Learning-Based Procedural Content Generation

Jonathan Roberts and Ke Chen, *Senior Member, IEEE*

***Abstract*—Procedural content generation (PCG) has recently become one of the hottest topics in computational intelligence and AI game research. While some substantial progress has been made in this area, there are still several challenges ranging from content evaluation to personalized content generation. In this paper, we present a novel PCG framework based on machine learning, named *learning-based procedure content generation* (LBPCG), to tackle a number of challenging problems. By exploring and exploiting information gained in game development and public player test, our framework can generate robust content adaptable to end-user or target players on-line with minimal interruption to their gameplay experience. As the data-driven methodology is emphasized in our framework, we develop learning-based enabling techniques to implement the various models required in our framework. For a proof of concept, we have developed a prototype based on the classic open source first-person shooter game, Quake. Simulation results suggest that our framework is promising in generating quality content.**

***Index Terms*—Content categorization, first person shooter, machine learning, on-line adaptation, player categorization, procedural content generation, public experience modeling, Quake.**

I. Introduction

THE video games industry has been expanding rapidly and even surpassed the movie industry in revenue [1]. The expectations of consumers have gradually increased to the point where players and critics demand cutting edge graphics, immersive game play and strong replay value from new releases. Generating game content is costly, as it requires many different skill sets and long periods of development. For example, generating the skeletal animation for a single character can take several months. It is therefore of the upmost importance to create content in timely and efficient manner.

Procedural content generation (PCG) is the process of generating content for a video game automatically using algorithms. A wide variety of content can be generated, e.g., art assets such as terrain and textures and even high-level game play structures such as storyline and characters. PCG not only has the potential to provide a basis built upon by developers but also can provide an endless stream of content for a player to extend the lifetime of the game. If used properly, it can reduce the amount of resources required to build a game and thus drive down costs. Although PCG often seems like a good approach, it may be very tricky to get right and could actually be more trouble than creating handcrafted content [2].

In recent years, a variety of approaches have been proposed to improve PCG. Most of such approaches generally fall under the umbrella of *search-based procedural content generation* (SBPCG), a terminology coined by Togelius *et al.* [3]. Given a potential game content space, SBPCG employs evolutionary or other metaheuristic search algorithms to explore the space effectively to find content appealing to players. This can result in more robust and trustworthy PCG algorithms that require far less manual configuration than traditional approaches. In the past few years, researchers have used different SBPCG techniques to adapt levels for games such as 2D platformers [4]–[7], first-person shooters [8], rogue-like games [9], racing games [10], and real-time strategy games [11] [12]. Some of the latest work in PCG was collected in a special issue of this journal [13]. Although substantial progress has been made in PCG, existing PCG techniques still encounter a number of challenging problems such as content representation, content space management, content quality evaluation, content generation efficiency and so on, which have been thoroughly summarized in [3].

It is well known that the same game content can elicit various emotional and cognitive experience for players of different types. As a result, *experience-driven procedure content generation* (EDPCG) [14] has become an active area in PCG research. The ultimate goal in the EDPCG is enabling game engines to generate personalized content that optimizes players' affective/cognitive experience. As pointed out by Yannakakis and Togelius [14], two main technical challenges in EDPCG are how to effectively model players' emotional and cognitive experience and how to efficiently find out quality content that optimizes experience according to their experience models. While different approaches to the EDPCG have been studied [14], there are still several open problems pertaining to two challenges in general, ranging from interruption of gameplay experience in data collection to optimization of the EDPCG algorithms in personalized content generation.

Machine learning is a data-driven methodology for knowledge acquisition/modeling and has been successfully applied in many different AI areas ranging from machine perception to natural language understanding. As pointed out by Lucas *et al.* in the preface of their edited book [15], "It would seem natural for an academic researcher to think that there were ample applications for learning algorithms in computer games. However, it is rather rare to see machine learning used in any published games ...". As reviewed in [16], the predominant learning technique used in computer games is still evolutionary learning while many versatile learning paradigms and algorithms have

Manuscript received August 29, 2013; revised February 24, 2014; accepted June 28, 2014. Date of publication July 11, 2014; date of current version March 13, 2015.

The authors are with the School of Computer Science, The University of Manchester, Manchester M13 9PL, United Kingdom (e-mail: robertsj@cs.manchester.ac.uk; chen@cs.manchester.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2014.2335273

yet to be explored so far. To the best of our knowledge, there is little work that systematically investigates the application of machine learning in PCG apart from few successful studies, e.g., [5].

Motivated by the typical commercial video game development cycles, we present a novel PCG framework by means of machine learning, termed the *learning-based procedural content generation* (LBPCG), in this paper. To address several challenging issues in PCG, our LBPCG explores and exploits knowledge and information gained from game developers and public testers via a data-driven methodology towards minimizing interruption to the end-user gameplay experience. Relying on generalization of component models trained with inductive learning, our framework tends to direct a game content generator towards generating robust yet appealing content to an arbitrary player. As a result, the LBPCG has several salient characteristics including avoiding hard coded content evaluation functions, effectively limiting a search space to the relevant content likely to elicit players' specific cognitive and affective experience for adaptivity, and minimizing interference to the end-user gameplay experience.

Our main contributions in this paper are summarized as follows: a) we propose a learning-based PCG framework to tackle a number of challenging problems in PCG; b) we develop learning-based enabling techniques to support the proposed framework; and c) we apply our proposed framework to a classic first-person shooter game, *Quake* [17], for a proof of concept, which may generate quality content appealing to the end-user as demonstrated in our simulation.

The rest of paper is organized as follows. Section II reviews the previous work that motivates our studies. Section III presents our LBPCG framework, and Section IV describes the enabling techniques to support the LBPCG. Section V reports simulation results. The last section discusses relevant issues.

II. Motivation

Commercial video game development is usually very well-structured, incorporating many work-flows for design, testing and deployment. Most companies have developers consisting of programmers, designers, artists, production managers and testers. Games will go through many iterations, where features are added, tested by engineers, then released for the test teams. After a game has reached a certain level of maturity, it is sometimes released to public testers. Besides bug finding, the other reason behind doing this is that in the games industry it is common for developers to be happy with the game they have created, but public opinion can be very different. Garnering public opinion is useful and can avoid the disastrous event of releasing a bad, expensively made, game to the public. After all production tests pass, the game goes "gold" and is released to the publishers for distribution to the end-user (target players).

It is well known that the risk of catastrophic failure and a lack of quality are among common concerns for existing PCG techniques [3] given the fact that a content generator may generate unplayable content or content that is not appropriate for anyone's tastes. Illegitimate content has some extremely undesirable property such as being unbeatable, e.g., a map that consists of rooms full of monsters that immediately kill the player no matter their skill level. Some efforts have been made to ward off this problem. For example, playability/acceptability issues have been formulated as a constrained content optimization problem [6], [11] that can be solved by the feasible-infeasible two-population genetic algorithm. While such methods effectively generate quality content, the fitness function on constraints has to be handcrafted and the content generation process might be less efficient due to intensive computing involved in the population-based evolutionary computation in general. In addition, studies in [7], [9] describe methods of combining off-line and online models, one for optimizing playability/winnability and the other for optimizing challenge/skill, with some success. The integrated model may avoid catastrophic failure and then adapt the generated content to suit a specific player online. Although some progress was made in an experiment with Infinite Mario [7], it is still unclear to what extent models of different functionality and purpose can be combined to form robust composite models.

In general, personalization of content for a target player is driven not only by factors such as playability and skill but also a player's affective and cognitive experience [14], which poses another big challenge to PCG. To generate the favored content for target players, players' styles/types need to be identified correctly and their cognitive/affective experience must be measured accurately. Obviously, it is undesirable to interfere with the gameplay experience of end users or target players, so asking them any questions such as "how much fun was that content?" should be avoided. To our knowledge, however, most of existing methods need to identify the target players' styles/types and to measure their cognitive/affective experience by learning from their feedback [19], [20] or behavior [14], which leads to a burden to target players. Moreover, existing methods often utilize predefined player types/styles, e.g., traits or personalities in terms of psychology, to categorize target players explicitly, which might not encapsulate all player types and also lead to difficulty to infer such predefined yet less operational players' types/styles from a computational perspective. Furthermore, players' subjective feedback could be quite noisy and inaccurate, which exacerbates difficulties.

To avoid modeling target players directly, it is possible to learn a generic playing style/type model from public testers' feedback and behavior [21], [22]. Nevertheless, data collection in such methods often undergoes an expensive and laborious process, as collecting data from members of the public often intuitively seems like the best way to build up realistic models of enjoyment in humans. Recently, a system was built up in order to predict human behavior based on gameplay statistics produced via a pool of 10 000 players [22]. Their study concluded that one of challenges faced in such work was the existence of many outliers, people who behave in an irregular fashion [22]. When crowd-sourcing is applied to PCG, how to deal with noisy feedback and outliers becomes an ongoing critical issue.

During game playing, it is possible that a players' notion of fun (or whatever other attribute is being optimized) may drift over time. This issue was touched upon in a recent study aiming to create an adaptive Super Mario [5]. On the one hand, their system was fairly successful in a test by switching the agent that

content is being optimized for half-way through. With the same experimental protocol on human players, however, it was reported that only 60% of players preferred their dedicated system over a purely random system [5]. Thus, it is yet another open problem in PCG to safeguard online algorithms against players who may change their mind over time.

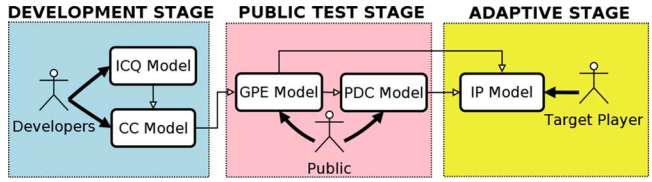

Fig. 1. Learning-based procedure content generation (LBPCG) framework.

III. LBPCG Framework

In this section, we first specify our problem statement resulting from our analysis of the challenges in state-of-the-art PCG and then propose a framework to address those issues. To illustrate our proposed framework, we further exemplify our general idea with a first-person shoot game, Quake.

A. Problem Statement

Motivated by commercial video game development cycles and the previous work reviewed in Section II, we formulate a number of challenging problems in PCG:

1) how to encode developers' knowledge on the content space represented via a game parameter vector to avoid completely undesirable content;
2) how to enable developers to self-define content categorization criteria to facilitate player categorization in an implicit yet flexible way;
3) how to learn robust yet "true" generic players' experience from noisy feedback and outlier detection;
4) how to exploit the information extracted from public testers' behavior and cognitive/affective experience and content categorization self-defined by developers to predict target players' preferred content while minimizing interruption to them,
5) how to tackle the concept-drift problem for target players, due to the change of players' experience, in the online adaptation.

While the aforementioned problems may be studied separately, we propose a *learning-based procedural content generation* (LBPCG) framework to tackle all the issues systematically.

B. Model Description

To tackle the issues specified in Section III-A, we believe that a process that mimics commercial video game development can provide an effective solution to PCG and therefore would divide a video game life cycle into three stages:*development* (involving developers), *public test* (involving public testers of different types) and *adaptive* (concerning target players). By taking this development process into account, we propose the LBPCG framework, which encodes developers' knowledge on game content in the development stage, models public players' experience in the public test stage and generates the content appealing to target players in the adaptive stage.

In the development stage, we specify two models, the *Initial Content Quality* (ICQ) and the *Content Categorization* (CC) to encode developers' knowledge, as depicted in Fig. 1. The ICQ model is designed to filter out illegitimate and unacceptable content of poor quality, and then the CC model is used to further partition the legitimate/acceptable content space with predefined content features by developers into meaningful subspaces that are likely to elicit various affective/cognitive experience for players of different types. We argue that these two tasks should be done by developers. On the one hand, playability/acceptablity assurance is essential in game development before releasing the product to any users. On the other hand, we believe that it is useful to discover salient content features somehow responsible for eliciting distinct cognitive/affective experience for players of different types/styles. By using such features, developers can flexibly categorize content and then associate the content of a specific category and typical playing behavior acting on the games in this category with certain styles/experience of interest. By doing so, the search space can be effectively limited for the personalized content generation. This is a key idea underlying the LBPCG framework, and the CC model provides the underpinning technique for other component models to carry out this idea, as described later. In general, the problems in the ICQ and CC models might be tackled in different ways, e.g., developers explicitly encode their knowledge into rules or constraints. However, developers may not fully understand a whole complex content space as there are often many game parameters and a formidable number of parameter combinations. In the LBPCG, we advocate the learning-based methodology to encode developers' knowledge implicitly instead of requiring developers to explicitly construct rules. We anticipate that the use of a data-driven methodology not only leads to an alternative approach to content quality control and content categorization but also could help developers better understand a complex game content space.

Public user testing has turned out to be one of the most important cycles in modern game development to enhance the final game product [23], [24]. Motivated by this development cycle, we propose two models, the *Generic Player Experience* (GPE) and the *Play-log Driven Categorization* (PDC), in the public test stage. As shown in Fig. 1, the GPE model is used to capture public players' behavior and feedback on their experience by playing a number of representative games well selected from different categories defined in the CC model. As public opinions may be different from developers' and their feedback is often subjective and noisy, the GPE model works on finding a "genuine" consensus on each selected game and assessing the conformity, i.e., similarity of an individual' experience to general public's on the same games played, to each public tester who provides feedback. By such a crowd-sourcing learning, the GPE model is expected to find the "popularity" of any acceptable game and to detect "outliers" in the cohort of public testers. The PDC model is designed to model cognitive/affective experience based on not only public testers' playing behavior but also the category of game content that elicits the experience,

which forms the key idea in our LBPCG. If carefully selected public testers involved in the GPE model well represent target players of different styles/types, the PDC model can be used in the adaptive stage to predict whether a target player prefers a specific category of games played based on their behavior after they have played a game in the category. As the GPE and the PDC models need to extract information and to model the experience from data, respectively, the data-driven learning naturally provides enabling techniques to solve those problems.

In the adaptive stage as shown in Fig. 1, we come up with the *Individual Preference* (IP) model to control a content generator with four models created in the development and the public test stages. The IP model works towards producing their preferred content online for a target player by minimizing interruption to their gameplay experience. By means of all the four underpinning models, the IP model deals with four nontrivial issues during content generation as follows: a) automatically detecting the categorical preference of a target player with the PDC model and the games selected by developers during the creation of the GPE model; b) assuring quality in subsequent generated games within a specific content category with the ICQ, the CC and the GPE models once the player type is determined; c) automatically detecting when concept-drift occurs with the PDC model and tackling it effectively, and d) dealing with crisis situations autonomously with a system failure avoidance mechanism. While existing techniques [3] may be used to carry out the IP model as discussed later on, we would develop more efficient techniques to enable the IP model to generate content online.

In summary, the motivations and the functionalities of the five component models in the LBPCG framework are listed in Table I. In Section III-C, we exemplify how to develop a LBPCG system to achieve the synergy among component models with a first-person shooter game, Quake [17]. This will be also used to facilitate presenting our learning-based enabling techniques in the next section.

TABLE I
SUMMARY OF THE LBPCG COMPONENT MODELS

Model	Motivation	Function
Initial **C**ontent **Q**uality (**ICQ**)	Content quality assurance via an evaluation function that filters out non-playable/unacceptable games of poor quality for any players.	**Input**: Game parameter vector **Output**: Yes/No decision with the confidence information
Content **C**ategorization (**CC**)	Categorizing content via an evaluation function that maps an acceptable game to a specific category defined by developers in order to facilitate the PDC model in categorizing target players and the IP model in limiting search space.	**Input**: Game parameter vector **Output**: Categorical label with the confidence information
General **P**layer **E**xperience (**GPE**)	Collecting behavior and feedback from public testers who play the games in different categories selected by developers. Estimating the feedback consensus on games played by multiple public testers and estimating the conformity of public testers with their feedback.	Data collection and crowdsourcing learning **Input**: Game parameter vector and feedback of public testers **Output**: Public feedback consensus on the games played and public testers' conformity
Play-log **D**riven **C**ategorization (**PDC**)	Categorizing target players via an evaluation function that predicts their experience based on their behavior and categorical information of games played (the positive experience implies that a target player favors games in the category).	**Input**: Game category and behavior (captured in play-log) **Output**: Positive/negative decision with the confidence information
Individual **P**reference (**IP**)	Applying other LBPCG component models to control a content generator to produce online content appealing to target players based on their historical behavior on games played before. Tackling crisis situations in content generation whenever the preferred content category fails to be detected.	-Detecting the preferred content category for a target player -Generating quality yet personalized content once the preferred content category is found -Detecting and tackling concept drift during gameplay -Avoiding a system failure

C. Exemplification

As illustrated in Fig. 2, the first-person shooter Quake [17] is a highly acclaimed open source game, which can be flexibly modified for different research purposes. OBLIGE [25] is a map generator for various Id Software games including Quake, which provides a command-line API that allows parameters to be set to control high-level attributes such as monster and weapon frequencies. For demonstration, we choose nine essential OBLIGE parameters to form a content description vector, including skill level, overall monster count, amount of health packs, amount of ammo, weapon choice and four parameters consolidating the control of different types of monsters, while we fix other parameters to a sensible value, e.g., the median value. Four parameters divide the various monsters into four monster sets: set 1 consists of Soldier and Dog, set 2 is composed of Knight, Scrag and Zombie, set 3 contains Fiend and Ogre, and set 4 has Shambler. The sets generally increase in difficulty, e.g., Shambler in set 4 is very hard to beat while Dog in set 1 is very easy to kill. With this subset of chosen parameters and the random seed, the content vector space consists of a total of 116 640 core games, with infinitely many variations on aesthetics and layout. To capture the gameplay behavior, we have made various changes to

Fig. 2. The classic first-person shooter Quake.

Quake engine such as the ability to output play-logs consisting of 122 features. These features include statistics such as the average number of monsters was killed per game tick, how much the mouse was moved in each direction and how many monsters of each type were killed, and so on. Once a game is played by a player, their gameplay behavior is automatically recorded in a play-log.

The ICQ model is designed to filter out unacceptable content of poor quality. OBLIGE may produce unacceptable content, e.g., some games have too many monsters but too little ammo. Therefore, the task of the ICQ model in Quake is to eliminate games of low quality by carving out a manifold of

acceptable content in the original content space of nine parameters. To encode developers' knowledge in an implicit way, we adopt the learning-based methodology. That is, by playing a few well-selected games, a developer labels them as either acceptable or unacceptable based on their knowledge. For annotated games, their parameter vectors and labels assigned by the developer are used as examples to train a binary classifier. Then the classifier would be able to evaluate any game parameter vector to decide its acceptability. To establish such a classifier, we develop a novel active learning method presented in the next section.

The CC model is designed to categorize the acceptable content in terms of content features self-defined by developers. For demonstration, we choose a generic feature: "difficulty" of five categories (Very Easy, Easy, Moderate, Hard, and Very Hard) for content categorization in Quake as we believe that this is likely to be something that most people will have a preference for. Note that for learning in the ICQ and the CC models, the only difference is that the ICQ model works on all games while the CC model takes only surviving ones, those passing the ICQ examination, into account. As presented in Section IV-B, the same active learning method as used in the ICQ learning may be applied to the CC learning while a multiclass classifier is required to assign one of five difficulty categories to any acceptable games.

In the GPE model, developers need to first select representative games of quality from each of five difficulty categories. For demonstration, we choose 100 representative games by fixing the random seed with the ICQ and the CC models; there are 20 games in each of five difficulty categories. Then 100 games are released to public testers. Ideally, the public testers should be a carefully selected cohort of players who are representative of different target players and each test player should play at least one game from each category. The gameplay behavior is recorded in the play-log for each game played. After playing each game, a public tester gives their feedback by answering some questions regarding the experience of interest, e.g., "Did you enjoy the game?". Once the data collection is completed, the GPE model uses a crowd-sourcing learning algorithm, presented in the next section, to create a game "popularity" evaluation function that gives the "popularity" scores to 100 selected games and to assign the conformity to each public player who provides feedback. Consequently, each of 100 selected games receives a score reflecting its "popularity" and each of participated public testers is assigned a confidence level indicating their conformity for outlier detection.

In the PDC model, the main task is to learn a mapping from both the categorical label of a game played by a public tester and their play-log to the experience of interest. In Quake, each survey in the data collection contains the difficulty category of a game played, the play-log of 122 attributes and a binary feedback (fun/not fun) given by the public tester. Thus, the information in each survey is used as an example to train a binary classifier with the input of 122 play-log attributes plus one categorical label to predict the positive/negative experience. For robustness, the conformity information gained in the GPE model is further exploited to deal with "noisy" examples coming from "outliers". In doing so, we develop an ensemble learning algorithm to establish the PDC classifier by taking the conformity information into account, as presented in the next section.

In the adaptive stage, the IP model is required to generate the content appealing to a target player online by applying all four component models strategically. The IP model is developed based on the following scenario. After a target player has played few of the same 100 games used in the public survey, their content categorical preference is determined via the PDC model that predicts the experience for each game played. By means of the ICQ, the CC and GPE models, quality games with variation in their preferred content category are generated subsequently. The PDC model always monitors any games played by a target player to test whether concept-drift occurs. Once a change is detected, the initial detection process is repeated for the target player to identify their new preferred content category. In case the PDC model fails to detect the preferred content category for a target player, the content "popularity" evaluation function achieved in the GPE model along with the ICQ model will be employed to deliver a stream of quality games that are likely to be enjoyed by most players.

IV. Enabling Techniques

In this section, we present our learning-based enabling techniques to support our LBPCG framework. For each model, we formulate the problem, present our solution and demonstrate how to be applied to Quake described in Section III-C.

While there are various content representations [3], our current work is confined to a class of representations characterized by a game parameter vector of D parameters denoted by $\boldsymbol{g} = (g_1, g_2, \cdots, g_D)$, which defines a content space $\mathcal{G}$ where $\boldsymbol{g} \in \mathcal{G}$. Furthermore, we assume that the gameplay behavior can be recorded by a play-log $\boldsymbol{l}$ of L event attributes denoted by $\boldsymbol{l} = (l_1, l_2, \cdots, l_L)$. Developers may choose F ($F \geq 1$) content features denoted by $\boldsymbol{c} = (c_1, \cdots, c_F)$ for content categorization. For example, in Quake, a game is characterized by $D = 9$ parameters in OBLIGE so that the content space contain $|\mathcal{G}| = 116,640$ core games, a play-log consists of $D = 122$ attributes and the developers choose $F = 1$ content feature, $c_1 =$ "difficulty" of five categories, for content categorization.

A. ICQ Learning

In general, the problem in the ICQ model is finding out a mapping for any $\boldsymbol{g} \in \mathcal{G}$ so that $\Phi_{ICQ} : \boldsymbol{g} \rightarrow \{+1, -1\}$ where $+1/-1$ indicates the parameter vector, $\boldsymbol{g}$, leads to acceptable or unacceptable content, respectively.

As developers are a limited yet expensive resource and a game content space is often too large to be exhaustively classified, the ICQ learning has to address two critical issues: how to select a "minimal" number of representative games for annotation and how to establish a classifier of good generalization. Active learning provides an effective way to combine annotating data and training a learner without need of many labeled data [26], while clustering analysis may discover the structure of data space [27]. We propose an active learning algorithm based on clustering analysis to address two critical issues, as summarized in Algorithm 1. The motivation behind our algorithm is as follows: By applying a clustering analysis algorithm, the content

space is partitioned into a number of subspaces. Then, a representative game from each subspace is played and labeled by developers with their gameplay experience and knowledge. All representative games from different subspaces form a validation set T_{ICQ} to evaluate the generalization of an active learner effectively. By exploiting the clustering analysis results and the information in T_{ICQ}, we can quickly find a few games from two different classes to label for training an active learner initially before applying a standard active learning procedure to train the ICQ classifier.

Algorithm 1 ICQ Learning

Clustering: Apply a clustering analysis algorithm to partition all games into a number of clusters and label a representative game from each clusters to form a validation set T_{ICQ}

Initialization: Label a few randomly selected games beyond T_{ICQ} and then train a binary classifier with the labeled examples

while test error on T_{ICQ} is unacceptable **do**

  for all $\boldsymbol{g} \in \mathcal{G}$ **do**

    Test $\boldsymbol{g}$ with the classifier of the current parameters

    Record its confidence on the label assigned to $\boldsymbol{g}$

  end for

  Find the game of the least confidence, $\boldsymbol{g}^*$

  Annotate $\boldsymbol{g}^*$ by developers if it was not annotated

  Re-train the classifier with $\boldsymbol{g}^*$ and its true label

end while

In Quake, we use the K-medoids algorithm with Euclidean distance [28], which is insensitive to initial conditions, for clustering analysis. K is set to 200 since it is a reasonable working load for a developer and sufficient to test the generalization. Thus, a developer plays and labels 200 medoids to form the validation set T_{ICQ}. For efficiency, we reduce the search space so that active learning can converge quicker. Hence, we randomly choose 100 games from each cluster to form a reduced content space of 20 000 core games. In the active learning, a nonlinear support vector machine (SVM) with the RBF kernel [29] was used in Algorithm 1 where it is initialized by two randomly chosen games from different classes. While there are different confidence measures [26], we use the probabilistic output of SVM based on the LibSVM implementation [30] to define our confidence measure for active learning: $|\Pr(+1 \mid \boldsymbol{g}) - \Pr(-1 \mid \boldsymbol{g})|$ where $\Pr(\pm 1 \mid \boldsymbol{g})$ is the probability that $\boldsymbol{g}$ is predicted to have the label ± 1. The active learning process is continued until the positive and negative error rates on T_{ICQ} converged, indicating the model has similar performance with respect to identifying both acceptable and unacceptable games.

B. Learning for CC

In general, the problem in the CC model is finding out a mapping for $\boldsymbol{g} \in \mathcal{G}_a$, a subspace containing all acceptable games, so that $\Phi_{\mathrm{CC}} : \boldsymbol{g} \rightarrow c_f$ for $f = 1, \cdots, F$. As each c_f takes categorical values, we can formulate this problem as multiclass classification.

As the CC learning encounters the same two issues in the ICQ learning, we adopt the same idea used in the ICQ learning again with a resource-sharing idea by making the maximal use of results achieved during the ICQ learning. For resource sharing, developers can assign categorical labels on content features simultaneously whenever they label a game as acceptable for the ICQ learning. Due to the locality property, the clustering results and acceptable games (also having categorical labels for each content feature) in T_{ICQ} may also help developers select representative acceptable games to label. In this way, we can efficiently achieve a validation set T_{CC} used in the CC active learning. Based on the resource sharing idea, we adapt the ICQ active learning algorithm for the CC learning as summarized in Algorithm 2.

Algorithm 2 Active Learning for CC

Initialization: (a) Based on the ICQ learning, label more acceptable games to ensure the validation set T_{CC} to have sufficient games on each category of c_f and (b) Label a few randomly selected games beyond T_{CC} and then train F classifiers with the labeled examples, respectively.

for all $c_f \in \boldsymbol{c}$ **do**

  while test error on T_{CC} is unacceptable in terms of c_f **do**

    for all $\boldsymbol{g} \in \mathcal{G}_a$ **do**

      Test $\boldsymbol{g}$ with the fth current learner

      Record its confidence on the label assigned to $\boldsymbol{g}$

    end for

    Find the game of the least confidence, $\boldsymbol{g}^*$

    Label $\boldsymbol{g}^*$ by developers if it was not annotated

    Update parameters of the fth learner with $\boldsymbol{g}^*$ and its ground-truth

  end while

end for

In Quake, the ICQ model is first applied to find an acceptable content subspace, $\mathcal{G}_a$. By using the resource sharing idea, we created a validation set T_{CC} consisting of 110 labeled games with at least 20 games from each difficulty category. For five-category classification, we decompose it into five binary classification subtasks. As the same as done in the ICQ active learning, we train five probabilistic SVM classifiers separately in the inner loop of Algorithm 2 for the CC active learning, one for each difficulty category. Again, we can apply the same initialization, confidence measure and stopping conditions to the CC active learning as described in Section IV-A.

C. GPE Learning

For data collection, N quality games denoted by $G = \{\boldsymbol{g}_1, \boldsymbol{g}_2, \cdots, \boldsymbol{g}_N\}$ are selected by developers, and P public tester participants are required. In survey, feedback may be often characterized by a nominal variable y. For a game $\boldsymbol{g}_n$ played by player p, we denote his/her feedback by $y_n^{(p)}$. As a multivalued nominal variable can be represented by multiple binary variables, we only take the binary feedback into account here; i.e., $y_n^{(p)} \in \{1, 0\}$ where $1/0$ indicates positive/negative feedback, respectively. Thus, the crowd-sourcing learning in the GPE model is formulated as follows: for any game $\boldsymbol{g}_n \in G$ played by public testers, finding the consensus experience $\hat{y}_n$ on this game with a confidence level γ_n and assigning a conformity factor to each public tester p. The conformity factor is defined by the *sensitivity* (i.e., similarity to the generic public in the positive feedback), $\alpha^{(p)}$, and the *specificity* (i.e., similarity to the generic public in the negative feedback), $\beta^{(p)}$ for public tester p. Intuitively, a larger $\hat{y}_n$ value assigned to game n implies that game n is enjoyed by more public testers. The higher α and β values assigned to public tester p, the closer to the public consensus the feedback of test player p is.

While there are many potential solutions to the crowd-sourcing learning, we adopt a generic algorithm [31] for our enabling technique. The algorithm treats the "true" label of an object to be annotated as missing data (i.e., "popularity" of N selected games in our context) and infers it from "noisy" labels given by multiple annotators (i.e., P public testers' feedback to the games they played in our context) with a maximum likelihood estimator defined on the likelihood function: $\Pr(G, \{y_n^{(p)}\}_{p=1}^P \mid \Theta) = \prod_{n=1}^N \{\Pr(\{y_n^{(p)}\}_{p=1}^P \mid y_n = 1 \mid y_n = 1, \{\alpha\}_{p=1}^P) \Pr(y_n = 1 \mid \boldsymbol{g}_n, \Theta) + \Pr(\{y_n^{(p)}\}_{p=1}^P \mid y_n = 1 \mid y_n = 0, \{\beta\}_{p=1}^P) \Pr(y_n = 0 \mid \boldsymbol{g}_n, \Theta)\}$, where $\Pr(y_n = 1 \mid \boldsymbol{g}_n, \Theta) + \Pr(y_n = 0 \mid \boldsymbol{g}_n, \Theta) = 1$ and $\Pr(y_n = 1 \mid \boldsymbol{g}_n, \Theta)$ is the probability of $y_n = 1$ for $\boldsymbol{g}_n$ estimated via a regressor $f(\boldsymbol{g}_n, \Theta)$ with the parameters Θ. The Expectation-Maximization (EM) framework is used to tackle missing data in this maximum likelihood problem, which leads to the algorithm [31], hereinafter to be referred to *Crowd-EM* summarized in Algorithm 3 in the context of our problem formulation. After the GPE learning is finished, the conformity factor $(\alpha^{(p)}, \beta^{(p)})$ is assigned to each of P public testers and a "popularity" score γ of any acceptable game $\boldsymbol{g} \in \mathcal{G}_a$ can be predicted via the trained regressor with $\gamma = f(\boldsymbol{g}, \Theta^*)$.

Algorithm 3 Crowd-EM for Learning GPE

Initialization

For $p = 1, \cdots, P$, set $\alpha^{(p)}(0) = \beta^{(p)}(0) = 0.5$.

For $n = 1, \cdots, N$, set $\gamma_n(0) = 1/P \sum_{p=1}^P y_n^{(p)}$.

Pre-train a chosen regressor, $f(\boldsymbol{g}, \Theta)$, by finding optimal parameters, $\Theta^*(0)$, with the training set of N examples, $\{(\boldsymbol{g}_n, \mu_n(0))\}_{n=1}^N$, and set $t = 1$.

E-Step

For $n = 1, \cdots, N$, calculate

$$h_n(t) = f(\boldsymbol{g}_n, \Theta^*(t-1)),$$
$$a_n(t) = \prod_{p=1}^P [\alpha^{(p)}(t-1)]^{y_n^{(p)}} [1 - \alpha^{(p)}(t-1)]^{1 - y_n^{(p)}},$$
$$b_n(t) = \prod_{p=1}^P [\beta^{(p)}(t-1)]^{1 - y_n^{(p)}} [1 - \beta^{(p)}(t-1)]^{y_n^{(p)}},$$
$$\gamma_n(t) = \frac{a_n(t) h_n(t)}{a_n(t) h_n(t) + b_n(t)[1 - h_n(t)]}.$$

M-Step

For $p = 1, \cdots, P$, update

$$\alpha^{(p)}(t) = \frac{\sum_{n=1}^N \gamma_n(t) y_n^{(p)}}{\sum_{n=1}^N \gamma_n(t)},$$
$$\beta^{(p)}(t) = \frac{\sum_{n=1}^N [1 - \gamma_n(t)](1 - y_n^{(p)})}{\sum_{n=1}^N [1 - \gamma_n(t)]}.$$

Re-train the chosen regressor, $f(\boldsymbol{g}, \Theta)$, by finding optimal parameters, $\Theta^*(t)$, with the training set of N examples, $\{(\boldsymbol{g}_n, \mu_n(t))\}_{n=1}^N$, and set $t = t + 1$.

Repeat both *E-Step* and *M-Step* **until** convergence.

In Quake, Algorithm 3 is directly applied to 100 games selected by developers and the "fun/not fun" feedback collected in the public survey where a nonlinear support vector regressor (SVR) with the RBF kernel [29] is employed to tackle the nonlinear regression problem. The algorithm will be terminated when it reaches a local maximum of the aforementioned likelihood function. During the public test (see Section V-A for more details on data collection), a public tester may not play all of 100 games used for the public survey. In this case, we simply substitute P, the number of all public testers, with P_n, the number of public testers who actually play game $\boldsymbol{g}_n$, as well as $N = 100$, the number of all games in G, with N_p, the number of games that public tester p actually plays, respectively, in Algorithm 3.

D. Learning for PDC

In general, the problem in the PDC model is finding out a mapping $\Phi_{PDC} : (\boldsymbol{l}, \boldsymbol{c}) \rightarrow y$. As y often takes a binary value indicating either positive or negative experience, we formulate it as a binary classification in terms of learning.

During public test, we assume that N_T surveys are accomplished so that N_T play-logs along with their corresponding feedback, $\boldsymbol{l}_i$ and y_i $(i = 1, \cdots, N_T)$, are recorded by a data collection system. By combining with content features $\boldsymbol{c}_i$ of each of the played games, we may construct a training data set, $\mathcal{D} = \{[(\boldsymbol{l}_i, \boldsymbol{c}_i), y_i]\}_{i=1}^{N_T}$. By taking the public testers' conformity

information, $(\alpha^{(p)}, \beta^{(p)})$, achieved in the GPE model into account, we can select examples from $\mathcal{D}$ based on their robustness. By setting thresholds for $\alpha^{(p)}$ and $\beta^{(p)}$, respectively, we can generate a training subset of examples from only public testers whose $\alpha^{(p)}$ and $\beta^{(p)}$ are above thresholds. Threshold setting is a challenging problem since there is no clear criterion; a low conformity score assigned to a player does not have to mean that the player is a "liar" but may suggest the player be an "outlier" who has distinct experience from the majority of public testers. To tackle this problem, we propose an ensemble learning algorithm, as summarized in Algorithm 4, for the PDC learning. The basic idea is using different thresholds to produce M $(M > 1)$ training subsets and then train M classifiers on those training data subsets via cross-validation, respectively. Accuracy measured on cross-validation would be used as weights to construct an ensemble classifier.

Algorithm 4 Learning for PDC

Initialization: Divide the training data set, $\mathcal{D}$, collected in public test into M training subsets, $\mathcal{D}_1, \cdots, \mathcal{D}_M$ by setting a number of thresholds for $\alpha^{(p)}$ and $\beta^{(p)}$, respectively. Choose a classifier, $f[(\boldsymbol{l}, \boldsymbol{c}), \Theta]$.

for $m = 1$ **to** $m = M$ **do**

Train $f[(\boldsymbol{l}, \boldsymbol{c}, \Theta_m)]$ on $\mathcal{D}_m$ via cross-validation by finding optimal parameter, Θ_m^*.

Record its accuracy, u_m, on cross-validation.

end for

Calculate weights:

$$w_m = \frac{\exp(u_m)}{\sum_{k=1}^{M} \exp(u_k)} \text{ for } m = 1, \cdots, M.$$

Construct the ensemble classifier:

$$F[(\boldsymbol{l}, \boldsymbol{c}), \Theta^*] = \sum_{k=1}^{M} w_k f[(\boldsymbol{l}, \boldsymbol{c}), \Theta_k^*].$$

In Quake, we employ the random forests (RF) [32] as a binary classifier since it has inbuilt cross-validation (out-of-bag error) and can identify useful features automatically during its learning. A RF binary classifier is trained on examples consisting of the difficulty category of a game and 122 attributes in the play-log resulting from playing the game as the features and the "fun/not fun" feedback given by the public testers as the label. To construct an ensemble learner, we need to establish different training subsets for multiple RF classifiers based on the public testers' conformity information. As a result, we choose four thresholds 0.0, 0.3, 0.6, and 0.9 on $\alpha^{(p)}$ and $\beta^{(p)}$ achieved in the GPE model, respectively, which results in 16 different training data subsets via the combination of different $\alpha^{(p)}$ and $\beta^{(p)}$ thresholds. Accordingly, 16 RF classifiers are first trained on 16 training subsets separately and then combined to complete the ensemble learning with Algorithm 4.

Fig. 3. The CATEGORIZE state in the IP model.

E. On-Line Generation With the IP Model

Based on the requirements described in Section III, the IP model is designed to tackle all problems in the adaptive stage of PCG. Here, we carry out the IP model with a state machine of three states, i.e., CATEGORIZE, PRODUCE, and GENERALIZE, to be used for preferred content category detection, quality yet personalized game generation as well as concept drift monitoring and system failure avoidance, respectively.

As illustrated in Fig. 3, the CATEGORIZE state is designed to detect a new target player's content preference. To detect his/her preferred content category reliably, the new target player needs to play a few games used in the GPE learning. Recall that after the GPE learning, N selected games grouped with their content features may be ranked based on their γ value. Once a target player has played a game, the PDC model uses his/her play-log along with the content features to predict his/her IP. If his/her behavior is consistent on games in a content category characterized by features $\mathbf{c}^*$, it suggests that the target player enjoys games in this specific category. Then, the IP model leaves from the current state for the PRODUCE state.

The PRODUCE state is shown in Fig. 4. It directs the IP model to use both the ICQ and the CC model to control the content generator to produce that quality content of variation in the specific game category determined in the CATEGORIZE state. Furthermore, the PDC model is employed to monitor whether a preference drift has taken place. Whenever such a concept drift is detected, the IP model will get back into the CATEGORIZE state.

To avoid a system failure, the GENERALIZE state shown in Fig. 5 is dedicated to a crisis situation that a target player's IP cannot be determined after many attempts in the CATEGORIZE state. Our solution is exploiting the ICQ, the CC, and the GPE models to generate quality games that are likely to be appreciated by public. This helps towards minimizing disappointment which may be experienced by playing randomly generated

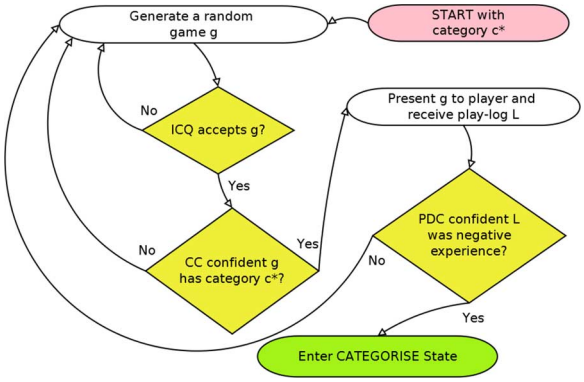

Fig. 4. The PRODUCE state in the IP model.

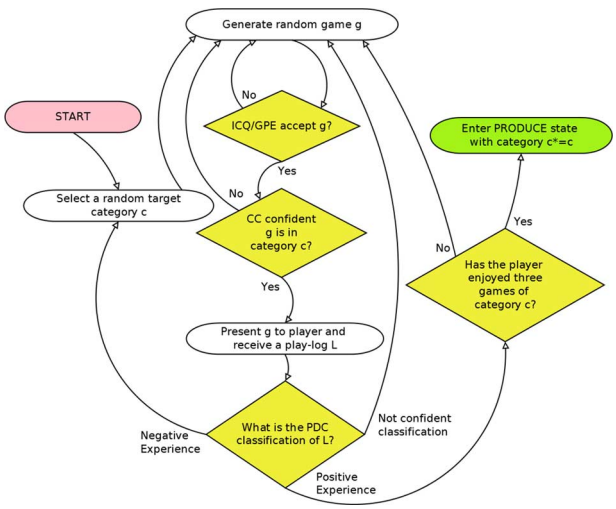

Fig. 5. The GENERALIZE state in the IP model.

games of low quality. By using evaluation functions achieved in above three models, a target player receives only those games, $\boldsymbol{g} \in \mathcal{G}_a$, of both a high confidence in content categorization and a large γ value produced by $f(\boldsymbol{g}, \Theta^*)$ in the GPE model. The PDC model is also employed to predict a target player's IP once a game was played as the same as done in the CATEGORIZE state.

In Quake, 100 selected games are grouped in five categories in terms of the content feature, difficulty, where 20 games in each category are ranked based on their γ values. The CATEGORIZE state uses the 100 games to detect a new target player's IP. Whenever the PDC model produces the positive prediction for two consecutive games of the same category played by the target player, the category is viewed as his preferred one and the IP model enters the PRODUCE state. If this situation does not occur after playing 10 games, the IP model will enter the GENERALIZE state. In the the PRODUCE state, the player is always provided with the quality games of the detected category using the ICQ and the CC models, while the PDC model oversees any preference change. In the GENERALIZE state, a target player always receives the quality game safeguarded by the ICQ, the CC, and the GPE models and will be brought into the PRODUCE state once the PDC model finds that the target player enjoys three games of the same difficulty category.

V. Simulation

In this section, we report simulation results on the Quake, a proof-of-concept prototype described in Section III-C, implemented with the enabling techniques presented in Section IV.

A. Data Collection

Both the ICQ and the CC models were trained by a single expert who played and labeled games during the active learning. For the GPE and PDC models, we put our prototype out to the internet to simulate a scenario that gets as varied a cross section of public testers as possible. To facilitate the public data collection, we developed a client/server architecture to collect surveys from people remotely. To collect data for the GPE learning, we choose 100 representative games for the public test via active learning. There are 20 games in each of five difficulty categories and the random seed was fixed so that all public testers can play exactly the same 100 games.

The client was distributed via the web and advertised on websites such as Reddit, Facebook and the OBLIGE forums. In total 895 surveys were recorded from a total of 140 people. Each game was played roughly nine times each and most of players played three to five games. However, one enthusiastic participant actually produced 154 surveys on their own while several played only one game. Our questionnaire consisted of two questions only: "Did you enjoy the level?" (yes/no) and "How do you rate it?" (Very Bad/Bad/Average/Good/Very Good). Play-logs received from the public testers are linked to their answers to two questions mentioned above and the binary answer to the first question is used to train the PDC model.

Our analysis reveals that approximately two thirds of surveys were positive feedback. Interestingly, the surveys showed that as game difficulty increases, the number of "Very Good" labels also increases, but the "Very Hard" category also has the most labels of "Very Bad." The middle difficulties "Easy" to "Hard" have the least number of people labeling them as "Very Bad." Additionally, as difficulty increases, less surveys are labeled as "Average", indicating more polarized view points. Further analysis showed that "Very Hard" and "Very Easy" games cause the most disagreement amongst participants, whereas the middle difficulties caused less disagreement. This suggests that the games were well selected since they caused controversy, potentially allowing us to distinguish between different types of players based on content category. Nevertheless, such a survey using unknown participants can by no means guarantee a representative of different target players. Hence, it is unlikely that the survey was able to capture entirely the relationship between behavior, content category and experience, since many participants did not encounter games from all difficulty categories.

B. Results on Component Models

For Quake, we first report results on four underpinning component models achieved from our simulation.

Fig. 6 illustrates test error rates on T_{ICQ} as the ICQ active learning process progresses. "+Error" and "−Error" stand for errors on positive and negative samples in T_{ICQ}, respectively. Also we report the *half total error rate* (HTER) defined as the average of "+Error" and "−Error" and denoted by "AvgError." It is observed from Fig. 6 that the overall error

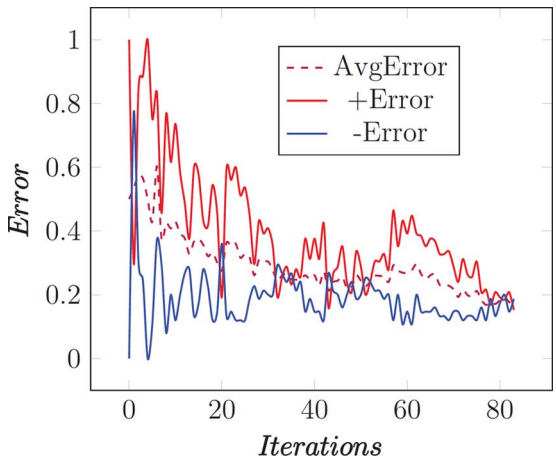

Fig. 6. Test errors in the ICQ active learning process.

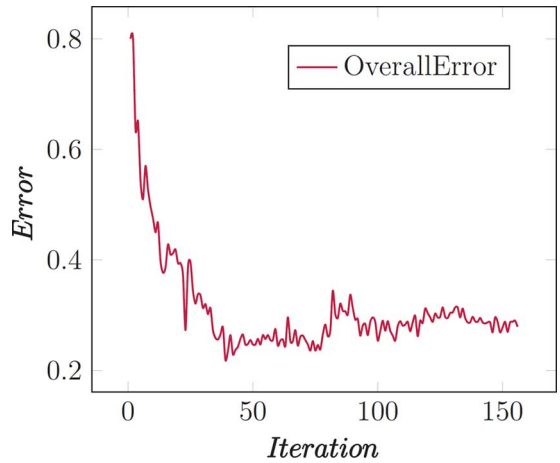

Fig. 7. Test error in the CC active learning process.

gradually decreases as more and more informative examples are presented and both positive and negative errors converge to an acceptable *equal error rate* (EER), approximately 19%, after 81 iterations. By applying the ICQ model to the content space, 37% core games are classified as unacceptable. Common features of unacceptable levels were those with no monsters and no ammo. In addition, games with an overwhelming number of monsters, but not much in the way of resources were also rejected. As we used the resource-sharing idea in the ICQ and the CC learning, we found that around 84% unacceptable games may be classified as "Very Hard." A feature selection experiment by training a RF classifier based on the active learning results revealed that the importance of nine parameters in the ICQ learning is ranked in the descent order as follows: ammo, monster set 4, monster set 1, health, monster set 2, skill, monster set 3, weapon choice and overall monster count.

As the CC model classifies the game content into five categories, we only report overall errors on T_{CC} during the CC active learning in Fig. 7 for clear visualization. We observed that the model reached its optimal accuracy at 41 iterations and training beyond this leads to over-fitting to a specific category. At an early stop at iteration 41, the overall error rate on T_{CC} was 22% and error rates for different difficulty categories were as follows: 17% on "Very Easy," 18% on "Easy," 35% on "Moderate," 25% on "Hard" and 20% on "Very Hard." A closer look at the confusion matrix revealed that almost all misclassification happened on two adjacent difficulty categories, e.g., "Hard" is misclassified as "Very Hard." Thus, such misclassification produces no catastrophic effect. "Very Easy" levels tend to consist of easier monsters from sets 1 and 2, i.e., soldiers, dogs, flying demons and knights. The "Very Easy" games generally have salient characteristics as follows: (a) "scarce" number of monsters in almost all levels; (b) none of the harder enemies (i.e., those contained in monster set 3 or 4) in most cases; and (c) no grenade launchers in weapon. In contrast, the "Very Hard" games are generally characterized by (a) an overwhelming amount of monsters; and (b) not much health or ammunition. Surprisingly, our statistical analysis reveals that the skill parameter has a very limited impact on the difficulty that the CC model learnt given the fact that most of the "Very Easy" games have the harder skill value! It seems that "skill" mainly affects the number of monsters that may not directly link to the difficulty, e.g., a few difficult skilled monsters are much easier to defeat than many easy skilled ones. Thus, this skill parameter seems to do nothing in characterizing the content difficulty apart from adding a few extract enemies sometimes.

For the crowding-sourcing learning in the GPE model, we did not use the data collected from the participant who made 154 surveys as it caused the learning to be strongly biased to his/her opinions and his/her feedback and play-logs are inconsistent for the same games that he/she played for several times. As there is no ground-truth on the popularity of the games and the conformity of public testers, we only report some interesting results summarized from the statistics produced by the games of high γ values and the public testers of high α and β values. Among the top 20 "popular" games of the highest γ values, there are five "Very Easy," six "Easy," six "Moderate," two "Hard," and one "Very Hard" games. All the top 20 games contain ammunition with the highest quantity and more health. The rocket launcher and super shotgun are the popular weapons. Monster sets 3 and 4 are not popular. In fact, none of monsters in set 4 appears in the top 20 games and monsters in set 3 only appear in 15% of the top 20 games. Also the skill parameter has no bearing on game popularity. For all the 100 games used in our public survey, the public testers generally do not like the Shambler monster as 89% games labeled with fun do not have monsters in set 4 who are quite hard to kill but can kill a player easily. The public testers also seemed to dislike monsters in set 3, i.e., Fiends and Ogres, who can make some annoying attacks, e.g., jumping and firing grenades, as 67% of the positive games contain none of them. Overall, there is no clear weapon preference and the public testers do not really like levels full of monsters that much as there are only an amount at 13%. In contrast, the public testers like lots of monsters in set 1 with 66% preferring the "more" attribute value. Also the public testers like lots of ammo but dislike maps with not much health.

To test the PDC model, we used 10-fold cross-validation on all the play-logs used in the GPE learning. As the RF provides a confidence value ranging from 0 to 1 on each decision, we

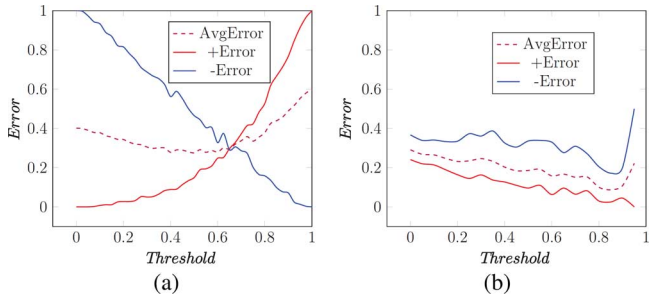

Fig. 8. Test errors of the PDC models. (a) Error rates versus confidence thresholds. (b) Error rates versus rejection thresholds.

further exploited such information to improve the PDC performance. By adjusting the decision boundaries with different confidence thresholds, we achieved different test results. It is immediately apparent from Fig. 8(a) that the positive/negative error rates converge to the EER of 29% at a confidence threshold of 0.61. During the test, we applied the confidence-based rejection to increase the accuracy. As depicted in Fig. 8(b), using a rejection threshold of 0.25 an HTER of 24% was achieved, where approximately 25% and 27% of samples from the positive and negative classes, respectively, were rejected. Given the nature of noisy public surveys and the fact that many public testers played very few games (less than five), we believe that overall performance is reasonable. As the RF also ranks the importance of 122 play-log attributes during the PDC learning, we observe that the most important attribute is the level completion, i.e., the further a player progresses into a game determines whether they like it or not. This is probably because a player is more likely to progress further in a game they enjoy. In addition, how many times a player dies is also important, along with how much they interact with the game. How many Grunts that are killed also seems to be influential. This might be because people like to kill lots of this easy monster. The number of Fiends in monster set 3 also influences player enjoyment as this is one of the hardest and the most annoying to kill. Apart from the above attributes, the following attributes also have the high importance in the PDC learning: total input events, in health/ticks, monsters killed, dogs killed, and total distance traveled.

C. Results on the Quake Prototype

We have tested our Quake prototype[1], i.e., the IP model, in comparison to two baselines: (a) *Random model*, which generates completely random games with OBLIGE and (b) *Skill model*[2], which presents "skill" balanced games by controlling the skill parameter in OBLIGE while allowing other parameters to be set randomly. As OBLIGE is a well designed content generator towards generating quality games, these two baseline models may be competitive to the IP model.

In the test, an extensive survey is done with the following protocol. A target player who never participated in the GPE survey was asked to play 30 games generated by invoking three models randomly, with 10 generated by each model. Before playing any game, a player was asked to answer two multiple-choice questions (MCQs) on his/her historical gameplay experience as follows: (i) *What kind of gamer would you describe yourself as?* ("I don't play games much—if at all," "I play games sometimes," "I play games regularly," "I'm a hardcore gamer"); and (ii) *How would you rate your skill level at action video games?* ("Inexperienced", "Average," "Good," "Very Good"). After playing each game, the player was asked five MCQs: (i) *Did you find the level fun?* ("No," "Yes"); (ii) *How difficult was that?* ("Too easy," "Just right," "Too hard"); (iii) *Do you want to see more levels like that?* ("No," "Yes"); (iv) *Do you think other people would enjoy it?* ("No," "Yes"); and (v) *How do you rate the level?* ("Very Bad," "Bad," "Average," "Good," "Very Good").

To evaluate the performance of three different models, we define three different evaluation metrics based on the feedback of target players. As the first one of five MCQs asked after each game played is the exactly same as being asked in the public test, we use the feedback to this question to form a simple metric, hereinafter named *Metric 1*, that the positive/negative answer scores one/zero. Also we define a metric with all the feedback to five MCQs, hereinafter named *Metric 2*. Except the last MCQ where "Very Bad" and "Bad" ratings score zero and other three ratings score one, the positive/negative answer scores one/zero in each of other four MCQs. As a result, a model receives five scores at maximum for each game it generates with Metric 2. As one of the goals in the IP model is automatically finding the preferred difficulty category of a target player, we further define a scoring metric based on the feedback to the first question (i.e., the binary "fun" feedback), hereinafter dubbed *Metric 3*, by taking the content category of each game played into account. For 30 games played by player p, let $N_{p,c}$ and $N^E_{p,c}$ denote the number of the games of difficulty category c played and the number of games of difficulty category c enjoyed by him/her (i.e., the number of positive feedback given by player p to games of c), respectively. Hence, the preference rate of player p for games of difficulty category c is $\rho_{p,c} = N^E_{p,c}/N_{p,c}$. For player p, let $N_{m,p,c}$ be the number of games of difficulty category c generated by model m. For model m and player p, our scoring metric $S_{m,p}$ is defined as

$$S_{m,p} = \sum_{c \in \mathcal{C}} \rho_{p,c} N_{m,p,c}$$

where $\mathcal{C}$ is the set of five content categories in Quake. Intuitively, a higher score awarded to a model indicates that the model produces more games of the difficulty category enjoyed by the player. Thus, we would use such a metric to measure the success of a model in terms of personalization.

In our simulation, we managed to get 14 reliable people to simulate target players for such a rather time-consuming test. Regarding the historical gameplay experience, their feedback indicates that there are three players who have little gameplay experience (players 3, 10, 13), six players who played games sometimes (players 1, 2, 6, 9, 11, 14), four regular game players (players 5, 7, 8, 12) and one hardcore gamer (player 4). In terms of the gameplay skill levels, their feedback reveals that there are three inexperienced (players 3, 10, 13), four averaged (players 2, 5, 6, 14), five good (players 1, 4, 7, 9, 11), and two very good (players 8, 12) players. According to the information extracted

[1]Online available: http://staff.cs.manchester.ac.uk/~kechen/lbpcg_quake/

[2]This model was suggested by anonymous reviewers who hypothesized that the "skill" parameter predominates other parameters for difficulty.

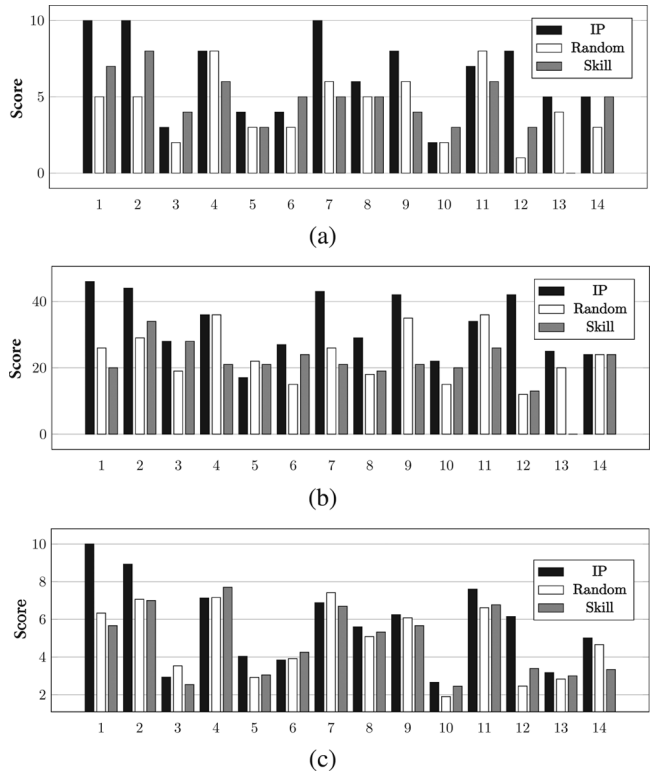

Fig. 9. Scores awarded to three different models by 14 target players with different score metrics. (a) Metric 1. (b) Metric 2. (c) Metric 3.

from their self-reported gameplay experience/skill levels, these participants should be able to represent target players of different types adequately.

Fig. 9 illustrates the scores awarded to three models measured with different metrics based on the feedback of 14 players. From Fig. 9(a), it is observed that the IP model won 10 players who awarded the highest scores in terms of Metric 1 but two was tied with other models. In contrast, the Random and the Skill models received only two and three highest scores but each model had one highest score tied with the IP model. According to the binary "fun" feedback, 64.3% (90 out of 140) games generated by the IP model were rated as "fun" totally, while 43.6% (61 out of 140) games and 45.7% (64 out of 140) games generated by the Random and the Skill models received the positive feedback, respectively. Similarly, Fig. 9(b) shows that the IP model received the highest scores from 11 players in terms of Metric 2 but three tied with other models. In contrast, the Random and the Skill models were awarded the highest scores in terms of Metric 2 by three and two players, respectively, but there were two players in each case who also rated other models as the tied winners. Fig. 9(c) shows those scores awarded to three models in terms of Metric 3 where the IP, the Random, and the Skill models won 10, 2, and 2 players, respectively. Overall, it is evident from Fig. 9 that the IP model outperforms other two models in terms of all three evaluation criteria.

We have done thorough analysis on all data collected during this simulation. Due to the limited space, we report only those nontrivial yet informative results here. The IP model deals with different situations with three states and the transition between different states would be important in understanding how our ideas work. Our data analysis shows that the CATEGORIZE state detected the "preferred" content categories for eight players (i.e., players 1, 2, 3, 4, 8, 10, 11, 12) but failed to do so for other six players after 10 games were played. In other words, the eight players were brought to the PRODUCE state after playing five games on average in the CATEGORIZE state. Five players (players 1, 2, 4, 8, 10, 12) stayed in the PRODUCE state with games for the detected preferred difficulty category but concept-drift was detected for the other three players (players 3, 10, 11) who went back to the CATEGORIZE state after playing six, seven and one games produced in the PRODUCE state, respectively. As the PDC model plays a critical role in the IP model, we also investigate its performance by using the players' binary "fun" feedback as the ground-truth. As a result, a nearly identical HTER was achieved regardless of whether the confidence threshold was applied; the HTER is 33.4% (+Error: 38.9%, −Error: 28.0%) without using the threshold and the HTER is 33.6% (+Error: 31.9%, −Error: 35.3%) by applying the threshold. In terms of HTER, the performance is approximately 10% worse than the PDC validation results reported in Section V-B, which might suggest the weakness of the uncontrolled random survey used in the GPE model. Although the similar performance was achieved overall regardless of the confidence threshold, applying the threshold to a specific player could reduce the error rate significantly, e.g., for player 9, the HTER was 93.8% without the threshold but sharply decreased to 50.0% with the threshold. This might suggest that the PDC Model should be enhanced to work better with different player types; outliers should be quickly detected and handled with additional mechanisms.

In general, our IP model seems to work better for players who had certain gameplay experience and good skill levels than for those who have little experience and poor skills. The highly successful examples include players 1 and 2 who self-reported to play games sometimes and to have either average or good skill level. The PDC model had error-free performance for player 1 and a low error rate (10%) for player 2. According to their feedback, the preferred category, "Very Easy," was identified when player 1 played four games in the CATEGORIZE state, while the preferred category, "Moderate," was not detected until player 2 played nine games in the CATEGORIZE state. Nothing particularly stood out with respect to the attributes of the types of games they liked. In general, the games played by player 1 have more ammo and health than not, and all the monsters came from sets 1 and 2. In the games played by player 2, 60% monsters came from sets 1 but there were monsters from other three sets. This player mainly used the weapon, nailgun, along with all other available weapons that were evenly used. Nevertheless, it is observed that those players who have little gameplay experience and poor skills did not seem to enjoy playing games regardless of models. Players 3, 10, and 13 in this class awarded the lowest overall scores to 30 games played by them as illustrated in Fig. 9. They seem quite sensitive to level variation within the same difficulty category. In addition, they tended to play a game in a shorter time and their behavior was extremely unstable, which makes the performance of the PDC model degraded considerably. For example, player 10 was given "Very Easy" games to play in the PRODUCE state but the PDC model

was not confident to decide whether this player still enjoyed this category after playing seven "Very Easy" games.

In summary, the Quake prototype carries out a proof of concept for our LBPCG framework. Simulation results suggest that the prototype is promising towards generating the content appealing to target players.

VI. Discussion

In this section, we discuss several issues arising from our LBPCG framework and simulations.

The ICQ model is key to any content generator for producing quality games while the CC model is useful in limiting the search space during content generation. Traditionally, such models were carried out with hand-crafted evaluation functions with prior knowledge, e.g., [6], [11]. In this paper, we explore a data-driven methodology by encoding developers' knowledge implicitly. The development in both methods inevitably involves developers' efforts; developers have to find constraints/rules on content space manually in traditional methods while developers need to annotate games in data-driven methods. In general, traditional methods may be more efficient if developers can encode their prior knowledge on content space explicitly with constraints/rules. However, such handcrafted evaluation functions often filter out quality games along with those of low quality. By taking the whole content space into account, we believe that the learning-based method might provide an alternative solution whenever it is hard for developers to find constraints/rules.

Our LBPCG framework provides a novel solution to user modeling for the EDPCG [14] via learning developers', public testers' and target players' preferences. In this context, the earlier work [5] has successfully explored machine learning for user modeling. Nevertheless, our work distinguishes from the previous work in the following aspects: (a) applying active learning in modelling developers' knowledge implicitly; (b) learning taking place in the different stages for developers and different users; (c) learning the preferences of public testers with quality games selected based on content categorization, which facilitates capturing typical gameplay behavior and eliciting experience of interest; and (d) minimizing the interruption of target players' gameplay experience with inductive learning models trained on public testers' data.

To generate the content appealing to target players online, the IP model strategically controls a content generator by using four component models sequentially. While our method working in the generate-and-test style is designed for computational efficiency, more sophisticated search-based techniques [3] might make use of four component models to generate quality contents. For instance, an evolutionary computation algorithm may combine all or some evaluation functions created in four component models along with other constraints, if there are, as a multiobjective fitness to evaluate the candidate content during search.

While the LBPCG framework is proposed towards tackling several challenging problems in PCG, there are still several issues to be addressed in future researches. First of all, the key idea underlying the LBPCG is allowing developers to define content features flexibly for meaningful content categorization in order to facilitate categorizing players and adapting the content to their preference. However, it is nontrivial for developers to choose the correct content features that can elicit different affective/cognitive experience for players of different types. Without prior knowledge, developers may need to acquire such knowledge first, which may have to involve the public testers prior to the content feature selection. In addition, the learning-based enabling techniques for the ICQ and the CC models require training examples, labeled representative games by developers. While it is an implicit way to encode developers' knowledge, it may be tedious and expensive in terms of labor cost. On the one hand, we would exploit agents or simulation-based methods [10] to lower the cost in annotation. On the other hand, the annotation process may involves veteran public testers, e.g., beta testers in the real industrial gaming "stress-test", who have the same knowledge as developers. Next, the enabling techniques presented in this paper are by no means sophisticated. There is no proof that the active learning algorithms used in the ICQ and the CC models always converge to the satisfactory performance with a few annotated examples. Recent studies [33] suggest that the rating-based feedback to a single game played in the public survey might cause less accurate and "biased" experience to be reported while ranking-based feedback by comparing a pair of games played can capture players' accurate and less "biased" experience. Unfortunately, our current enabling techniques in the GPE, the PDC and the IP models do not enable the LBPCG to use the ranking-based survey. In addition, more effective yet robust enabling techniques for the IP model should be explored in detecting a target player's type/style. Finally, our proof of concept is also subject to limitation as follows: (a) a single developer was used in the ICQ and the CC models; (b) the random survey did not necessarily contain participants who are representative of target players and many of the participants played no games in some difficulty categories in the GPE model; and (c) the IP test was less thorough as there were only 14 target players and each played 10 games.

In conclusion, we have presented the LBPCG framework and the enabling techniques towards generating the content appealing to target players. For a proof-of-concept, we have developed a Quake prototype to demonstrate how our LBPCG framework works. The preliminary results in our simulation are promising. In our ongoing research, we will be investigating the outstanding issues discussed above and exploring state-of-the-art machine learning and other relevant techniques to improve the enabling techniques.

Acknowledgment

The authors would like to thank the action editor and three anonymous reviewers for their invaluable comments that significantly improved the presentation of this manuscript. Also the authors are grateful to all anonymous public players for their feedback in the public survey and all the target players who participated in our simulation.

References

[1] H. Wallop, "Video games bigger than film," *The Telegraph* 2009 [Online]. Available: http://www.telegraph.co.uk/technology/video-games/6852383/Video-games-bigger-than-film.html

[2] J. Ludwig, "Procedural content is hard," *Website* 2007 [Online]. Available: http://programmerjoe.com/2007/02/11/procedural-content-is-hard/
[3] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.
[4] A. Laskov, "Level generation system for platform games based on a reinforcement learning approach," M.Sc. thesis, School of Inform., The Univ. Edinburgh, Edinburgh, U.K., 2009.
[5] N. Shaker, G. N. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," in *Proc. AIIDE*, 2010.
[6] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *Proc. Int. Conf. Appl. Evol. Comput.*, 2010.
[7] N. Shaker, G. N. Yannakakis, J. Togelius, M. Nicolau, and M. O'Neill, "Evolving personalized content for Super Mario Bros using grammatical evolution," in *Proc. AIIDE*, 2012.
[8] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *Proc. Int. Conf. Appl. Evol. Comput.*, 2011.
[9] J. Togelius, T. Justinussen, and A. Hartzen, "Compositional procedural content generation," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012.
[10] J. Togelius, R. Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *Proc. IEEE Conf. Comput. Intell. Games*, 2007.
[11] A. Liapis, G. N. Yannakakis, and J. Togelius, "Neuroevolutionary constrained optimization for content generation," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011.
[12] A. Liapis, G. N. Yannakakis, and J. Togelius, "Limitations of choice-based interactive evolution for game level design," in *Proc. AIIDE*, 2012.
[13] J. Togelius, J. Whitehead, and R. Bidarra, "Guest editorial: Procedural content generation in games," *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 3, pp. 169–171, 2011.
[14] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Trans. Affect. Comput.*, vol. 2, no. 3, pp. 147–161, 2011.
[15] *Artificial and Computational Intelligence in Games*, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Wadern, Germany: Dagstuhl Follow-Ups, 2013, vol. 6.
[16] H. Munoz-Avila, C. Bauckhage, M. Bida, C. Congdon, and G. Kendall, "Learning and AI Games," in *Artif. Comput. Intell. Games*, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Wadern, Germany: Dagstuhl Follow-Ups, 2013, vol. 6, pp. 33–43.
[17] "Product Release—Quake (iD Software)," 1996.
[18] G. N. Yannakakis, M. Maragoudakis, and J. Hallam, "Preference learning for cognitive modeling : A case study on entertainment preferences," *IEEE Trans. Syst. Man Cybern. (Part A: Syst. Hum.)*, vol. 39, no. 6, pp. 1165–1175, Nov. 2009.
[19] G. N. Yannakakis and J. Hallam, "Real-time game adaptation for optimizing player satisfaction," *IEEE Trans. Comput. Intell. AI in Games*, vol. 1, no. 2, pp. 121–133, 2009.
[20] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation," *Comput. Intell.*, vol. 2, no. 1, pp. 54–67, 2010.
[21] J. Gow, R. Baumgarten, P. Cairns, S. Colton, and P. Miller, "Unsupervised modelling of player style with LDA," *IEEE Trans. Comput. Intell. AI in Games* , vol. 4, no. 3, pp. 152–165, 2012.
[22] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yannakakis, "Predicting player behavior in tomb raider: Underworld," in *Proc. IEEE Conf. Comput. Intell. Games*, 2010.
[23] R. J. Pagulayan, K. Keeker, D. Wixon, R. L. Romero, and T. Fuller, "User centered design in games," in *The HCI Handbook*. Mahwah, NJ, USA: Lawrence Erlbaum Associates, 2003, pp. 883–906.
[24] K. Isbister and N. Schaffer, *Game Usability: Advancing the Player Experience*. Boston, MA, USA: Morgan Kaufman, 2008.
[25] A. Apted, "Oblige Level Maker" 2010 [Online]. Available: http://oblige.sourceforge.net
[26] B. Settles, "Active learning," *Syn. Lec. AI Mach. Learn.*, vol. 6, no. 1, pp. 1–144, 2012.
[27] P. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York, NY, USA: Wiley, 2001.
[28] L. Kaufman and P. J. Rousseeuw, "Clustering by means of Medoids," in *Statistical Data Anal. Based on the L1-Norm and Related Methods*, Y. Dodge, Ed. Amsterdam, The Netherlands: North-Holland, 1987, pp. 404–416.
[29] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge, U.K.: Cambridge Univ. Press, 2001.
[30] H. Lin, C. Lin, and R. C. Weng, "A note on Platt's probabilistic outputs for support vector machines," *Mach. Learn.*, vol. 68, no. 3, pp. 267–276, 2007.
[31] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoui, and L. Moy, "Learning from crowds," *J. Mach. Learn. Res.*, vol. 11, pp. 1297–1322, 2010.
[32] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[33] G. N. Yannakakis and J. Hallam, "Rating vs. preference: A comparative study of self-reporting," in *Affective Computing and Intelligent Interaction*. New York: LNCS, Springer, 2011, vol. 6974.

Jonathan Roberts received the B.Sc. degree in computer science from the University of Hull and was awarded the Computer Science prize for the best all round result in 2004. He received the M.Sc. degree in mathematical logic and the Foundations of Computer Science at the University of Manchester in 2005. After several years working as a software engineer, he returned to the University of Manchester, where he received the Ph.D. degree in computer science in 2014.

His research interests include video games, procedural content generation and machine learning.

Ke Chen (M'97–SM'00) received the B.Sc., M.Sc., and Ph.D. degrees in computer science in 1984, 1987, and 1990, respectively.

He has been with The University of Manchester, U.K., since 2003. He was with The University of Birmingham, Peking University, The Ohio State University, Kyushu Institute of Technology, and Tsinghua University. His current research interests lie in machine learning, machine perception and their applications in intelligent system development including computer games.

Dr. Chen has been on an editorial board of several academic journals including *Neural Networks* (2011–present) and the IEEE TRANSACTIONS ON NEURAL NETWORKS (2005–2010). He was a Technical Program CoChair of the International Joint Conference on Neural Networks (IJCNN'12) and has been a member of the Technical Program Committee of numerous international conferences. In 2008 and 2009, he chaired the IEEE Computational Intelligence Society's Intelligent Systems Applications Technical Committee and the IEEE Computational Intelligence Society's University Curricula Subcommittee. In 2012 and 2013, he was a member of IEEE Biometrics Council AdCom. He is a recipient of several academic awards including the NSFC Distinguished Principal Young Investigator Award in 2001 and JSPS Research Award in 1993.