Integrated Approach to Personalized Procedural Map Generation Using Evolutionary Algorithms

William L. Raffe, Fabio Zambetta, Xiaodong Li, and Kenneth O. Stanley

***Abstract*—In this paper, we propose the strategy of integrating multiple evolutionary processes for personalized procedural content generation (PCG). In this vein, we provide a concrete solution that personalizes game maps in a top–down action-shooter game to suit an individual player's preferences. The need for personalized PCG is steadily growing as the player market diversifies, making it more difficult to design a game that will accommodate a broad range of preferences and skills. In the solution presented here, the geometry of the map and the density of content within that geometry are represented and generated in distinct evolutionary processes, with the player's preferences being captured and utilized through a combination of interactive evolution and a player model formulated as a recommender system. All these components were implemented into a test bed game and experimented on through an unsupervised public experiment. The solution is examined against a plausible random baseline that is comparable to random map generators that have been implemented by independent game developers. Results indicate that the system as a whole is receiving better ratings, that the geometry and content evolutionary processes are exploring more of the solution space, and that the mean prediction accuracy of the player preference models is equivalent to that of existing recommender system literature. Furthermore, we discuss how each of the individual solutions can be used with other game genres and content types.**

***Index Terms*—Hierarchical optimization, interactive evolutionary computation, neuroevolution of augmenting topologies, personalized game maps, procedural content generation, recommender systems.**

I. Introduction

VIDEO game players come from many backgrounds.[1] The study by Drachen *et al.* [1] of player types in the game *Tomb Raider: Underworld* (Eidos Interactive, 2008) shows that, even within a single game, players' preferences can be varied and there are many ways of achieving the same goal. Given this, it is becoming increasingly difficult to design a game that will accommodate a broad range of player preferences. In many modern games, the only way that a player's preferences can alter the mechanics of the game is through manually set difficulty levels that determine the skill of nonplayable characters [2], the time limit of each round of the game [3], or the number of aids afforded to the player [4]. However, these systems are usually reported to the player as a short list of nominal settings and so do not perform well for those players whose abilities do not match any of the provided settings [5].

Instead of requiring the player explicitly set the difficulty level and other settings, approaches such as dynamic difficulty adjustment [4] and adaptive AI [2] have explored ways of mining player experience data, so that the preferred settings can be learned and adapted implicitly. More recently, experience-driven procedural content generation (EDPCG) [6] has gained academic interest as a means of incorporating learned player preferences into a game by algorithmically creating content that is suitable for that player.

The aim of this work is to provide the player with personalized experiences through procedurally generating the maps of a game. This is conducted in an online EDPCG manner in which the player's preferences are learned and used during the course of the game. Furthermore, we investigate the decomposition of the search-based map generation process into the *geometry* (the noninteractive elements such as boundaries, architecture, and clutter) and the *content* (the interactive elements such as enemy characters and pick-ups), with different representations, generative processes, evolutionary cycles, and means of capturing player preferences for each. The solutions for each of these are discussed in Section III and include the geometry being represented as a tree structure and the content density being calculated by a compositional pattern-producing network (CPPN) [7]. While we borrow from the existing literature by evaluating the geometry through interactive evolutionary computing (IEC) [8], the density of content cannot as easily be visualized to the player. Instead, CPPN candidates are evaluated with a learned per-player preference model that is built around the paradigm of a content-based recommender system (RS) [9]. This approach was motivated by the comparison between the RS and adaptive gameplay research fields, provided by Medler [10]. The implementation evaluated here is a starting point for future investigations into utilizing existing RS techniques in EDPCG applications.

While the generative stages of numerous constructive PCG and SBPCG systems use multiphase approaches, SBPCG solutions are usually conducted in a single-search format. By integrating distinct searches, the dimensionality of the search space and the fitness landscape can be divided, with dedicated representations and fitness evaluation techniques used for each search. The use of multiple optimization processes in an

Manuscript received June 14, 2013; revised November 06, 2013; accepted July 16, 2014. Date of publication July 22, 2014; date of current version June 12, 2015.

W. L. Raffe, F. Zambetta, and X. Li is with the School of Computer Science and IT, RMIT University, Melbourne, Vic 3001, Australia (e-mail: william.raffe@rmit.edu.au; fabio.zambetta@rmit.edu.au; xiaodong.li@rmit.edu.au).

K. O. Stanley is with the Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816 USA (e-mail: kstanley@eecs.ucf.edu).

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.2014.2341665

[1]Entertainment Software Association: http://www.theesa.com/facts/index.asp

SBPCG solution has been briefly explored in the past by Cook and Colton [11] but, to the best of our knowledge, ours is the first solution of its kind in the EDPCG domain.

This paper extends upon previous work [12] that introduced our personalized procedural map generation solution and gave an initial qualitative analysis via the examination of the experiences of three sample players. Here, we further the evaluation of this system with substantially more player data that is the result of an unsupervised public experiment that aimed to test the integrated solutions in a real-world setting. The details of the experiment and metrics used for evaluation are given in Section IV and the results of the experiment a subsequent discussion is provided in Section V.

In summary, the four main contributions of this paper are: 1) we propose the strategy of integrating multiple search cycles in an SBPCG framework, with a focus on decomposing a map into geometry and content density and optimizing them separately; 2) we analyze the applicability of using a CPPN to calculate the quantity and layout of content in a map; 3) we investigate the evaluation of map candidates with an RS based per-player preference model; and 4) we demonstrate how to approach rigorous statistical evaluation of PCG systems in real-world conditions.

Developing approaches to personalized content generation within complete games remains a wide open area of research wherein conventions and standards are still being established. In this context, the comprehensive description and analysis in this paper makes an important contribution to building an initial critical mass of work from which a systematic science can be developed and provides a representative example of statistically evaluating a PCG system from noisy real-world player data. Thus, this work is not only relevant for its individual success, but also as a data point in the larger enterprise of learning to build satisfying games that generate their own content. Furthermore, the individual solutions presented here each afford opportunities for generalization to other contexts, which are discussed in Section VI. The conclusion in Section VII also discusses intended future work into the use of collaborative filtering for player modeling and improving the diversity of maps through novelty search.

II. Background

In this section we discuss background knowledge and related work regarding decomposed search-based map generators and personalized PCG. Additionally, as one of the core principles of this work is the use of an RS as a player model, a brief background of RS in games is provided.

A. Evolving Game Maps

Search-based PCG (SBPCG) has risen in academic popularity in recent years as an alternative to traditional rule-based constructive PCG techniques, especially in the generation of game maps. SBPCG solutions use a generate-and-test approach, evaluating generated maps and using the current best to improve future map candidates. Togelius *et al.* [13] provide an extensive taxonomy and survey of such techniques. From their work, one noticeable finding is that evolutionary algorithms have been a common choice in SBPCG as they offer a means of iteratively improving on previously found good content. Out of the existing solutions that generate game maps, each focuses on different game genres, has different genetic representations, and has different heuristics for evaluating the maps that are generated, highlighting the current creative exploration of the field.

Other than using evolutionary computation, another common trait among many SBPCG solutions for generating maps is that they use a single evolutionary framework. Multiphased generation techniques have been successfully applied to constructive PCG approaches, such as how Tutenel *et al.* [14] use multiple PCG solutions to create complete buildings, how Uriarte and Ontanón [15] use various stages with specific constraints to generate maps in a real-time strategy game, or how Dormans and Bakkes [16] aid developers in first structuring the mission in a map and then procedurally generating the surrounding space. Treanor *et al.* [17] even build playable games by taking a user created graph-based description of the game, forming this in a list of minor game mechanics, and then combining these into a complete set of interacting rules and mechanics. These PCG solutions use a divide-and-conquer approach, decomposing a problem into more manageable subproblems and using unique and appropriate solutions for each.

However, this type of decomposition is not commonly applied to SBPCG solutions. Some solutions, such as that of Togelius *et al.* [18], address multiple aspects of a game map in the genotype and fitness metrics, while others use separate generative processes during the genotype-to-phenotype conversion [19]. However, only a single evolutionary framework is used and therefore the authors search for solutions to multiple aspects of map generation at the same time, using global fitness metrics and breeding operations.

In cases where the generative process can be expressed as a sequence of minimally interacting stages, each stage can potentially contain its own search. Cook and Colton [11], [20] apply this idea of decomposition by optimizing the geometry of maps, the layout of player and nonplayer characters, and the rule-sets of a game in multiple evolutionary strands that proceed in parallel. Our approach is similar in that the problem is decomposed into subproblems with distinct generative and search processes, but differs in that hierarchical optimization is used. That is, one evolutionary cycle is completed first and then the resulting solution is used as an input or a constraint to the next evolutionary cycle.

Finally, we take this opportunity to also explore the application of a CPPN representation [7] and neuroevolution of augmenting topologies (NEAT) [21] to map generation. CPPN-NEAT has previously been used to generate game content, including the trajectory of particle weapons [22], 3D models of star-ship hulls [23], and the shape and color of flower petals [24], but to the best of our knowledge has not been used in the procedural generation of maps, specifically in calculating quantities of content throughout the map.

B. Personalized Game Content

While some SBPCG solutions establish fitness evaluation metrics based upon hypothesized criterion of high quality content, others capture a player's preferences, skill, mood, or other affective states in order to evaluate the appropriateness of the content relative to that particular player, thus personalizing the content to that player. These techniques, recently termed experience-driven PCG (EDPCG) [6], commonly use SBPCG as a base approach to iteratively improving content to cause a desired response in the user such as enjoyment or frustration. Examples of these are the use of multiuser IEC to give players control over the types of race tracks they play [8] or more subtly monitoring which weapons the player interacts with to determine parents when evolving particle weapons [22].

An alternative to allowing the player's input to directly affect the fitness evaluation is instead to model the player's affective state. Player modeling techniques capture information about the player through subjective feedback, in-game activity, or physiological responses [6] and can either be used to report statistics about the players in order to better formulate a fitness evaluation metric or can be used directly to predict whether potential content will be appropriate for the player [25].

As an example, Shaker *et al.* [26] train neural networks offline to model emotional states of players in a platform game. These models are then used during gameplay to optimize controllable parameters of future maps given the playing style of the player in previous maps. In this approach, a single model is created for each emotional state for all players. In contrast, Togelius *et al.* [27] model individual players by constructing an agent to mimic a player's driving style and then using this agent to evaluate tracks. The taxonomy provided by Smith *et al.* [25] highlights that the use of learned per-player models to generate game content (especially maps) is rare. Under their taxonomy, our approach uses an individual induced generative reaction model, for which only Polymorph [28] is listed as a comparable solution, which combines a universal model of map segment difficulty with individual models of player skill.

We use a combination of methods to capture and use the preferences of individual players; borrowing from the literature by using IEC to give players direct control over the evolution of the map geometry while using a trained per-player preference modeling technique to evaluate the distribution of content in a map. We believe that per-player models better capture the individuality of each player by preventing the preferences of other players from skewing the learning process.

C. *Recommender Systems in Games*

In this paper, the player modeling process is framed as that of an RS. The goals of RS [29] closely align with those of personalized PCG; both involve finding appropriate items (content) that maximize a utility for a user (player) based upon items that the user has previously interacted with or rated. In player preference modeling, that utility is the player's enjoyment of the content. Medler [10] has previously made a similar qualitative comparison between RS and adaptive gameplay. While Medler identifies potential future work towards applying RS to adaptive games, no example solution is implemented or experimented on.

Despite the similarities between RS and personalized PCG, the relationship between the two fields is typically implicit. Of those that explicitly state the use of an RS, Berkovsky *et al.* [3] use collaborative filtering to set the time limit of a game map to the average completion time of similar players. Zook and Riedl [30] use a tensor factorization technique that is popular in collaborative filtering to model a player's change in skill over time and Yu and Riedl [31] use a prefix based collaborative filter to select a sequence of plot points in a game's narrative. Our approach differs in that we use a content-based RS as a per-player preference model, use explicit player feedback to determine preferences, and use the RS player model during fitness evaluation in an evolutionary cycle, thus using SBPCG as a means of reducing the number of items evaluated by the RS.

III. Personalized Procedural Map Generation

In this section we first describe the game that is used as a test-bed and then detail the geometry representation and its evaluation through IEC, the CPPN representation of the content density of a map, and the novel RS player model that is used to determine the fitness of CPPN-NEAT candidates. This solution was first described in [12].

A. *PCG: Angry Bots*

The game that is utilized in our experiments is titled *PCG: Angry Bots*. It is built upon the *Angry Bots* technical demonstration provided free with the Unity game engine.[2] The game is a single-player action-shooter played from a top–down perspective and has mechanics similar to those of the multiplayer game *Alien Swarm* (Valve Corporation, 2010). A screenshot of the game being played is shown in Fig. 1.

The goal of the game is simply to get from one end of a map to the other. The maps are not typically complicated enough to be classified as mazes and thus the challenge that the player will experience is primarily a result of computer controlled enemies they encounter.

Algorithm 1 outlines the gameplay and map generation system. Players first sign into their accounts, which store all their player data. If this is the player's first map, then the map is randomized, otherwise it is optimized using their player data. If a map is randomized, both the geometry and content density is randomly generated and the player experiences the map immediately.

If the map is to be optimized, then the last geometry that was played is evolved, producing seven offspring, which are presented to the player through IEC. Once the player has selected a geometry, CPPN-NEAT produces CPPN candidates, which take coordinates from the selected geometry to calculate a content density. The geometry and content are then combined to produce map candidates from which features are extracted. These features are classified by an RS-based player model and the probability of the map being liked is assigned to the CPPN candidate. This is repeated ten thousand times or until a perfect fitness is found, at which time the elite CPPN is used to calculate the optimized content density.

Once a map has been generated (through random generation or optimization), it is rendered and experienced by the player. After play, the player is asked to rate the map on a six point scale. This rating and features of the map that was played are then used to update the RS player model. The player then chooses whether to randomize or optimize the next map and the process repeats.

[2]Unity Game Engine: http://unity3d.com

Fig. 1. Screenshot of gameplay in *PCG: Angry Bots*.

Algorithm 1 An overview of the main game cycle.

```
1: player := PlayerLogin()
2: if (player is newPlayer)
3:   generationMethod := randomize
4:   CPPN := new CPPNObject(randomPopulationFlag)
5:   RS := new RSObject()
6: else
7:   generationMethod := optimize
8:   CPPN := new CPPNObject(LoadLastPopulation())
9:   RS := new RSObject(LoadPlayerData())
10: while (playing)
11:   if (generationMethod is randomize)
12:     geometry := RandomGeometry()
13:     content := RandomContent(geometry)
14:   else if (generationMethod is optimize)
15:     geometryCandidates := Evolve(geometry)
16:     Display(geometryCandidates)
17:     geometry := GetPlayerInput(geometrySelection)
18:     while (evaluations < 10000)
19:       cppnCandidate := CPPN.NeatEvolution()
20:       content := CPPN.Process(geometry, cppnCandidate)
21:       mapCandidate := Combine(geometry, content)
22:       features := ExtractFeatures(mapCandidate)
23:       fitness := RS.ProbabilityOfLike(features)
24:       CPPN.UpdatePopulation(cppnCandidate, fitness)
25:       if (fitness is 1.0)
26:         break
27:       evaluations++
28:     end while
29:     content := CPPN.Process(geometry, CPPN.GetElite())
30:   end else if
31:   map := Combine(geometry, content)
32:   RenderMap(map)
33:   PlayMap()
34:   rating := GetPlayerInput(ratingSelection)
35:   features := ExtractFeatures(map)
36:   RS.UpdatePlayerModel(features,rating)
37:   generationMethod := GetPlayerInput(methodSelection)
38: end while
```

B. Geometry and Content Density Optimization

There are two evolutionary cycles used to search for a map in *PCG: Angry Bots*; one for geometry and one for content density. These two components are minimally linked through hierarchical optimization by first generating the geometry and then using it to calculate the content densities. In the context *PCG: Angry Bots*, the geometry determines the length of the game and provides opportunities for limited exploration, while the location and quantity of content (enemies and pick-ups) within the geometry is the primary determinant of the difficulty of the map.

The geometry of a map in *PCG: Angry Bots* is generated by connecting premade room and corridor templates together in a fixed n-ary tree structure. In this structure, an example of which is shown in Fig. 2, each edge is a corridor and each node is a room. In this implementation, $n = 3$ simply because there are no room templates with more than four doors. There is exactly one node in which the player starts (the start room) and one node that they must reach to successfully complete the map (the exit room). The path between these rooms must have a minimum of two rooms and there is no maximum. There is a random chance

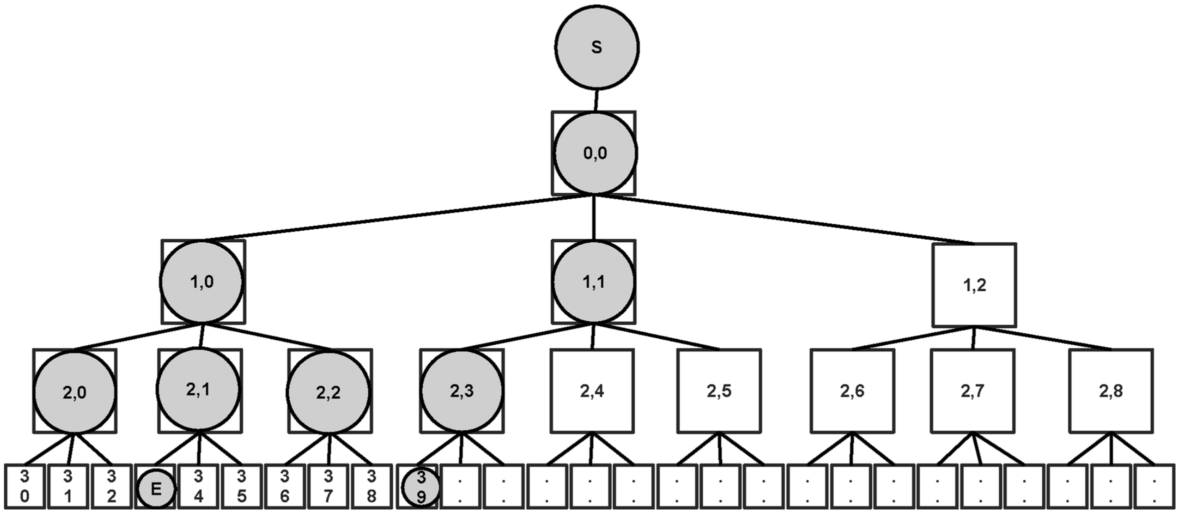


Fig. 2. Example of a map's geometry represented as a fixed n-ary tree. *S* is the starting room and *E* is the exit room. Each square is a potential node location. Each circle is an instantiated node (room).

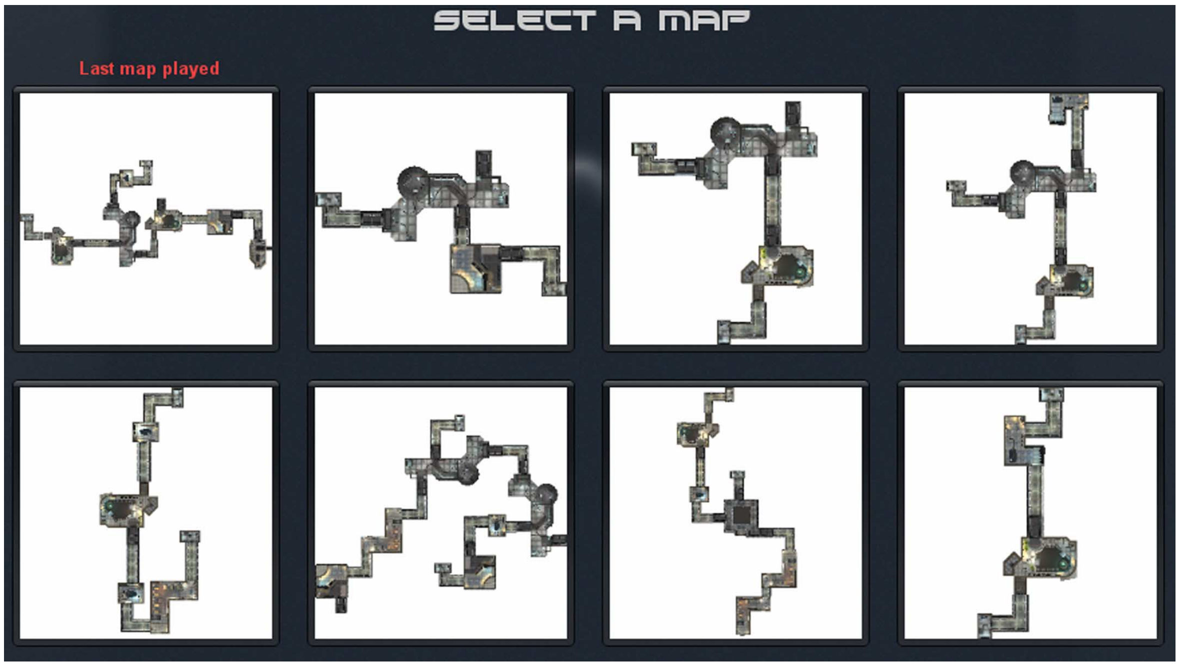


Fig. 3. Geometry selection menu in *PCG: Angry Bots*. The menu facilitates the use of IEC.

that each node will have a branch, with less of a chance further along branches to prevent bloat.

The tree is referred to as "fixed" because when a map is created during the random generation of the first evolutionary population, each node in the tree is labeled with $[depth, sibling]$ coordinates. From there, a node's label is only altered minimally by all future mutation operations. For example, in Fig. 2, if node $(2, 0)$ is removed, nodes $(2, 1)$ and $(2, 2)$ are not relabeled. This is done to limit the impact of geometry mutations on the content density search process. Mutation is conducted by adding new nodes and creating branches from them, removing nodes and all children that don't lead to the exit node, or the permutation of a node's reference to a room template. Each offspring map is validated to make sure that no rooms or corridors will overlap once the map is rendered for play. If the validation fails, then the offspring is discarded and mutation is attempted again. It takes, on average, 30 ms on a 3.07 GHz quad-core processor to mutate and validate a single offspring.

Fitness evaluation of the geometry is conducted through IEC. For every map that a player experiences, they are shown a menu screen such as the one depicted in Fig. 3. This menu shows the geometry of the previous map that was played as well as the first seven valid offspring of that map. The player is then required to choose the geometry of the map that they wish to play next, which then becomes the parent for the next generation of geometries.

During this process, the player has an opportunity to memorize the geometry. While this limits exploration potential for players, it was decided that this was a positive trait. As the geometries are not mazes and the only rewards for exploration

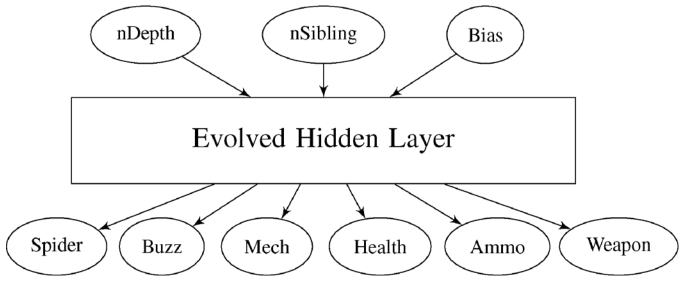


Fig. 4. Input and output structure of the CPPN used for calculating the quantity of each content type in each room of a map. From top to bottom are the input layer, hidden layer, and the output layer. Both inputs are normalized to the range of $[0, 1]$ and outputs are in the ranges of $[-1.0, 1.0]$, which are discretized to one of the four predetermined content quantity settings.

are additional pick-ups, there is already limited exploration potential. By showing the geometry through IEC, experienced players can memorize it and, at junctions of multiple paths, decide whether it is a good idea to pursue the potential of extra pick-ups (of which the player is unsure of) or continue straight to the exit of the map.

Once the player has selected a geometry through IEC, the densities of content across the map are optimized. In *PCG: Angry Bots*, there are six types of content: three enemies (*Spider Bots*, *Buzz Bots*, and *Mechs*) and three pick-ups (Ammo, Health, and New Weapons).

The manually constructed room templates utilized for the geometry generation also define the maximum quantity and location of each content type and so the quantity of each content type must be calculated for each room. This is done to ensure that there is a logical layout of content in each room and thus no validation process is needed. Each content type in each room can have a discretized setting of *None*, *Low*, *Medium*, or *High*. Choosing a setting for each content type in each room is done by using the coordinates of each node in the fixed n-ary tree of the chosen geometry as input to a CPPN [7].

Fig. 4 shows the input and output structure of the CPPN network. The input to the network utilizes the depth and sibling coordinates of each node in the user selected geometry, as well as a bias value of 1.0. Both inputs are normalized to the range of $[0.0, 1.0]$ by dividing the depth by the maximum depth of the current geometry tree and the sibling coordinate by the maximum number of siblings at the corresponding depth. There are six outputs, one for each of the content types, that each provides a value between $[-1.0, 1.0]$, which is in turn discretized into one of the earlier mentioned settings. If there is no connection to an output node, then the output value will default to the 'Medium' setting.

The hidden layer of the CPPN is optimized through NEAT [21]. The *SharpNEAT* [32] implementation of CPPN-NEAT was used here. For each player selected geometry, 10 000 fitness evaluations are conducted to optimize the content density. There were 50 candidates per generation, with no repeated evaluations, and five species. This gives CPPN-NEAT ample time to find an acceptable solution. Sine, Gaussian, bipolar sigmoid, and linear activation functions were used with equal probability of being chosen at each node. An absolute complexity regulation strategy was used with a complexity threshold of 50. This allows for the CPPN candidates to expand and shrink quickly and encourages exploration while also restricting the maximum CPPN size. Finally, all CPPNs were acyclic networks as we have no need of the internal memory properties of a cyclic network. On average, an entire round of CPPN-NEAT evolution, including calculating the content density of, extracting features from, and classifying 10 000 map candidates, takes roughly 6 s.

IEC was judged to be a sufficient method of evaluating geometry candidates because the geometry can be clearly previewed to the player and a player can quickly decide their preferences regarding the shape and size of the map. While this approach could also be potentially used for evaluating the content density, this will require more effort from the player as they will need to study each map candidate carefully, interpreting a legend (such as those used later in Fig. 5) and making a roughly calculated decision. However, we wanted the player to convey their preferences easily to reduce interruptions to gameplay. Thus, each CPPN candidate is instead evaluated by first calculating the content density for every room of the geometry, then extracting features of the map and evaluating it through a per-player preference model, which is described next.

C. Player Modeling With Recommender Systems

We frame the player modeling process as that of a traditional *content-based RS [29]*[29]. More specifically, a model-based approach is used in which a player rates each map that they play and that rating, along with extracted map features, are used to train a classifier. That classifier is then used to predict the likelihood that the player will like a content density that results from a CPPN candidate. Note that a content-based RS calculates similarities between the items (maps) that a user (player) has already rated and those that they have not yet experienced. Every user is considered in isolation and their recommendations are not influenced by the actions of other users. This is opposed to a collaborative filter [29], which compares similarities between users. Collaborative filters are difficult to use in this context because they require items to be corated and given the enormous size of the solution space of the earlier described generative systems, it is rare that two players will experience the same map.

A Naive Bayes (NB) classifier was used due to its reported success in other RS applications [33] and initial testing on sample data generated by our research team that indicated that NB performed well for the training set sizes that players are likely to produce in *PCG: Angry Bots*. The following 18 features are extracted from each map candidate and used to predict whether the player will enjoy the map:

- Enumerated Sums – The settings {None, Low, Medium, High} are enumerated as {0, 1, 2, 3} and a sum of each of the 6 content types across the entire map is calculated.
- Room Composition Counts – The six content types are condensed to two categories, Enemies (E) and Pick-ups (P), and the four settings are condensed into Low and High. This creates 4 possible room types: LowE-LowP, LowE-HighP, HighE-LowP, and HighE-HighE. The quantity of each room composition is counted throughout the map.
- Room Transition Counts – As with the Room Compositions, Content types and settings are reduced. There are 4 enemy transition types: LowE-to-LowE, LowE-to-HighE,

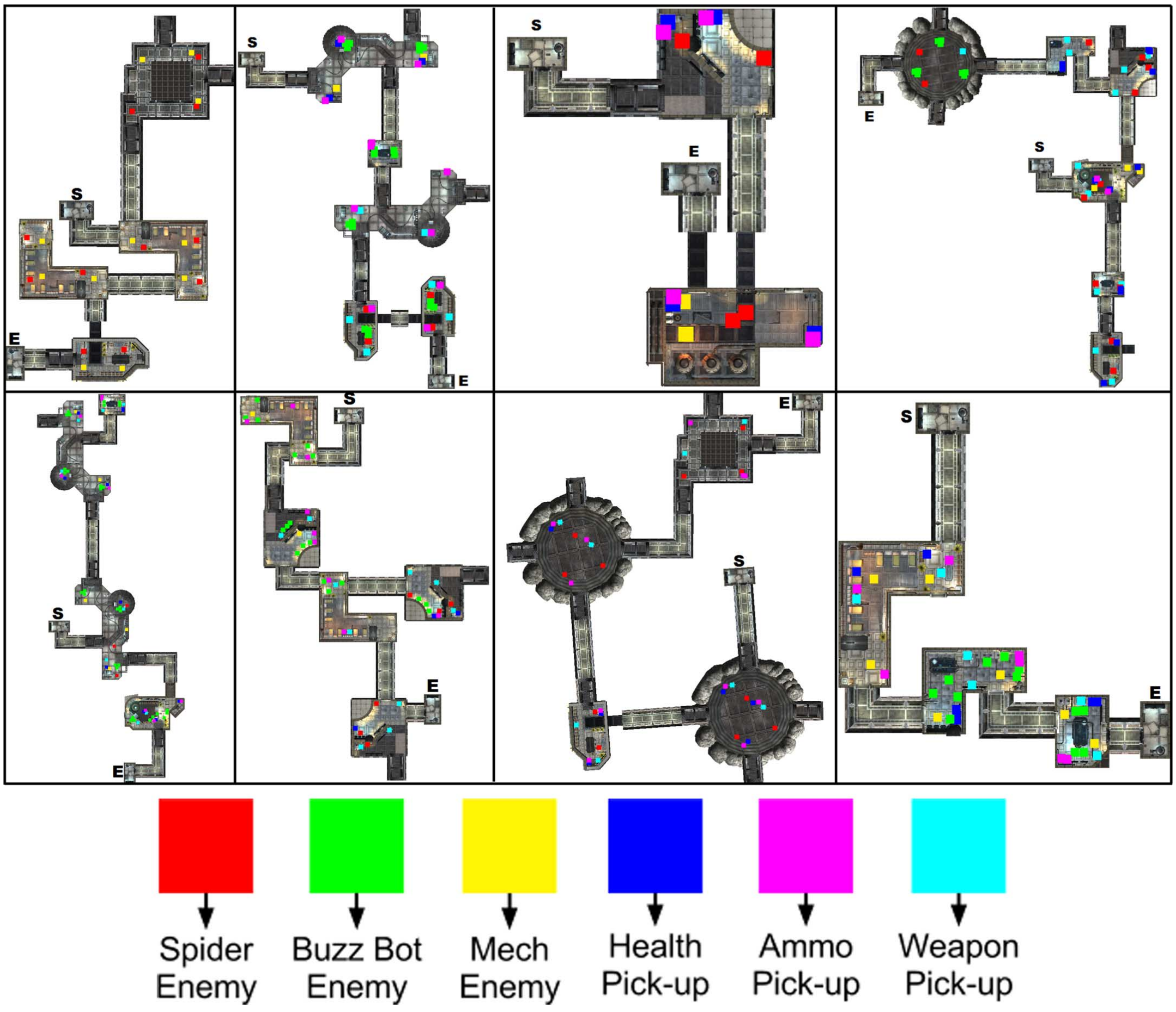


Fig. 5. Examples of well rated maps. Each colored square represents a single piece of content, corresponding to the provided legend.

HighE-to-LowE, and HighE-to-HighE. There are also four similar transition types for pick-ups. Each edge of the geometry tree will belong to one enemy transition type and one pick-up transition type.

The above feature values are all normalized by dividing by the total number of rooms in the map. The NB independent feature assumption doesn't hold true for some of these features, but this doesn't appear to negatively impact the performance of the classifier. This behavior has also been witnessed by Domingos and Pazzani [34].

These features were determined through expert knowledge and without the rigorous feature evaluation conducted in previous procedural map generation studies [26]. This is because we wanted to evaluate the system's performance in suboptimal circumstances, which would allow the system to be quickly adapted for other games given expert knowledge. If the system can be shown to perform well under these conditions, then it is likely that the performance would increase with a more thorough feature selection process.

A trade-off between the granularity of a multiclass setup and the performance of a binary class setup was achieved by converting the player's nominal rating into a weighted binary class value. The player ratings {Very Bad, Bad, Poor, Fair, Good, Very Good} are evenly divided into two classes {Dislike, Like} and given weights of {2, 1, 0.5, 0.5, 1, 2} respectively. Therefore, a rating of 'Very Bad' or 'Very Good' will have a strong influence over the classifier while ratings of "Poor" and "Fair" will only make weak contributions to the classes.

This type of rating system was chosen so that the system would be coherent with much of the literature in the RS field. However, Yannakakis and Hallam [35] provide a comparative analysis of post reporting techniques for player modeling and found that a pairwise preference approach introduces less error. The results presented in [12] show that there is indeed an issue associated with the order that maps are experienced, especially as a player's preferences and skill change rapidly in the first few maps. However, it was found that while the classifier may take a sudden penalty for contradicting training data, this is typically quickly negated by further training data that aligns with the new preferences.

D. Random Generation

In *PCG: Angry Bots*, after each map is completed and the player has rated it, they have the option of generating the next map through the use of the mechanisms described above or by completely randomly generating the map. Every map played during the experiment fits into one of these categories and is referred to as either an *optimized* map or a *randomized* map. The option to randomly generate a map was provided as a means of introducing variety if the player felt that they were being given similar maps over and over. This was in anticipation of overspecialization in the evolutionary optimization or RS player model; an issue that eventuated during the experiment and is discussed later.

If the player chooses to randomly generate a map, then a tree with a random number of nodes between the start and exit rooms is produced and each node is given a chance to have branches. The tree is then validated in the same way that a mutated tree is, regenerating the entire tree if the validation fails. Once the tree is constructed, the content setting of each content type of each node in the tree is randomly assigned, with each setting having an equally likely probability of being chosen.

Player ratings are also gathered for randomized maps in order to further train the RS player model. These randomized maps and their ratings act as a baseline comparison in the results below. The random generation algorithm used here is a plausible PCG solution and, as results show later, is quite capable of providing enjoyable experiences to players. By using this approach we are able to compare our implementation to game genres such as rogue-likes that successfully utilize similar constrained random map generators that ensure playable and minimally enjoyable maps, but otherwise do not draw upon computational intelligence to improve the quality of the maps.

IV. Experiment Setup and Evaluation Metrics

The *PCG: Angry Bots* game was made available to play online[3] and general invitations to participate were distributed through various social networking channels. This experiment was anonymous and unsupervised, and therefore participants could play the game for as little or as long as they wanted. Information about the maps that were played, player ratings, and player activity was recorded. As well as playing the game, participants were given the option to complete a survey[4] on their experience.

In this section we describe the metrics used to evaluate the collected data, the results of which are reported in Section V. This user study was primarily conducted to test the fixed n-ary tree and CPPN representations in a real game situation, to analyze the experiences that players had and their preferences, and to evaluate the accuracy of an RS per-player preference model. This section describes the nonparametric statistical tests, the content balance calculations, and the approach to mean classification accuracy used to evaluate these properties of the system.

A. Statistical Analysis Tests

For the following discussion, the set of all optimized maps and the set of all randomized maps are referred to as S_{Opt} and S_{Rand} respectively. Statistical analysis of these two sets is provided through the following nonparametric tests. Unless otherwise stated, these tests are conducted on the ratings provided by the players, rather than the binary classes used by the NB classifier.

We first use the chi-squared (χ^2) test of independence to test whether the rating of a map is dependent on the set (S_{Opt} or S_{Rand}) it belongs to. The null hypothesis (H_0) states that the two sets are independent of the ratings received. To reinforce this, we then use the Mann-Whitney U test [36] to test whether or not maps from S_{Opt} receive statistically better ratings than those from S_{Rand}. Here, H_0 states that there is no difference between the two sets and the alternative hypothesis (H_1) states that S_{Opt} performs better than S_{Rand}.

B. Content Balance

Visualizing content trends among all players can be difficult when considering all six content types. Thus, for much of the evaluation of the content densities in this paper, we summarize the six content types into their broader categories: enemies and pick-ups. The assumption made here is that increasing the number of enemies of any kind will increase the difficulty of a room or map equally and increasing any pick-up quantity will reduce difficulty equally. While this assumption isn't entirely accurate (e.g., ammo is more valuable than health in this game), it does give a clear impression of the types of content distributions being experienced by players.

With this in mind, the change in difficulty throughout a map can be summarized by first enumerating each of the four content settings {None, Low, Medium, High} as {0, 1, 2, 3} and calculating the content balance of a room as the sum of all pick-ups minus the sum of all enemies. Therefore, a reduction in the content balance value from one room to the next indicates that the difficulty is increasing and vice versa. If the content balance is calculated for every room on the direct path between the start and end rooms, we can categorize the linear experience as increasing in difficulty, decreasing in difficulty, fluctuating between the two, or holding a constant difficulty across all rooms.

C. Mean Prediction Accuracy

The performance of the RS player model is deduced by comparing the predicted rating of a map with the rating that the player provided (on the binary class scale of {Dislike, Like}) and updating a confusion matrix for that player. Accuracy [37] is then calculated for each player by dividing the number of correct predictions by the total number of predictions at a given map index. Once accuracy is calculated for every player at every map index that they experienced, the mean accuracy of all players at each map index is calculated. A map index is simply the order that a map was played in. For example, map index 3 is the third map played by a player.

A slight variation on the above accuracy measurement is k-fold cross validation. Here, the ratings so far from a player at a given map index are divided into k partitions (folds), using $k - 1$ folds as training data and the last fold as the test set to calculate accuracy. This is repeated multiple times, allowing each fold to be the test set once, and then taking the mean accuracy of all tests. Once this is done for each player at a given map index, the mean value of all players is calculated. More specifically, we use leave-one-out cross validation (LOOCV) where k equals the map index, thus using a single map and

[3]PCG: Angry Bots download: http://goanna.cs.rmit.edu.au/ wraffe/ExperimentHome.html

[4]Survey questions: http://goanna.cs.rmit.edu.au/ wraffe/QualtricsSurvey.pdf

rating pair as the test set in each fold. Due to the small training set sizes, LOOCV is used to allow as much data as possible to be used in training each fold. At a given map index, accuracy describes the real performance of a classifier so far, while LOOCV instead indicates a classifier's ability to generalize to unseen test data given the quantity of training data currently available.

As CPPN-NEAT is given ample time to search for a content density candidate that the NB classifier will predict to be liked, negative predictions are rare. Thus, precision shows similar results to accuracy and recall typically remains perfect after the first true-positive that a player experiences. Therefore, these metrics and subsequent measures, such as F1 and area under ROC, are not reported.

D. Comparative Classifiers

In the results below we show the performance of the NB classifier that was used in the game along with a few additional classifiers that have been utilized in previous content-based RS research [9], [33]. The first is a multilayer perception with one hidden layer containing 12 hidden nodes, trained through backpropagation. The second is the IB1 nearest neighbor algorithm [38], the third is a J48 decision tree [39], and the fourth is a random forest classifier [40]. Finally, two runs of a random classifier are provided to show a baseline comparison. This classifier randomly predicts a class for each map of each player. If this classifier were to be used as the RS in the actual game, it would cause all optimized maps to also be randomly generated.

The user experiment was only conducted with NB and so the maps in S_{Opt} are a result of its use. The other classifiers are evaluated offline. This may cause some undesirable effects in the learning capabilities of the other classifiers. However, these plots give a general indication of the performance of the player model if one of the other classifiers had been used.

V. Results

The results of the *PCG: Angry Bots* experiment are provided in this section by using the metrics described above. In [12] we provided an initial high level qualitative examination of the types of maps that were being generated and the experiences that individual players were having. This was done by examining the playing time of three sample players. In contrast, the results presented here show a broader, more aggregate view of the experiences that a larger sample of players had by utilizing the full set of player data that was collected during the public experiment.

A. General Results

Fig. 5 shows just a few of the maps that were rated well by players. Due to the large solution space of both the geometry and the content quantities, there is a wide variety in maps that were played and no exact map was played by more than one player. As it is hard to establish correlation between such varied maps on a visual basis, readers are referred to [12] for examples of the kind of maps that were evolved for specific players.

Table I shows general results of the experiment. In total, 202 participants played at least one map or more. However, many users only played a handful of maps with the median number of maps played being 4. Meanwhile, the number of users that played ten maps or more was 20. A total of 893 maps were played, of which 570 were optimized and 323 were randomized. Users successfully completed 502 of those maps by reaching the exit, while the rest were skipped over. Players completed 67.89% of optimized maps and only 35.60% of randomized maps, a difference that is significant at $p = 0.005$ and with a medium Cohen effect size [41] of 0.31, determined from a Mann-Whitney U test where H_0 states that the number of completed optimized maps is less than or equal to the number of completed randomized maps. This suggests that optimized maps were more feasible for players' skills and therefore more likely to be completed.

TABLE I
General Results From the User Study. S_{Opt} is the Set of All Optimized Maps (Using IEC, CPPN-NEAT, and RS Player Models). S_{Rand} is the Set of All Randomly Generated Maps

Players (>= 1 Map)	202	Players (>= 10 Maps)	20
Maps Played	893	Maps Completed	502
Median Maps Played	4	Mean Maps Played	5.7
Opt Maps Played	570	Rand Maps Played	323
Opt Maps Completed	387	Rand Maps Completed	115
Mean S_{Opt} Rating	3.87	Mean S_{Rand} Rating	3.73
Median S_{Opt} Rating	4 (Fair)	Median S_{Rand} Rating	4 (Fair)

While the median of S_{Opt} and S_{Rand} are the same and the mean ratings appear to be similar, there is in fact a slight statistical difference between the two. Unfortunately, there is no clear aggregate view of ratings improving in S_{Opt} as the map index increases. This is due to fluctuating ratings caused by changing player preferences and skill, as well as the overspecialization of the RS player models and CPPN-NEAT evolution leading to repetitive experiences or evolution pushing optimization too far in one direction. This means that at any map index, some players are having positive experiences and other are having negative ones, leading to a constant median rating of either ‘Poor’ or ‘Fair’ at every map index. These fluctuating ratings were observed in the sample player analysis in [12].

Instead, Fig. 6 shows the percentage of maps that have a certain rating in either S_{Opt} or S_{Rand}. Both sets approximate a normal distribution, however, the ratings in S_{Opt} show a slight positive bias, with the ‘Fair’ and ‘Good’ ratings being selected most often, while the ratings in S_{Rand} are slightly more skewed to the negative side, with “Fair” and “Poor” being the most used options. More “Good” and “Very Good” ratings were given to S_{Opt}, as well as slightly more ‘Bad’ and ‘Very Bad’ ratings, but less Poor' and ‘Fair’. This suggests that the personalized map generation is generally performing better than a random baseline, but that it also polarized ratings. That is, typically the system is performing well, but when it is underperforming, the maps are worse than the random baseline.

In the χ^2 test, H_0 (the two sets are independent of the ratings they received) was rejected with $p = 0.001$, which suggests that the two sets are not related and that the ratings are dependent on whether the map was optimized or not. As one of our sample sets is a random baseline, this tells us that our algorithm is not simply generating maps at random or that the ratings they are receiving are not a matter a chance. For the Mann–Whitney U test, H_0 (S_{Opt} is worse or equal to S_{Rand}) was rejected with

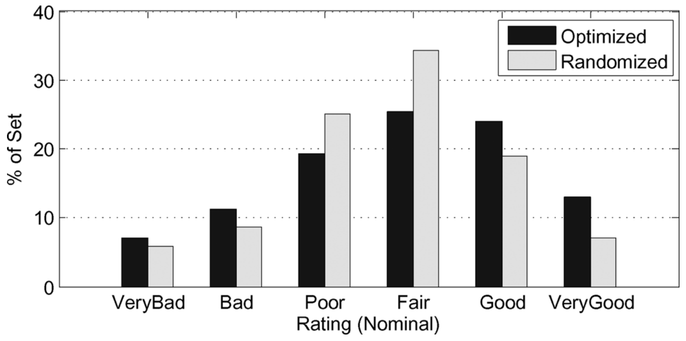


Fig. 6. Percentage that each rating makes up of the optimized (set S_{Opt}) and the randomized (set S_{Rand}) maps.

TABLE II
SURVEY QUESTION ASKING RESPONDENTS TO RATE THEIR LAST FEW MAPS IN COMPARISON TO THEIR FIRST FEW

	Number of Respondents	
Answer	All Responses	Played >= 7 Maps
Much Worse	0	0
Worse	4	0
About the Same	10	1
Better	22	9
Much Better	6	6
Total	42	16

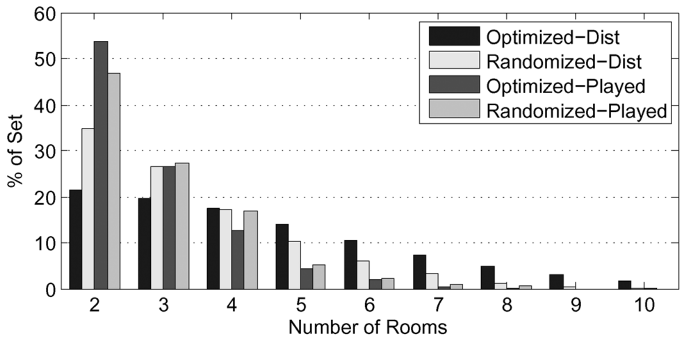
$p = 0.033$, indicating that the optimized maps are receiving better ratings than the randomly generated ones.

Unfortunately, in both of these tests the effect size is small. For the χ^2 test, the ϕ coefficient [42] is only 0.17, a small effect size when judged by Cohen's effect size bands [41]. Meanwhile, using the Mann–Whitney effect size estimate suggested by Grissom and Kim [43], if an optimized map and a randomized map were chosen from their sets at random, there is a 0.54 probability of the optimized map having a better rating; only slightly better than an equal chance.

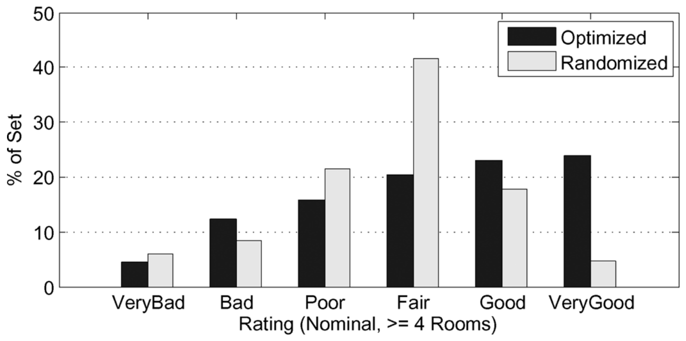
Finally, a total of 42 players responded to the optional post-play survey. It should be noted that because the survey was optional, survey respondents were likely to be more invested in the game or experiment than other players, which may affect the survey results. Indeed, the average respondent played three more maps than the average nonrespondent. However, there were no significant differences between the rating distributions, geometry sizes, and content of maps played by the two groups.

In the survey, participants were asked to rate their experience towards the end of their play time as compared to their experience at the beginning. The first column of Table II shows the possible answers to that question. From this group, 68% of respondents said that their experience got either better or much better. The majority of the remainder said that their experience didn't change much and only four respondents said it got worse. No respondents said that their experience got much worse.

The positive bias of these responses matches that of the nominal rating histogram for optimized maps in Fig. 6 and the responses themselves are statistically significant. The one-sample Wilcoxon test [44] was used to test against an expected median of 3 (‘About the Same’), with H_0 stating that the median of the responses is less than or equal to this hypothesized median. Here, $p < 0.001$, indicating that the responses are significantly



(a)



(b)

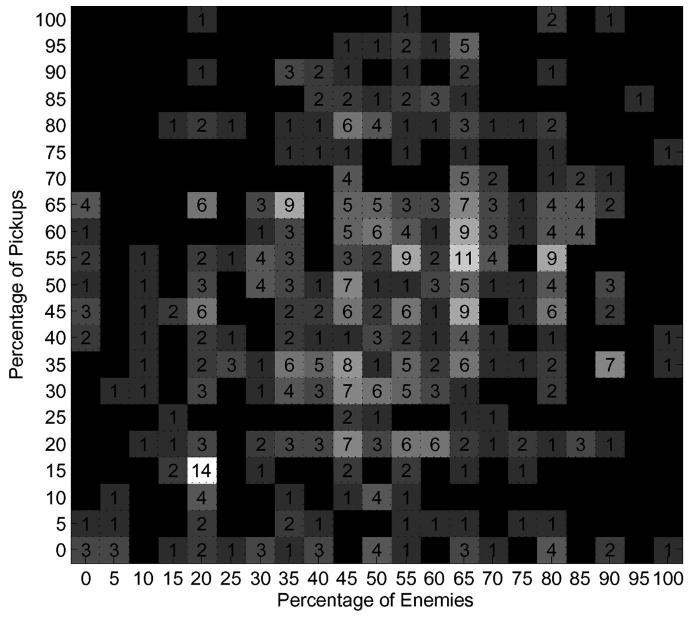
Fig. 7. (a) *-Dist: The percentage of maps with the specified number of rooms given 100 000 valid mutated and randomly generated geometries. *-Played: The percentage of maps with the specified number of rooms that were played. (b) The rating distribution of maps that had 4 or more rooms.

better than ‘About the Same’. The Cohen-comparable effect size value here is 0.65, indicating a strong result.

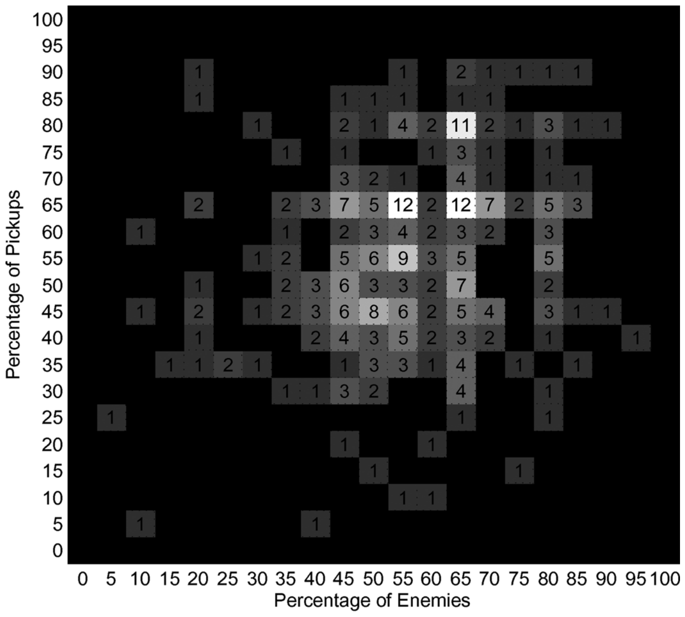
It is also worth noting that in terms of reported enjoyment versus the number of maps played, there is a Spearman correlation of 0.52 at significance $p < 0.001$. Many of the respondents that answered that their experience got worse or stayed the same did not play many maps. The third column of Table II includes only respondents who played seven or more maps. All the respondents that indicated that the game got worse have been eliminated and all, but one of the “About the Same” respondents have been removed. Alternatively, none of those that stated that their experience got much better have been removed.

B. Geometry

Fig. 7(a) shows the distribution of the number of rooms in maps created from mutation (“Optimized”) and random generation (“Randomized”), taken from 100 000 valid maps generated through each technique. Small maps are favored during both mutation and random generation because validation fails more often as maps get bigger. However, mutated maps are more likely to be successfully validated. For random generation, validation succeeded 37.07% of the time for maps with only two rooms, 7.06% of the time for maps with 5 rooms, and only 1.8% of the time for maps with only 10 rooms. The success rate was higher for mutation with 83.28%, 43.42%, 6.59%. Mutation is also slightly more inclined towards growth due to the branching process of the addition operator and so when combined with the validation success rate produces a more even distribution in small map sizes than random generation. If more large maps were desired, the mutation probability could be adjusted to favor



(a)



(b)

Fig. 8. Heat maps of all (a) optimized and (b) randomized maps that were played, plotted in the feature space of percent of enemies against percent of pick-ups. All black areas indicate that no maps were played with the corresponding enemy and pick-up percentages. (a) Optimized maps. (b) Randomized maps.

the addition operator even more or the branching rate could be forced to higher ranges.

Fig. 7(a) also shows the distribution of geometry sizes that were actually played during the experiment. Despite being given opportunities for larger maps in most IEC cycles, players chose shorter maps more often, with a strong emphasis on two room geometries. Oddly though, this result conflicts with the responses to a survey question that asked how large a respondent's ideal map would be. Only 4.76% of respondents said their ideal map would be small (2 to 4 rooms), while the remainder of players' ideals were close to evenly split between medium (5 to 7 rooms) and large (more than 7 rooms) maps. Fig. 7(b) supports this survey results by showing the rating distribution for maps with four or more rooms (113 optimized maps and 84 randomized maps). Larger optimized maps were more likely to be rated well rather than badly and received a higher proportion of "Good" and "Very Good" ratings than randomized maps. For maps with three rooms or less, the rating distribution closely resembles that of the histogram earlier presented Fig. 6.

C. Content Density

Here we show a few qualitative results that pertain to the use of a CPPN to calculate the content settings of a map. Firstly, it's worth noting there is no statistical correlation between the complexity of the geometry (number of rooms) and the complexity of the CPPN used to calculate the content settings of a map. There is also no correlation between the player's rating and the complexity of the CPPN. This implies that the complexification process of CPPN-NEAT is not being utilized to its fullest potential, which is likely a result of the discrete input and output of the CPPN representation.

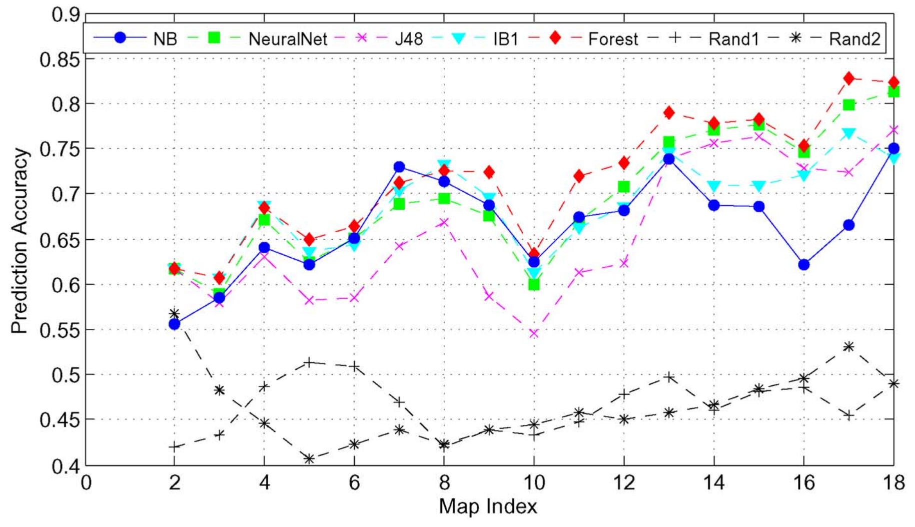
Fig. 8 shows heat maps for all optimized maps and for all randomized maps in the feature space of percent of enemies versus percent of pick-ups, grouped into 5% bins. These are the percentages of the number of enemies or pick-ups in a map out of the maximum allowable number of enemies or pick-ups in that map.

TABLE III
DIFFICULTY TREND OF A MAP CROSS THE RATINGS RECEIVED. VALUES ARE SHOWN AS A PERCENTAGE OF ALL MAPS IN THE RESPECTIVE DIFFICULTY TREND. ALSO SHOWN ARE THE RATINGS FOR MAPS WITH ONLY TWO ROOMS AND THE PERCENTAGE OF MAPS IN EACH TREND THAT MAKE UP S_{Opt}

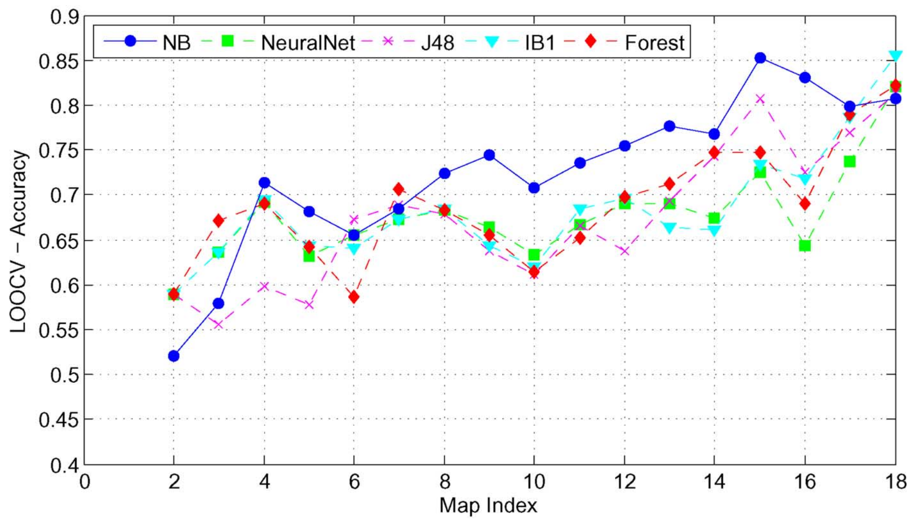
	Increase	Decrease	Fluctuate	Constant	2 Rooms
Very Bad	5.13%	7.02%	5.08%	11.43%	7.33%
Bad	6.41%	22.81%	8.48%	14.29%	10.56%
Poor	23.08%	19.30%	8.48%	14.29%	20.82%
Fair	29.49%	24.56%	22.03%	11.43%	26.69%
Good	20.51%	12.28%	32.20%	40.00%	23.75%
Very Good	15.38%	14.03%	23.73%	8.56%	10.85%
% of S_{Opt}	13.68%	10%	10.35%	6.14%	59.83%

What is immediately obvious from these heat maps is that the CPPN representation is allowing for more exploration of this solution space. With randomized maps, while the setting of each content type in each room of a map is uniformly random and independent, the sum of the settings across the map results in the Gaussian distribution shown in Fig. 8(b). The CPPN representation, however, is generating more maps that have consistently lower or higher levels of content in every room, thus producing more consistent patterns of content throughout a map. To improve the random baseline in future work, it would be beneficial to first randomly generate a maximum content density for enemies and pick-up and then randomly activate content in the rooms of geometry until this maximum density is reached.

In using the content balance metric described earlier in Section IV-B, Table III shows the four mentioned difficulty trends cross tabulated with the ratings given to each map. This only includes the 229 optimized maps that had three or more rooms between the start and end of the map, as the fluctuating difficulty trend is not possible with only two rooms. The final



(a)



(b)

Fig. 9. Mean classification scores of NB and comparative classifiers. (a) Mean prediction accuracy for given map index (S_{Opt} only). (b) Mean accuracy using LOOCV at the given map index (both S_{Opt} and S_{Rand}). All players except two were included in these plots.

column shows the ratings given to maps with only two rooms. Additionally, the final row of the table shows the percentage of maps with the given trend in S_{Opt}.

Of interest in these results is that 77.96% of maps with fluctuating difficulty were liked, with a rating of ‘Fair’ or better. Additionally, 23.73% of fluctuating maps were rated ‘Very Good’. Both of these are higher proportions than any other difficulty trend. In a Mann-Whitney U test where H_0 is that fluctuating maps receive worse or equal ratings than the other three difficulty trends, the null hypothesis was rejected at $p = 0.003$ and if one map of each was drawn at random there is a 0.62 probability of the fluctuating map having a better rating; this was not the case for equivalent tests for any of the other difficulty trends. This indicates that maps that have fluctuating difficulty are more likely to be enjoyed.

Following from this, maps with increasing difficulty are the second most likely type to be enjoyed, with a total of 65.38% of those maps being liked. Next were maps with only two rooms, with 61.29% of these small maps being liked. Also, as was highlighted in the previous section, despite the moderate ratings and the responses to the user survey, maps with only two rooms were played much more often than larger maps, making up nearly 60% of all ratings. Out of those maps with three or more rooms, the distribution of difficulty trends that were played is relatively even. Finally, while the constant difficulty trend had the highest proportion of “Good” ratings, it also had the lowest percentage of “Very Good” ratings and had the second highest proportion of maps that were disliked (59.99% liked), with maps with decreasing difficulty having the worst ratings overall (50.87% liked).

D. Player Model Prediction Accuracy

Fig. 9 shows the mean accuracy and LOOCV for the first 18 map indices for the various classifiers. The means are calculated from all players who played more than two maps, except for two. The next subsection details the reasons for excluding these two players from the plots. The mean accuracy measurement only includes predictions on maps in S_{Opt}. For LOOCV, however, both S_{Opt} and S_{Rand} are included in the test sets to thoroughly test the classifier's ability to generalize.

Fig. 9(a) shows that all classifiers experience a steep improvement in accuracy early on. The NB classifier compares well with the other classifiers at this stage. However, the mean accuracy of NB drops after 15 maps while the other classifiers generally continue to improve. The classifiers are all performing better than both runs of the random classifier, which both appropriately have an accuracy measure of around 0.5 for all map indices.

Both the random forest and the neural network classifiers are performing well. However, the downside to the use of the neural network is that time requirements for training the classifier sharply increase with the number of training samples. Thus, a simple improvement to the current implementation may be to use a random forest classifier in the RS player model and not the current NB classifier.

Fig. 9(b) shows the results of the LOOCV. The NB classifier typically generalizes better than the other classifiers for most training set sizes. However, in comparison with the earlier discussed accuracy plot, it may be that generalization is not needed in this system and a classifier that is overfitting is not necessarily a bad outcome.

1) Filtering Skewed Data: Two players were removed from the data set before the mean prediction accuracy was calculated. These two players experienced more maps than any other player, with the longest having played 86 maps and the second longest having played 23 maps. However, both players experienced a poor early game, with both players' classifiers predicting opposite classifications for most early optimized maps. Both players also responded to the optional survey and both identified themselves as being inexperienced game players. Upon inspection of gameplay data, it appears as though they have both provided contradictory ratings for similar map at the start of play that may be a result of their developing preferences and skill.

Despite this, both players stated that their experience improved over time, which is reflected in the accuracy plots of these two players (not shown here) as prediction accuracy begins to improve after about 20 maps. However, as all other players experienced 18 maps or less, this late improvement does not stop the data of these two players from skewing all other player data.

E. Results Discussion

The hierarchical optimization reduced the complexity of each subproblem, but also made evaluation more difficult. As the experiment tested all the solutions simultaneously in a real-world situation, attributing successes and limitations to individual solutions is nontrivial. Despite this, below is a discussion of some of the potential causes of the results presented earlier.

The statistical analysis of the ratings suggests that the optimized maps are, on average, receiving better ratings than randomly generated ones. However, the effect size is small and it's not clear whether this is due to poorly performing player models or a lack of variety in potential maps caused by the game design and constrained discrete generative processes. It's also possible that the statistical difference is due to IEC keeping the player more engaged or that the players sought to self-validate their IEC choices. A growing body of work hints that involving the player in the authorship of content through IEC improves their experience [8], [22], [24]. The present work adds yet another perspective on the possibilities for involving players in this way and raises questions about when the player's contribution is most essential and most satisfying.

The fixed n-ary tree representation produced a similar shaped distribution of room counts to random generation, but had a more even likelihood of medium and large geometries. Despite being given the opportunity of larger geometries during IEC and survey responses and ratings indicating preferences for medium and large sized maps, players instead mostly chose maps with only two rooms. This may be due to more small maps being shown during IEC, causing an unintended alteration of the player's subconscious preferences. Alternatively, it may be that players chose smaller maps due to previous bad experiences with larger ones. In both approaches, it is more likely for smaller maps to have an appropriate content balance (and therefore challenge) in every room, making them more likely to be completed.

The CPPN-NEAT approach to calculating content densities explored more of the solution space than random generation by creating more logical patterns of content throughout a map. It was also found that maps with fluctuating difficulty were rated well more often than other difficulty trends. This aligns with map design theory that promotes sequences of high and low intensity gameplay [45]. It should be noted that as a map grows in size it is more likely to have a fluctuating difficulty trend, which when combined with the CPPN patterns may explain why large optimized maps were more likely to be rated well.

While the mean prediction accuracy of the RS player models can be improved upon, the results are what we expected and are comparable with past studies on using NB classifiers for content-based RS [33]. However, there are also other recommender algorithms [29] which may provide a significant improvement.

Finally, 75.62% of survey respondents indicated that the variety of maps was fair or worse. This limitation may simply be due to the reuse of the same assets and mechanics in every map. Addressing this issue would either require better game design or the procedural generation of other aspects of the game. Alternatively, the lack of variety may be due to the RS or CPPN-NEAT being stuck in a local optimum, providing the player with similar content densities and, therefore, repetitive experiences. This type of overspecialization is a common issue in the RS field [29].

VI. Generalizing the Solutions

With creative thinking, a range of SBPCG problems can be decomposed. Some cases are more obvious, such as generating multiple types of game content that play significantly different roles in player's experience. For example, using separate evolutionary processes and preference models for weapon properties and the appearance of armor; or optimizing a narrative and then searching for a map to contain it, similar to the workflow of Hartsook *et al.* [19] However, less obvious cases are those where the result appears to be a single piece of content, such as the complete maps being generated in this paper. Another such example may include generating game mechanics by searching for scoring rules and win conditions separately. Some of these examples require information to be shared between evolutionary cycles, which can be achieved with a hierarchical approach or a coevolutionary one [11]. Decomposition should also be considered if the problem is a low-dimensional multiobjective optimization of game content, as it may be better to solve subproblems separately than to use weighted fitness functions or Pareto front methods.

However, problems should not be decomposed to the point where the resulting subcontent don't individually affect the player's experience. For example, wall arrangements and door placements both affect the player's navigation and exploration potential and so can be optimized as a single objective. Using multiple evolutionary processes can also be computationally expensive if the genetic operators, fitness evaluation, and genotype-to-phenotype conversion are not efficient.

The abstract concept of combining premade geometry templates to form a larger geometry has been a popular approach in the past [46]. However, the fixed n-ary tree representation of the geometry is the least generalizable of the subsolutions due to it being designed to work well with *PCG: Angry Bots* and the CPPN representation. It is suitable for use in structured linear games with maps in which there is one entry point and one exit point and where each node of the tree is self-contained. Examples of this are the Portal series (Valve Corporation, 2007), which presents a sequence of isolated physics puzzles to solve, and dungeons in roguelikes. It can also be used with exterior environments, such as heightmap terrains, if each geometry template has high walls on some borders to stop the player from leaving the valid area of play.

Using IEC to optimize the geometry of a map is suitable when knowing the map shape prior to play is beneficial (e.g., racing games) and geometry candidates can be clearly previewed, but is not appropriate when the exploration of unknown areas is a key entertainment factor of the game (e.g., many role playing games). Additionally, in our deployed system, it was more likely for small maps to be shown in the IEC interface and we observed that this may have influenced player preferences. When using IEC to generate game content, the interface should present a diverse range of candidates. This allows the player to naturally express their preferences without pressure from the system. In *PCG: Angry Bots*, diversity could be enforced by ensuring that no two map candidates in a single generation have the same number of rooms.

In this paper, we used locations on the geometry as the input to a CPPN and translated the output as a quantity of content at that location. In this case, the inputs were the coordinates of nodes in the geometry tree. However, other forms of location tags on the geometry or Cartesian coordinates could be used. It may also be possible to first determine the content layout and then procedurally generate the surrounding geometry, similar to how Dormans and Bakkes [16] build spaces around mission objectives using grammars. For the output of the CPPN, each piece of content can be given a separate output node of the network or one node can specify the type of content while the other specifies the quantity or difficulty setting of that type of content at the given input location. Here the output values were converted into discretized ordinal settings, but they could also be converted to integers to specify an exact quantity.

Currently the CPPN is being used to calculate discrete settings in small sized maps. However, many existing CPPN applications calculate continuous outputs from either continuous or high resolution inputs [22], [24]. In any of those applications, using discretized input or output may have reduced the visible patterns visible in the output. Using continuous inputs and outputs for content density should improve the variety of maps by producing more detailed patterns, but will likely also allow for invalid maps.

As with the fixed n-ary tree, this solution currently suits self-contained linear game maps. The content quantities are calculated before the map begins and therefore the map can become empty when the player moves through it. In games where space is to be reused, a mechanism would need to be introduced to periodically reintroduce content. In multiplayer experiences, it would also be necessary to determine the quantity and location of respawn points for the players.

The RS player model can be used during personalized PCG of game maps in most singleplayer game genres that have self-contained maps. This solution can also be used in multiplayer games, where the best candidate map is the one that has the highest likelihood of being congenial to all participating players. It can be easily adjusted to work with any dungeon or arena style gameplay, but not for open world landscapes or continuous gameplay where there is no logical break in play to gather subjective player feedback. Furthermore, the RS player model can be generalized to other forms of game content by keeping in mind that a player is synonymous with an RS user and the potential content solutions are the RS items. It can be used almost entirely as it is presented here, with the only required change being made to the content features, which can be designed with expert knowledge. The content should be described as completely as possible with as few features as possible.

It is important for the player to have some conscious preferences over the game content. For example, the player may not have any preferences over the shape of terrain, only an expectation that it's of a certain quality. Meanwhile, most players have strong preferences over the types of weapons that they use in a game and so can report this in subjective feedback. Additionally, a content-based RS is better suited for ongoing, repetitive experiences, such as experiencing multiple maps or weapons during the lifetime of play. However, for one-off experiences, such as the narrative in a game that may only be played once, it is better to utilize the experiences of others through collaborative filtering, similarly to the work of Yu and Riedl [31].

Subjective player feedback was used to train the RS player model here, but the player's actions during gameplay can also be monitored to deduce their preferences; a common approach in both RS [29] and EDPCG [6]. However, doing so can introduce bias as the designer must interpret how a player's actions reflect their emotions [26]. Finally, if a content-based RS is used, then a classifier and class setup must be chosen. A random forest classifier appears to be a strong choice and a weighted-binary class setup gives a suitable trade-off between player reporting granularity and classifier performance.

A. Generalizing the Methodology

The user experiment in this paper was administered in an unsupervised public format. This was done to evaluate the proposed solutions in a natural, relaxed environment. Participants could play for as long as they wanted, at a convenient time, in anonymity, and without the pressure of performing in front of a research team. This simulates the way in which commercial games may be beta tested within the community and avoids unintentional experimental biases where possible. Such experiments are better suited to evaluating the commercial viability of a solution over structured, supervised user experiments. They also allow for data to be gathered from a larger and more diverse audience than could otherwise be recruited to on-site lab testing, especially for small research teams.

Unfortunately, this approach does reduce the amount of experimental control and leads to noisy player data. Controlled environments allow for easier evaluation of the resulting data and for more definitive conclusions because variables can be fine-tuned during participation and users can be probed for further information. However, the data is not gathered from the entirety of the user experience, which, for games, involves player's enjoying the activity as a form of entertainment rather than as a task to be completed under supervised conditions.

In the unsupervised setting, proper statistical analysis of the resulting data becomes an important means of discerning meaning from the noisy player data. In a personalized PCG solution, we are primarily concerned with detecting whether the system is providing a better experience over a random baseline, whether the players' preferences are being properly captured and utilized, and the generative range of the various PCG components. These are evaluated here by the metrics described in Section IV. Conveniently, the subjective player feedback can be used to not only construct a player model, but also as a means of evaluating a system's performance.

Finally, administering postplay surveys in an unsupervised public experiment can be difficult, especially when the period of play is indefinite (i.e., there is no end to the game). The survey can be forced onto the player after a specified time, but this invasive act may encourage a premature stop in play. On the other hand, making the survey optional avoids interruption to play, but makes it easier for the player to ignore or forget it. As a design choice, we used the latter setup as the survey data was of secondary interest compared to the interaction data.

VII. Conclusion and Future Work

In this paper we investigated procedurally generating maps of a 3D action-shooter game to match the preferences of individual players. The map generation and optimization process was decomposed into two integrated searches; one for geometry and the other for content densities. The geometry was represented by a fixed n-ary tree and evaluated through IEC, while the density of content was indirectly represented by a CPPN and a content-based RS was used as the fitness evaluator to CPPN-NEAT. An unsupervised public experiment showed that while decomposition allowed for appropriate solutions to be devised for each search, it also made the attribution of performance more difficult. However, results indicated that the systems were performing well, but that there is room for improvement. Additional findings include players choosing smaller geometries despite reporting that they enjoyed larger ones and that players generally preferred maps with a fluctuating difficulty. Our current implementation somehow lacks variety in experiences, which may be due to limitations in gameplay design, the discretized generative processes, or overspecialization during optimization.

Planned future work currently includes testing the RS approach to player modeling in other contexts, both as it is presented here as well as including advances from the RS research field. Applying a collaborative filter to a domain in which there are no coratings is difficult, but the results here show that there are commonalities between players that could be exploited with a modified collaborative filter. This would be an efficient way of improving the quality of recommendations after fewer maps, while still using a per-player preference models. We also wish to investigate the mitigation of overspecialization with a novelty function [47] during CPPN-NEAT. However, simply looking for novelty may result in many maps that a player doesn't enjoy. Using a feasible-infeasible dual population [48] could allow for maps that are predicted to be disliked to be placed in the infeasible population and explore novelty only in maps that are predicted to be enjoyed.

The study of PCG, and more specifically procedural map generation, remains to be a topic that is open to creative solutions. While many games and content types require customized solutions, best practices are starting to emerge for generative PCG, such as the use of World Machine (World Machine Software, 2014) or SpeedTree (Interactive Data Visualization Inc, 2014) in commercial games. However, the personalization of games through PCG is only recently gaining research interest and is yet to produce any community leading solutions. Hence, we look forward to future investigations in this emerging field that has the potential to change the way that we develop and experience games.

Acknowledgment

The experiment in this paper uses graphical, audio, and gameplay scripting assets from the Unity Game Engine (Educational license). The authors would like to thank the anonymous reviewers whose time and effort has helped improve this paper.

References

[1] A. Drachen, A. Canossa, and G. N. Yannakakis, "Player modeling using self-organization in Tomb Raider: Underworld," in *Proc. IEEE Symp. Comput. Intell. Games (CIG 2009)*, Sep. 2009, pp. 1–8.

[2] M. C. Machado, E. P. Fantini, and L. Chaimowicz, "Player modeling: Towards a common taxonomy," in *Proc. 16th Int. Conf. Comput. Games (CGAMES)*, Jul. 2011, pp. 50–57.

[3] S. Berkovsky, J. Freyne, M. Coombe, and D. Bhandari, "Recommender algorithms in activity motivating games," in *Proc. 4th ACM Conf. Recommender Syst.*, 2010, pp. 175–182.

[4] R. Hunicke and V. Chapman, "Ai for dynamic difficulty adjustment in games," in *Proc. Challenges Game Artif. Intell. AAAI Workshop*, 2004, pp. 91–96.

[5] D. Johnson and J. Wiles, "Effective affective user interface design in games," *Ergonomics*, vol. 46, no. 13-14, pp. 1332–1345, Oct. 2003.

[6] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Trans. Affective Comput.*, vol. 2, no. 3, pp. 147–161, Jul. 2011.

[7] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Program. Evolvable Mach.*, vol. 8, no. 2, pp. 131–162, Jun. 2007.

[8] L. Cardamone, D. Loiacono, and P. L. Lanzi, “Interactive evolution for the procedural generation of tracks in a high-end racing game,” in *Proc. 13th Annu. Conf. Genetic Evol. Comput.*, 2011, pp. 395–402.

[9] M. J. Pazzani and D. Billsus, *Content-Based Recommendation Systems*. Berlin , Germany: Springer-Verlag, 2007, vol. 4321, Lecture Notes in Computer Science, ch. 10, pp. 325–341.

[10] B. Medler, “Using recommendation systems to adapt gameplay,” *Int. J.Gaming Comput.-Mediated Simul. (IJGCMS*, vol. 1, no. 3, pp. 68–80, 2009.

[11] M. Cook and S. Colton, “Multi-faceted evolution of simple arcade games,” in *Proc. IEEE Symp. Comput. Intell. Games (CIG)*, 2011, pp. 289–296.

[12] W. L. Raffe, F. Zambetta, and X. Li, “Neuroevolution of content layout in the PCG: Angry Bots video game,” presented at the IEEE Congr. Evol. Comput. (CEC),, 2013.

[13] 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, Sep. 2011.

[14] T. Tutenel, R. M. Smelik, R. Lopes, K. J. de Kraker, and R. Bidarra, “Generating consistent buildings: A semantic approach for integrating procedural techniques,” *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 3, pp. 274–288, Sep. 2011.

[15] A. Uriarte and S. Ontanón, “PSMAGE: Balanced map generation for StarCraft,” presented at the 2013 IEEE Conf. Computat. Intell. Games (CIG), , 2013.

[16] J. Dormans and S. Bakkes, “Generating missions and spaces for adaptable play experiences,” *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 3, pp. 216–228, Sep. 2011.

[17] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, “Game-o-matic: Generating videogames that represent ideas,” in *Proc. 3rd Workshop Procedural Content Generation Games*, 2012, p. 11.

[18] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, and G. N. Yannakakis, “Multiobjective exploration of the starcraft map space,” in *Proc. IEEE Symp. Comput. Intell. Games (CIG)*, 2010, pp. 265–272.

[19] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, “Toward supporting stories with procedurally generated game worlds,” in *Proc. IEEE Symp. Comput. Intell. Games (CIG)*, Aug. 2011, pp. 297–304.

[20] M. Cook, S. Colton, and J. Gow, “Initial results from co-operative co-evolution for automated platformer design,” in *Appl. Evol. Comput.* . Berlin, Germany: Springer-Verlag, 2012, vol. 7248, Lecture Notes Comput. Sci., ch. 20, pp. 194–203.

[21] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolu. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[22] E. J. Hastings, R. K. Guha, and K. O. Stanley, “Evolving content in the Galactic Arms Race video game,” in *Proc. IEEE Symp. Comput. Intell. Games (CIG)*, 2009, pp. 241–248.

[23] A. Liapis, G. N. Yannakakis, and J. Togelius, “Adapting models of visual aesthetics for personalized content creation,” *IEEE Trans. Comput. Intell. AI in Games*, vol. 4, no. 3, pp. 213–228, Sep. 2012.

[24] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley, “Combining search-based procedural content generation and social gaming in the Petalz video game,” in *Proc. 8th Artif. Intell. Interactive Dig. Entertainment Conf.*, 2012, pp. 63–68.

[25] A. M. Smith, C. Lewis, K. Hullett, G. Smith, and A. Sullivan, An Inclusive Taxonomy of Player Modeling Univ. California, Santa Cruz, CA, USA, 2011, Tech. Rep. UCSC-SOE-11-13.

[26] N. Shaker, G. N. Yannakakis, and J. Togelius, “Towards automatic personalized content generation for platform games,” presented at the Proc. AAAI Conf. Artif. Intell. Interactive Digital Entertainment (AIIDE), 2010.

[27] J. Togelius, R. De Nardi, and S. M. Lucas, “Towards automatic personalised content creation for racing games,” in *Proc. IEEE Symp. Comput. Intell. Games (CIG)*, 2007, pp. 252–259.

[28] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, “Polymorph: Dynamic difficulty adjustment through level generation,” in *Proc. 2010 Workshop Procedural Content Generat. Games*, 2010, p. 11.

[29] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, Jun. 2005.

[30] A. E. Zook and M. O. Riedl, “A temporal data-driven player model for dynamic difficulty adjustment,” presented at the Proc. AAAI Conf. Artif. Intell. Interact. Digital Entertainment, 2012.

[31] H. Yu and M. O. Riedl, “A sequential recommendation approach for interactive personalized story generation,” in *Proc. 11th Int. Conf. Autonom. Agents Multiagent Syst.*, 2012, vol. 1, pp. 71–78.

[32] C. Green, “Phased Searching with NEAT: Alternating between Complexification and Simplification,” 2004 [Online]. Available: http://sharpneat.sourceforge.net/phasedsearch.html

[33] M. Pazzani and D. Billsus, “Learning and revising user profiles: The identification of interesting web sites,” *Mach. Learn.*, vol. 27, no. 3, pp. 313–331, 1997.

[34] P. Domingos and M. Pazzani, “Beyond independence: Conditions for the optimality of the simple bayesian classi er,” in *Proc. 13th Int. Conf. Mach. Learn.*, 1996, pp. 105–112.

[35] G. N. Yannakakis and J. Hallam, *Ranking vs. Preference: A Comparative Study of Self-Reporting*. Berlin, Germany: Springer-Verlag, 2011, vol. 6974, Lecture Notes Comput. Sci., ch. 47, pp. 437–446.

[36] H. B. Mann and D. R. Whitney, “On a test of whether one of two random variables is stochastically larger than the other,” *Annal. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 1947.

[37] G. Shani and A. Gunawardana, *Evaluating Recommendation Systems*. Berlin, Germany: Springer-Verlag, 2011, ch. 8, pp. 257–297.

[38] D. W. Aha, D. Kibler, and M. K. Albert, “Instance-based learning algorithms,” *Mach. Learn.*, vol. 6, no. 1, pp. 37–66, Jan. 1991.

[39] J. R. Quinlan, *C4. 5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993, vol. 1.

[40] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[41] J. Cohen, “A power primer,” *Psychol. Bulletin*, vol. 112, no. 1, pp. 155–159, 1992.

[42] H. Cremér, *Mathematical Methods of Statistics (PMS-9)*. Princeton, NJ, USA: Princeton Univ. Press, 1999, vol. 9.

[43] R. J. Grissom and J. J. Kim, *Effect Sizes for Research: Univariate and Multivariate Applications*, 2nd ed. New York, NY, USA: Routledge, 2012.

[44] F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics*, vol. 1, no. 6, pp. 80–83, Dec. 1945.

[45] E. Adams, *Fundamentals of Game Design*, 2nd ed. New York, NY, USA: New Riders, 2009.

[46] 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 in Games*, vol. 3, no. 1, pp. 1–16, Mar. 2011.

[47] J. Lehman and K. O. Stanley, “Exploiting open-endedness to solve problems through the search for novelty,” *Artif. Life*, vol. 11, p. 329, 2008.

[48] A. Liapis, G. N. Yannakakis, and J. Togelius, “Enhancements to constrained novelty search: Two-population novelty search for generating game content,” in *Proc. Genetic Evol. Comput. Conf.*, 2013.

William L. Raffe received the B.Sc. degree (Hons) in computer science and the Ph.D. degree in computer science from the School of Computer Science and Information Technology at RMIT University, Australia, in 2008, 2009, and 2014, respectively.

He is currently a Postdoctoral Research Fellow at the same institution, investigating adaptive play in virtual-physical environments. He is also a sessional lecturer and a member of the RMIT Evolutionary Computing and Machine Learning Group. His research interests include personalized procedural content generation, adaptive game A.I., player preference modeling, and exergaming.

Fabio Zambetta received the M.S. and Ph.D. degrees in computer science from the University of Bari, Bari, Italy, investigating the use of 3-D personae as adaptive intelligent interfaces.

He is currently a Senior Lecturer with the School of Computer Science and Information Technology, RMIT University in Melbourne, Melbourne, Australia. His current research interests include procedural generation of game play, player modeling, and GPU computing.

Dr. Zambetta is an Associate Editor for ACM CIE, and he is an ACM and IEEE Member. He also serves on the IEEE Games Technical Committee.

Xiaodong Li (M'03–SM'07) received the B.S. degree in information science from Xidian University, Xi'an, China, in 1988, and the Dip.Com. and Ph.D. degrees in information science from the University of Otago, Dunedin, New Zealand, in 1992 and 1998, respectively.

He is currently an Associate Professor with the School of Computer Science and Information Technology, RMIT University, Melbourne, Australia. His research interests include evolutionary computation (in particular, evolutionary multiobjective optimization, evolutionary optimization in dynamic environments, large scale optimization, and multimodal optimization), neural networks, complex systems, and swarm intelligence.

Dr Li is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the *Journal of Swarm Intelligence* (Springer), and the *International Journal of Swarm Intelligence Research (IJSIR)*. He is currently the Chair of the IEEE CIS Task Force on Large Scale Global Optimization and a Vice-Chair of the IEEE CIS Task Force on Swarm Intelligence. He is a Member of the editorial board of the journal of *Softcomputing* (Springer), and a Member of the Technical Committee on *Soft Computing, Systems, Man and Cybernetics Society*, IEEE. He is an Advisor on the Scientific Advisory Board of *SolveIT Software*. He is a Vice-Chair of IEEE Victorian Section CIS Chapter, Melbourne, Australia. He is the recipient of the 2013 ACM SIGEVO Impact Award.

Kenneth O. Stanley received a B.S.E. degree from the University of Pennsylvania, Philadelphia, PA, USA, in 1997, and the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 2004.

He is an Associate Professor in the Department of Electrical Engineering and Computer Science at the University of Central Florida, Orlando, FL, USA. He is an inventor of the Neuroevolution of Augmenting Topologies (NEAT), HyperNEAT, and novelty search algorithms for evolving complex artificial neural networks. His main research contributions are in neuroevolution (i.e., evolving neural networks), generative and developmental systems (GDS), coevolution, machine learning for video games, and interactive evolution.

Dr. Stanley won best paper awards for his work on NEAT, NERO, NEAT Drummer, FSMC, HyperNEAT, ES-HyperNEAT, adaptive HyperNEAT, novelty search, and Galactic Arms Race. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, is on the editorial board of the *Evolutionary Computation Journal*, and on the ACM SIGEVO Executive Committee. He is also a Cofounder and the Editor-in-Chief of aigameresearch.org.