Learning Constructive Primitives for Real-Time Dynamic Difficulty Adjustment in *Super Mario Bros*

Peizhi Shi and Ke Chen, *Senior Member, IEEE*

***Abstract*—Among the main challenges in procedural content generation (PCG), content quality assurance and dynamic difficulty adjustment (DDA) of game content in real time are two major issues concerned in adaptive content generation. Motivated by the recent learning-based PCG framework, we propose a novel approach to seamlessly address two issues in *Super Mario Bros* (SMB). To address the quality assurance issue, we exploit the synergy between rule-based and learning-based methods to produce quality game segments in SMB, named constructive primitives (CPs). By means of CPs, we propose a DDA algorithm that controls a CP-based level generator to adjust the content difficulty rapidly based on players' real-time game playing performance. We have conducted extensive simulations with sophisticated SMB agents of different types for thorough evaluation. Experimental results suggest that our approach can effectively assure content quality in terms of generic quality measurements and dynamically adjust game difficulty in real time as informed by the game completion rate.**

***Index Terms*—Constructive primitive (CP), content quality assurance, dynamic difficulty adjustment (DDA), machine learning, procedural content generation (PCG), real-time adaptation, Super Mario Bros (SMB).**

I. Introduction

PROCEDURAL content generation (PCG) aims to generate game content automatically via algorithms [1], [2]. Recently, generating personalized game content has become an active research area in PCG [3]. A variety of content adaptation techniques have been proposed and applied to different game genres ranging from platform games to first person shooter [3]. Among them, dynamic difficulty adjustment (DDA) that adapts content difficulty for an individual player is a major strategy for game content adaptation.

Super Mario Bros (SMB) is a classic two-dimensional (2-D) platform game [4], which has become one of the most popular test beds for PCG research [5], [11], [36]. In SMB, a player runs from the left side of the screen to the right side, fights enemies, and rescues the Princess Peach. SMB has a number of game elements suitable for deploying PCG techniques for content adaptation, e.g., enemies, coins, tubes, and cannons. As a result, several adaptive SMB level generators have been studied [6]–[11].

Manuscript received August 3, 2016; revised March 21, 2017, June 5, 2017, and July 31, 2017; accepted August 11, 2017. Date of publication August 15, 2017; date of current version June 14, 2018. (*Corresponding author: Ke Chen.*)

The authors are with the School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K. (e-mail: shipa@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.2017.2740210

While substantial progress has been made, there are still several open problems in SMB content or level adaptation. First, content quality assurance is still a fundamental issue in SMB level adaptation since the low-quality game content inevitably leads to negative gameplay experience [2], [12], [14]. Filtering out the low-quality content in personalized content generation is extremely challenging as adaptive content often has to be generated online. Next, the existing game adaptation techniques can be generally divided into two categories: model-free or model-based [3], [15]. In a model-free approach, a player model is established for adaptation based on human players' gameplay data, e.g., playlog and controllable parameters in a game, and their subjective feedback, e.g., affective states such as "fun," while player model in a model-based approach is derived from psychological emotion theories, e.g., [18]–[20]. To the best of our knowledge, most of the existing SMB adaptation methods fall into the model-free category and model-based adaptation has yet to be investigated for SMB level adaptation. In particular, the real-time model-based DDA that the content difficulty is adjusted dynamically within a level generation still remains unexplored in SMB level adaptation research up to now.

Motivated by the learning-based PCG (LBPCG) framework [14], where a quality evaluation function may be learned from games/levels annotated by game developers, we recently proposed an approach [21] for generating short game segments of high quality, named constructive primitives (CPs), for SMB based on a modified version of *Infinite Mario Bros* [5]. In this approach, easy-to-design rules are first employed to remove apparently unappealing game segments and a content quality evaluation function working on CPs is then established via machine learning. This hybrid method would allow for addressing the quality evaluation issue more effectively. Nevertheless, this CP learning idea was neither investigated sufficiently nor evaluated thoroughly in our previous work [21]. In addition, our CP-based approach was only preliminarily applied to online level generation to demonstrate its usefulness. Hence, its potential still needs to be explored further.

In this paper, we further develop our hybrid CP generation idea with sufficient justification and thorough evaluation. For model-based adaptation, we come up with a novel performance-driven real-time DDA algorithm to explore the potential of our CP-based generation idea by using SMB as a test bed. We have thoroughly evaluated the quality of our CPs via generic yet objective quality measurements and a comparative study. Subsequently, the effectiveness of our proposed CP-based DDA algorithm is thoroughly evaluated via simulations with

sophisticated SMB agents of different types. Experimental results suggest that our data-driven evaluation function effectively encodes multiple quality-related criteria in an implicit way and our model-based DDA algorithm working on CPs yields favorable results in real-time adaptation.

The main contributions of this paper are summarized as follows.

1) The further development of our hybrid approach to quality assurance of short game segments in SMB [21], which lays the foundation for a new level generation approach in different scenarios.
2) A novel CP-based DDA algorithm for real-time adaptation in SMB.
3) A thorough evaluation of our hybrid approach to quality assurance and our CP-based DDA algorithm via simulation.

The remainder of this paper is organized as follows. Section II reviews the relevant works. Section II presents our approach in generating CPs in SMB. Section IV describes our CP-based DDA algorithm. Section V reports experimental results, and the last section discusses remaining issues and relates ours to previous works.

II. Related Work

In this section, we review the relevant works that motivate our study presented in this paper.

Constructive PCG has been widely applied in adaptive SMB level generation [6]–[9], [11], where human experts design a set of constructive rules that can be converted into high-level game parameters corresponding to concrete game levels. In particular, there exist several constructive approaches in SMB [8], [9], [11] that first generate game segments by exploring the local properties and then merge segments to generate a complete game level via a set of constructive rules. In general, constructive rules were handcrafted with developers' elaborate knowledge on different content features and their remarkable skills in rule formulation. While constructive rules may make content adaptation easier, they have to ensure that the generated content is playable and appealing to individual players, which heavily relies on the understanding of content space and sophisticated knowledge/skills in rule formulation. To ensure that the generated content is playable and appealing to individual players, the rule design is rather difficult especially when the content space is complex. Unlike existing constructive SMB generators, our work presented in this paper retains the favorable properties of constructive generators but addresses the content quality assurance issue in a different manner.

DDA is a major technique used in adaptive content generation, where the difficulty of content is dynamically adjusted to match an individual player's skill level and/or to optimize their cognitive/affective experience during gameplay [3], [15]. As described in Section I, DDA can be carried out by either a model-free or a model-based method [3], [15]. To the best of our knowledge, however, all the existing SMB level adaptation methods [6]–[10] are model-free; in other words, no model-based DDA has been studied for SMB. Nevertheless, model-based DDA has been studied widely in a variety of games, including real-time strategy [23], prey/predator [22], fighting [27], role playing [25], and car racing [26] games, especially for nonplayer character (NPC) adaptation. In general, model-based DDA works on the assumption that maximum gameplay enjoyment occurs with the content of a moderate challenge [18]–[20], [22], and can be carried out by different techniques such as evolutionary algorithms (EAs), e.g., [22], [23], reinforcement learning (RL), e.g., [25], [27], and their combination, e.g., [26]. In general, EAs are capable of finding proper game content based on given score metrics from large high-dimensional content space [24] but less efficient in carrying out the real-time DDA due to the population-based computation. In contrast, RL enables a generator to make a rapid adaptation but is more suitable for low-dimensional space exploration [24]. Some efforts [26] have been made to exploit the synergy between EAs and RL in a car racing game. Motivated by the existing model-based DDA approaches and taking the nature of our CPs into account, we employ RL to come up with a model-based DDA algorithm for real-time adaptive SMB level generation. Here, we emphasize that our algorithm works on game geometry adaptation rather than NPC behavior adaptation studied in most of existing model-based DDA approaches.

III. CP Generation

In this section, we first describe our motivation and main ideas. Then, we present our approach in learning CPs in SMB.

A. Overview

Existing SMB level generators work on a huge content space that contains all the complete procedural levels. As there are an enormous variety of combinations among game elements and structures at a procedural level, such a content space inevitably leads to a greater challenge in managing quality assurance and making content adaptation. Nevertheless, a complete procedural level in SMB may be decomposed into a number of short segments as done in [8], [9], [12], [16], [41]. Thus, partitioning a procedural level into fixed-size game segments results in a new content space of lower complexity, which allows us to explore the SMB content space from a different perspective. It is anticipated that the content space of short game segments would facilitate a constructive PCG approach in tackling content quality assurance and real-time content adaptation effectively.

For quality assurance, there are generally two methodologies in developing such a mechanism in PCG [1], [14]: deductive versus inductive. To adopt the deductive methodology, game developers have to understand the content space fully and have skills in formulating/encoding their knowledge into rules or constraints explicitly. In the presence of a huge content space, however, it would be extremely difficult, if not impossible, to understand the entire content space, which might lead to less accurate (even conflicted) rules/constraints used in PCG. Nevertheless, we observe that some constraints in SMB are easy to identify. For example, overlapped tubes are unacceptable and easily detected with a simple rule. On the other hand, a learning-based PCG (LBPCG) framework [14] was proposed, where an

Fig. 1. Constructive primitive (CP) generation process for SMB.

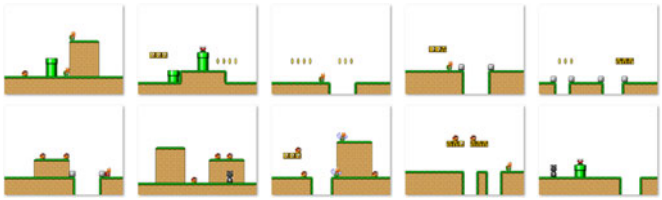

Fig. 2. Typical 20×15 game segment instances.

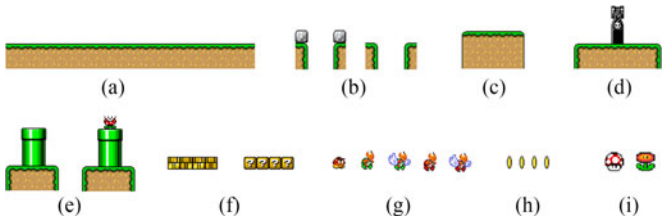

Fig. 3. Game design elements. (a) Initial platform. (b) Gaps without/with rock decoration. (c) Hill. (d) Cannon. (e) Tubes without/with flower enemy. (f) Boxes without/with coins/powerup. (g) Enemies. (h) Coins. (i) Mushroom and fire flower.

inductive methodology is advocated for quality assurance via learning from annotated data. As video game content is observable but less explainable, it is easier for developers to judge the quality of games (especially in the presence of complex content elements and structures) via visual inspection (applying their knowledge implicitly) than to formulate their knowledge into rules formally with sophisticated skills. Thus, a quality evaluation function may be learned from games/levels annotated by developers. This observation suggests that exploiting the synergy between rule-based and learning-based methods would allow for addressing the quality evaluation issue more effectively.

With the motivation described above, we propose a hybrid approach for generating CPs (quality yet controllable game segments) in SMB, where both simple rules and a learning-based method work together. Fig. 1 illustrates the main steps of our approach. First, game developers choose a region that covers main variations of game elements and structures in SMB from the entire content space via controllable parameters. Then, all the game segments in this region are evaluated by a set of easy-to-design conflict resolution rules. Finally, subsequent data-driven quality evaluation function deals with more complicated quality issues. Surviving game segments form CPs to facilitate generating adaptive SMB levels.

B. Content Space and its Representation

A recent study [12] suggests that in SMB, game segments of 20 in length and 15 in height (i.e., the approximate size of a screenful of tiles in SMB) are sufficient to express rich yet diverse types of game elements and structures in a procedural level for 2-D platform games. As exemplified in Fig. 2, such game segments are naturally specified by a 2-D grid similar to an image. However, this leads to a 300-D content space of massive redundancy, e.g., the uniform background. Instead of the 2-D grid representation, we employ a list of design elements, atomic units in PCG, as our content representation, including initial platform, gaps without/with rock decoration, hill, cannon, tubes without/with flower enemy, boxes without/with coins/powerup,enemies, coins, and mushroom and fire flower, as illustrated in Fig. 3. By using this representation, we not only specify the content space concisely but also gain the direct control on low-level geometrical features, e.g., coordinates of tubes and coins. As listed in Table I, 85 content features used in our representation are grouped into 15 categories, where a content feature is used to specify and control a design element in a segment. Our representation is similar to those used in the previous work [35], [42]. In this representation, $x, y, width, height,$ and $type$ refer to x, y coordinates, width, height, and type of each design element, while w_{before} and w_{after} refer to width of the platform before and after each tube/cannon. Among these features, types of gap, tube, boxes, and enemies are nominal features, while the rest are ordinal features. In our content space, the design elements in each type are sorted in a decreasing order along the x dimension. However, there are several redundant parameters in this representation, e.g., if the number of cannons is one, then the parameters for the second and third cannons will be useless. In this case, we simply set all the meaningless parameters to zero. Here, we emphasize that the design elements listed in Table I are fully controllable; game developers can control relevant content features to generate a specific game segment. For example, a mountainous segment may be generated by setting the number of hills to 1 or 2 and other features randomly, while a pure linear segment is generated by setting the number of hills, y coordinate of tubes, and cannons to zero.

TABLE I
CONTENT FEATURES

ID	Description
1	$height$ of initial platform
2	number of gaps
3–11	x, $width$ and $type$ of the first–third gap
12	number of hills
13–18	x, $width$ and $height$ of the first and second hill
19	number of cannons
20–34	x, y, $height$, w_{before} and w_{after} of the first–third cannon
35	number of tubes
36–53	x, y, $height$, w_{before}, w_{after} and $type$ of the first–third tube
54	number of boxes
55–62	x, y, $width$ and $type$ of the first and second boxes
63	number of enemies
64–78	x, y and $type$ of the firstfifth enemy
79	number of coins
80–85	x, y and $width$ of the first and second coins

While design element parameters in Table I have a huge range that specifies the entire content space, we confine our concerned content space to a nontrivial region by setting the

maximum number of gaps, hills, tubes, cannons, boxes, coins, and enemies to 3, 2, 3, 3, 2, 2, and 5, respectively, in a game segment. Consequently, there are roughly 9.72×10^{37} different game segments in the content space used in our experiments. This content space is sufficient to generate content with a variety of geometrical features, level structures and difficulties required by SMB. We believe that a larger content space may contain more unplayable or aesthetically unappealing game segments. For instance, a 20×15 game segment containing more than five enemies or more than three cannons/gaps is too difficult for human to play and appears aesthetically unappealing if there are more than three tubes inside.

C. Conflict Resolution

As the content representation described in Section III-B results in overlapped design elements, there are a large number of game segments that contain conflicting design elements in our content space. For instance, "*...Tube(6, 0, 2, 0, 0, flower)...Cannon(6, 0, 4, 0, 0)...*" represents a game segment of at least one tube and one cannon, where x, y, $height$, w_{before}, w_{after}, and $type$ of this tube are 6, 0, 2, 0, 0, and $flower$, and the x, y, $height$, w_{before}, and w_{after} of this cannon are 6, 0, 4, 0, and 0. As both the tube and the cannon share the same x coordinate, the cannon and the tube are overlapped in this game segment. This conflict makes the segment aesthetically unappealing.

To address this issue, we first discard those game segments of geometrically conflicted design elements via a set of simple rules in our approach. Our rules for the above situations are similar to those presented in [35] and [42], i.e., any overlapping among gap, enemy, tubes, cannons, boxes, and coins are forbidden, whilst hills of different heights are allowed to be overlapped and enemy/tube/cannon may also be overlapped with hills in a game segment.

Application of the above-mentioned rules to the entire content space leads to a tailored content space. Learning CPs merely works on this tailored content space.

D. Learning CPs

After filtering out those obviously unappealing game segments via rules described in Section III-C, the tailored content space still contains a lot of low-quality segments, e.g., segments of unreachable coins and boxes, unplayable segments, segments of unbalanced resources, and aesthetically unappealing structures. Inspired by the LBPCG framework [14], we would learn a quality evaluation function from annotated game segments to detect unplayable/unacceptable segments. Thus, the problem in our CP learning is formulated as binary classification. For such a classifier, its input is the 85-D feature vector of a game segment and its output is a binary label that predicts the quality of a game segment.

To carry out the aforementioned idea, however, we have to tackle a number of nontrivial problems. After conflict resolution, there are still a huge number of game segments in the tailored content space. It is computationally intractable to deal with all the segments in this content space. To tackle this problem, *sampling* appears to be a suitable technique as it can generate a much smaller dataset of the same properties owned by the original content space. To make a learning-based quality evaluation function, training examples are essential but have to be provided by game developer(s). Although the use of sampling leads to a computationally manageable dataset, annotating all the segments in this sampled dataset is not only laborious and time-consuming but also may not be necessary if their distribution can be estimated. Clustering analysis provides a manner for exploring the distribution and the structure underlying the sampled dataset. Furthermore, we adopt the active learning methodology that enables us to train a classifier with only a small number of most informative game segments annotated by game developer(s) during active learning. To carry out an effective active learning, we exploit the clustering results to minimize the number of annotated game segments required by active learning since game segments residing in all the different clusters are likely to hold the main properties of the entire tailored content space. Thus, clustering analysis not only facilitates active learning but also reveals nontrivial properties of a content space, which allows for using other techniques, e.g., visualization, to understand a large content space and to identify the cause of misclassification in active learning (cf., Section V-A).

Fig. 4. Constructive primitive learning process.

As depicted in Fig. 4, our CP learning process consists of three stages: sampling (to be computationally tractable), clustering analysis (to explore the distribution and the structure underlying the sampled dataset), and active learning (to establish an effective classifier with minimal labeled game segments backed by the clustering analysis). The main techniques used in each stage of our CP learning are presented below.

1) Sampling: For sampling, we apply the simple random sampling (SRS) with replacement [28] to the tailored content space for a manageable dataset. In comparison with other sampling approaches, SRS is unbiased and enables each game segment in this tailored content space to be selected with the equal probability. Moreover, it is capable of handling unknown data distribution without using any prior knowledge. Thus, SRS becomes a proper sampling technique for our approach as we do not know the accurate distribution of our tailored content space.

In sampling, we have to address the issue on the size of a sample that can retain all nontrivial properties underlying the tailored content space. Due to its unknown distribution, we need to apply the sample size determination (SSD) algorithm suggested in [28] and [29] to determine the size of our sample. According to the SSD algorithm, the proper sample size *N* for a given dataset is estimated by

$$N \approx \frac{z^2\delta^2}{d^2} = \frac{1.96^2 \times 49.7749}{0.10^2} = 19121.32 \approx 19\,000 \quad (1)$$

where z stands for the z-score of 1.96 for a 95% confidence interval, and d is 0.10 that refers to a small tolerable difference. These values are default in the SSD algorithm. δ^2 refers to

the maximum variance of the content space among different features. The value of δ^2 is 49.7749, which is approximated by the largest variance estimated on several samples of the tailored content space [29]. With the approximation in (1), we sample the tailored content space to yield a sampled dataset of 19 000 game segments.

2) Clustering: For clustering analysis, we employ the CURE algorithm [30] on the sampled dataset since it is applicable to a dataset of unknown data distribution [31] and can uncover the structure underlying data by automatically determining the proper number of clusters underlying a dataset [32]. To use the CURE, one has to preset four hyperparameters: the number of clusters, sampling rate, shrink factor, and the number of representative points. By using the dendrogram tree resulting from this hierarchical clustering algorithm, the number of clusters is automatically decided by the longest *k*-cluster lifetime defined in [32]. The rest of the parameters are set to defaults as suggested in [30]; i.e., 0.5 for shrink factor, 2.5% for sampling rate, and 10 representative points, respectively.

As there are three different feature types in our content representation, i.e., numeric, nominal, and ordinal, we employ the mixed-variable distance metric [31] in the CURE. The mixed-variable distance $d(i, j)$ between objects i and j is defined by

$$d(i,j) = \frac{\sum_{f=1}^{F} d_{ij}^{(f)}}{F}$$

where F denotes the number of features in the content representation and $d_{ij}^{(f)}$ refers to the contribution of feature f to the overall distance between objects i and j. Suppose the value of f for the ith object is x_{if}, the distance regarding this feature $d_{ij}^{(f)}$ is measured with a metric dependent on its feature type.

1) When f is *numeric*: $d_{ij}^{(f)} = (|x_{if} - x_{jf}|)/(\max_h x_{hf} - \min_h x_{hf})$, where h runs over all objects in the dataset for feature f.
2) When f is *nominal*: $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$ and $d_{ij}^{(f)} = 1$ otherwise.
3) When f is *ordinal*: compute the $z_{if} = (r_{if} - 1)/(M_{if} - 1)$, where $r_{if} \in \{1, \ldots, M_f\}$ are the corresponding ranks of M_f ordered states that f has. Then, an ordinal feature is converted into numeric via z_{if}.

The above treatment used in the mixed-variable distance metric normalizes distances contributed by different features [31], respectively. These normalized distances tend to eliminate the bias introduced by a feature of a large permissible range and make the overall distance $d(i, j)$ within a range of $[0, 1]$.

Clustering analysis with the CURE algorithm on the sampled dataset leads to 58 clusters, and the detailed experimental results are reported in Section V-A1.

3) Active Learning: For active learning, a validation set that holds main properties of the content space is required to evaluate/monitor the generalization performance during active learning. By considering the segment distribution information, we first select a small number of segments randomly from each cluster to form such a validation set of 800 segments totally, where the number of segments selected from each cluster is proportional to the cluster size. The first author of this paper, an experienced game player, labels all 800 game segments in the validation set via visual inspection; up to 5 s are taken in annotating any segments in our experiments. As a result, a binary label is assigned to a segment to indicate its quality; $+1/-1$ is a label of good/poor quality. The qualitative labeling criteria are based on the annotator's own knowledge/experience and guided by the existing literatures. Below, we summarize the main aspects considered during the annotation.

1) *Playability*: A path should exist between the entrance of the game level and the exit, which allows players to finish the game [1].
2) *Resource balance*: Design elements within a quality segment should be distributed in terms of balance [2], [39], [42].
3) *Difficulty curve*: A quality segment should not contain unexplainable difficulty spikes/curve [2], [42].
4) *Reachability*: All the resources (e.g., coins and boxes) in a quality segment should be reachable [13].
5) *Aesthetic properties*: A quality segment should contain meaningful, aesthetically appealing, and reasonable structures [2], [12], [39].

Here, we emphasize that the annotator labels a game segment via visual inspection only by taking into account the above-mentioned aspects. In other words, the annotator merely applies his knowledge/experience implicitly in visual inspection (without any explicit yet well-formulated rules in mind).

In general, there are two error types in binary classification: false negative (type-I error), where a high-quality segment is misclassified as low quality and false positive (type-II error), where a low-quality segment is misclassified as high quality. Obviously, a type-II error could result in a catastrophic effect, while a type-I error is less harmful as it simply shrinks the content space slightly. As a result, we formulate our classification as a cost-sensitive learning problem, where the type-II error incurs a higher cost in learning. By looking into several state-of-the-art classification techniques, we found that the weighted random forest (WRF) [33], a cost-sensitive oblique random forest [34] classifier, fully meets our requirements for active learning. As an ensemble learning approach, a WRF works based on decision trees trained on randomly partitioned data subsets for classification. Such a classifier can handle a dataset of different feature types and offers the feature importance information during active learning, which is extremely useful for developers. To address the cost-sensitive issue, the WRF applies class weights to the information gain during finding optimal splits. In the terminal nodes of each tree, the class prediction is determined via weighted majority vote. The final class prediction of the WRF is also determined via the weighted vote of individual trees. In our experiments, the "optimal" hyperparameters of the WRF are decided via validation suggested in [33]: 1.3:1 for cost ratio, 30 trees, 5 combined features, 30 feature groups selected at each node, and 9 in depth.

Active learning is undertaken on the entire sampled dataset excluding 800 segments used to form the validation set described at the beginning in Section III-D3. We initially choose 100 segments randomly and label them via visual inspection. Then, we use these 100 examples to train an initial WRF. Based

Algorithm 1: Active Constructive Primitive Learning.

Input: Sampled data set U and clustering results on U.
Output: WRF binary classifier.
Initialization: Based on the clustering analysis results, create a validation set V of 800 examples.
Active Learning:
Annotate 100 segments randomly selected from U via visual inspection to form a training set L.
Train WRF on L to obtain an initial binary classifier.
repeat
  for all $x_i \in U$ **do**
    Label x_i with the current WRF.
    Calculate the uncertainty score s_i of x_i.
  end for
  Annotate 100 segments of the highest uncertainty score in U to form a new training set L.
  Re-train the WRF with the examples in L.
until The overall accuracy on V does not increase.
return Classifier WRF.

on the initial WRF, our active learning works on uncertainty sampling for efficiency. In each iteration of active learning, we find 100 segments of the highest uncertainty scores defined by $s_i = 1 - P(\hat{y}|x_i)$, where $\hat{y}$ is the predicted label of segment x_i and $P(\hat{y}|x_i)$ is the probability of this prediction yielded by the current WRF. Then, we annotate those 100 segments to form new examples for retraining the WRF. The active learning carries on until the accuracy of this WRF on the validation set no longer increases. Our active learning algorithm is summarized in Algorithm 1. The detailed results on active learning are reported in Section V-A2.

Once the evaluation function is learned, it will be used (along with the conflict resolution rules described in Section III-C) to produce CPs in a generate-and-test way. In the next section, we develop an algorithm that combines proper CPs via controllable parameters, which leads to a new adaptive level generator for real-time DDA in SMB.

IV. CP-Based Dynamic Difficulty Adjustment

In this section, we propose a DDA algorithm by applying the CPs described in Section III to real-time content adaptation. This algorithm works on the model-based assumption, i.e, maximum gameplay enjoyment occurs with the content of a moderate challenge [18]–[20], [22], and the appropriateTable II challenge level can be reflected by the game playing performance.

For the model-based DDA, it is essential to define game difficulty levels and to measure the game playing performance. In our CP-based approach, the difficulty-level definition is motivated by the difficulty/leniency metric presented in [35], which suggests that the difficulty of an SMB level may be determined by main design elements. In our work, we use enemies, gaps, cannons, and flower tube to define five difficulty levels by controlling relevant features manually. We further use 14 existing SMB agents [36] to refine our difficulty categories according to their completion rates on different CP-based games. To test

TABLE II
Difficulty-Level Parameter for Game Adaptation

Level	Description
1	number of enemies $= 0$; number of gaps ≤ 2; number of flower tube $= 0$; number of cannons $= 0$; width of gap < 3; gaps without rock
2	number of gaps $= 0$; number of enemies $= 1$; number of cannons $= 0$; number of flower tube $= 0$
3	number of gaps ≤ 1; width of gap < 3; $1 \leq$ number of enemies ≤ 2; number of cannons $= 0$
4	number of cannons ≤ 1; $1 \leq$ number of enemies ≤ 4
5	number of cannons ≥ 1; number of enemies ≥ 3

Algorithm 2: Generating a CP of a Specified Difficulty Level.

Input: A specified difficulty level.
Output: A CP of this specified difficulty level.
Generation:
Choose a region of interest from the content space via difficulty parameters described in .
repeat
  Randomly select a game segment g from the region of interest.
  Evaluate game segment g via conflict resolution rules and data-driven evaluation function.
until Evaluation results are positive.
return Game segment g

the effectiveness of our difficulty level definition, each agent plays 30 games at each difficulty level. The averaging complete rates at difficulty levels 1–5 are 0.71, 0.54, 0.46, 0.42, and 0.38, respectively. It suggests that our difficulty level definition summarized in Table II can considerably affect the completion rate. As described in Algorithm 2, a CP of the specified difficulty level can be generated by controlling the content features shown in Table II with the support of the conflict resolution rules and the data-driven quality evaluation function described in Section III. Measuring a player's performance may be complicated if all different factors have to be considered. In this paper, we only take into account the survivability on CPs, which will be discussed later on.

In our work, we make use of CPs for real-time DDA, i.e., the performance is measured locally on a CP rather than on a procedural level and the DDA is done instantly in response to a player's performance on the last played CP. This is naturally a sequential decision process as only the performance regarding the last played CP affects the difficulty level of the next CP to be generated. Thus, we formulate this as a Markovian decision process (MDP) [37] and our goal is hence to find an optimal policy that maximizes the expected long-term performance via adjusting the difficulty of generated CPs. To attain this goal, we define a regret ρ at time T (i.e., after T CPs have been played) as the absolute difference between an expected survival rate and

the expectation of rewards

$$\rho = \left| \theta_{\text{opt}} - \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} r_t\right] \right| \tag{2}$$

where θ_{opt} is a preset optimal survival rate on any CP in a level and r_t is a binary reward: $r_t = 1$ if the player survives on the CP played at time t and 0 otherwise. For instance, $\theta_{\text{opt}} = 0.8$ means that the performance of the survival rate 0.8 is expected for any CP in an adaptive level. For a given θ_{opt}, levels of proper difficulties can be generated for a player to gain the expected performance. Thus, minimizing the regret in (2) is key to our DDA. To solve this optimization problem, we employ the Thompson sampling [38], an effective and efficient heuristic method used in binary reward case in MDP.

Let θ_i denote a player's survival rate at difficulty level i ($i = 1, \ldots, 5$). Thus, a player survives with the probability θ_i and reciprocally dies with the probability $1 - \theta_i$ when they play a CP of difficulty level i. During gameplay, one plays a number of CPs of different difficulty levels sequentially. When a CP of difficulty level $a_t = i$ is completed at time t, a binary reward is assigned. Given the player's survival rate θ_i corresponding to a_t, the reward likelihood is

$$P(r_t | a_t = i, \theta_i) = \theta_i^{r_t}(1 - \theta_i)^{1 - r_t}. \tag{3}$$

Furthermore, let $D_T = (a_t, r_t)_{i=1}^{T}$ denote the historical profile regarding corresponding difficulties and rewards after T CPs have been played. By using a conjugate prior, $\theta_i \sim \text{Beta}(1, 1)$, the posterior distribution of survival rate based on the likelihood in (3) is $P(\theta_i | D_T) \propto \prod_{t=1}^{T} P(r_t | a_t, \theta_i) P(\theta_i)$, which leads to $\theta_i | D_T \sim \text{Beta}(\alpha_i + 1, \beta_i + 1)$, where α_i and β_i are the number of survives and deaths when playing the CPs of difficulty level i in D_T.

For content adaptation, we follow the typical setting in real SMB games: at the beginning, a player is put in small state, i.e., the weakest form of Mario, where the player is not allowed to use powerful weapons (e.g., throwing fireballs) and then turns into other states by powering up with a mushroom or fire flower. Based on the gameplay information recorded in D_T, CPs are randomly produced according to $\text{Beta}(\alpha_i + 1, \beta_i + 1)$ and the CP of difficulty level i is chosen as a game segment to play if it results in the least regret defined in (2). After a CP of difficulty level i is played, the posterior probability $\text{Beta}(\alpha_i + 1, \beta_i + 1)$ is updated based on the performance of the CP. Thus, this content adaptation process continues until a player quits. Our real-time DDA is summarized in Algorithm 3.

Algorithm 3: CP-Based DDA for Real-Time Content Adaptation.

Input: Optimal survival rate θ_{opt}.
Initialization: $t \leftarrow 0$, $\alpha_i \leftarrow 1$ and $\beta_i \leftarrow 1$ for $i = 1, \cdots, 5$.
repeat
  $t \leftarrow t + 1$.
  if $t == 1$ or $r_{t-1} == 0$ **then**
    For each $i \in \{1, \cdots, 5\}$, sample θ_i from $Beta(\alpha_i, \beta_i)$.
    Choose action $a_t = \arg\min_i |\theta_{\text{opt}} - \theta_i|$.
  else
    For each $i \in \{\max(1, a_{t-1} - 1), a_{t-1}, \min(a_{t-1} + 1, 5)\}$, sample θ_i from $Beta(\alpha_i, \beta_i)$.
    Choose action $a_t = \arg\min_i |\theta_{\text{opt}} - \theta_i|$.
  end if
  Generate the chosen CP of a_t using Algorithm 2.
  if $r_t == 1$ **then**
    $\alpha_{a_t} \leftarrow \alpha_{a_t} + 1$. //the player survives
  else
    $\beta_{a_t} \leftarrow \beta_{a_t} + 1$. //the player dies and a new game starts
  end if
until Gameplay stops.

V. Experimental Results

In this section, we report experimental results on the CP learning and the real-time DDA. The game engine adopted in our experiments is a modified version of the open-source *Infinite Mario Bros* used in the *Mario AI Championship* [11], [36]. The source code used in our experiments as well as more experimental results not reported in this paper are publicly available on our project website.[1]

[1]http://staff.cs.manchester.ac.uk/~kechen/CP-Mario.html

TABLE III
Relationship Between Clusters and Main Features

#Clusters / Value / Feature	0	1	2	3	4	5
Number of Gaps	56(46)	54(9)	21(3)	6(0)	-	-
Number of Hills	57(48)	49(9)	30(1)	-	-	-
Number of Cannons	50(34)	55(15)	33(9)	13(0)	-	-
Number of Tubes	51(39)	45(16)	17(3)	9(0)	-	-
Number of Boxes	39(22)	53(19)	33(17)	-	-	-
Number of Enemies	43(9)	47(15)	52(10)	55(6)	53(7)	42(11)
Number of Coins	50(30)	56(19)	33(9)	-	-	-
Difficulty Level	-	26(3)	29(0)	41(6)	54(42)	36(7)

A. Results on CP Learning

Based on the learning algorithms described in Section III, we report results on clustering analysis and active learning to explore our content space and to demonstrate the benefit of our data-driven evaluation function.

1) Cluster Analysis: Application of the CURE clustering algorithm described in Section III-D2 leads to 58 clusters. While all the clustering results are available on our project website, Table III summarizes the nontrivial properties of clusters on seven content features that mainly affect the appearance and styles of game segments. As a content feature has a permissible value range, we hence report the properties of clusters based on each permissible value of a feature in Table III, where $M(N)$ denotes that there are M clusters containing those game segments of a specific value for a given content feature and N out of those M clusters have more segments of this value than those of different values regarding this content feature. For example, the first element in Table III is with respect to all the segments containing no gap, i.e., the number of gaps equals zero, and $56(46)$

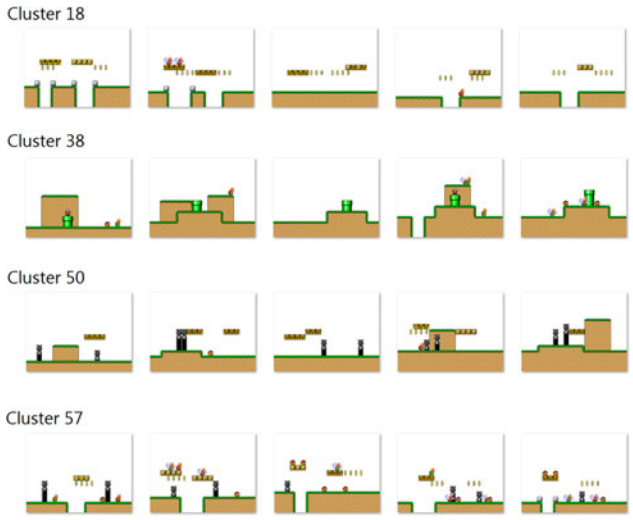

Fig. 5. Exemplary game segments in different clusters.

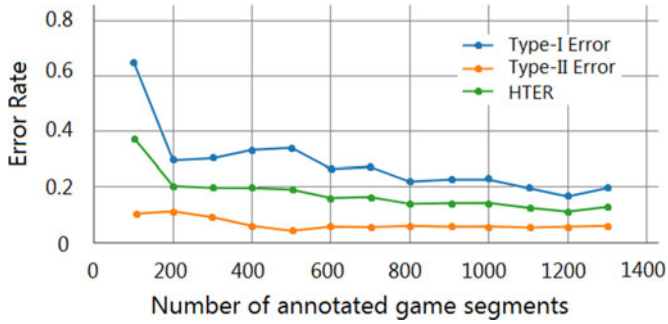

Fig. 6. Performance evolution on the validation set during active learning.

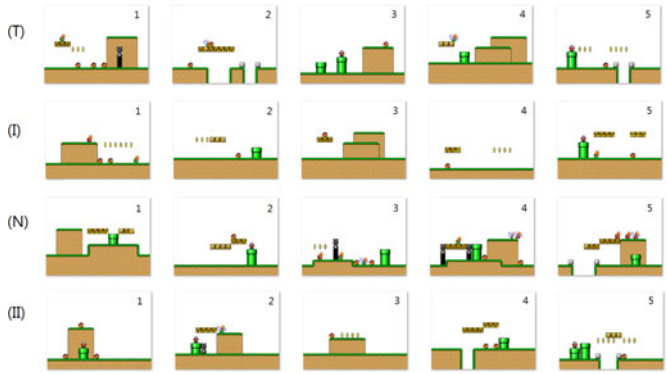

Fig. 7. Test game segments labeled by our classifier. (T) Correctly classified positive segments. (I) Segments leading to type-I error. (N) Correctly classified negative segments. (II) Segments leading to type-II error.

indicates that such segments appear in 56 clusters, where 46 out of those 56 clusters have more segments containing no gap than those with gaps.

In general, M states the number of clusters a feature is distributed and N indicates the number of clusters this feature is predominant. Hence, both M and N facilitate the understanding of the segment distribution in the tailored content space. For instance, M is 57, 49, and 30, respectively, corresponding to the value of feature NumHills $= 0$, 1, and 2. It suggests that those segments with the feature NumHills of different values are widely spread in many clusters. In contrast, the corresponding N values are 48, 9, 1, respectively, which indicates hills of different numbers are not distributed uniformly in those clusters containing hills. It is observed from Table III that segments of different content features are not distributed uniformly, and the tailored content space contain more segments specified by only a few design elements since segments with many design elements are likely to commit to illegitimate overlapping and hence removed by the conflict resolution rules prior to clustering. Thus, the use of conflict resolution rules causes a bias to segments of fewer design elements in our CP generation. The clustering results also reveal the distribution of segments of different difficulty levels (cf., Table II). Here, we adopt the same notation to report the distribution of segments of difficulty levels 1–5 in Table III. For instance, 26(3) in the bottom row of Table III indicates that there are 26 clusters containing segments of level 1, where only three clusters contain more segments of level 1 than those belonging to other levels.

In Fig. 5 we provide some examplary game segments in different clusters. The following is observed:

1) segments in cluster 18 have at least one set of boxes and two sets of coins;
2) segments in cluster 38 include at least one tube;
3) segments in cluster 50 contain at least two sets of boxes and one cannon;
4) there are at least three enemies, one cannon, and one set of boxes in segments belonging to cluster 57.

The exemplification shown in Fig. 5 suggests that our clustering analysis is meaningful since segments within the same cluster contain similar content while those in different clusters vary in terms of geometry and structures. Hence, we use the clustering results to facilitate the subsequent active learning.

2) Active Learning: Fig. 6 illustrates the evolutionary performance of our active learning on the validation set described in Section III-D3, including type-I and type-II error rates as well as their average, the half total error rate (HTER). It is shown in Fig. 6 that active learning reaches an optimum at 1200 queries with a total of 1300 queries. While the optimal HTER is around 11.18%, the corresponding type-I and -II error rates are around 16.67% and 5.69%, respectively. Fortunately, those segments resulting in type-II error do not include unplayable game segments in our experiments; in other words, all the unplayable segments in validation set were classified correctly.

To demonstrate our active learning results vividly, we show several correctly classified and misclassified segments in Fig. 7. For clarification, segments are correctly classed if their labels assigned by our classifier are consistent with the developer's labels and those are misclassified otherwise. Segments in Fig. 7(T) are correctly labeled as positive. These high-quality segments contain playable structures, reachable resources, and a meaningful combination capturing an area of challenge and conveying a sense of progression. However, these segments generated by our approach do not seem to appear in handcrafted SMB levels.

TABLE IV
TOP 20 MOST IMPORTANT FEATURES IN THE CONSTRUCTIVE PRIMITIVE LEARNING

Rank	Description	Rank	Description	Rank	Description	Rank	Description
1	x coordinate of the first enemy	6	$width$ of the first hill	11	$width$ of the first box	16	$width$ of the second box
2	y coordinate of the first box	7	y coordinate of the first cannon	12	number of enemies	17	$type$ of the second box
3	x coordinate of the first hill	8	$type$ of the first enemy	13	y coordinate of the first enemy	18	x coordinate of the second enemy
4	$height$ of the first hill	9	$width$ of the second coin	14	$width$ of the first hill	19	y coordinate of the second enemy
5	number of hills	10	x coordinate of the first box	15	y coordinate of the first tube	20	y coordinate of the second coin

Segments in Fig. 7(I) should have been labeled as positive since these segments look like appealing short episodes in a procedural level. Unfortunately, they are misclassified as negative by our classifier, which leads to type-I error. Segments in Fig. 7(N) are correctly classified as negative: Fig. 7(N).1 consists of unplayable structure which does not allow a player to pass through; Fig. 7(N).2 contains simple structures and hence lacks a sense of progression; Fig. 7(N).3 and 5 appear aesthetically unappealing since arbitrarily lumped enemies, gap, boxes, and tubes are less meaningful; and Fig. 7(N).4 contains unreachable boxes. In contrast, segments in Fig. 7(II) should have also been labeled as negative but misclassified by our classifier, which results in type-II error. It is seen that all game elements, i.e., enemy, cannon, tube flower, in Fig. 7(II).1 and 2 are located in a narrow zone in the middle of those game segments. In literature [2], such a situation is named "unexplainable difficulty spikes" that indicates a sudden difficulty increase and hence classified as low quality [2], [42]. Enemy, coins, and hill in Fig. 7(II).3 are concentrated in a narrow zone in the middle of the segment, which is also regarded as low quality in literature [39]. Fig. 7(II).4 and 5 appear aesthetically unappealing by visual inspection.

By means of clustering results, we find that, to a great extent, the nature of the WRF accounts for misclassification as it needs to optimize the information gain in a statistical sense [31], [34]. We observe that game segments located nearby in the content space (i.e., game segments of a short distance) tend to be assigned the same label by our active learner. For instance, Fig. 7(I).5 is a negative example located in cluster 37. The rest of the segments from cluster 37 are also assigned as negative by our classifier. A similar problem occurs on Fig. 7(I).2 and 4. In addition, segments of similar content but distributed in different clusters are also likely to be misclassified. For instance, Fig. 7(N).4 is similar to Fig. 7(II).4 and 5 as they all have boxes above gaps, tubes, and enemies, but these segments are located in different regions of content space due to the combination with other design elements. Likewise, Fig. 7(I).1 and 3 are in a similar situation. Such segments look similar with human visual inspection but often spread over different regions in the content space. This observation resulting from clustering analysis suggests that we are encountering a well-known yet challenging multimodal classification problem, where data within a specific class may distribute in different clusters [31], and it is hard for a classifier including the WRF to yield the error-free performance. Regardless of the WRF, "rare segments," e.g., Fig. 7(II).1–3, are highly prone to misclassification as such segments are scattered widely with low density in the content space and hence less likely to be drawn in queries during active learning.

As a WRF can automatically find proper content features that maximize the information gain [34], we use the WRF to measure and rank the importance of content features in our CP learning. As a result, Table IV lists the top 20 most important features. It is observed from Table IV that the position and geometrical information on game elements, e.g., their width and height, are among the most important features in our data-driven quality evaluation. Moreover, we find that the first design element, e.g., gap, hill, or enemy, plays a more important role than the subsequent design elements in a segment (cf., Section III-B). Due to the nature of the WRF, those content features often lead to small information gain and are hence likely to have a low importance score.

In summary, the above experimental results demonstrate that the evaluation function learned from the sum total of 2100 labeled game segments (1300 for active learning and 800 for validation) effectively eliminates low-quality game segments in terms of its performance on the validation set.

B. Quality Assessment of CPs

While the effectiveness of our data-driven evaluation function has been assessed from a machine learning perspective in Section V-A, we further examine the quality of CPs generated with this evaluation function by using a number of generic yet objective content quality measurements that have been already validated thoroughly in [39].

In our experiment, we employ the objective quality metrics [39] that statistically measure content quality based on a number of procedural levels generated by a level generator, including strategic resource control ($\boldsymbol{f}_s$), area control ($\boldsymbol{f}_a$), area control balance ($\boldsymbol{b}_a$), and exploration ($\boldsymbol{E}$). Among these metrics, $\boldsymbol{f}_s$ measures how close treasures, e.g., coins, boxes, are to enemies; $\boldsymbol{f}_a$ measures how far the placement of design elements, e.g., enemies, tubes, cannons, are away from each other; $\boldsymbol{b}_a$ measures how balanced a distribution of design elements is; and $\boldsymbol{E}$ measures the exploration efforts to reach the exit from entrance. To a great extent, $\boldsymbol{E}$ indicates the content geometric complexity. In our experiment, the $\boldsymbol{E}$ scores are normalized to the range of [0, 1]. The formal definition of these metrics can be found in [39]. Apart from above metrics, we further use Peter Lawford's A* agent [36] to evaluate the playability ($\boldsymbol{P}$) of game levels as this agent can survive on almost all playable levels regardless of their difficulty levels.

As a CP is a short game segment rather than a complete level, it is impossible to make a direct comparison between CPs and complete levels produced by other level generators. To enable

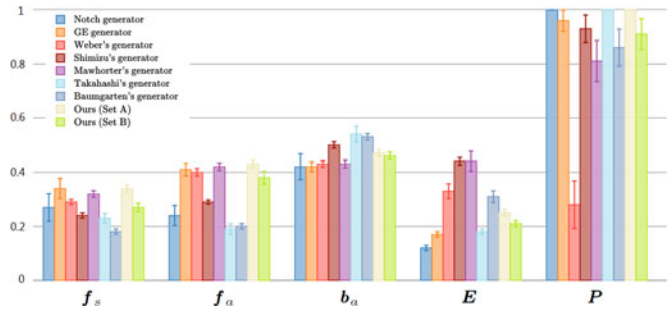

Fig. 8. Quality assessment results with different metrics for various SMB level generators.

the comparison, we generate a complete level by cascading CPs or game segments. In our experiments, we thus generate two sets of procedural levels: Sets A and B. In Set A, a level is composed of randomly selected CPs that are classified as high quality by our data-driven evaluation function, while a level in Set B consists of those randomly generated game segments that survive from conflict resolution rules but are rejected by our evaluation function. For a thorough evaluation, we compare our approach with those sophisticated SMB level generators, notch [5], and grammar evolution (GE) [35], as well as several generators developed for *Mario AI Championship* [11]. Each of the generators, including ours, generates 100 procedural levels for evaluation, where a level has a size of 200 in width and 15 in height. In other words, a CP-based level is composed of ten game segments of equal size: eight CPs, one initial and one ending segments. In our experiments, the default parameters as suggested in their works are always used for a fair comparison.

We conduct a Wilcoxon rank sum test to confirm a hypothesis that our CP-based levels in Set A outperform other SMB level generators stated above in terms of the aforementioned objective quality metrics (via a pairwise comparison). Fig. 8 shows the 95% confidence interval for the mean of quality scores in terms of different metrics.[2] From Fig. 8, it is evident that our levels in Set A generally yield favorable results in terms of different metrics. Our levels in Set A result in a higher $\boldsymbol{f}_s$ score than those generated by all others apart from GE. This suggests that acting as their "guardians," treasures, e.g., coins and boxes, in our CP-based levels are close to enemies [39]. Our levels in Set A further lead to a higher $\boldsymbol{f}_a$ score than all others and a $\boldsymbol{b}_a$ score not lower than all apart from Shimizu's, Takahashi's and Baumgarten's, which implies that design elements, e.g., enemies, tubes, cannons, in our CP-based levels are reasonably distributed in light of balance. As described in Section III-D3, resource balance is one of our labelling criteria, which accounts for higher $\boldsymbol{f}_a$ and $\boldsymbol{b}_a$ scores on Set A over Set B. In general, $\boldsymbol{E}$ reflects the complexity of game geometry. The $\boldsymbol{E}$ scores of levels in Sets A and B appear relatively low due to the content space adopted in our experiments (cf., Section III-B), e.g., there are two hills at most, which leads to less complex levels than those used in other works. For playability measured by $\boldsymbol{P}$, our levels in Sets A along with Notch's and Takahashi's levels do not produce unplayable levels while others do in this experiment. The much higher $\boldsymbol{P}$ value on Set A over Set B benefits from active learning as playability is considered in labeling segments for training examples.

In summary, this comparative study suggests that our data-driven evaluation function works well in generating quality game content. In particular, the substantial gain for levels in Set A over those in Set B clearly shows the effectiveness of our active learning in content quality assurance.

C. Results on Real-Time DDA

According to [7], the use of agents for a DDA test may benefit from stable agents' behavior, their diversified playing styles, and a wide range of skills. In our simulations, we hence employ 14 sophisticated yet different SMB agents (submitted to the Gameplay track of the Mario AI Competition [36]), Lawford's, Sloane's, Baumgarten's, Ellingsen's, Lopez's, Schumann's, Polikarpov's, Forward, Erickson's, Perez's, CyberNeuron, Scared, Tulacek's and Oliveira's agents, to test our CP-based real-time DDA algorithm. For performance evaluation, we use the completion rate and the online learning behavior.

1) Completion Rate: Progression is one of the most commonly used criteria to evaluate a DDA algorithm [36]. As a result, we adopt completion rate, a variant of progress defined in [36], for performance evaluation. Completion rate refers to the ratio of the actual distance travelled over to the length of a game level being played, that is, the distance from the x-coordinate of the start to the x-coordinate where the player died divided by the total width of the level. For reliability, each of 14 agents plays on four sets of adaptive games generated by Algorithm 3 corresponding to different optimal survival rates, $\theta_{\text{opt}} = 0.70, 0.80, 0.87, 0.95$, where each set consists of 200 levels and a level is limited to a maximum length of 200. For a baseline, we also randomly generate 200 levels of the same size and refer them to static games as no DDA is applied and each agent also plays all the games in this static game set.

Fig. 9 shows the 95% confidence interval for the mean of completion rates achieved by 14 agents on four aforementioned game sets. As shown in Fig. 9, Lawford's, Sloane's, and Baumgarten's agents outperform other agents in terms of the averaging completion rate, thanks to the A* algorithm used in their implementation. Hence, we regard these three agents as "skillful players" and, accordingly, all the rest as "novices." It is observed from Fig. 9 that our real-time DDA algorithm works well for all the novice agents given the fact that their completion rates on adaptive game sets are considerably higher than those on the static game set, which implies easier levels were dynamically generated to improve their performance. Nevertheless, the DDA performance varies for three skillful agents. While the completion rates of Baumgarten's agent are achieved as expected, our algorithm does not work well for Lawford's and Sloane's agents. According to [36], these two agents almost always survive from all the procedural levels regardless of difficulty. Thus, our algorithm almost never finds games from the current content space

[2]The $\boldsymbol{P}$ value here differs from that in [40] due to different Mario agents and start state settings; they used Baumgarten's agent in fire state, while we use Lawford's agent in small state.

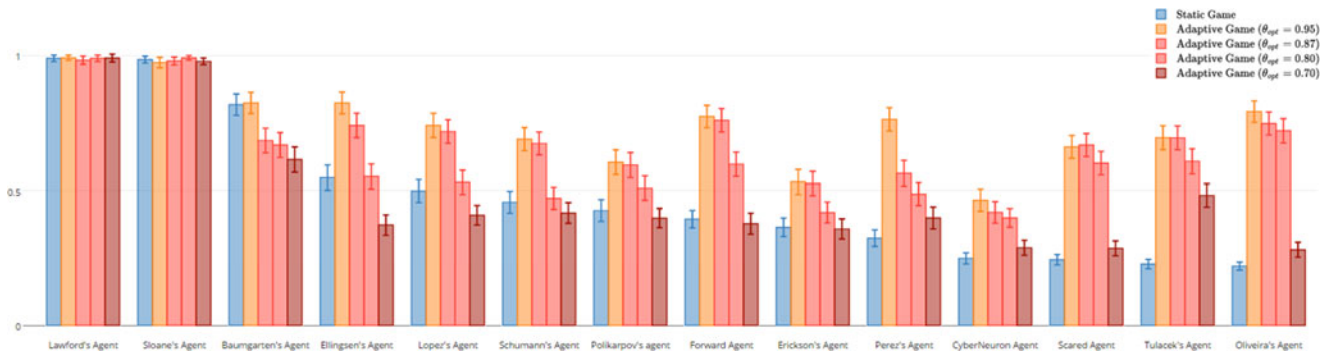

Fig. 9. Results on completion rates yielded by 14 agents on static and adaptive game sets with different θ_{opt}.

(cf., Section III-B) to challenge them unless this content space is expanded by including much more difficult games. From Fig. 9, it is also evident that a higher value of θ_{opt} generally leads to a higher completion rate, which suggests our algorithm works properly for DDA.

2) Online Learning Behavior: We further examine the online learning behavior of our DDA algorithm with three representative agents: Oliveira's, Lopez's, and Baumgarten's agents. Oliveira's and Lopez's agents are novices but have different playing styles, while Baumgarten's agent is a skilful player. In our simulation, each of three agents played in a gameplay episode consisting of 50 successive game levels generated by our DDA algorithm by setting $\theta_{\mathrm{opt}} = 0.80$ and 0.70, where a game level is composed of up to 10 CPs and, in other words, its length is limited to 200 at maximum. For comparison, three agents also played an episode of 50 randomly generated static game levels of the same length. For reliability, we repeated this simulation for 30 trials and the mean and the standard error of completion rates achieved by three agents are shown in Fig. 10.

From Fig. 10(A) and (C), it is seen that the averaging completion rates of Oliveira's and Lopez's agents are significantly higher than those on static games ($p < .005$ in both cases) after playing 5 and 14 game levels, respectively, thanks to the adaptation that generates the levels appropriate to Oliveira's and Lopez's agents. In contrast, the completion rates of Baumgarten's agent gradually decrease as observed from Fig. 10(E), where the averaging complete rates on adaptive game levels are always significantly lower than those on their static counterparts after 17 levels were played ($p < .005$), thanks to the adaptation that keeps generating more and more difficult levels. From Fig. 10(A), (C), and (E), it is also seen that the convergence time of Lopez's and Baumgarten's agents is longer than that of Oliveira's agent since the survival rate of those two agents on static games is closer to the optimal survival rate $\theta_{\mathrm{opt}} = 0.80$. This suggests that the convergence time for a specific agent could be minimized by finding a proper yet personalized optimal survival rate θ_{opt}. Furthermore, it is clearly seen from Fig. 10 that the game completion rates on adaptive games resulting from $\theta_{\mathrm{opt}} = 0.70$ are lower than those on their counterparts corresponding to $\theta_{\mathrm{opt}} = 0.80$ due to increasing difficulties in adaptive games for $\theta_{\mathrm{opt}} = 0.70$. The above simulation results indicate that our DDA algorithm is able to generate easy or challenging games flexibly by controlling the optimal survival rate θ_{opt}.

For demonstration, we illustrate exemplified procedural levels generated by our DDA algorithm for 14 different agents in Fig. 11. In those levels shown in Fig. 11(A)–(C) for three skillful players, Lawford's, Sloane's, and Baumgarten's agents, there are more enemies, cannons, and gaps than those generated for other 11 agents. Those levels shown in Fig. 11(D)–(K) are at the intermediate difficult level, where there are several enemies, gaps, cannons, and flower tubes, which tends to match their skill capacities of Ellingsen's, Lopez's, Schumann's, Polikarpov's, Forward, Erickson's, Perez's, and CyberNeuron agents. In those levels shown in Fig. 11(L)–(N) for complete novices, Scared, Tulacek's, and Oliveira's agents, there exist fewer enemies (two of the three levels have no enemies at all) than those generated for more skillful agents as illustrated in Fig. 11(A)–(K). It is observed from Fig. 11(L)–(N) that those levels generated for three complete novices are much less appealing than others. In our simulations not reported here due to the limited space, we notice that three complete novices generally do not perform well for most of the game levels generated by all the SMB level generators described in Section V-B including ours in terms of game completion given the fact that they are very easily killed by gaps, enemies, tube flowers, and cannons due to a lack of required skills. To enable three complete novices of very limited skills to maintain a survival rate toward $\theta_{\mathrm{opt}} = 0.8$, our DDA algorithm has to find those CPs of few gaps without tube flowers, cannons, and enemies in constructing proper adaptive games, which is responsible for those unappealing levels shown in Fig. 11(L)–(N).

In summary, it is evident from the simulation results reported above that our CP-based DDA algorithm works effectively in dynamically adjusting the content difficulty in response to agents' local performance of playing a level in real time.

VI. Discussion

In this section, we discuss the issues arising from our work and relate ours to previous works.

As elucidated in Section III-D, clustering analysis facilitates understanding our content space and generating a validation set for active learning. A fundamental requirement for effective active learning demands a validation set that consists of only a

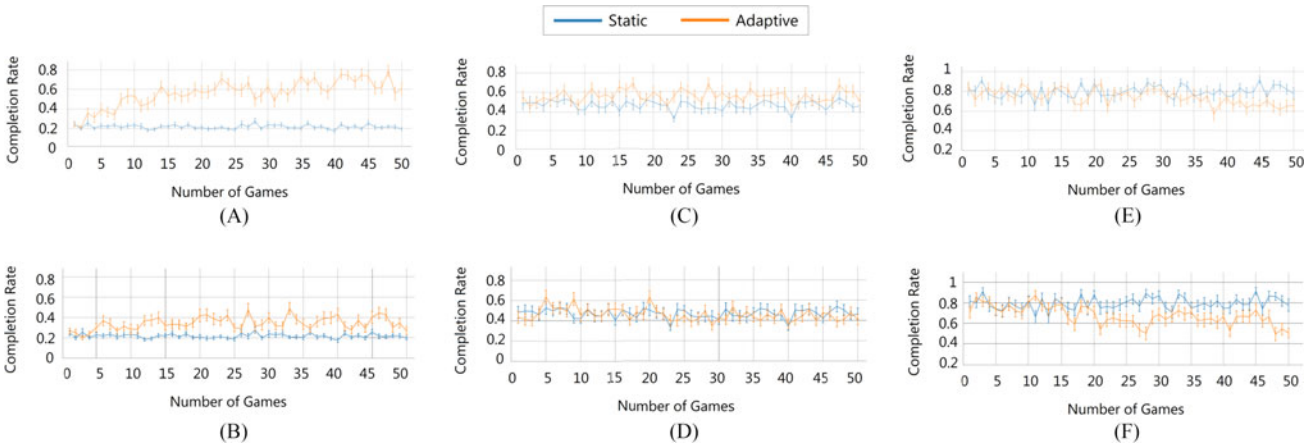

Fig. 10. Completion rates of different agents in a gameplay episode of 30 levels: adaptive versus static. (A) Oliveira's agent ($\theta_{\mathrm{opt}} = 0.8$). (B) Oliveira's agent ($\theta_{\mathrm{opt}} = 0.7$). (C) Lopez's agent ($\theta_{\mathrm{opt}} = 0.8$). (D) Lopez's agent ($\theta_{\mathrm{opt}} = 0.7$). (E) Baumgarten's agent ($\theta_{\mathrm{opt}} = 0.8$). (F) Baumgarten's agent ($\theta_{\mathrm{opt}} = 0.7$).

small number of representative game segments holding as many statistical properties of the content space as possible. There might be an alternative yet more efficient way[3] to generate such a validation dataset. Instead of clustering analysis, one could apply their prior knowledge to choose several key content features and then use those features to partition the content space into a number of regions corresponding to different combinations of their feature values. Likewise, a validation set is formed by choosing a small number of segments from each of the different regions resulting from this partition. While the partition method is more efficient in generating a validation set, our statistical analysis on the representativeness [28], a measurement on the statistical properties that a sample inherits from the sampled population, suggests that a validation set generated with clustering analysis has higher representativeness than that yielded by the aforementioned content space partition method. Despite a computational burden, clustering analysis may be rewarded with a more informative validation set for active learning.

In terms of game content quality, automatically generated SMB levels are generally worse than those handcrafted levels [2]. Apart from constructive approaches reviewed in Section II, there are other approaches in SMB to address this issue but most of them handcraft evaluation functions for quality assurance, e.g., [35], [41], and [42]. However, formulating a complete set of heuristic rules is challenging since game content is observable but less explainable and hard to abstract. Instead, Reis *et al.* [12] merged human-annotated game segments to form quality levels. Unlike ours, theirs has to make a great deal of effort in obtaining enough annotated segments for level generation and needs to address the controllability issue in addition. On the other hand, other researchers aim to generate high-quality levels with human-authored levels rather than domain knowledge. Dahlskog and Togelius [16] used design patterns learned from human-authored SMB levels to generate levels. Snodgrass and Ontañón [43] learned Markov chains from human-authored SMB levels to formulate constructive rules for level generation. Summerville and Mateas [44] used the LSTM to learn sequences from existing human-authored procedural levels and the path information. However, formulating reliable yet effective constructive rules or diverse design patterns from human-authored levels without any domain knowledge might be very difficult since there are a limited number of such levels; there are only 32 human-authored levels in SMB [16], [43]. Thus, additional information, e.g., path information, has to be required to assure the playability of game levels [44]. In contrast, we explore and exploit the synergy between rule-based and learning-based methodologies to produce quality building blocks, *CPs*. Our data-driven evaluation function implicitly encodes multiple quality-related criteria via developers' visual inspection on training segments. Thus, our approach significantly distinguishes from handcrafting constraints, e.g., [35], [41], and [42], formulating constructive rules, e.g., [6]–[9], and learning models/patterns from a limited number of human-authored levels [16], [40], [43], [44]. As a learning-based approach, however, ours does not yield the error-free performance, which implies low-quality segments could appear in a procedural level. In general, different developers may have different opinions on game quality. Although this subjectivity is a general issue regardless of methodologies, our current work is limited as only one developer is employed to annotate data for active learning. Nevertheless, this issue may be tackled via crowdsourcing, which has been attempted in [12].

For real-time DDA, we hypothesize that there is a strong relationship between performance, e.g., survival rate, and gameplay experience based on the previous studies [18]–[20], [22]. Hence, we formulate our DDA as a binary reward MDP problem and solve it with Thompson sampling. It is noticed that some model-based approaches [25]–[27] proposed under other game genres also treat DDA an MDP problem. In their approach, however, the degree of player's survival/winning rate, e.g., Q-learning and dynamic scripting, is not tunable directly. It is also unclear how to use these approaches to tackle a binary reward MDP problem. Nevertheless, our work in DDA is subject to a number of limitations.

1) Our proposed DDA algorithm is model-based and the hypothesis based on psychological theories has yet to be confirmed by human players in SMB.

[3] An anonymous reviewer suggested a partition method.

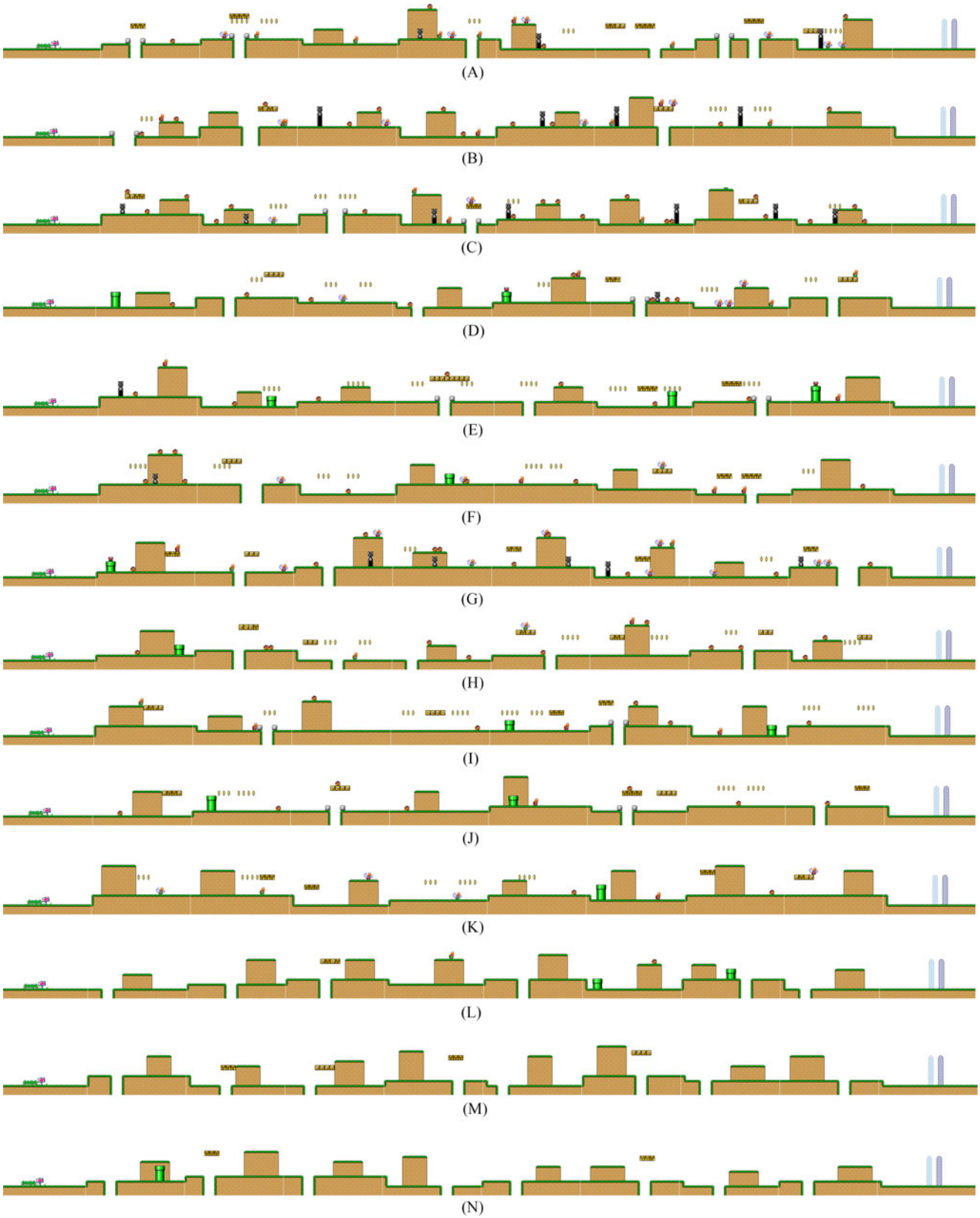

Fig. 11. Exemplar procedural levels generated by our CP-based DDA algorithm for 14 different agents when the optimal survival rate θ_{opt} is set to 0.8. (A) Lawford's agent. (B) Sloane's agent. (C) Baumgarten's agent. (D) Ellingsen's agent. (E) Lopez's agent. (F) Schumann's agent. (G) Polikarpov's agent. (H) Forward agent. (I) Erickson's agent. (J) Perez's agent. (K) CyberNeuron agent. (L) Scared agent. (M) Tulacek's agent. (N) Oliveira's agent.

2) The adaptation process does not seem rapid for agents.
3) It still remains unknown on how to find a proper optimal survival rate for an individual player.

In addition, our rule-based difficulty level definition for game segments (cf., Table II) tends to explore the whole content space for generating diverse games but seem less accurate, which accounts for less stable performance in adaptation.

The previous works [8], [9] have investigated segment-based experience-driven content adaptation in SMB, where they learned a mapping from player's behavior, e.g., playlog, and the segment properties, e.g., controllable parameters, onto the player's affective states. By means of their methods, our CP-based DDA algorithm can be straightforwardly converted into a model-free algorithm for experience-driven DDA. Based on the experience on a CP self-reported by a player, we can establish a mapping function required by a model-free method in the same manner. Moreover, our CP-based generator could adapt game content even more easily since all the properties of game content can be directly controlled via CPs. Unfortunately, the existing model-free methods in SMB [8], [9] entirely rely on the self-reported feedback on short segments, which severely interrupts the gameplay experience [3]. Thus, such an extension does not seem promising. On the other hand, the use of subjective feedback on an entire procedural level has also been studied in SMB, e.g., [6], [7], [10], which could minimize interruption of the gameplay experience. Nevertheless, such techniques are not applicable to real-time DDA studied in this paper. Hence, it remains as an open problem how to adapt content via CPs in the presence of subjective feedback on an entire procedural level.

In conclusion, we have presented a novel approach to real-time DDA in SMB via learning CPs and carried out a thorough evaluation. Experimental results suggest that our data-driven content quality evaluation function works well for content quality assurance and our CP-based DDA algorithm is capable of adjusting content difficulty in real time to match different agents's performance. As our CPs are fully controllable, they are also applicable to online level generation in SMB as demonstrated in our previous work [21]. With the insight into CPs gained in this paper, our previous CP-based online SMB level generation method certainly needs to be further investigated for improvement. In our ongoing work, we are going to tackle all the remaining issues discussed above by exploring state-of-the-art machine learning techniques and alternative content representations. Furthermore, we would explore the feasibility in extending our approach to other video game genres, e.g., first-person shooter.

ACKNOWLEDGMENT

The authors would like to thank the action editor and anonymous reviewers for their valuable comments that improve the presentation of this manuscript. The authors would like to thank N. Shaker for providing SMB procedural level images used in our comparative study and to A. Liapis who responded to our enquiries for useful discussions. An extended abstract of the "CP generation" method described in Section III was presented at the IEEE Conference on Computational Intelligence in Games (CIG'16).

REFERENCES

[1] 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 Games*, vol. 3, no. 3, pp. 172–186, Sep. 2011.
[2] J. Togelius *et al.*, "Procedural content generation: Goals, challenges and actionable steps," *Artif. Comput. Intell. Games*, vol. 6, pp. 61–75, 2013.
[3] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Trans. Affective Comput.*, vol. 2, no. 3, pp. 147–161, Jul.–Sep. 2011.
[4] *Super Mario World*, Nintendo EAD, Kyoto, Japan, 1990 (Game).
[5] P. Persson, "Infinite Mario Bros.," 2008. [Online]. Available: http://www.mojang.com/notch/mario/
[6] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 1, pp. 54–67, Mar. 2010.
[7] N. Shaker, G. N. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," in *Proc. 6th AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2010, pp. 63–68.
[8] S. Bakkes and S. Whiteson, "Towards challenge balancing for personalised game spaces," in *Proc. FDG Workshop Procedural Content Gener.*, 2014, pp. 1–7.
[9] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: A model for dynamic level generation," in *Proc. Artif. Intell. Interact. Digit. Entertainment*, 2010, pp. 138–143.
[10] 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. Artif. Intell. Interact. Digit. Entertainment*, 2012, pp. 304–311.
[11] N. Shaker *et al.*, "The 2010 Mario AI championship: Level generation track," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 4, pp. 332–347, Sep. 2011.
[12] W. M. P. Reis, L. H. S. Lelis, and Y. Gal, "Human computation for procedural content generation in platform games," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 99–106.
[13] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the Mario AI framework," in *Proc. Int. Conf. FDG*, 2014, pp. 1–8.
[14] J. Roberts and K. Chen, "Learning-based procedural content generation," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 1, pp. 88–101, Mar. 2015.
[15] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. Andre, "Player modeling," *Dagstuhl Follow-Ups*, vol. 6, pp. 45–59, 2013.
[16] S. Dahlskog and J. Togelius, "A multi-level level generator," in *Proc. IEEE Conf. Comput. Intell. Games*, 2014, pp. 1–8.
[17] P. Mawhorter and M. Mateas, "Procedural level generation using occupancy regulated extension," in *Proc. IEEE Conf. Comput. Intell. Games*, 2010, pp. 351–358.
[18] T. W. Malone, "What makes things fun to learn? heuristics for designing instructional computer games," in *Proc. 3rd ACM SIGSMALL Symp. 1st SIGPC Symp. Small Syst.*, 1980, pp. 162–169.
[19] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. New York, NY, USA: Harper Collins, 1990.
[20] R. Koster, *A Theory of Fun for Game Design*. Phoenix, AZ, USA: Paraglyph Press, 2005.
[21] P. Shi and K. Chen, "Online level generation in Super Mario Bros via learning constructive primitives," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.
[22] G. N. Yannakakis and J. Hallam, "Towards optimizing entertainment in computer games," *Appl. Artif. Intell.*, vol. 21, pp. 933–971, 2007.
[23] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an RTS game using rtNEAT," in *Proc. IEEE Conf. Comput. Intell. Games*, 2008, pp. 87–94.
[24] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 653–668, Dec. 2005.
[25] P. Spronck, M. Ponsen, I. S. Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Mach. Learn.*, vol. 63, no. 3, pp. 217–248, 2006.
[26] C. H. Tan, K. C. Tan, and A. Tay, "Dynamic game difficulty scaling using adaptive behavior-based AI," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 4, pp. 289–301, Dec. 2011.
[27] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Extending reinforcement learning to provide dynamic game balancing," in *Proc. IJCAI Working Reason. Represent. Learn. Comput. Games*, 2005, pp. 7–12.

[28] S. K. Thompson, *Sampling* 2nd ed. New York, NY, USA: Wiley, 2002.
[29] *NIST/SEMATECH e-Handbook of Statistical Methods*, 2003. [Online]. Available: http://www.itl.nist.gov/div898/handbook/
[30] S. Guha, R. Rastogi, and K. Shim, "CURE: An efficient clustering algorithm for large databases," *ACM SIGMOD Rec.*, vol. 27, no. 2, pp. 73–84, 1998.
[31] J. Han and M. Kamber, *Data Mining.* San Francisco, CA, USA: Morgan Kaufmann, 2001.
[32] A. L. N. Fred and A. K. Jain, "Combining multiple clusterings using evidence accumulation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 6, pp. 835–850, Jun. 2005.
[33] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," Univ. California, Berkeley, CA, USA, Tech. Rep. 666, 2004.
[34] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[35] N. Shaker, M. Nicolau, G. N. Yanakakis, J. Togelius, and M. O'Neill, "Evolving levels for Super Mario Bros using grammatical evolution," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 304–311.
[36] J. Togelius, S. Karakowskiy, and R. Baumgarten "The 2009 Mario AI Competition," in *Proc. IEEE Congr. Evol. Comput.*, 2009, 1–8.
[37] R. Bellman, "A Markovian decision process," *J. Math. Mech.*, vol. 6, pp. 679–684, 1957.
[38] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, pp. 285–294, 1933.
[39] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2013, pp. 30–36.
[40] S. Snodgrass and S. Ontañón, "A hierarchical MdMC approach to 2D video game map generation," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2015, pp. 205–211.
[41] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha, "Launchpad: A rhythm-based level generator for 2-D platformers," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 1, pp. 1–16, Mar. 2011.
[42] N. Sorenson, P. Pasquier, and S. DiPaola, "A generic approach to challenge modeling for the procedural creation of video game levels," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 229–244, Sep. 2011.
[43] S. Snodgrass and S. Ontañón, "Experiments in map generation using markov chains," in *Proc. Int. Conf. FDG*, 2014, pp. 225–232.
[44] A. Summerville and M. Mateas, "Super Mario as a string: Platformer level generation via LSTMs," in *Proc. 1st Conf. DiGRA FDG*, *arXiv preprint arXiv:1603.00930*, 2016.

Peizhi Shi received the B.Sc. degree in computer science from the Guilin University of Electronic Technology, Guilin, China, in 2010, and the M.Sc. degree in software engineering from the University of Science and Technology of China, Hefei, China, in 2013. He is currently working toward the Ph.D. degree in computer science at the University of Manchester, Manchester, U.K.

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

Ke Chen (M'97–SM'00) received the B.Sc. and M.Sc. degrees from Nanjing University, Nanjing, China, in 1984 and 1987, respectively, and the Ph.D. degree from Harbin Institute of Technology, Harbin, China, in 1990, all in computer science.

Since 2003, he has been with The University of Manchester, Manchester, U.K., where he leads the Machine Learning and Perception Lab. He was with The University of Birmingham, Peking University, The Ohio State University, Kyushu Institute of Technology, and Tsinghua University. He was also a Visiting Professor at The Hong Kong Polytechnic University and a Visiting Senior Researcher in Microsoft Research Asia. He has published more thab 200 papers in peer-reviewed journals and conferences. His current research interests include machine learning, machine perception, computational cognitive modeling, and their applications in intelligent system development including learning-based video game development.

Dr. Chen has been an Associate Editor and 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 Co-Chair 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 received several academic awards including the NSFC Distinguished Principal Young Investigator Award in 2001 and the JSPS Research Award in 1993.