Game-based Data Capture for Player Metrics

Aline Normoyle and **John Drake**
University of Pennsylvania
3451 Walnut Street
Philadelphia, PA 19104

Maxim Likhachev
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, 15213

Alla Safonova
Disney Research Pittsburgh
4720 Forbes Avenue
Pittsburgh, PA 15213

Abstract

Player metrics are an invaluable resource for game designers and QA analysts who wish to understand players, monitor and improve game play, and test design hypotheses. Usually such metrics are collected in a straightforward manner by passively recording players; however, such an approach has several potential drawbacks. First, passive recording might fail to record metrics which correspond to an infrequent player behavior. Secondly, passive recording can be a costly, laborious, and memory intensive process, even with the aid of tools. In this paper, we explore the potential for an *active* approach to player metric collection which strives to collect data more efficiently, and thus with less cost. We use an online, iterative approach which models the relationship between player metrics and in-game situations probabilistically using a Markov Decision Process (MDP) and solves it for the best game configurations to run. To analyze the benefits and limitations of this approach, we implemented a system, called GAMELAB, for recording player metrics in Second Life.

Introduction

The collection of player data is an integral part of game development, particularly for design and playtesting. Many modern approaches to playtesting involve the collection of numerous player metrics (such as the number of times a player uses a particular weapon or percentages of time a player spends in different activities) which are then analyzed by designers and QA specialists (Kennerly 2003; DeRosa 2007; Luban 2009; Ambinder 2009). Even after a game is released, such data is useful for identifying cheating, monitoring the economy, and designing new downloadable content (Kennerly 2003). The most straightforward way to collect such metrics is to passively record players, but such an approach has potential disadvantages. First, a naive approach might fail to capture desired data when it corresponds to an infrequent player behavior. Secondly, the acquisition of this data from playtesters can be a costly, laborious, and memory intensive process, even with the aid of tools.

Thus, in this paper, we explore an *active* approach for the acquisition of metrics for describing player behaviors. Our approach exploits the inherently malleable nature of games to target data collection to those metrics which are most uncertain and would therefore benefit most from having more data. The approach is based on the idea that most modern games consist of numerous scenarios (such as levels, missions, quests, or mini-games) in which players have opportunities to display different sets of behaviors. By modeling the relationship between game scenarios and the types of metrics which can be collected from each, we can configure game scenarios based on the metrics we want to collect. In some cases, the relationship between game scenario and metric is trivial (for example, a player might only be able to swim in environments with deep water); however, in cases where players have choices, we cannot assume a simple relationship. Player behaviors can vary significantly based on skill level, interest, and personality; and designers often do not have a good *a priori* grasp of how players will play.

The approach in this paper employs an iterative, online method for data collection. We create initial estimates for each metric by running each of our game scenarios once. This information is also used to initialize a model of which games provide samples for each metric. This model is formulated as a Markov Decision Process (explained in detail later). From this MDP, we can compute a locally optimal policy which recommends the best game scenario to run next. We then run this scenario, extract data for our metrics, update our MDP model, and then compute a new recommendation.

The approach in this paper is inspired by ideas from optimal experiment design (Chaloner and Verdinelli 1995; Settles 2012), which aims to target experiments towards collecting data with the highest uncertainty, where uncertainty is usually defined in terms of variance. In these setups, the goal of running additional experiments is to reduce the uncertainty in the data. Similarly, the goal of our setup is to select games for the purposes of reducing uncertainty in our metrics, where we measure uncertainty in terms of confidence intervals. Our approach for making game recommendations is similar to (Shani, Brafman, and Heckerman 2002), which used a Markov Decision Process (MDP) to model consumer choices as a sequential optimization problem. This MDP was then solved to produce a policy for choosing product advertisements. Using a different MDP formulation, we model the numbers of samples collected

from different game scenarios. The advantages of such an approach over passive data collection is that we can focus our collection efforts on metrics with high uncertainty (measured in terms of confidence intervals, variance, or fewest number of samples). Specifically, if a metric corresponds to a rare behavior, the model will identify how best to obtain samples for it. Overall, the entire set of metrics can be potentially collected with greater accuracy while running fewer games. This could be a huge benefit when running more games might mean missing deadlines, spending more money, or requiring massive data storage.

Most systems for metric collection implement some degree of automation. A typical workflow might consist of a human analyst who specifies the desired metrics and then an automated system which then records data and generates corresponding reports and graphs (perhaps using a fixed schedule (Kennerly 2003) or game scenarios chosen a priori (Luban 2009)) . The technique in this paper enhances data collection by automating the choice of game scenarios; however, our method does not aid in analyzing the collected results, nor with determining which quantities are most important to measure.

To analyze the benefits and limitations of this approach, we implemented a set of mini-games in Second Life from which we culled five metrics: the distances between people standing in either narrow or wide spaces; the timing of lane crossings for slow and fast traffic; and the choice of whether to use a health kit based on health level. For data collection, we implemented a system (called GAMELAB) which can run the mini-games, collect data, and then extract these metrics. With this system, we ran 70 games and collected data from 179 international participants over a period of 5 weeks.

Related Work

For games, data collection is usually focused on the collection of player data. Such data can be used to train Bots and NPCs (Reeder et al. 2008; Priesterjahn et al. 2007; McPartland and Gallagher 2008; Bauckhage et al. 2007; Sharifi, Zhao, and Szafron 2010; Tastan and Sukthankar 2011) to customize gameplay to a player's ability (Robin Hunicke 2004; Gilleade and Dix 2004; Ponsen et al. 2005; Charles et al. 2005) or to customize gameplay to a player's preferences (Shaker, Yannakakis, and Togelius 2010; Eunyoung Ha and Lester 2012).

However, the focus of this paper is the use of player data for playtesting and design, where it is particularly useful for identifying problems and improving gameplay (Kennerly 2003; DeRosa 2007; Luban 2009; Ambinder 2009). Playtesting research has focused on more effective development and test cycles (Medlock et al. 2002), better automated methods for logging, tracking, and reporting player data (Kennerly 2003; DeRosa 2007; Kim et al. 2008), automated methods for effectively training game agents (Alexander 2002), novel ways to organize and visualize the massive amounts of data collected with logging (Chittaro, Ranon, and Ieronutti 2006; Andersen et al. 2010; Moura, el Nasr, and Shaw 2011), and better models for understanding collected player data (Tychsen and Canossa 2008; Drachen and Canossa 2009; Mahlmann et al. 2010). In this work, we aim to enhance current automated logging and tracking methods, which collect data passively, so that playtesters can collect metrics more quickly. To our knowledge, we are the first to explore an active game recommendation approach for such a goal.

The goal of most active learning methods is to reduce manual effort and cost (for good overviews, please see (Chaloner and Verdinelli 1995; Settles 2012)). Such a goal might correspond to reducing the number of samples that a human needs to label (for example, in domains where labeling is time-consuming such as text classification (Roy and Mccallum 2001; Hoi, Jin, and Lyu 2006), image classification and retrieval (Tong and Chang 2001; Hoi et al. 2006), and speech recognition (Tur, Hakkani-Tur, and Schapire 2005; Riccardi and Hakkani-Tur 2005)) or it might correspond to reducing the number of experiments to run. For example, (King et al. 2004) describes a "robot-scientist" that uses an active learning strategy to isolate gene functions in the yeast *Saccharomyces cerevisiae* with minimal cost. (Krishnamurthy 2002) describes an active learning algorithm for dynamically computing a schedule for sensors using a Hidden Markov Model. (Cooper, Hertzmann, and Popović 2007) uses an active learning approach for the creation of character controllers that aims to reduce the amount of motion data that needs to be captured. In this work, we investigate whether an active learning approach might reduce time and cost for playtesting and metric collection.

Overview

For data collection, we built an online framework for autonomously collecting player metrics, which we call GAMELAB (Figure 1). Testers input a set of desired metrics and a set of game configurations. GAMELAB then iteratively collects data to improve the estimates for each metric, specifically, it

1. computes a game schedule for recommending the next game to run
2. runs the recommended game and extracts new samples
3. updates its model and metric estimates, and repeats

In the next two sections, we will describe how we model the relationship between metrics and games as an MDP and then use this MDP to compute a game schedule.

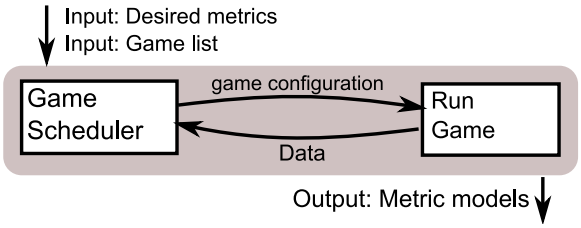


Figure 1: *GAMELAB Overview. Testers input a set of metrics and a set of game scenarios. The system outputs distributions for the metrics. At each iteration, the system computes a game schedule which is then used to select a game for the next testing activity. Samples are extracted after each game, metric models updated, and a new schedule is computed.*

Game Scheduler

We envision a scenario in which an analyst or designer wishes to collect a set of player metrics. Such metrics could aid testing a hypothesis or correspond to game parameters which require tuning. Our goal is to collect these metrics efficiently by running as few game sessions as possible.

We formulate this problem as follows. First, let us define the M metrics we wish to collect as the set $\mathcal{P}$ (for parameters). Secondly, let us define the set of possible game configurations as $\mathcal{G} : \mathbb{T} \times \mathbb{E} \times \mathbb{S}$, where $\mathbb{T}$ enumerates game types, $\mathbb{E}$ environment configurations, and $\mathbb{S}$ game type settings. $\mathcal{G}$ formalizes how we can tune our given game set for data collection. For example, tweaking the game environment or the number of hazards might affect the numbers of samples we receive for our game metrics. Let us assume that we have K games to choose from, where for each game $g_k \in \mathcal{G}$ we have an estimate for the likelihood $P_k(p_m)$ of obtaining p_m number of samples for a parameter m, $1 \leq m \leq M$.

In our current system, we model the decision of which game to run next using a Markov Decision Process, or MDP. Recall that an MDP is a 4-tuple, consisting of states, actions, transition probabilities, and rewards. In our formulation, states s_i correspond to the numbers of samples we have collected so far for each metric.

$$s_i = \{p_1, p_2, \ldots, p_m, \ldots, p_M\}$$

State nodes are organized in a simple tree structure (Figure 2). Each level in the tree corresponds to a choice to be made after each game. The outcome of this choice determines which node in the next level we transition to. Because the depth of the tree determines how far into the future we consider outcomes, we also refer to the tree depth as the amount of lookahead employed by the schedule. Note that some states will be repeated between tree levels, but that this structure allows us to compute a corresponding policy very efficiently (linear in the number of states).

Actions g_k correspond to running a game. Possible action outcomes consist of the numbers of samples we might receive when we run g_k. We can estimate the probabilities for each outcome based on the data we have received so far and in some cases, a priori knowledge. Suppose $\lambda_{k,m}$ is the average number of samples that we collect from game g_k for the m-th parameter. As we run games, we update this estimate for $\lambda_{k,m}$.

We can also estimate our transition probabilities based on $\lambda_{k,m}$. Because we are dealing with counts, we can estimate the probability of receiving samples for the m-th parameter with a Poisson distribution with mean $\lambda_{k,m}$. An experiment outcome corresponds to the number of samples we receive for each parameter. Thus, if we assume that the acquisition of samples is independent for each parameter, then the probability of a particular outcome will be the product of the probabilities of receiving samples for each parameter. As a technical note, we split the possible counts of samples into bins to keep the number of transitions and states tractable. Put together, the probability of a transition from state s_i to state s_j is given by

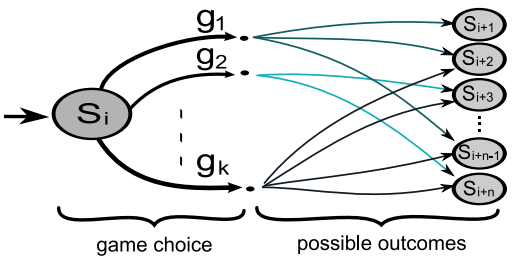


Figure 2: *Game scheduling. States represent the numbers of samples collected so far for each metric. Running a game g_k provides additional samples, leading to a new state in the next level of the MDP.*

$$P_k(s_i, s_j) = \prod_{m=1}^{M} P_k(\theta_m^a \leq p_m < \theta_m^b \mid \lambda_{k,m})$$

where each θ_m corresponds to a range of counts $[\theta_m^a, \theta_m^b)$ for the m-th metric, and where $P_k(\theta_m^a \leq p_m < \theta_m^b \mid \lambda_{k,m})$ is the probability that the number of samples p_m will lie in the interval θ_m if we run the game g_k. Note that in some cases, a game g_k is incapable of providing samples for a metric. In this case, we define a special case corresponding to zero samples such that

$$P_k(\theta_m^a \leq p_m < \theta_m^b \mid \lambda_{k,m}) = \begin{cases} 1 & \text{if } \theta_m^a = \theta_m^b = 0; \\ 0 & \text{otherwise.} \end{cases}$$

The optimal policy, in the discounted expected cost sense, $\pi(s_i)$ consists of the best game to run when in state s_i, given by

$$\pi(s_i) = \arg\max_{g_k} \sum_{s_j} P_k(s_i, s_j)(R_k(s_i, s_j) + \gamma v(s_j)) \quad (1)$$

where $v(s)$ is defined recursively as

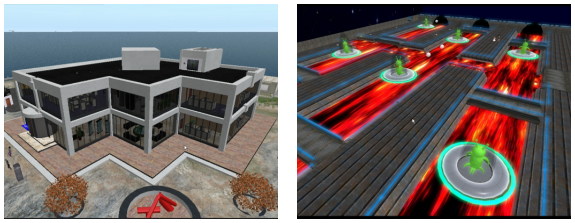
$$v(s) = \sum_{s_j} P_{\pi(s)}(s, s_j)(R_{\pi(s)}(s, s_j) + \gamma v(s_j)) \quad (2)$$

with $v(s) = 0$ when s is a leaf node.

Above, s_j is the resulting number of samples after running a game g_k, $0 < \gamma < 1$ is a discount factor, P is the probability of transitioning from s_i to s_j, and R is the estimated reward for transitioning from s_i to s_j.

The optimal policy π is solved by iterating through each tree node once, evaluating $v(s_i)$ at each state on a level-to-level basis according to equations 1-2, starting at the leaf level and working up to the root. Once we have $\pi(s_i)$, we simply need to look up which action to perform based on our current state s_i.

Note that to initialize the schedule, we rely on estimates of the numbers of samples we are likely to receive for each game g_k and if these initial estimates are non-representative of the true values, the resulting policy will not be a good schedule. To guard against poor initialization, the system might periodically run a random game choice.



(a) (b)

Figure 3: *Environments. 3(a) shows the exterior of our office environment. Players have access to both floors and the roof. 3(b) shows the interior of the alien teleportation facility. Saucers appear at spawn points and then travel along the red lanes of lava. Players can traverse these lanes at crosswalks.*

Reward Function

The choice of our reward function R will affect how the schedule prioritizes metrics. A simple choice might be a reward based purely on sample counts. The advantage of such a choice is that no assumptions need to be made about our metrics. Alternatively, a reward based on confidence intervals assumes an underlying model for each metric. For example, when estimating a mean, we might assume a normal distribution for it. This assumption allows us to estimate confidence intervals easily as a function of the number of samples and their variance.

In our experiments, we fit a Gamma distribution $\Gamma(\alpha, \beta)$, chosen because it fit our initial 14 experiment set well, whose data was noisy and showed skew. A Normal distribution, or any other, might alternatively be used. Given this, we define our reward function as

$$R(s_i, s_j) = \begin{cases} 0 & \text{if } s_j \text{ is a leaf node;} \\ \sum_{m=1}^{M} [\text{CI}(p_m) - \text{CI}(p_m + \theta_m)] & \text{otherwise.} \end{cases}$$

where θ_m is the additional samples we receive if we transition to s_j, and CI denotes a confidence interval as a function of the number of samples. (θ_m is assumed based on the start θ_m^a of the interval $[\theta_m^a, \theta_m^b)$.) We compute $\text{CI}(p_m + \theta_m)$ assuming the variance of samples remains constant. The advantage to assuming an underlying model is that we avoid collecting more data unnecessarily. For example, a game might provide lots of samples for a metric, but if that metric already has a small confidence interval, obtaining more samples is not necessary. However, if our underlying model does not fit the data well, the policy will not perform well.

Proof of Concept

To analyze our method, we implemented a set of mini-games in Second Life from which we culled five metrics: the distances between people standing in either narrow or wide spaces; the timing of lane crossings for slow and fast traffic; and the choice of whether to use a health kit based on health level. To collect data, we ran our framework online over the course of five weeks, approximately 23 hours in total. During this time, we ran 70 games which attracted 179 participants worldwide, 80 of which corresponded to unique people (determined by avatar ID). Games consisted of scavenger hunts within two different environments: a building

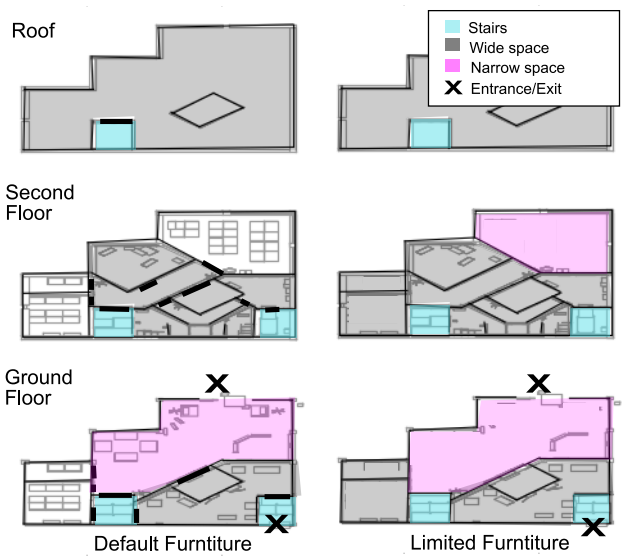


Figure 4: *Office environment. Floors are shown from bottom to top. The left map shows the categorization of wide and narrow spaces used for behavior detection when furniture is present. The right shows the categorizations when furniture is hidden. The presence of furniture in some of the rooms affects steering; thus, we ignore these rooms when collecting standing samples.*

with hazards and an orbital platform filled with lanes of space traffic.

Each environment contained switches for changing the amounts of wide and narrow spaces (see Figure 4 and Figure 5). For the office building, we can block rooms, toggle the presence of furniture, and enable hazards, such as lasers and toxic waste. Locking rooms and hiding furniture changes the numbers of narrow and wide spaces during the hunt (see Figure 4). Enabling hazards allows us to collect health kit data. For the orbital platform, where we collect crossing behaviors, we can specify the saucer speed, currently 8 m/s for slow traffic and 12 m/s for fast traffic.

Experiments

Participants

Participants were a diverse group of players recruited from the Second Life community. Of the 63 participants who filled out our questionnaire, we had 39% male, and 61% female; 53% reported being between the ages of 18-25, 29% between the ages of 25-45, 16% between the ages of 45-65, and 2% over the age of 65. 75% of participants reported living in the United States. The remaining 25% hailed from Italy, Germany, Colombia, Argentina, Chile, Brazil, the United Kingdom, Canada, and Australia. Participants won in-game objects and Linden dollars for playing.

Data Collection

We ran games in batches of three as part of a single hour-long activity, advertised to Second Life residents through

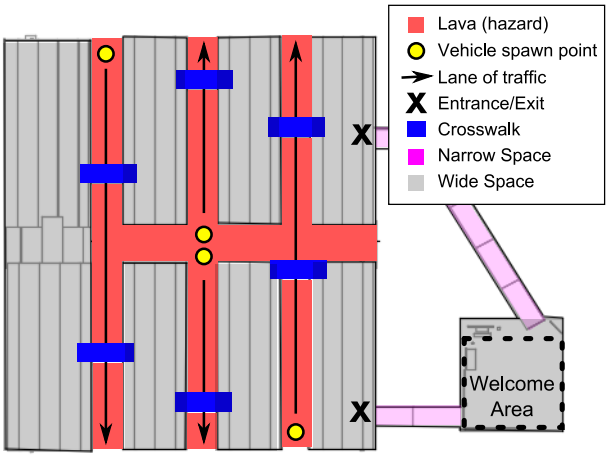


Figure 5: *Alien teleportation facility. Red indicates lanes filled with lava over which saucers fly. Blue shows the locations of crosswalks. Saucer spawn points and directions of traffic are also shown.*

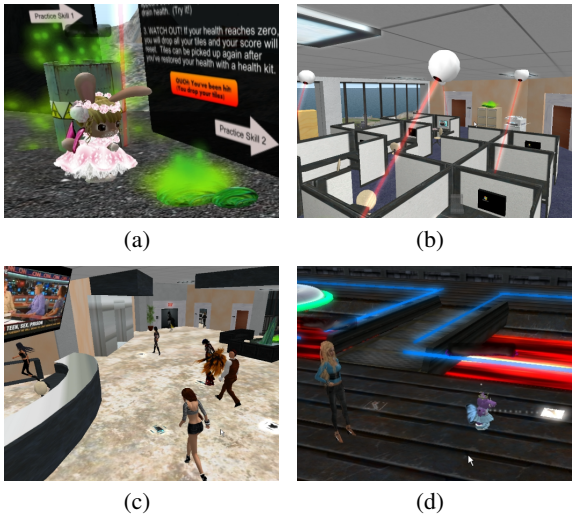


Figure 6: *Participants and gameplay. In 6(a), a tiny participant browses our tutorial area. 6(b) shows lasers in one of the cubical areas. 6(c) shows participants racing through the main foyer of the office building to collect tiles. 6(d) shows tile collection during the traffic game.*

the events calendar. Figure 6 shows screenshots taken during several of our activities.

A Second Life Bot automatically welcomed people and directed them to equip a Heads-up Display (HUD). The HUD showed rules, an IRB consent form, and game state, such as the amount of time left for the game. A tutorial area allowed participants to practice using our game widgets. At the end of each activity, the system gave participants a questionnaire via instant message, calculated final scores, and then sent prizes and Linden dollars based on their performance. A human assistant was also present during each activity to answer questions and wait for participants to be ready before starting each game.

Given a game configuration, GAMELAB automatically initialized the game environment and logged the state of participants, including positions, orientations, behavior state, and game actions, to a database. With this system, we collected multiple games of each of our 14 configuration types (70 in total) for running the experiments in the next section.

Effect of scheduling

We compare the results obtained using our computed game schedule against a brute force schedule where we ran each game configuration with a fixed order. Both schedules are initialized with the same set of 14 games, in which each game configuration was run once. In our game set, we have two traffic games (one with slow traffic and one with fast traffic) from which we collect crossing and standing samples, six configurations of hazard games (each with different numbers of narrow and wide spaces) from which we collect health and standing samples, and six configurations of token collection games (each with different numbers of narrow and wide spaces) from which we collect standing samples. The brute force scheduler runs these 14 configurations in the same order, whereas GAMELAB runs a computed policy with 1-step lookahead (a pilot study showed that a greater lookahead provided no benefit for this particular metric and game set).

After initialization, the brute force schedule selects the same set of 14 games in the same order, while the GAMELAB schedule selects three fast traffic games, three slow traffic games, and eight hazard games (six run with most furniture hidden, and two run with furniture). A comparison of the changes in confidence interval, summed over all parameters, is shown in Figure 7. Though both the brute force and GAMELAB schedule improve over time, the GAMELAB schedule beats the brute force schedule on 3 parameters (fast lane crossing, slow lane crossing, and narrow standing behaviors), ties for one parameter (wide standing behaviors), and loses for one parameter (health kit usage).

The greatest benefit of the scheduling algorithm is that it automatically reasons about differences between game configurations. For example, most narrow waiting behaviors occurred on the walkways to the alien teleportation facility, making this the best choice for collecting narrow behaviors. This was surprising to us, given that the office building contained many more narrow spaces; however, further analysis revealed that participants were rarely stationary inside the building. This fact, combined with the need for crossing

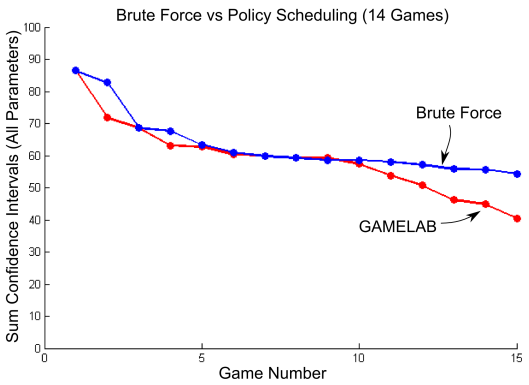


Figure 7: *Comparison of sum of confidence intervals. We compare the brute force schedule (blue line) against a 1-step-lookahead schedule (red line). Both schedules were initialized with 14 games, one for each configuration. The plots compare the change in the sum of confidence interval after an additional 14 games. The GAMELAB schedule more quickly reduces the confidence intervals across all parameters.*

samples, made the traffic game configurations good choices at the start and contributed to the policy's ability to perform better than the brute force schedule on these parameters.

A major reason why the policy performed better than the brute force schedule is that almost half our game set did not contribute many needed samples. The brute force schedule wasted time running these games whereas the policy did not. When we remove these games, the brute force schedule does equally well. This is not surprising. The true benefit of using the policy is to handle cases when some games provide samples unevenly for metrics.

Discussion

In this paper, we present an approach for the active acquisition of player metrics based on a set of flexible game configurations. Our prototype shows the potential for scheduling to be used to reduce the amount of time and cost necessary for collecting a variety of metrics using a variety of game scenarios. As data is collected, GAMELAB builds a model of how metrics correspond to game situations. This model enables GAMELAB to automatically detect how to record rarely occurring metrics (such as narrow standing behaviors in our proof-of-concept) as well as focus on the most uncertain metrics. Alternatively, an analyst would need to manually look at the data periodically to make these decisions. However, the effectiveness of the policy will depend on the accuracy of the initial estimates for P_k and possibly on the underlying models for each metric.

Though our experiments show that our policy performs better than a brute force approach, the brute force schedule still makes progress over time. The additional complexity of configuring games to be used in a scheduling policy may not be worth it when it is easier to simply run more games. Additionally, one can imagine many cases where designers are interested in estimating where people do what they do. The final MDP transition probabilities will offer some insights into this, but it will not provide detailed spatial information well, for example, heat maps of where deaths occur. The technique in this paper is best for collecting the statistics about the use of objects or about the amount of player interactions. Future work might adapt this technique for different types of metrics or for modeling differences between individual players, so that data collection might take advantage of different play styles. Future work may also adapt this technique to post-release data collection, where this technique might be used to select where and when to collect data, rather than to change the settings of released games and environments. Lastly, given that our current analysis is with a toy proof-of-concept, we would like to further investigate the potential for this technique in more complex games and environments.

Acknowledgments

This work was supported by NSF Grant IIS-1018486. We also wish to thank Benedict Brown, Ben Sunshine-Hill, Alex Shoulson and the anonymous reviewers for their comments.

References

Alexander, T. 2002. *GoCap: Game Observation Capture*. AI Game Programming Wisdom, Charles River Media.

Ambinder, M. 2009. Valve's approach to playtesting: The application of empiricism. In *Game Developer's Conference*.

Andersen, E.; Liu, Y.-E.; Apter, E.; Boucher-Genesse, F.; and Popovic, Z. 2010. Gameplay analysis through state projection. In *Foundation of Digital Games*.

Bauckhage, C.; Gorman, B.; Thurau, C.; and Humphrys, M. 2007. Learning human behavior from analyzing activities in virtual environments. *Machine Learning* 12:3–17.

Chaloner, K., and Verdinelli, I. 1995. Bayesian experimental design: A review. *Statistical Science* 10:273–304.

Charles, D.; McNeill, M.; McAlister, M.; Black, M.; Moore, A.; Stringer, K.; Kcklich, J.; and Kerr, A. 2005. Player-centred game design: Player modelling and adaptive digital games. In *Proceedings of the Digital Games Research Conference*.

Chittaro, L.; Ranon, R.; and Ieronutti, L. 2006. VU-Flow: A visualization tool for analyzing navigation in virtual environments. *IEEE TVCG* 12:1475–1485.

Cooper, S.; Hertzmann, A.; and Popović, Z. 2007. Active learning for real-time motion controllers. *ACM Trans. Graph.* 26.

DeRosa, P. 2007. Tracking player feedback to improve game design. *Gamasutra*.

Drachen, A., and Canossa, A. 2009. Analyzing spatial user behavior in computer games using geographic information systems. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, MindTrek '09, 182–189.

Eunyoung Ha, Jonathan Rowe, B. M., and Lester, J. 2012. Goal recognition with markov logic networks for player-adaptive games. In *AAAI*.

Gilleade, K. M., and Dix, A. 2004. Using frustration in the design of adaptive videogames. In *Advances in Computer Entertainment*, 228–232.

Hoi, S. C. H.; Jin, R.; Zhu, J.; and Lyu, M. R. 2006. Batch mode active learning and its application to medical image classification.

In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, 417–424.

Hoi, S. C. H.; Jin, R.; and Lyu, M. R. 2006. Large-scale text categorization by batch mode active learning. In *Proceedings of the 15th international conference on World Wide Web*, 633–642.

Kennerly, D. 2003. Better game design through data mining. *Gamasutra*.

Kim, J. H.; Gunn, D. V.; Schuh, E.; Phillips, B.; Pagulayan, R. J.; and Wixon, D. 2008. Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. In *SIGCHI*, CHI '08, 443–452.

King, R. D.; Whelan, K. E.; Jones, F. M.; Reiser, P. G. K.; Bryant, C. H.; Muggleton, S.; Kell, D. B.; and Oliver, S. G. 2004. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature* 427:247–252.

Krishnamurthy, V. 2002. Algorithms for optimal scheduling and management of hidden markov model sensors. *Signal Processing, IEEE Transactions on* 50(6):1382 –1397.

Luban, P. 2009. The silent revolution of playtests. *Gamasutra*.

Mahlmann, T.; Drachen, A.; Canossa, A.; Togelius, J.; and Yannakakis, G. N. 2010. Predicting player behavior in tomb raider: Underworld. In *Computational Intelligence and Games (CIG)*.

McPartland, M., and Gallagher, M. 2008. Learning to be a bot: Reinforcement learning in shooter games. In *AIIDE*.

Medlock, M.; Wixon, D.; Terrano, M.; Romero, R.; and Fulton, B. 2002. Using the RITE method to improve products: a definition and a case study. Technical report, Usability Professionals Association.

Moura, D.; el Nasr, M. S.; and Shaw, C. D. 2011. Visualizing and understanding players' behavior in video games. In *SIGGRAPH 2011 Game Papers*, SIGGRAPH '11, 2:1–2:6.

Ponsen, M.; Munoz-avila, H.; Spronck, P.; and Aha, D. W. 2005. Automatically generating game tactics via evolutionary learning. *AI Magazine* 27:2006.

Priesterjahn, S.; Kramer, O.; Weimer, E.; and Goebels, A. 2007. Evolution of human-competitive agents in modern computer games. In *Congress on Evolutionary Computation (CEC)*.

Reeder, J.; Gita, S.; Georgiopoulos, M.; and Anagnostopoulos, G. C. 2008. Intelligent trading agents for massively multi-player game economies. In *AIIDE*.

Riccardi, G., and Hakkani-Tur, D. 2005. Active learning: theory and applications to automatic speech recognition. *Speech and Audio Processing, IEEE Transactions on* 13(4):504 – 511.

Robin Hunicke, V. C. 2004. AI for dynamic difficulty adjustment in games. In *AIIDE*.

Roy, N., and Mccallum, A. 2001. Toward optimal active learning through sampling estimation of error reduction. In *In Proc. 18th International Conf. on Machine Learning*, 441–448. Morgan Kaufmann.

Settles, B. 2012. *Active Learning*. Morgan & Claypool Publishers.

Shaker, N.; Yannakakis, G. N.; and Togelius, J. 2010. Towards automatic personalized content generation for platform games. In Youngblood, G. M., and Bulitko, V., eds., *AIIDE*. The AAAI Press.

Shani, G.; Brafman, R. I.; and Heckerman, D. 2002. An MDP-based recommender system. In *Journal of Machine Learning Research*, 453–460. Morgan Kaufmann.

Sharifi, A.; Zhao, R.; and Szafron, D. 2010. Learning companion behaviors using reinforcement learning in games. In *AIIDE*.

Tastan, B., and Sukthankar, G. 2011. Learning policies for first person shooter games using inverse reinforcement learning. In *AIIDE*, 85–90.

Tong, S., and Chang, E. 2001. Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, 107–118.

Tur, G.; Hakkani-Tur, D.; and Schapire, R. E. 2005. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication* 45(2):171 – 186.

Tychsen, A., and Canossa, A. 2008. Defining personas in games using metrics. In *Conference on Future Play: Research, Play, Share*, Future Play '08, 73–80.