, and goes back home after. On weekends, she goes to church in the morning, and goes back home afterwards.

In this hypothetical example, our Tiered Behavior Architecture can express Eve's two very different behavior requirements using two circumstances, depending on what happened to Adam. In each circumstance, there are two schedules, one for weekdays, and one for weekends. For each objective in a schedule, there are one or more roles. As an example, when Eve is off-work and wants to have a drink, she can choose from a set of taverns to go to (a set of roles to satisfy her objective). With a probability selector, she can choose each role with a probability that the game designer specified beforehand. While the current behavior system in Skyrim is capable of expressing these behaviors by adding complex filter conditions and constraints to each single role and putting all roles into one schedule, our Tiered Behavior Architecture can express Eve's behaviors in a much simpler and more structured way.

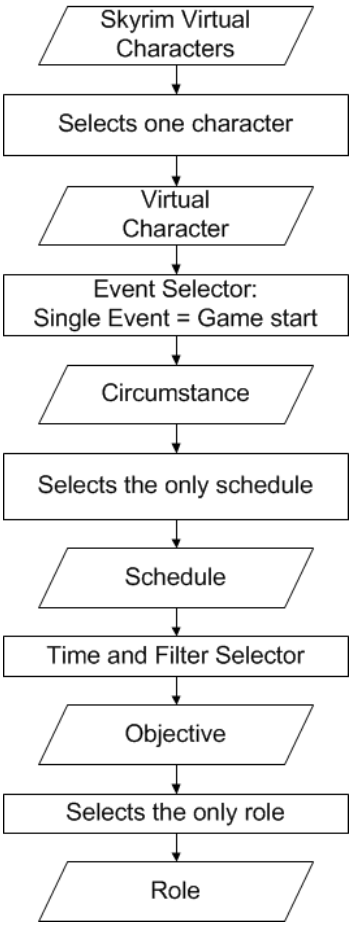

Figure 3. One way of mapping a virtual character in Skyrim to the Tiered Behavior Architecture.

Moreover, the current Skyrim system does not dynamically choose roles based on the most current state of the game world. For example, as a character walks home, if a tree falls and blocks the only path home, the character would simply stand in front of the tree. Path-finding failure is not the only mechanism to cause a role to fail. Events, such as a tavern being burned down, will disable roles such as the tavern patron as well.

Our Tiered Behavior Architecture checks the availability of a role in the game before assigning it to an objective. The mechanism to do this is to add a filter to the probability-based role selector that removes a role that is unavailable before spinning. Also, if the chosen role becomes unavailable in transition, a different role is selected immediately. Using the previous example, if a fallen tree blocks the only path home, to satisfy the "sleep" objective, the character would pick a different role, such as sleeping at a friend's house, or a tavern. If the tree falls as the character is walking home, the character will immediately switch roles upon seeing the tree. This eliminates certain bad behaviors that really hurt the playing experience, as our user study in Section 6 shows. It is the designer's responsibility to ensure that every objective can be satisfied by at least one default role, regardless of what happens at game time. An implementation could also include a "last resort" role, such as "idle" or "sit on the ground" that satisfies every objective, but is only selected if no other roles are available. This would prevent a crash but is little better than current practice of

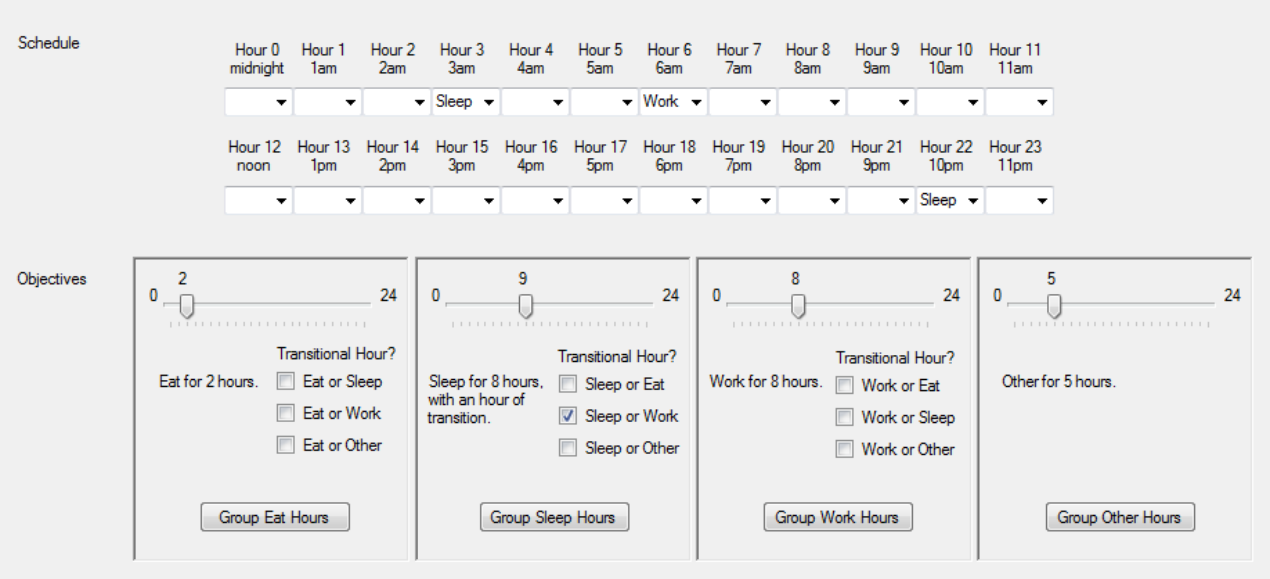

Figure 4. An implementation of the Behavior Architecture model, showing a part of the cyclic scheduling user interface.

having characters blocked. Therefore designers should pick more reasonable default roles for each objective, such as "sleep on the ground" for the sleep objective, and "stand and eat something" for the eat objective.

4. ARCHITECTURE IMPLEMENTATION

We have implemented the Tiered Behavior Architecture model in a prototype behavior generation tool. A game designer is able to first specify the circumstances for a virtual character at various levels of specificity. Within each circumstance, a 24-hour Timeline is utilized for each schedule. At each hour, the designer only chooses one of many objectives. For illustration purposes, we will present only four objectives: Eat, Sleep, Work, and Other. The "Other" objective includes social and other entertainment roles. These four objectives are chosen based on common daily schedules of virtual characters. There is nothing that prevents different objectives to be defined in a different virtual world.

The Timeline lets the designer specify constraints at specific hours as desired. This gives the designer total control over the behaviors of a virtual character. If the designer fills in all 24 hours with objectives, then the virtual character will do exactly as the designer specifies, with no emergent behavior in objectives, only probabilistic roles for each objective. If the designer wants even more control, only a single role needs to be provided for each objective. The underlying cyclic scheduler then fills in objectives on the Timeline based on designer constraints.

Figure 4 shows a screenshot of a part of the implementation. In a typical scenario, the designer will provide only constraints that are important to the story. For example, the constraints shown in Figure 4 are that the character must be asleep at hours 3 and 22 and must sleep for a total of 9 hours. The character must be working at hour 6, and must work for 8 hours. The checkmark beside "Sleep or Work" for the Sleep objective indicates that there is an hour of transition between Sleep and Work, so the character can choose to go to work (at most) an hour early on some days, creating some *stochasticity* in the daily schedules. In a separate settings window (Figure 5), the designer can specify how the objective hours should be grouped into consecutive blocks. In this example, the designer can specify that the 8 hours that the character works can be split into a 3-hour block and a 5-hour block, but that a non-working hour does not need to be inserted between the two blocks. This objective can be further split into smaller blocks, and the designer may require a non-working block between working blocks by checking that option.

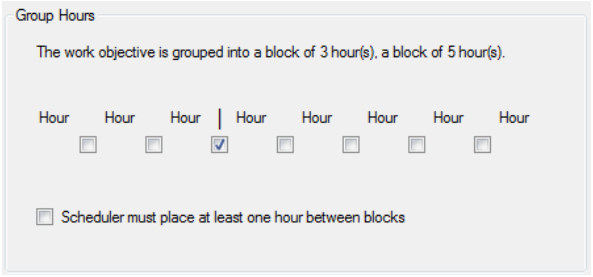

Figure 5. A Group Hours settings window.

Note that it is possible for the designer to express constraints that are inconsistent such as Sleep for 8 consecutive hours, but Sleep at hour 0 and at hour 9. In this case, the designer is informed offline by the cyclic scheduler that the scheduling constraints are unsatisfiable.

The designer is also able to specify how the role selector chooses the roles at run time. Under each objective, there can be a number of roles to satisfy the objective. For example, for the Eat objective, there can be a number of ways to fulfill this objective: Eat at home, Eat at a friend's place, Eat at a tavern, etc. The designer can define *multiple roles* per objective.

Due to the dynamism of story-based games, not all roles are available during the course of the game-play. If the virtual character has no friends available at a particular time, then "Eat at a friend's place" would not be a viable option. Similarly, if the only tavern in town burns down, then "Eat at a tavern" would not be a viable option after that incident. Even if an option is available, the designer may want to control how often the virtual character chooses it. The system architecture must support *dynamic roles* during game play. Typically the number of dynamic variables is small for each character, and the characters will share many of these dynamic variables. For example, a

character may check a faction list to find "friends", and a tavern's status variable may be checked to see if it is usable.

The system needs a way to give the designer direct control in how the role selector chooses a role given the available roles at the current game time, depending on what has happened in the game so far. In essence, the designer needs to have the ability to specify the percentage chance of choosing each role relative to every subset of available roles. The designer is able to make these choices in a separate window not shown in Figure 4.

5. SAMPLE RESULTS

Figure 4 is a typical example as specified by a designer. Again, the virtual character has to sleep at hours 3 and 22, sleep for 9 consecutive hours with one hour of either sleep or work, work at hour 6 for a total of 8 hours, and eat for 2 non-consecutive hours.

With these requirements, the scheduling system generated the schedule of objectives (done off-line) shown in Figure 6. We also show two consecutive days of potential roles (which would actually be generated on-line based on game context) that the virtual character could perform. The generation process took less than one second, but performing this task offline allows the designer to check the schedule and iterate if required. The roles in the two potential days shown in Figure 6 are for demonstration purposes only for this paper, since they would be dynamically generated on-line as the game is played, and different each time according to the current game context and player actions. Note that the dynamic selector must ensure consistency between hour 23 and hour 0 of the next day. If a single objective spans the day transition, the selector will map this spanning objective to the same role. For example, this will prevent a character from generating two different roles for the spanning objective, such as "sleep at inn" and "sleep at home".

Schedule:	Day 1:	Day 2:
0 : Sleep	Sleep at home	Sleep at inn
1 : Sleep	Sleep at home	Sleep at inn
2 : Sleep	Sleep at home	Sleep at inn
3 : Sleep	Sleep at home	Sleep at inn
4 : Sleep / Work	Sleep at home / Work at market (seller)	Sleep at inn / Work at Tavern (server)
5 : Work	Work at market (seller)	Work at Tavern (server)
6 : Work	Work at market (seller)	Work at Tavern (server)
7 : Work	Work at market (seller)	Work at Tavern (server)
8 : Work	Work at market (seller)	Work at Tavern (server)
9 : Work	Work at market (seller)	Work at Tavern (server)
10 : Other	Entertain on road (wanderer)	Entertain on road (wanderer)
11 : Other	Entertain on road (wanderer)	Entertain on road (wanderer)
12 : Other	Entertain on road (wanderer)	Entertain on road (wanderer)
13 : Other	Entertain on road (wanderer)	Entertain on road (wanderer)
14 : Other	Entertain on road (wanderer)	Entertain on road (wanderer)
15 : Eat	Eat at friend's	Eat at tavern
16 : Work	Work at City Gate	Work at City Gate
17 : Work	Work at City Gate	Work at City Gate
18 : Work	Work at City Gate	Work at City Gate
19 : Eat	Eat at friend's	Eat at tavern
20 : Sleep	Sleep at inn	Sleep at friend's
21 : Sleep	Sleep at inn	Sleep at friend's
22 : Sleep	Sleep at inn	Sleep at friend's
23 : Sleep	Sleep at inn	Sleep at friend's

Figure 6. An example of generated schedule with objectives (first column) and two consecutive days of potential roles (second and third columns).

6. STUDIES AND EXPERIMENTS

To verify the Tiered Behavior Architecture model, two studies (expressiveness and quality) and one experiment (performance) were designed.

6.1 Expressiveness

The first study asked the question: can the proposed architecture express the behaviors used by state-of-the-art virtual characters in current commercial games? To answer this question we picked the game with arguably the most mature daily schedules for its virtual characters, Skyrim.

We examined the AI of Skyrim, specifically at how the virtual characters behave on a daily schedule. We have replicated the behaviors with our proposed Tiered Behavior Architecture. Since there are an unlimited number of virtual characters in the game (some virtual characters are dynamically generated), we looked at only named characters that persisted in the game world. An inspection of all of the named characters in Skyrim [14] reveals that all of their daily schedules can be generated by this architecture. However, for our study we selected the eighty-five characters from a sample large city, Solitude. These include characters from many different professions in the game. A scan of the code using the Skyrim Creation Kit for each of these characters confirms that our Tiered Behavior Architecture is able to express the behaviors of all the Solitude virtual characters.

Here is an example of one of the most complex behaviors in Solitude (and they are comparable to the most complex behaviors elsewhere in Skyrim). For the character Greta, if her husband Addvar is dead, she will go to the market at 6am and stay for 14 hours selling goods before going back home for the night. Otherwise, if the player completed the quest "Return to Grace", she will go to a temple at 6am and stay for 9 hours. At 3pm, she will go wander around near a well for 3 hours before going back home. If the above quest is not completed, she will sleep until 8am, do some housework until 3pm, and then go wander around the well as before. These can be expressed with three different schedules in our Tiered Behavior Architecture, managed by two circumstances, "Addvar is dead" and "Return to Grace is complete".

Figure 7. Greta, the main character in each set of videos in the user study, is leaving her house in this screenshot.

6.2 Quality of Behaviors

This second study was a user study that asked the question: are the behaviors created by our proposed architecture a viable alternative to typical commercial game virtual character behaviors? In this study, participants were asked to watch six sets of game videos (in random order) in which the behaviors of the observed virtual character were generated by different methods. Each set of videos focused on the daily lives of one observed character over three days. All six observed characters look identical (to Greta in Figure 7), and they live in identical world settings. To help the participants focus on the high-level behaviors, a single city populated by virtual characters was presented (Solitude in Skyrim). As the observed character walks into local scenes (for example, a tavern building), a fade-out/fade-in effect was used to show only the transitions to and from the local scene. Activities inside local scenes were not presented.

After watching the main characters, participants were asked to rank and rate the characters according to believability of behaviors. Some demographic information was also gathered. The six behavior variations are listed in Table 1. The difference between a fixed schedule and a stochastic schedule is that a stochastic schedule supports a maximum plus or minus one hour duration for each objective. Dynamic roles imply that the roles are constantly checked for validity and dynamically switched if one becomes unsatisfiable. SS and MS behaviors are default Skyrim behaviors. MSSMDR showcases the most complex capabilities of the Tiered Behavior Architecture model.

Table 1. The six behavior variations.

Behavior	Details
SS	Fixed Single Schedule, with Single Roles
SSS	Stochastic Single Schedule, with Single Roles
MS	Fixed Multiple Schedules, with Single Roles
MSS	Stochastic Multiple Schedules, with Single Roles
MSSMR	Stochastic Multiple Schedules, with Fixed Multiple Roles
MSSMDR	Stochastic Multiple Schedules, with Multiple Dynamic Roles

Here is a description of the actual behaviors:

SS – Greta goes from her house to her market stall at 6am. She goes to the "Angeline's Aromatics" tavern at 3pm, then goes home at 6pm. The schedule is the same for three days.

SSS – This schedule is the same as SS except that the times of transition are stochastic, meaning that each time she goes to a place, she can leave any time (up to one hour) earlier than specified in the schedule.

MS – Here Greta has the same schedule as SS on the first two days and has a different schedule on day 3, where she goes from her house to church at 6am. She goes to the same tavern at 12 noon, then goes home at 6pm.

MSS – This schedule is the same as MS except that the times of transition are stochastic by one hour.

MSSMR – This schedule extends the MSS schedule with multiple roles for each objective. Instead of going to only the "Angeline's Aromatics" tavern, Greta chooses between this tavern and a "Bits and Pieces" tavern. Instead of working only at the market, she chooses between the market job and a bard job.

MSSMDR – This schedule extends the MSSMR schedule with dynamic roles, so that Greta is able to dynamically switch roles to go to a friend's house for the night upon seeing that the road to her own house is rendered inaccessible by fallen trees.

Note that Greta's default behavior in Skyrim can be represented as an MS behavior, except that instead of changing schedules according to the day of the week she changes schedules after some game events, such as when her husband is dead. Most Skyrim characters have SS behaviors, but some have MS behaviors that depend on game events.

Our study had 80 participants, who were undergraduate students taking a first year psychology class. There were 50 females and 30 males. Of these, 9 of the females were gamers and 41 were non-gamers, while 18 of the males were gamers and 12 were non-gamers. A gamer in this context is defined as someone who plays story-based video games at least once a week.

The resulting averages of rankings and ratings are presented in Table 2. Ranking scores are from 1 to 6: for each participant response, the highest ranked behavior received a score of 6, the second highest ranked received a score of 5, etc. Rating scores range from 1 to 4, with 4 being highest. The trends of rankings and ratings are consistent with each other, with MSSMDR as the best, indicating that stochasticity, multiple schedules, and multiple dynamic roles together make the best behaviors.

Table 2. Average ratings and rankings of the behaviors.

Behavior	Average Ranking Score	Average Rating Score
SS	2.54	2.01
SSS	2.64	2.05
MS	3.26	2.43
MSS	3.40	2.54
MSSMR	3.90	2.55
MSSMDR	5.26	3.35

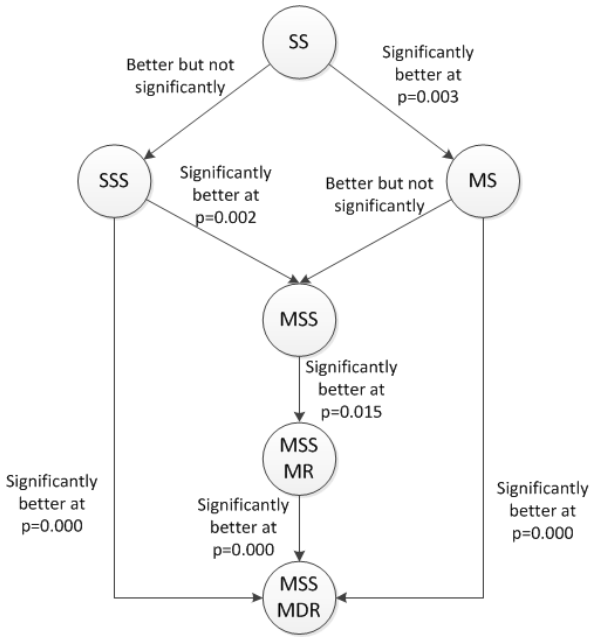

Figure 8. Statistical significance diagram comparing the rankings of the six behaviors with 95% confidence.

ANOVA shows that there are statistically significant differences in the results at 95% statistical confidence (p-value < 0.05). Paired T-tests at a confidence of 95% indicate that MSSMDR is better than each of the other alternatives. Figure 8 is a graphical illustration of the results presented in Table 3. Starting from SS, adding a multiple-schedule to get to MS is significantly better. Adding stochasticity to either SS or MS is better, but not significantly. One potential reason for some people not perceiving stochastic schedules as more realistic could be the belief that

individuals usually go to work at the same time every single day. This is consistent with some feedback we collected at the end of the user study, in which there were comments such as "if the character does the same thing at the same time, I rather consider it more natural than otherwise."

Finally, adding both the multiple roles and dynamic roles produce significantly better results. The raw p-values of the t-tests are shown in Tables 3 and 4.

Table 3. T-tests of ranking scores, showing the p-values.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	0.000	0.000	0.002	0.003	0.314
SSS	0.000	0.000	0.002	0.006	
MS	0.000	0.005	0.233		
MSS	0.000	0.015			
MSSMR	0.000				

Table 4. T-tests of rating scores, showing the p-values.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	0.000	0.000	0.000	0.000	0.347
SSS	0.000	0.000	0.000	0.000	
MS	0.000	0.092	0.053		
MSS	0.000	0.448			
MSSMR	0.000				

6.3 Performance of Architecture

We wanted to determine whether the overhead of dynamic scheduling and the new behaviors introduced would perceptibly reduce frame rates. Based on the studies, we followed the Greta character throughout the city of Solitude. On a high-end gaming computer the frame rates usually varied from 59 to 60 FPS whether our Tiered Behavior Architecture was enabled or not. Other factors affected the frame rate more than the architecture. For example, whether the architecture was enabled or not, the frame rate dropped to 52 FPS when children were playing nearby. On a more modest computer the frame rates varied from 9 to 15 FPS in both cases. On the high-end computer an ENB [7] was used to measure frame rate, and on the low-end computer FRAPS [8] was used. Our Tiered Behavior Architecture did not affect the performance in a measurable way on either computer.

7. FUTURE WORK

While we can show that the Tiered Behavior Architecture model is expressive and the behaviors generated from the model produce the best behaviors among alternatives, we would like to examine another aspect: would a game designer find the implementation of our proposed architecture easy to use and powerful enough to specify the kinds of behaviors needed, compared to existing methods for creating behaviors of virtual characters? As future work, we would like to conduct user studies that compare the use of our architecture tools with manual scripting and other behavior generation methods.

8. CONCLUSIONS

In this paper we propose a new architecture model and its implementation for game designers that allows them to create behaviors for virtual characters in story-based games, without having to learn programming skills. The model allows for multiple circumstances, schedules, and objectives, as well as stochasticity in the schedules, and dynamically chosen roles to satisfy objectives. We devised a set of user studies and experiments to validate the expressiveness, quality and performance of the proposed Tiered Behavior Architecture. Note that we focused on a particular group of selectors and filters, but the model can use any kind of a selector that selects an element from a collection and any kinds of filters that act on collections.

9. ACKNOWLEDGMENTS

This research was supported by GRAND NCE, the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta's Informatics Circle of Research Excellence (iCORE). We thank the anonymous reviewers for their feedback. We also thank members of the BELIEVE research team at the University of Alberta.

10. REFERENCES

[1] Bertz, M. 2011. *The Technology Behind The Elder Scrolls V: Skyrim*. http://www.gameinformer.com/games/the_elder_scrolls_v_skyrim/b/xbox360/archive/2011/01/17/the-technology-behind-elder-scrolls-v-skyrim.aspx

[2] Bethesda Softworks LLC. *The Elder Scrolls IV: Oblivion*, 2006. [Video Game].

[3] Bethesda Softworks LLC. *The Elder Scrolls V: Skyrim*, 2011. [Video Game].

[4] Botea, A, Müller, M., Schaeffer, J. 2007. Fast planning with iterative macros. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, India, 1828-1833.

[5] Champanard, A. 2012. *Understanding the Second-Generation of Behavior Trees*. http://aigamedev.com/insider/tutorial/second-generation-bt/

[6] Chung, M. Buro, M. and Schaeffer, J. 2005. Monte Carlo Planning in RTS games. In Simon Lucas and Graham Kendall, editors, *Proceedings of the IEEE Symposium on Computational Intelligence and Games*.

[7] ENB. 2014. http://enbdev.com/

[8] FRAPS. 2014. http://www.fraps.com/

[9] Kelly, J. P., Botea, A., and Koenig, S. 2007. Planning with Hierarchical Task Networks in Video Games. In *Proceedings of the ICAPS-07 Workshop on Planning in Games*.

[10] Kelly, J. P., Botea, A., and Koenig, S. 2008. Offline Planning with Hierarchical Task Networks in Video Games. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, Stanford, CA: AAAI Press, 60-65.

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

[12] Maxis. *The Sims*, 2000. [Video Game].

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

[14] Skyrim:People. 2014. The Unofficial Elder Scrolls Pages. http://uesp.net/wiki/Skyrim:People

[15] Zhao, R, Szafron, D. 2011. Generating Believable Virtual Characters Using Behavior Capture and Hidden Markov Models. *Advances in Computer Games 13 Conference*, Tilburg, The Netherlands. Lecture Notes in Computer Science 7168, Springer 2012, 342-353.