Automatic Playtesting for Game Parameter Tuning via Active Learning

anonymous

ABSTRACT

Game designers use human playtesting to gather feedback about game design elements when iteratively improving a game. Playtesting, however, is expensive: human testers must be recruited, playtest results must be aggregated and interpreted, and changes to game designs must be extrapolated from these results. Can automated methods reduce this expense? We show how active learning techniques can formalize and automate a subset of playtesting goals. Specifically, we focus on the low-level parameter tuning required to balance a game once the mechanics have been chosen. Through a case study on a shoot-'em-up game we demonstrate the efficacy of active learning to reduce the amount of playtesting needed to choose the optimal set of game parameters for a given (formal) design objective. This work opens the potential for additional methods to reduce the human burden of performing playtesting for a variety of relevant design concerns.

Categories and Subject Descriptors

pick

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

pick

Keywords

pick

1. INTRODUCTION

update top part w/FDG info

Iterative game design practices emphasize the centrality of playtesting to improve and refine a game's design. Human playtesters provide valuable feedback on audience reactions to a game. Playtesting is often claimed to be "the single most important activity a designer engages in" [7]. Test data informs designers of how real players may react to the game in ways that self-testing, simulations, and design analysis may not. Playtesting, however, is expensive—developers must recruit players, devise design experiments, collect game play and subjective feedback data, and make design changes to meet design goals.

We ask the question: can we reduce the cost of the playtesting process by automating some of the more mundane aspects of playtesting? To address this problem we examine a subset of playtesting questions focused on "parameter tuning." Parameter tuning involves making low-level changes to game mechanic settings such as character movement parameters, power-up item effects, or control sensitivity. Games based on careful timing and reflexes depend on well-tuned parameters, including racing, platforming, shoot-'em-up, and fighting game genres. Addressing the problem of parameter tuning requires a means to automatically select a set of potentially good parameter settings, test those settings with humans, evaluate the human results, and repeat the process until a pre-defined design goal is achieved.

Our primary insight is to model playtesting as a form of active learning (AL). Active learning [19] selects among a set of possible inputs to get the best output while minimizing the number of inputs tested. We define the "best output" as a parameter tuning design goal and treat a set of game design parameters as an "input." Minimizing the number of inputs tested minimizes the number of playtests performed, saving human effort and reducing costs. This paper makes three contributions toward machine-driven playtesting:

1. Formulating efficient playtesting as an AL problem
2. Defining a set of playtesting goals in terms of AL metrics
3. Demonstrating the efficacy of AL to reduce the number of playtests needed to optimize (1) difficulty-related and (2) control-related game parameters in a case study of a shoot-'em-up game

We believe machine-driven playtesting is a novel use of machine learning in games. Unlike prior work in dynamic difficulty and adaptive games we focus on the case of deciding on a fixed design for future use. Our approach can be applied to iterative, online game adjustments to more rapidly converge on the right set of game parameters. Unlike prior work on game design support tools using simulations or model-checking we focus on the problem of efficiently working with a set of human testers. Our approach complements these tools for early-stage exploration with late-stage refinement.

In this paper we first compare our approach to related work on game design support. We next define a set of parameter tuning design goals and relate them to AL methods. Following this we describe a case study of automated playtesting using a shoot-'em-up game. After describing the game we present results showing AL reduces playtesting costs using human data collected from an online study. We conclude with a discussion of the limitations, applications, and potential extensions to this playtesting approach.

update

2. RELATED WORK

Two research areas are closely related to machine playtesting: offline game design tools and online game adaptation. Offline game design tools enable designers to explore possible game designs by defining a high-level space of games through a design language. Online game adaptation changes game designs in real-time based on player actions or game state.

2.1 Offline Game Design Tools

Offline game design tools have evaluated game designs using simulated players and formal model-checking. Simulation-based tools use sampling techniques to test aspects of a game design for "playability," typically defined as the ability for a player to reach a given goal state with the current design parameters. Model-checking tools define game mechanics in a logical language to provide hard guarantees on the same kinds of playability tests.

Bauer et al. [2] and Cook et al. [6] use sampling methods to evaluate platformer game level playability. Shaker et al. [20] combine a rule-based reasoning approach with simulation to generate content for a physics-based game. Simulation approaches are valuable when design involves an intractably large space of possible parameters to test and can serve as input to optimization techniques. Model-checking approaches provide guarantees on generated designs having formally defined properties—typically at the cost of being limited to more coarse design parameter decisions. Smith et al. [22], Butler et al. [4], and Horswill and Foged [10] use logic programming and constraint solving to generating levels or sets of levels meeting given design constraints. Jaffe et al. [12] use game-theoretic analysis to understand the effects of game design parameters on competitive game balance.

Our approach to automated playtesting is intended to complement these approaches to high-level early design exploration with low-level optimization of game parameters and tuning. Further, we focus on a generic technique that applies to cases with human testers in the loop, crucial to tuning game controls or subjective features of games. Offline design tools currently enable designers to formally define and enforce properties of a game design across all possible player behaviors in a specific game or space of designs. To date these tools have emphasized techniques for ensuring game designs have desired formal properties that meet designer intent. Machine-driven playtesting enables players to provide feedback to designers on *expected* player behaviors in a game. Developing machine-driven playtesting techniques affords designers insight into how human audiences interact with designer intent, complementing an understanding of whether and how a game matches formal criteria.

2.2 Online Game Adaptation

Online game adaptation researchers have used both hand-crafted rules and data-driven techniques. Hunicke and Chapman [11] tracked the average and variance of player damage and inventory levels and employ a hand-crafted policy to adjust levels of enemies or powerups. Systems by Magerko et al. [14], El-Nasr [18], and Thue et al. [24] model players as vectors of skills, personality traits, or pre-defined "player types" and select content to fit players using hand-crafted rules. Hand-crafted rules enable designers to describe fine-tuned details of how to adjust a game toward design goals. However, designers must fully describe how to tune the game and rules are often sensitive to minor changes in game settings.

To bypass the brittleness of rules others have employed data-driven techniques that optimize game parameters toward design goals. Hastings et al. [9], Shaker et al. [21], Liapis et al. [13] and Yu and Riedl [26] model player preferences using neuro-evolutionary or machine learning techniques and optimize the output of these models to select potential game parameters. Harrison and Roberts [8] optimize player retention and Zook and Riedl [28] optimize game difficulty using similar techniques.

Automated playtesting extends these approaches with principled methods to guide the process of designing hand-crafted rules or optimizing game parameters. When hand-crafting rules, automated playtesting informs the choice of which rule parameters to use. When optimizing models learned from data, automated playtesting informs the choice of which next set of parameters to test during the optimization process. We argue research to date has ignored the problem of reducing "sample complexity"—the number of data points (human playtests) needed to train a model. Active learning makes the trade-off in playtesting between "exploring" potentially valuable game design settings and "exploiting" known good solutions with small changes explicit. Thus, AL complements online game adaptation through reducing the number of mediocre or bad sets of game parameters players experience before arriving at good parameter settings without changing the underlying models used.

2.3 Active Learning in Games

There are other uses of AL in games. Normoyle et al. [15] use AL to recommend sets of useful player metrics to track; we pick design settings to improve player experience. Rafferty et al. [16] optimize game designs offline to learn the most about player cognition; we use an online process and explore a variety of alternative AL approaches. Our approach complements prior uses of AL for game design by focusing on efficiently improving designs for player behavior and experience. We extend these efforts with a wider variety of AL methods while addressing the cost of playtesting.

3. PLAYTESTING AS ACTIVE LEARNING

Our goal is to automate mundane playtesting tasks by efficiently choosing game designs for players to test. Active learning (AL) provides a generic set of techniques to guide the playtesting process of choosing a set of design parameters to test toward achieving a design goal. Playtesting typically involves trade-offs between testing designs that are poorly understood (exploration) and refining designs that are known to be good but need minor changes (exploitation). Active learning captures this intuition through explicit models of the exploration-exploitation trade-off. In this section we characterize playtesting in terms of AL. We provide intuitions behind AL, but full mathematical treatments are available through the references.

Machine-driven playtesting—AL for game design—involves (1) a design model for how a design works, (2) a design goal, and (3) a playtesting strategy for how to change a design to achieve a design goal. Formally, a design model is a *function* that captures the relationship between game design parameters (input) and game metrics (output: e.g. specific player behaviors or subjective responses). The design goal is an *objective function* specificying what game metrics are desired. Playtesting strategies choose what design to test next using an *acquisition function* that uses information from the design model and goal to predict and value possible playtest outcomes.

Design model functions come in two forms: regression and classification. *Regression* models capture continuous outputs—e.g. how the rate of enemy firing in a shoot-'em-up influences the number of times a player is hit or how the height of jumps in a platformer influences how long it takes players to complete a level. *Classification* models capture discrete outputs—e.g. which of a pair of control settings a player preferred in a racing game or which choice a player made when given a set of options in a choose-your-own adventure. Objective functions specify how to value different outputs—e.g. wanting players to be hit a certain number of times or wanting players to agree that one set of controls is good. Note that many design goals can be formulated as goals for playtesting: the challenge lies in defining a useful metric for measuring these goals through player feedback or in-game behavior.

Acquisition functions differ for regression and classification models. In the next sections we provide intuitive definitions of several acquisition functions—references to the relevant mathematical literature are provided. Our survey of regression models covers key methods along the exploration-exploitation spectrum. For classification models we cover the most common frameworks for AL with discrete data that are intended to mitigate the impact of highly variable data.

3.1 Regression Models

Acquisition functions choose which input parameters to test next to most efficiently maximize an objective function. Acquisition functions vary along a spectrum of exploration—playtesting by picking designs that are least understood—and exploitation—picking designs that are expected to be best. We consider four acquisition functions for regression models: (1) variance, (2) probability of improvement, (3) expected improvement, and (4) upper-confidence bounds. These acquisition functions have been developed in the field of Bayesian experimental design and apply generally to any regression model with a probabilistic interpretation [5, 3]. Regression models are useful when design goals fall along a continuous scale; we examine player behavior, specifically studying performance as damage taken.

Regression acquisition functions include:

Variance Exploration by picking the input with greatest output uncertainty [3].

Probability of improvement (PI) Exploitation by picking the input most likely to have an output that improves over the previous best [3].

Expected Improvement (EI) Balances exploration-exploitation by picking the input with greatest combined probability and amount of improvement over the previous best [3].

Upper Confidence Bound (UCB) Balances exploration-exploitation by picking the input with greatest combined expected value and uncertainty to gradually narrow the space of inputs [23].

3.2 Classification Models

Classification models are useful when design goals involve discrete choices; we examine player subjective ratings, specifically studying preference choices when picking between sets of controls. Classification models are primarily concerned with increasing certainty in predicting outcomes—improving the model of how the design works. We consider five acquisition functions for classification models: (1) entropy, (2) query-by-bagging (QBB) vote, (3) query-by-bagging (QBB) probability, (4) expected error reduction, and (5) variance reduction. These acquisition functions have been developed for classification models; several—entropy, QBB probability, and expected error and variance reduction—require probabilistic predictions.

Entropy Reduces uncertainty by picking the input with greatest output uncertainty according to entropy—a measure of information needed to encode a distribution [19].

Query-By-Bagging (QBB) Reduces uncertainty by picking the input with most disagreement among copies of a classification model trained on random subsets of known results [19]. **QBB vote** picks the input with the largest difference between its top two output options [19]. **QBB probability** picks the output with greatest average uncertainty across the models [1].

Expected Error Reduction Reduces uncertainty by picking the input that, if used to train the model, leads to the greatest expected reduction in classification error [19].

Variance Reduction Same as expected error reduction, but seeks to reduce variability in outputs rather than error [19].

4. GAME DOMAIN

We sought to assess how well AL could reduce the number of playtests needed to achieve a design goal. To conduct a case study of machine-driven playtesting we developed a simple shoot-'em-up game (Figure 1). Shoot-'em-up games emphasize reflexes and pattern recognition abilities as a player maneuvers a ship to dodge enemy shots and return fire. In general, arcade games serve as an ideal starting domain for low-level parameter tuning:

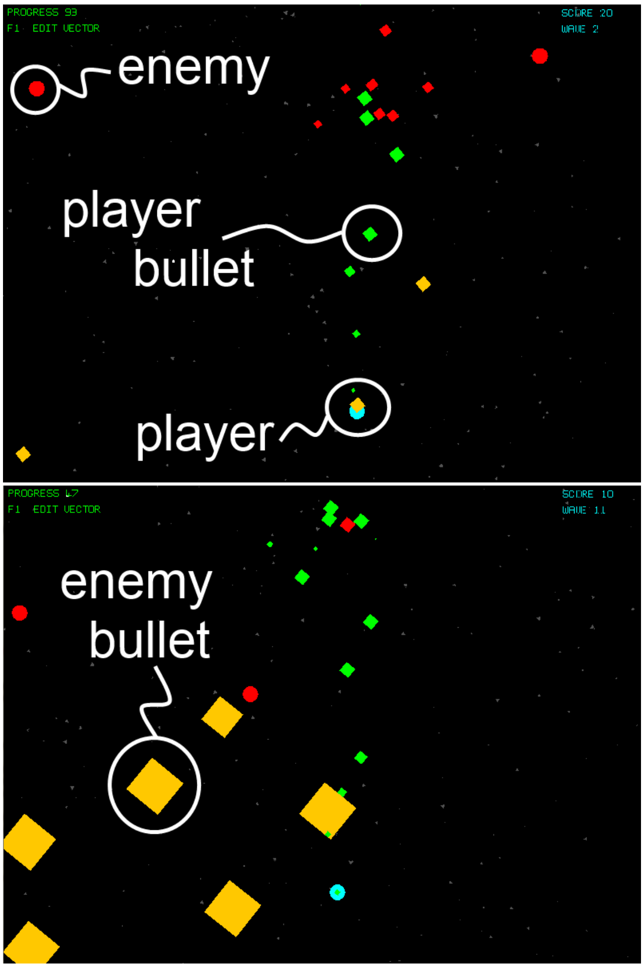

Figure 1: Study game interface illustrating player, enemies, and shots fired by both at two points along adaptation process.

- There are a number of parameters that can potentially interfere with each other: size and speed of enemies and enemy bullets, rate of enemy fire, player speed, player rate of fire, etc.
- The game can be played in a series of waves, enabling our system to naturally test game parameter settings and gather player feedback.
- Action-oriented gameplay reduces the complexity of player long-term planning and strategizing.
- A scoring system makes gameplay goals and progress clear, unlike domains involving puzzle-solving or aesthetic enjoyment of a game world or setting.

In the case of shoot-'em-up games, we tested two different kinds of game design goals: (a) player game play behavior goals and (b) player subjective response goals. Player game play behavior goals cover cases where designers seek particular play patterns or outcomes—e.g. player success rates or score achieved. Subjective responses goals cover cases where designers desire specific player subjective feedback—e.g. getting good user ratings on the feel of the controls.

The shoot-'em-up game involves space ship combat over a series of waves. During each wave a series of enemies appear that fire bullets at the player. To test AL for regression we set a game play behavior design goal (objective function) of the player being hit exactly six times during each wave of enemies (output) and tuned enemy parameters (input). We varied the size of enemy bullets, speed of enemy bullets, and rate that enemies fire bullets. Increasing bullet size requires the player to move more carefully to avoid bullets. Faster bullets require quicker player reflexes to dodge incoming fire. More rapid firing rates increase the volume of incoming fire. Together these three parameters govern how much players must move to dodge enemy attacks, in turn challenging player reflexes. Getting approximate settings for these parameters is easy, but fine-tuning them for a desired level of difficulty can be challenging.

To test AL for classification we set a subjective response design goal (objective function) of the player evaluating a set of controls as better than the previous set (output) and tuned player control parameters (input). We varied two ship movement parameters: drag and thrust. Drag is the "friction" applied to a ship that decelerates the moving ship at a constant rate when it is moving—larger values cause the ship to stop drifting in motion sooner. Thrust is the "force" a player movement press applies to accelerate the ship—larger values cause the ship to move more rapidly when the player presses a key to move. Combinations of thrust and drag are easy to tune to rough ranges of playability. However, the precise values needed to ensure the player has the appropriate controls are difficult to find as player movement depends on how enemies attack and individual player preferences for control sensitivity (much like mouse movement sensitivity). After wave of enemies a menu asked players to indicate if the most recent controls were better, worse, or as good/bad as ("neither") the previous set of controls. We provided a fourth option of "no difference" for when players could not distinguish the sets of controls, as opposed to "neither" where players felt controls differed but had no impact on their preferences.

5. EXPERIMENTS

Our experiments tested whether AL could reduce the number of human playtests needed to tune design parameters compared to a random sampling approach. Random sampling is the standard baseline used to evaluate the efficacy of AL models for improving an objective function for a fixed number of inputs [19]. Random sampling is similar to A/B testing approaches that capture large amounts of data before acting on the results.

In two experiments we used the AL acquisition functions given above for regression by tuning enemy parameters and for classification by tuning player controls, respectively. For both experiments we first built a dataset by providing human players with random sets of parameters and recording behavior or subjective responses, respectively. The experiments had AL methods use this data as a potential pool of playtests to run and evaluated how well those methods could pick a sequence of playtests to best achieve the design goals.

In the regression study we used Gaussian Processes (GPs), the standard non-linear regression function used in the Bayesian experimental design literature. Gaussian Processes generally yield good models with few playtests (samples) and have computationally inexpensive analytic formulations for many of our acquisition functions. In the classification study we used three different objective functions—Gaussian Processes (GP), kernel support vector machines (KSVM), and optimized neural networks ("neuro-evolution", NE). KSVMs and NE are common classification approaches, whereas GPs are not. Kernel methods (e.g. KSVMs and GPs) are a popular machine learning technique previously used in player modeling [27] and optimized neural networks have been widely used in preference learning [25].[1] Note that NE does not produce probabilistic predictions, limiting the acquisition functions it can be paired with.

5.1 Data Collection

We empirically evaluated AL by deploying two versions of our game online. We publicized the game through websites and local emailing lists and did not offer compensation to participants. To collect data on patterns of play over time we asked participants to try to play at least 10 waves of the game, though we did not enforce this requirement.

For analysis we only used data from players who played at least 10 waves total. This ensures we avoid data from players who were unable to reliably run the game. For our regression experiment this resulted in data from 138 players and 991 waves of the game total (using all waves each player played). For our preference experiment we had 57 players, 47 of these provided only binary responses of "better" or "worse" and we limited our analysis to this subset of players to yield 416 paired comparisons. We only used preference responses during the first 10 waves of play to avoid collecting many positive responses from those who were highly engaged in the game. Note that we did not collect preference comparisons for the first wave of the game as players could not yet compare control settings.

5.2 Experiment Design

Using this data we performed 10-fold cross-validated experiments to measure how well a playtesting strategy (acquisition function) could achieve a design goal (objective function) given a set of design parameters (input). For regression we trained a GP (design model) using the three enemy parameters (input: bullet speed, bullet size, and firing rate) to minimize the squared difference between the number of times the player was hit and a desired rate of 6 times (objective function). We squared the difference to more steeply penalize sets of parameters with greater differences from the ideal. For classification we trained a GP, KSVM, or NE (design model) with four control parameters (input: current and previous drag and thrust) to maximize the prediction quality (as F1 score) on whether the preference rating was "better" or "worse" (objective function). We discarded ratings that were not in these two classes as our data had too few samples to make a comparison (only 10/57 players ever responded in other categories).

For each cross-validation fold we first set aside a randomly selected 10% of the data for evaluating objective function performance. Next we randomly sampled 30 inputs and outputs from the other 90% of the dataset to create a training data set; the remaining dataset samples formed the training pool. Within each fold we then repeated the following process:

1. Train the regression or classification model on the training data set.
2. Evaluate the objective function for that model on the testing data set.
3. Use the acquisition function to pick a new input sample from the training pool (without yet knowning the sample output) to improve the objective function.
4. Move the selected sample (including the true output) from the training pool into the training data.
5. Return to the first step and repeat the process until the maximum number of training samples are used.

We used a maximum of 300 training samples in both regression and classification.[2]

[1] For computational reasons we employ a simple gradient-based optimization method for network structure, size, and weights, rather than the more common neuro-evolutionary approaches. We did not find any performance differences between the two optimization approaches in initial tests on our data.

6. RESULTS

Overall our results show AL is a promising approach for reducing the number of playtests needed to achieve a design goal. For enemy parameter tuning (a regression problem) we found acquisition functions that balance exploration and exploitation (UCB and EI) have the best performance. For control tuning (a classification problem) we found acquisition functions that tolerate high variance (e.g. QBB and entropy) have strong performance. No single acquisition function, objective function, or acquisition-objective function pair was optimal across cases and number of playtests. These results align with previous work in AL showing that many data-specific properties impact AL efficacy [17]. Below we provide further details with an emphasis on how AL impacted the need for playtesting.

6.1 Regression

Our regression experiments show strong results for AL to design for players having a desired level of performance for almost all acquisition functions. Having a clear behavioral objective (being hit a number of times in the game) was likely a strong contributor. We found UCB and EI to be most effective (Table 1). Both methods explicitly balance exploring and exploiting potential test designs, suggesting parameter tuning objectives involve a balance between considering alternative parameter settings and refining a given setting.

get figure w/relative improvements

Methods that balance exploration and exploitation—UCB and EI—performed well, other AL methods did not (Figure 2). Variance only performed well with many samples; explained by the need to explore heavily before having a good enough design model. When tuning many parameters at once it is easy to find many sets of uncertain (but bad) parameters, leading to poor performance with few samples. Over time EI gradually worsened while UCB maintained better performance. As more samples are gathered UCB reduces exploration while EI eventually begins to make poor

[2] For computational reasons we only used a maximum of 280 samples for 280 NE methods in classification.

Table 1: Comparison of GP mean squared error with various acquisition functions. Lower values indicate better performance.

acquisition function	80 samples	280 samples
random	136	124
variance	121	**120**
PI	133	123
EI	**112**	127
UCB	**107**	**117**

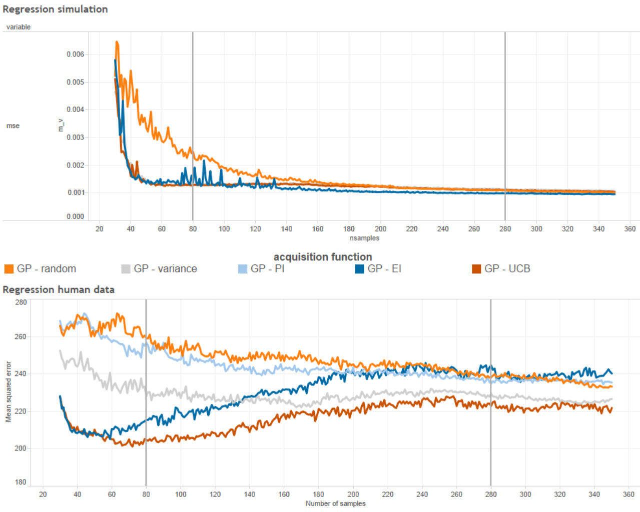

Figure 2: Regression experiments GP mean squared error on predicting number of times player is hit vs number of samples used in training for different acquisition functions. Lower values indicate better performance. Top figure uses simulation data, bottom uses human data. Lines demarcate points used in Table 1.

playtest choices. Approximately 100 samples were needed to train the successful AL methods to their peak performance; random sampling never achieved this level of performance on our data set. Overall this clearly demonstrates AL can enhance playtesting efficacy, perhaps beyond what would happen through simply A/B testing and collecting data.

Our regression experiments demonstrate the power of AL to reduce the amount of playtesting required and better achieve design goals. Methods that balance exploration and exploitation—EI and UCB—show the greatest efficacy and suggest a design process of gradual refinement is optimal. These results make a strong case for AL applied to optimizing low-level in-game behaviors, such as difficulty in terms of in-game performance.

6.2 Classification

Our classification experiments show AL improves models of subjective player preferences with both probabilistic and non-probabilistic acquisition functions. Methods that tolerate high variance—entropy, QBB vote and probability, and expected error reduction—have the strongest performance (Table 2). These acquisition functions succeed by overcoming the noise inherent in human playtest data, particularly when using few playtests. Our results show AL methods are

Table 2: Comparison of acquisition-objective function combination accuracy with few (100) and many (200) playtest samples.

acquisition function	100 samples			200 samples		
	GP	KSVM	NE	GP	KSVM	NE
random	0.720	0.684	0.673	0.773	0.709	0.718
entropy	**0.763**	**0.731**	N/A	0.763	0.751	N/A
QBB vote	**0.758**	**0.746**	**0.703**	0.780	**0.777**	**0.760**
QBB prob	0.749	0.724	N/A	**0.792**	**0.782**	N/A
error red	**0.761**	0.702	N/A	**0.795**	**0.772**	N/A
var red	0.660	0.667	N/A	0.725	0.723	N/A

effective even with more complex data and can improve a variety of baseline design models (GPs, KSVMs, and NE).

replace

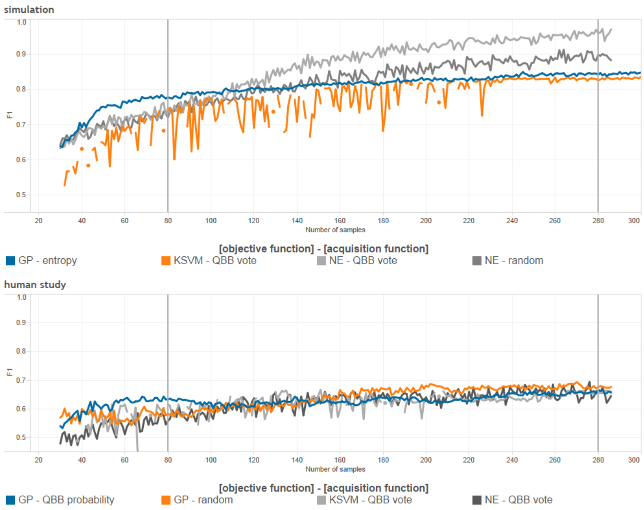

Figure 3: Classification F1 score vs number of samples used in training for different objective and acquisition function combinations. Higher values indicate better performance. Top figure uses simulation data, bottom uses human data.

Entropy, QBB vote and probability, and error reduction all improved classification quality (as F1 score) over random sampling (Figure 3). QBB methods (especially vote) were effective at both few and many samples. Entropy was only effective with few samples while error reduction was most effective with more samples. Expected error reduction depends on predicting future outcomes and thus requires more initial data before becoming effective. Variance reduction had poor performance; similarly to the variance acquisition function for regression a large number of possible parameters causes difficulty in effectively reducing variability in responses. We speculate that preference responses are typically noisy due to people shifting preferences or disagreeing on a common design as preferrable (e.g. looking control sensitivity or inversion in first-person games).

Comparing the design models, we found GPs had the best performance with random sampling, followed by NE and then KSVMs. Overall GPs with QBB probability or expected error reduction did best, followed by KSVMs with either QBB method and then NEs using QBB vote. Using AL methods provided the largest performance boost for

KSVMs, though GPs and NE also benefited.

Our classification experiments thus demonstrate AL can reduce the amount of playtesting needed even for subjective features of a design such as control settings. Reducing playtest costs requires acquisition functions (e.g. entropy, QBB, and error reduction) that mitigate the noise inherent in preference response data. AL always improved over random sampling across different design model approaches, though the best acquisition functions varied.

7. CONCLUSIONS

We have shown how playtesting for low-level design parameter tuning can be automated using active learning. AL techniques can reduce the expense of playtesting to achieve a design goal in terms of the designs tested. In some cases AL may get better results for a design goal than simple A/B testing could accomplish. Machine-driven playtesting can thus complement the strengths of human game designers by allowing them to focus on high-level design goals. Machine-driven playtesting has great promise for broadening how automation enhances existing game design practices across the design iteration process.

We believe machine-driven playtesting can yield valuable insights for how to best perform playtests. Designers may be able to learn better playtest techniques through considering how machines best achieve these tasks. For example, in AL for regression we found the UCB acquisition function was best. UCB uses a process of initially exploring widely before shifting toward greater exploitation; UCB chooses designs expected to elicit both high-quality and high-variability results to narrow the space of alternatives. Balance or other feature-tuning techniques may thus benefit from similar methods of parallel tests on multiple alternatives based on balanced consideration of their value and how well they are understood. In the future we hope to see automated techniques for other aspects of design — recognizing design flaws or assigning credit to aspects of a design—for a variety of player behavioral and subjective outcomes.

8. ACKNOWLEDGMENTS

9. REFERENCES

[1] N. Abe and H. Mamitsuka. Query learning strategies using boosting and bagging. In *International Conference on Machine Learning*, 1998.

[2] A. W. Bauer, S. Cooper, and Z. Popovic. Automated redesign of local playspace properties. In *8th International Conference on the Foundations of Digital Games*, 2013.

[3] E. Brochu. *Interactive Bayesian optimization: learning user preferences for graphics and animation.* PhD thesis, University of British Columbia, 2010.

[4] E. Butler, A. M. Smith, Y.-E. Liu, and Z. PopoviÄĞ. A mixed-initiative tool for designing level progressions in games. In *ACM Symposium on User Interface Software and Technology*, 2013.

[5] K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Statistical Science*, 10 (3):273–304, 1995.

[6] M. Cook, S. Colton, and J. Gow. Initial results from co-operative co-evolution for automated platformer design. In *EvoGames 2012*, 2012.

[7] T. Fullerton, C. Swain, and S. Hoffman. *Game design workshop: a playcentric approach to creating innovative games.* Morgan Kaufmann, 2008.

[8] B. Harrison and D. L. Roberts. Analytics-driven dynamic game adaption for player retention in scrabble. In *IEEE Conference on Computational Intelligence in Games*, pages 1–8. IEEE, 2013.

[9] E. Hastings, R. K. Guha, and K. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1:245–263, 2009.

[10] I. Horswill and L. Foged. Fast procedural level population with playability constraints. In *Proceedings of the Eighth Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[11] R. Hunicke and V. Chapman. AI for dynamic difficulty adjustment in games. In *AAAI Workshop on Challenges in Game Artificial Intelligence*, 2004.

[12] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popovic. Evaluating competitive game balance with restricted play. In *Proceedings of the Eighth Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[13] A. Liapis, H. P. Martınez, J. Togelius, and G. N. Yannakakis. Adaptive game level creation through rank-based interactive evolution. In *IEEE Conference on Computational Intelligence in Games*, volume 4, pages 71–78. Springer, 2013.

[14] B. Magerko, B. Stensrud, and L. Holt. Bringing the schoolhouse inside the box - a tool for engaging, individualized training. In *25th Army Science Conference*, 2006.

[15] A. Normoyle, J. Drake, M. Likhachev, and A. Safonova. Game-based data capture for player metrics. In *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[16] A. N. Rafferty, M. Zaharia, and T. L. Griffiths. Optimally designing games for cognitive science research. In *34th Annual Conference of the Cognitive Science Society*, pages 280–287, 2012.

[17] A. I. Schein and L. H. Ungar. Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265, 2007.

[18] M. Seif El-Nasr. Interaction, narrative, and drama: Creating an adaptive interactive narrative using performance arts theories. *Interaction Studies*, 8(2):209–240, 2007.

[19] B. Settles. *Active learning*, volume 6. Morgan & Claypool Publishers, 2012.

[20] M. Shaker, N. Shaker, and J. Togelius. Evolving playable content for cut the rope through a simulation-based approach. In *9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013.

[21] N. Shaker, G. N. Yannakakis, and J. Togelius. Crowdsourcing the aesthetics of platform games. *IEEE Transactions on Computational Intelligence and AI in Games*, 5:276–290, 2013.

[22] A. M. Smith, B. Eric, and Z. Popovic. Quantifying over play: Constraining undesirable solutions in puzzle design. In *8th International Conference on the Foundations of Digital Games*, 2013.

[23] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning*, 2010.

[24] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen. Interactive storytelling: A player modelling approach. In *3rd AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2007.

[25] G. Yannakakis and J. Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, 2011.

[26] H. Yu and M. O. Riedl. Personalized interactive narratives via sequential recommendation of plot points. *IEEE Trans. Computational Intelligence and AI in Games*, forthcoming, 2013.

[27] H. Yu and T. Trawick. Personalized procedural content generation to minimize frustration and boredom based on ranking algorithm. In *7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2011.

[28] A. Zook and M. O. Riedl. A temporal data-driven player model for dynamic difficulty adjustment. In *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.