Automatic Playtesting for Game Parameter Tuning via Active Learning

anonymous

ABSTRACT

Game designers often 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. We argue that active learning techniques can be used to 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" [5]. 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 central insight is to model playtesting as a form of active learning (AL). Active learning [17] 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." 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

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 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 testing with late-stage refinement.

Foundations of Digital Games ???

In the remainder of the paper we first compare our approach to related efforts at 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. Examples include support for generating or evaluating design in terms of hard constraints (what must be true) or soft optimization (what to achieve to the best degree possible). Online game adaptation changes game designs in real-time based on player actions or game state. Examples include designer authored rules or data-driven player models coupled to optimization techniques.

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. [1] and Cook et al. [4] use sampling methods to evaluate platformer game level playability. Shaker et al. [18] 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. Active learning can enhance simulation approaches by guiding the sampling process toward spaces of parameters likely to be of greater value.

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. [20], Butler et al. [3], and Horswill and Foged [8] employ logic programming and constraint solving to generating levels or sets of levels meeting given design constraints. Jaffe et al. [10] employ game-theoretic analysis to understand the trade-offs of game design parameters for balance in a competitive game.

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 [9] 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. [12], El-Nasr [16], and Thue et al. [22] 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. [7], Shaker et al. [19], Liapis et al. [11] and Yu and Riedl [24] 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 [6] optimize player retention and Zook and Riedl [26] optimize game difficulty using similar techniques.

Automated playtesting extends these approaches by providing principled methods for guiding 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 that research to date has ignored the problem of reducing "sample complexity"—the number of data points (human tests) needed to train a model. Active learning makes explicit the trade-off in playtesting between "exploring" potentially valuable game design settings and "exploiting" known good solutions with small changes. 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.

not sure where to slot this in

Our approach complements prior uses of AL for game design by focusing on efficiently improving designs for player behavior and experience. Normoyle et al. [13] use AL to recommend sets of useful player metrics to track; we pick design settings to improve player experience. Rafferty et al. [14] optimize game designs offline to learn the most about player cognition; we use an online process and explore a variety of alternative AL approaches. We extend these efforts with a wider variety of AL metrics and tasks while addressing the cost of playtesting.

3. 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 the following sections we will define AL in the playtesting context and provide the intuition behind several common AL methods. Our aim is to characterize playtesting in terms of AL; full mathematical treatments are available through the references.

3.1 Playtesting as Active Learning

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). 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.

3.2 Regression Models

We consider four acquisition functions for regression models: (1) variance, (2) probability of improvement, (3) expected improvement, and (4) upper-confidence bounds. The *variance* acquisition function picks designs based purely on the uncertainty of the design goal value [2]. Intuitively, variance encodes a playtesting strategy of attempting any design that is poorly understood until the entire design space is well-understood. Variance approaches are pure exploration and are useful when design information on many possibilities is needed.

Probability of improvement (PI) improves on uncertainty by selecting the design that is most likely to gain some improvement of the best design observed so far [2]. Intuitively, probability of improvement is a playtesting strategy of attempting the design that seems most likely to improve over what has been tried so far. Probability of improvement approaches are pure exploitation and are effective when performing minor tweaks to already-effective designs.

Expected improvement (EI) enhances PI by weighting the probability of each design improvement by the amount of improvement [2]. Unlike probability of improvement or variance, EI balances between exploration and exploitation. Intuitively, EI seeks designs that are likely to have large improvements over the existing design. By balancing exploration and exploitation EI is a more general-purpose technique useful for playtesting during many stages of the development cycle.

Upper Confidence Bound (UCB) methods select designs based on the upper bound on design quality according to a model of design quality and variability in that quality [21]. Designs that are expected to be high quality but are also poorly understood (and thus highly variable) are given priority. By testing these designs UCB narrows down the space of designs to those known to be good or unknown but likely to be bad. In the UCB approach cited above the importance of variability is gradually scaled down as more playtests are run. Intuitively, this capture the notion of initially exploring a variety of possible designs and gradually converging to exploit the most promising designs.

3.3 Classification Models

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. *Entropy* acquisition functions choose designs that are most uncertain according to entropy [17]. Entropy measures the amount of information needed to encode a distribution. Intuitively, this captures how uncertain a model is about the possible (discrete) outcomes. Entropy sampling acts similarly to variance-based sampling for regression.

Query-By-Bagging (QBB) approaches emphasize disagreement among potential models [17]. First, multiple copies of the same objective function model are trained using random selections of the playtest data gathered so far. This captures the notion of guessing how the model might be if slightly different sets of playtests had occurred. Second, these models predict the outcome of new playtests. The final choice of design to test is the one these models most disagree on. Intuitively, this captures the idea of picking designs that are most likely to be poorly understood.

Two varieties of QBB are relevant: QBB vote and QBB probability. In *QBB vote* each objective function model votes on outcomes, giving a single choice. When only two options are possible (e.g. new controls were better or worse than previous) the design with the greatest disagreement between two binary outcomes is chosen. When multiple options are possible (e.g. a multi-way branch in a choose-your-own-adventure), the design with the greatest difference between the top two options is chosen. In *QBB probability* each objective function model estimates how probable each outcome is, giving a weight to all choices. The weights of all models are averaged and the point with lowest probability is chosen.

find citation

Expected error reduction and variance reduction attempt to maximize the quality of objective function predictions, rather than reduce variance or disagreement [17]. In *ex-*

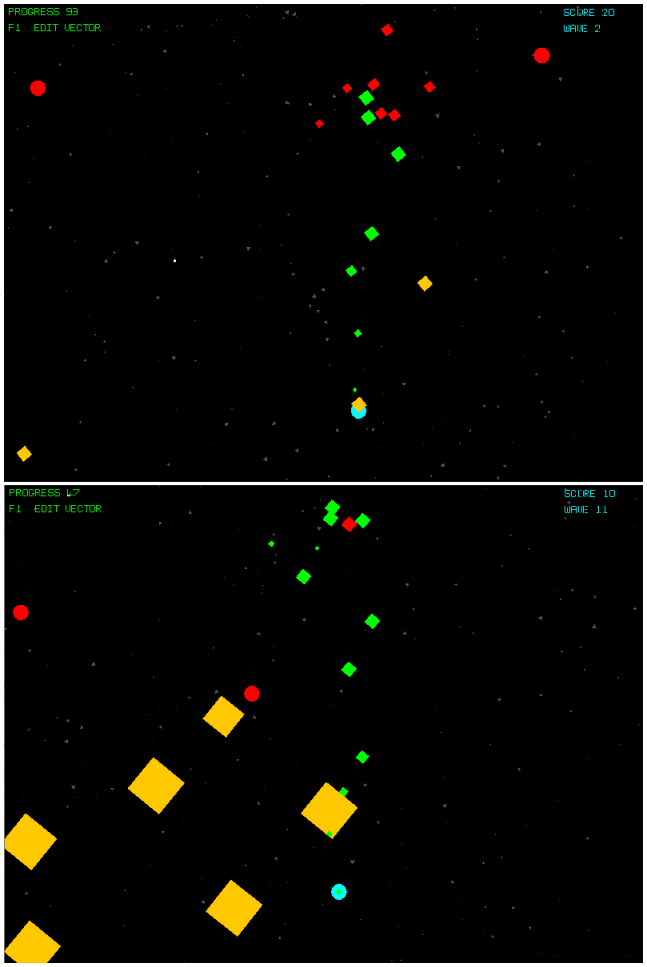

Figure 1: Study game interface illustrating player, enemies, and shots fired by both at two points along adaptation process.

pected error reduction an unlabeled data point is selected and assigned one of the possible labels. This fake data point is used to train a new objective function model and the error of that model is calculated over all remaining unlabeled data points. Expected error reduction then picks the data point that results in a model with the least average ("expected") error when considering all possible label assignments. Intuitively, expected error reduction is guessing how a new playtest could result and picking the playtest that will most improve the model of how people play. *Variance reduction* performs a similar process but minimizes the variance of predictions over data points, rather than the error. Intuitively, a the model is picking a playtest most likely to increase certainty about how future playtests will go.

4. GAME DOMAIN

In order 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: player game play behavior goals and 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 is based on 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 sought to test whether the AL framework 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 [17]. 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 tuning enemy parameters or tuning player controls. 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 each study we employed three different objective functions— Gaussian Processes (GP), kernel support vector machines (KSVM), and optimized neural networks ("neuro-evolution", NE). We chose these objective functions as they cover several player modeling approaches used in this domain: GPs are a common model in AL contexts [2], kernel methods (particularly KSVMs) are a popular machine learning technique previously used in player modeling [25], and optimized neural networks have been widely used in preference learning [23].[1] All three of these objective functions can be trained in regression or classification modes. KSVMs do not have probabilistic predictions in regression mode and NE do not produce probabilistic predictions, limiting the acquisition functions they are 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 having more 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 controls all players agree on as good is difficult to obtain. Combined across experiments, 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 paramters 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 rate of making errors. 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 along 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. Playtest 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. Every objective function 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.

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 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.

[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.

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 our analysis we only retained data from players who reached at least 10 waves of play total. This subsetting 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. 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 retained only 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 after 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 [15]. Below we provide more details on these results with an emphasis on how AL impacted the need for playtesting.

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.

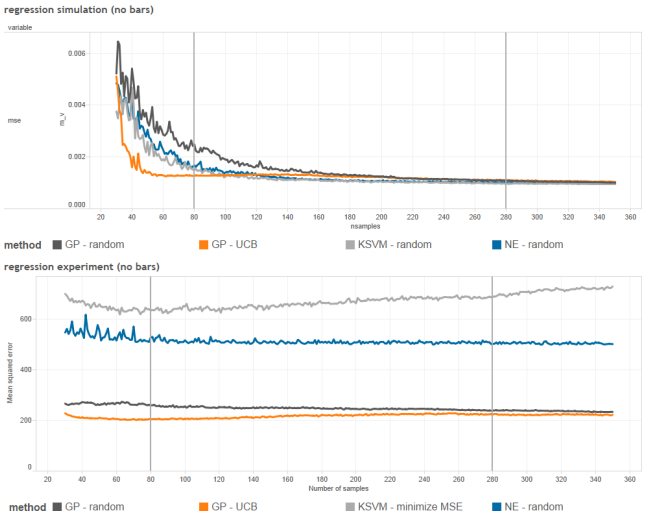

Figure 2: Regression mean squared error on predicting number of times player is hit vs number of samples used in training for different objective and acquisition function combinations. Lower values indicate better performance. Top figure uses simulation data, bottom uses human data.

In both simulation and human data we found GP-UCB and EI to be most effective. Both methods explicitly balance between exploring and exploiting designs, suggesting many parameter tuning objectives involve a balance between considering alternative parameter settings and refining a given setting.

Table 1: Comparison of acquisition-objective function combination accuracy with few (80) and many (280) playtest samples of simulated data (units 10^{-2}).

acquisition function	MSE 80 samples			MSE 280 samples		
	GP	KSVM	NE	GP	KSVM	NE
random	1.91	1.58	1.62	1.36	1.29	1.31
variance	1.50	-	-	1.39	-	-
PI	8.32	-	-	6.38	-	-
EI	1.60	-	-	1.30	-	-
UCB	1.48	-	-	1.38	-	-

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 a bad set of parameters and fail to distinguish among many possible alternatives. Roughly 40 playtests were needed to tune the three parameters related to player performance against enemies when using GP-UCB, EI, or variance; random sampling require 175 playtests for comparable performance. More playtests marginally improved AL performance.

[2]To avoid clutter only the best acquisition function results are shown for each objective function, along with the best-performing random sampling (among objective functions) as a baseline for comparison.

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

acquisition function	MSE 80 samples			MSE 280 samples		
	GP	KSVM	NE	GP	KSVM	NE
random	136	379	268	124	374	261
variance	121	-	-	120	-	-
PI	133	-	-	123	-	-
EI	112	-	-	127	-	-
UCB	107	-	-	117	-	-

In human studies GP-UCB and EI both performed well, other AL methods did not (Figure 2). We focus our analysis on the GP objective function as NE for regression was unable to outperform a random sampling approach which was worse than all GP method results. Gaussian Processes were developed for regression settings so it is not surprising that neural networks—developed for classification—would show relatively poor performance.

Variance's low 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 is better able to reduce exploration while EI eventually begins making 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 is a clear demonstration that 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 be effective but is more challenging in subjective domains. In simulation data AL methods—particularly entropy sampling—show significant improvements over random sampling. Human data did not show strong improvements for AL methods and only marginal performance improvements for random sampling. These results suggest AL works for preference learning when the domain can be modeled as a preference choice, but that our human data was not of this sort.

In simulation QBB vote and entropy sampling showed strong performance gains across all three objective functions (Figure 3). Previous work has found QBB models are best for high variance data sets [15]. 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 these situations.

In human studies only specific acquisition and objective function combinations achieved substantial improvements

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

acquisition function	MSE 80 samples			MSE 280 samples		
	GP	KSVM	NE	GP	KSVM	NE
random						
QBB vote						
QBB prob						
entropy						
error red.						
var. red.						

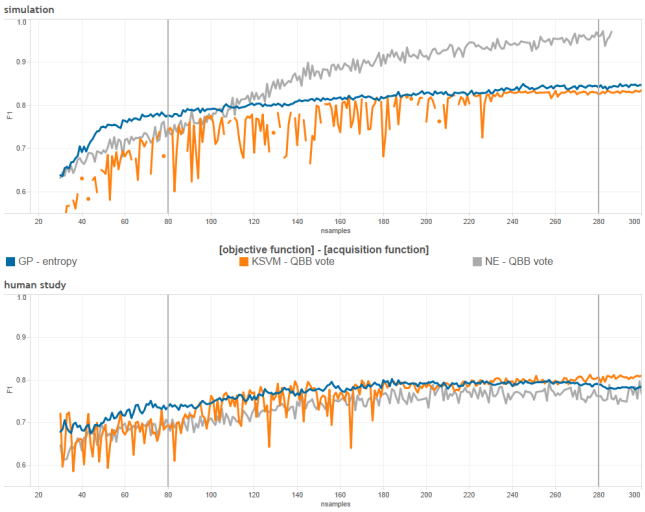

Figure 3: Classification F1 score vs number of samples used in training for different objective and acquisition function combinations. Higher values indicate better performance. Top figure uses simulation data, bottom uses human data.

over random sampling (Figure 3). Gaussian Processes showed strong early (by 80/300 maximum samples) learning with entropy-based sampling. All other GP sampling strategies (including random) reached competitive performance with more (160) samples. Kernel SVMs showed a trend of steady improvement across all methods. Entropy alone showed weaker performance—at the level of random sampling—and only with large sample sizes. Neural network optimization failed to improve with non-random sampling strategies.

Across methods GPs with entropy-based sampling showed the strongest performance with few samples while KSVMs with decision-boundary sampling was strongest with more samples. Together these results suggest GPs are excel with few samples by rapidly exploring potentially valuable options. Gaussian Processes also showed the best baseline performance with random sampling, meaning they effectively model the problem domain at hand and likely contributing to their strong performance with fewer samples. With more samples KSVMs become more effective—in general Bayesian methods (e.g. GPs) tend to show strong performance with little data before converging to similar results as non-Bayesian approaches (e.g. KSVMs). That GPs showed somewhat worse performance with large sample sizes (0.76 vs 0.80 F1 scores) merits further investigation but may be

Table 4: Comparison of acquisition-objective function combination accuracy with few (80) and many (280) playtest samples.

acquisition function	MSE 80 samples			MSE 280 samples		
	GP	KSVM	NE	GP	KSVM	NE
random						
QBB vote						
QBB prob						
entropy						
error red.						
var. red.						

partially due to the small number of total samples employed. Our classification experiments 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. entropy sampling or query by bagging) that mitigate the high variance inherent in subjective response data.

7. CONCLUSIONS

8. ACKNOWLEDGMENTS

9. REFERENCES

[1] 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.

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

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.