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 focus on 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" in terms of a parameter tuning design goal and treat a setting for a game design as an "input." Minimizing the number of inputs tested minimizes the number of playtests performed. This paper makes three contributions toward machine-driven playtesting:

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

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

Foundations of Digital Games ???

be applied to 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 approaches. Following this we describe a case study of automated playtesting using a shoot-'em-up game. After describing the game used we present results showing the efficacy of AL to model a known simulated player model and human data collected from an online study. We conclude with a discussion of the limitations, applications, and potential extensions to this playtesting approach.

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 reduce the playtesting burden by efficiently choosing game designs to test on players. 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 intuition behind AL, but full mathematical treatments are available through the references.

Machine-driven playtesting—active learning for game design—uses a model of how a design works to choose how to change that design to achieve a design goal. Formally, the design goal is known as the *objective function* and captures the relationship between input (game design parameters) and desired output (game design goals: specific player behaviors or subjective responses). An *acquisition function* takes information on expected outputs from the objective function and uses it to decide on the next set of inputs to test.

Objective functions come in two forms: regression and classification models. *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. Note that many design goals can be formulated as goals for playtesting: the primary 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 functions. 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 that balance between exploration of possible designs and exploitation to refine high quality designs. 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. 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]. 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 Picks the input with greatest output uncertainty [3]. Corresponds to picking the design that is hardest to predict the outcomes from. Exploration.

Probability of improvement (PI) Picks the input that most likely to improves the over the best previous output [3]. Corresponds to picking the design most likely to improve over the current best. Exploitation.

Expected Improvement (EI) Picks the input by weighting the probability of output improvement by the amount of improvement [3]. Corresponds to picking the design with largest expected improvement. Balances exploration and exploitation, but more computationally costly.

Upper Confidence Bound (UCB) Picks the input with combined greatest expected value and uncertainty [23]. Corresponds to picking designs that seem high quality but are poorly understood to gradually narrow the space of design to be known good or expected bad but uncertain. The approach cited gradually reduces the weighting of uncertainty to help converge on an answer. Balances exploration and exploitation.

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. We consider five acquisition functions for classification models: (1) entropy, (2) query-by-bagging voting, (3) query-by-bagging probability, (4) expected error reduction, and (5) variance reduction. These acquisition functions have been developed for general classification models with only entropy requiring probabilistic predictions.

Entropy Picks input with greatest output uncertainty according to entropy—a measure of information needed to encode a distribution [19]. Corresponds to picking designs where player choices are most uncertain. Exploration.

Query-By-Bagging (QBB) Trains multiple copies of a classification model on random subsets of existing input-output data and picks next input with greatest disagreement among those models [19]. Corresponds to picking designs with greatest variability in expected outcomes when viewed based on different subsets of playtest data. Exploration. Two varieties are relevant:

- **QBB vote** Each model votes on predicted output for each potential input. The input with the greatest difference between its top two output options is then picked.
- **QBB probability** Each model predicts the probability of each predicted output for each potential input. The input with most uncertain averaged output probability is then picked [1].

Expected Error Reduction Picks the input that, if used to train the model, will give greatest expected reduction in model error when predicting on the remaining inputs. Corresponds to picking designs that will most improve prediction of the design outcomes over the design as a whole. Balances exploration and exploitation but very computationally expensive. The algorithm performs the following steps:

1. Pick a potential input and assign one of the possible outputs to make a "fake" training point.
2. Train a new classification model using the fake point and compute the error over the remaining input points.

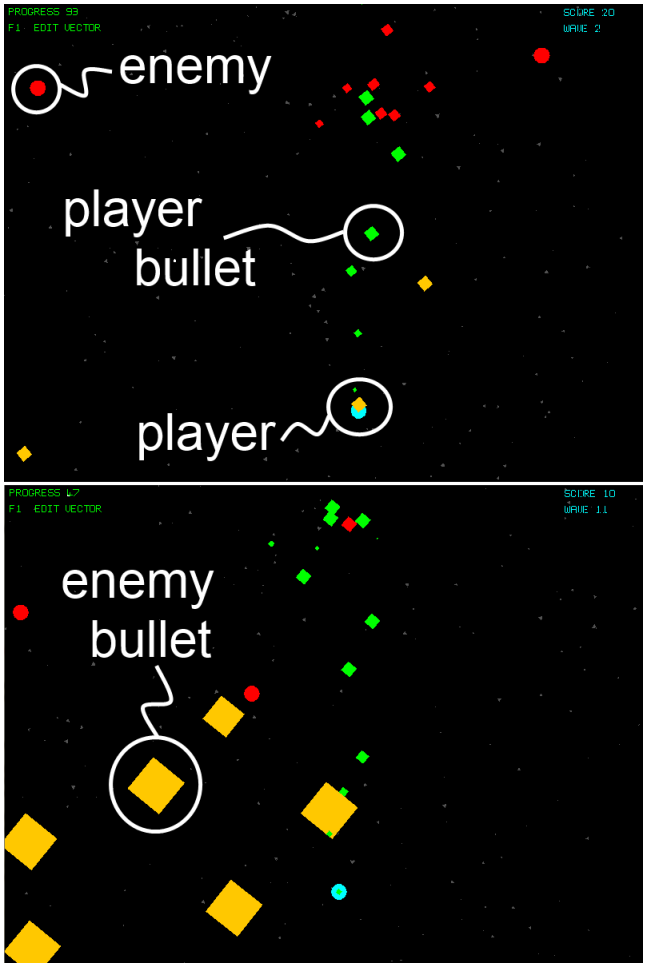

Figure 1: Study game interface illustrating player, enemies, and shots fired by both at two points along adaptation process.

3. Repeat (1) and (2) for all possible outputs and compute the average ("expected") error across possible outputs.
4. Repeat (1) through (3) for all inputs.
5. Pick the input with greatest difference between expected error and current model expected error.

Expected Variance Reduction Same as expected error reduction, but seeks to reduce variability in outputs rather than error. Corresponds to picking designs that lead to greatest reduction in uncertainty about the design space over time.

4. GAME DOMAIN

We sought to assess the efficacy of AL at reducing 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:

- 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 of the player being hit exactly six times during each wave of enemies and tuned enemy parameters. 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 of the player evaluating a set of controls as better than the previous set and tuned player control parameters. 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 different" 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 method used to evaluate the efficacy of AL models for improving model fits for a fixed number of playtests [19]. This 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. Both experiments involved a preliminary test on simulated data followed by testing with human participants. The simulation experiments allowed us to verify that our methods would work in principle; humans studies show our method can apply to real-world contexts.

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 samples (playtests) and computationally inexpensive analytic formulations of 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 (particularly 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.

Our results show AL can reduce the number of human playtests needed. For enemy parameter tuning (a regression problem) we find GPs with UCB or EI have the best performance in both simulation and on human data. Variance performs well in simulation but poorly with humans; likely due to human players' greater variability in playing behavior than our simulated models.

For control tuning (a classification problem) we find QBB vote and entropy have competitive performance in simulation. On human data GPs show good performance using QBB probability with few (roughly 80) samples while KSVMs show good performance using QBB probability with more (roughly 280) samples; NE showed the best performance overall and little difference among techniques used. Our control tuning results show a generic set of good controls for all players is difficult to obtain. Together these results show AL has promise for helping to automate low-level parameter tuning.

In the following sections we detail our experimental design for both the simulated players and human participants. We then discuss the results of our tests in detail.

5.1 Simulation Methods

We devised two simple models of how players might respond to different design parameters to verify our AL approach on ground truth information. Our regression model treats players as having an underlying set of skills related to each enemy tuning parameter along with a cross-skill tolerance for differing challenge demands. Greater differences between player skills and enemy parameters lead to larger differences from being hit at a base rate. Our classification model treats players as having preferences for each of the control tuning parameters with a cross-parameter tolerance for differences from preference. Preference choices are based on the difference between the ideal set of parameters and design control settings.

Our regression model is a probabilistic model of player behavior in terms of underlying player skills and design parameters. Playtest players have three independent skills for enemy bullet size, bullet speed, and firing rate; all three skills are sampled from a normal distribution with a variance term capturing variability in skill. Taking the difference between the player-specific skills and the design parameters, then scaling by the error in player skills produces an estimated rate of being hit by enemies in our game. When training simulated models we generated a fixed ideal playtester and allowed the AL model to choose sets of enemy parameters along a 10 x 10 x 10 grid.

Our classification model is a probabilistic model of player responses in terms of underlying control preferences and design parameters. Simulated players have two independent preferences for force and drag parameters; both are sampled from a normal distribution with a variance term capturing variability in preference. When given a set of design parameters the model performs a two-stage comparison process. First, each individual design parameter is compared to the desired parameter for the model by taking a cumulative normal distribution centered at the difference of the parameters and scaled by the player variance term. Differences below the player error threshold yield a "no difference" result; positive or negative differences above the threshold yield "better" or "worse" responses, respectively. Second, the individual parameter responses are combined into the final model response. If both responses are the same then that response is given. If one response is "no difference" then the other response is given. Otherwise the model responds with "neither." When training simulated models we generated a fixed ideal playtester as before and varied sets of controls using a 5 x 5 x 5 x 5 grid. Each test point consisted of both current and previous wave control parameters.

Using these simulation models we performed 10-fold cross-validated experiments to measure how accurately an objective function could predict playtester output given a set of design parameters. For the regression setting the GP was trained to use only the three enemy parameters to predict the squared difference between the number of times the player was hit and the desired rate of 6 times. We square the difference to penalize sets of parameters that differ from the ideal more strongly as they become further from the desired outcome. For the classification setting the objective functions were trained with the current and previous settings for drag and thrust to predict whether the preference rating was "better" or "worse." We discarded ratings that were not in these two classes as human data had too few samples to make a comparison (only 10/57 players ever responded in other categories).

In each cross-validation fold we generated a set of 30 initial randomly sampled design parameter settings and used the simulated models to generate playtester responses. These settings and responses formed the initial training data set. An additional set of 100 testing points was generated in the same way. We then created the grid of points the AL

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

model could sample, forming the training pool. Finally, we created a set of 100 randomly sampled design parameters and Within each fold we then repeated the following process:

1. Train the objective function on the training data set.
2. Test the objective function on the testing data set and measure the error.
3. Use the acquisition function to pick a point from the training pool to improve the objective function model.
4. Generate a player response for that training pool point and remove it from the training pool.
5. Return to the first step and repeat the process.

For regression we measured error as the mean squared error from the desired rate of being hit. For classification we measured error as the F1 score—a combination of model accuracy and coverage of sample data.

5.2 Human Study Methods

To perform an empirical evaluation of our methods we deployed two versions of our game online. We publicized the game through websites and local emailing lists and did not offer compensation to participants. In order 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 reached at least 10 waves of play 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. We only used preference responses during the first 10 waves of play to avoid collecting many positive responses from those who remain engaged in the game.

Our AL evaluation used this dataset with the same process as described for the simulation approaches above. We trained each objective function to predict preference responses using only the control settings for the most recent wave and the control settings for the previous wave. Note that we did not collect preference comparisons for the first wave of the game as players could not yet compare control settings.

For each cross-validation split we constructed a test set of data by randomly selecting a subset of points. After this we then randomly sampled to choose the seed data set from only data remaining after removing the test set. The training pool was the data points remaining after removing the seed data points.

6. RESULTS

Overall our results show AL is a promising approach for reducing the number of playtests needed to train an accurate predictive model. In both simulation studies AL almost always reduced the number of playtests needed. In both human studies AL was able to reduce the number of playtests in some, but not all, cases. No single acquisition function, objective function, or acquisition-objective function pair was optimal across cases. These results align with previous work in AL showing that many data-specific properties impact the results [17]. Below we provide further details with an emphasis on how AL impacted the need for playtesting.

Table 1: Regression experiment: GP mean squared error at 80 or 280 samples with various acquisition functions. Simulation results are in units of 10^{-2}. Lower values indicate better performance.

acquisition function	simulation		human data	
	80	**280**	**80**	**280**
random	1.91	1.36	136	124
variance	1.50	1.39	121	120
PI	8.32	6.38	133	123
EI	1.60	**1.30**	112	127
UCB	**1.48**	1.38	**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.

6.1 Regression

Our regression experiments both show strong results for AL. Having a clear behavioral objective (being hit a number of times in the game) was likely a strong contributor. In both simulation and human data we found GP-UCB and EI to be most effective (Table 1). Both methods explicitly balance exploring and exploiting designs, suggesting many parameter tuning objectives involve a balance between considering alternative parameter settings and refining a given setting.

In simulation all AL methods performed well except for PI (not shown as it distorts the graph scale) (Figure 2). As PI is a pure exploitation strategy it focuses on playtesting parameter settings that are highly certain. Because we are tuning three parameters in a fine-grained space this makes it easy to find many bad parameters and fail to distinguish among many possible alternatives. Roughly 50-60 playtests were needed to tune the three parameters related to player performance against enemies when using GP-UCB, EI, or

Table 2: Comparison of acquisition-objective function combination accuracy with few (80) and many (280) simulated playtest samples.

acquisition function	100 samples			200 samples		
	GP	KSVM	NE	GP	KSVM	NE
random	0.732	0.750	0.762	0.795	0.790	0.858
entropy	**0.791**	0.730	N/A	**0.829**	**0.804**	N/A
QBB vote	**0.790**	0.749	**0.776**	0.819	**0.805**	**0.919**
QBB prob	0.691	0.703	N/A	**0.827**	0.766	N/A
error red	0.691	0.739	N/A	0.785	0.774	N/A
var red	0.717	0.731	N/A	0.798	0.770	N/A

acquisition function	100 samples			200 samples		
	GP	KSVM	NE	GP	KSVM	NE
entropy	**0.059**	-0.021	N/A	**0.033**	**0.014**	N/A
QBB vote	**0.058**	-0.001	0.014	0.024	**0.015**	**0.061**
QBB prob	-0.041	-0.047	N/A	**0.031**	-0.024	N/A
error red	-0.041	-0.011	N/A	-0.009	-0.015	N/A
var red	-0.015	-0.019	N/A	0.003	-0.019	N/A

variance; random sampling required 175 playtests for comparable performance. More playtests marginally improved AL performance.

In human studies GP-UCB and EI both performed well, other AL methods did not (Figure 2). Variance's lower performance is explained by the tendency to focus on highly uncertain parameter settings—in a high-dimensional space it is easy to find many sets of uncertain bad parameters, leading to poorer performance. Over time EI's performance gradually decayed while GP-UCB maintained better performance. As more samples are gathered GP-UCB reduces exploration while EI eventually begins to make poor playtest choices. Approximately 60 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. Methods that balance exploration and exploitation—EI and GP-UCB—show the greatest efficacy across simulation and human studies. 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 can reduce testing needs when using few samples, but that subjective classification is generally more challenging. In simulation data AL methods almost always yielded improvements over random sampling. On human human data QBB methods improved GP and KSVM performance with few samples; all methods (including random sampling) were comparable with more samples. These results suggest AL has benefits in preference choices when data is sparse, but less value when more data is available. Players may differ enough in preferred controls to prevent learning a good model across all players.

In simulation QBB vote and entropy showed strong performance gains across all three objective functions (Figure 3, Table 2). Previous work has found QBB models are best for high variance data sets [17]. Entropy sampling is also effective for high-variance situations. The complexity of our simulation model made it a highly variable function to optimize and demonstrates the value of AL methods that can handle

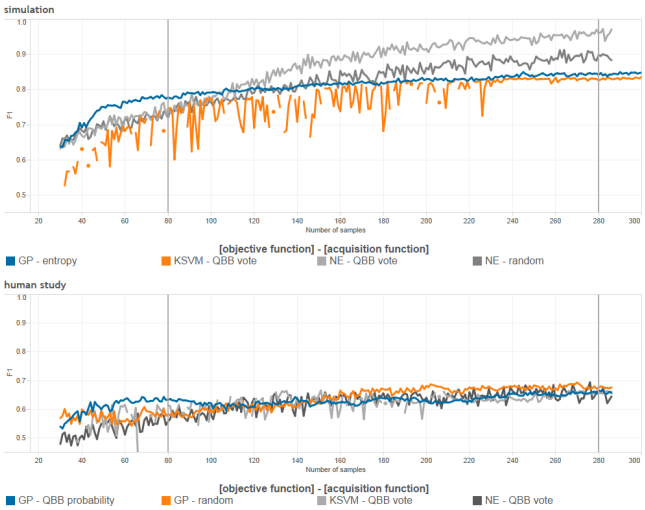

Figure 3: Classification F1 score vs number of samples used in training for different objective and acquisition function combinations. Higher values indicate better performance. Top figure uses simulation data, bottom uses human data.

Table 3: Comparison of acquisition-objective function combination accuracy with few (80) and many (280) human 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

acquisition function	100 samples			200 samples		
	GP	KSVM	NE	GP	KSVM	NE
entropy	**0.043**	**0.046**	N/A	-0.010	0.042	N/A
QBB vote	**0.038**	**0.062**	**0.030**	0.007	0.068	**0.042**
QBB prob	0.029	0.040	N/A	**0.019**	**0.074**	N/A
error red	**0.041**	0.018	N/A	**0.023**	**0.065**	N/A
var red	-0.060	-0.017	N/A	-0.046	0.016	N/A

these situations. Among objective functions NE showed the best performance; GPs had strong early performance before plateauing to similar performance as KSVMs.

In human studies only GPs (with QBB vote or probability) and KSVMs (with QBB vote) improved over random sampling and only with few samples (Table 3). These results suggest AL is able to mitigate high variability when possible but using a greater portion of the total data forces additional noise that cannot be mitigated. When using random sampling GPs performed best, followed by NE and then KSVMs. As GPs are designed for smaller sample sizes these results are not surprising. The relatively poor performance of NE here suggests the model was overfitting simulation data and unable to accommodate the noise in human results.

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, however, requires acquisition functions (e.g. QBB or entropy) that mitigate the high variance inherent in preference response data. Exploring methods that learn

multiple possible distinct control settings is a promising direction for future work.

7. CONCLUSIONS

We have shown how low-level design parameter tuning playtesting can be automated. Using AL techniques can reduce the expense of playtesting in terms of the number of input points (designs) tested. 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. In the future we hope to see automated techniques for other aspects of design like 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.