# A Multi-Faceted Surrogate Model for Search-based Procedural Content Generation

Daniel Karavolos, *Student Member, IEEE,* Antonios Liapis, *Member, IEEE,* and Georgios Yannakakis, *Senior Member, IEEE*

***Abstract*—This paper proposes a framework for the procedural generation of level and ruleset components of games via a surrogate model that assesses their quality and complementarity. The surrogate model combines level and ruleset elements as input and gameplay outcomes as output, thus constructing a mapping between three different facets of games. Using this model as a surrogate for expensive gameplay simulations, a search-based generator can adapt content towards a target gameplay outcome. Using a shooter game as the target domain, this paper explores how parameters of the players' character classes can be mapped to both the level's representation and the gameplay outcomes of balance and match duration. The surrogate model is built on a deep learning architecture, trained on a large corpus of randomly generated sets of levels, classes and simulations from gameplaying agents. Results show that a search-based generative approach can adapt character classes, levels, or both towards designer-specified targets. The model can thus act as a design assistant or be integrated in a mixed-initiative tool. Most importantly, the combination of three game facets into the model allows it to identify the synergies between levels, rules and gameplay and orchestrate the generation of the former two towards desired outcomes.**

***Index Terms*—Procedural Content Generation, Search-based PCG, Surrogate Model, Deep Learning, Shooter Games**

## I. Introduction

GAME design practitioners must integrate a variety of creative domains, such as visuals, music and narrative, to bring about an intended experience in a player. Liapis *et al.* have identified six different facets of games which require distinct creative input: rules, levels, visuals, audio, narrative and gameplay [1]. When designing a new element for a game, a designer must consider its effect on players' experience, as well as how it matches with the other content. For example, switching from a game rule that allows players to regenerate health continuously to a rule that a player colliding with a specific powerup heals a preset number of hit points will have vast repercussions across facets. For starters, the number of hit points healed (which is a detail of the above rule) may greatly affect the priorities of players in seeking this powerup out, and may unbalance other parts of the ruleset such as the relative power of different weapons. Moreover, the powerup will need a mesh and textures to signify its function, while the game levels will need to include such powerups at critical locations which must also be carefully considered. Finding how one element of the game design (e.g. a visual asset, a level design choice, or a game rule) will influence the design of other game elements as well as the player's experience is a core challenge for a human designer — let alone a computer.

Procedural content generation (PCG) has largely sidestepped the issue by focusing on a single type of content at a time, and running the generative algorithm in a well-defined space that does not require knowledge about the rest of the game. For instance, a level generator for *StarCraft* (Blizzard, 1998) already has established what the possible resources, the units' statistics, or the textures of the game are. The generator thus must only consider how to distribute spatially game elements that are otherwise pre-authored [2]. When the need for multiple generators does arise, their dependencies are carefully crafted. For example, *Dwarf Fortress* (Bay 12 Games, 2006) has many generators that contribute their content independently, while in *No Man's Sky* (Hello Games, 2016) the important information of one generator is passed to the next generator as a sequential pipeline.

This paper proposes a framework for combining the generation of levels and rulesets by evaluating the joint artifacts with a model of gameplay. This model has been trained via deep learning on gameplay logs to automatically extract features and find an approximate mapping between game levels, rules and gameplay outcomes. The model does not offer the precision of a simulation, but its real-time feedback makes it a useful design tool. The computational model is thus able to identify interrelations between the three facets without needing to produce and test the outcome, following a similar process to a human designer who drafts and corrects content without playtesting every iteration. The model can be used as a surrogate for actual game playthroughs, informing generators who adapt content of all contributing facets (in this case levels and rules). The ability of the framework to both predict gameplay outcomes from the game design specifications and to automatically create new designs renders it a computational creator, which can easily work alongside a human creator in a mixed-initiative design task [3]. Indeed, the experiments in this paper show the benefits of this approach for fine-tuning an initial design towards designer-specified gameplay goals.

The proposed framework is tested on a two-player, competitive first-person shooter, which was chosen for its relatively straightforward gameplay. The gameplay facet is quantified as two outcomes: the score balance between the two opponents and the duration of the game. The level facet is the arena in which the game takes place and includes the placement of powerups that affect the players' survivability. The rules facet is represented by the character classes of

All authors are with the Institute of Digital Games, University of Malta, Msida, MSD 2080, Malta

Manuscript received August 19, 2018; revised X.

the two players. *Character classes* are a common way to group game mechanics in multiplayer shooter games such as *Team Fortress 2* (Valve, 2007) and *Battlefield: Bad Company* (Electronic Arts, 2008). Character classes offer players different gameplay options (e.g. scouting or area control) and may have a different survivability and movement speed as well as signature weapons that balance their affordances. A computational model that maps levels and rules to gameplay outcomes is trained via deep learning on a rich corpus of simulated playthroughs. Moreover, this model is used to adapt either facet individually or orchestrate the generation of both. Experiments show that better gameplay is obtained when both facets are adjusted simultaneously, in an orchestrated way.

This paper builds on earlier work, where a model was trained to find associations from levels and rules to gameplay in a more abstract game [4], as well as surrogate-based generative methods for the current game framework [5], [6]. Unlike those earlier studies, in this paper the corpus of gameplay simulations is balanced while each gameplay outcome is predicted via its own, individual model (rather than using one model with two outputs). More importantly, this study extends previous work by (a) introducing a multi-objective approach where the two target gameplay outcomes are treated as independent objectives, (b) performing for the first time *surrogate-based orchestration* where both levels and rules are adapted simultaneously. Both of these new additions are shown to perform better as adapting both levels and rules can result in balanced matches with much fewer changes in class parameters or levels than when adjusting only one facet.

## II. Related Work

Procedural content generation has been used in the game industry for almost 40 years, and it has been an active field of academic study for over a decade. With its long history, it is not surprising that a wide variety of algorithmic techniques have been used, such as generative grammars, constraint solving and searching the design space [7]. This paper focuses on search-based algorithms [8], as they allow the generative process to be guided via evaluations of the intermediate content. These evaluations can be based on the explicit choices of the designer [9] or the player [10], a computational model of preference [11], [12], game simulations [13], [14] or static fitness functions [15], [16].

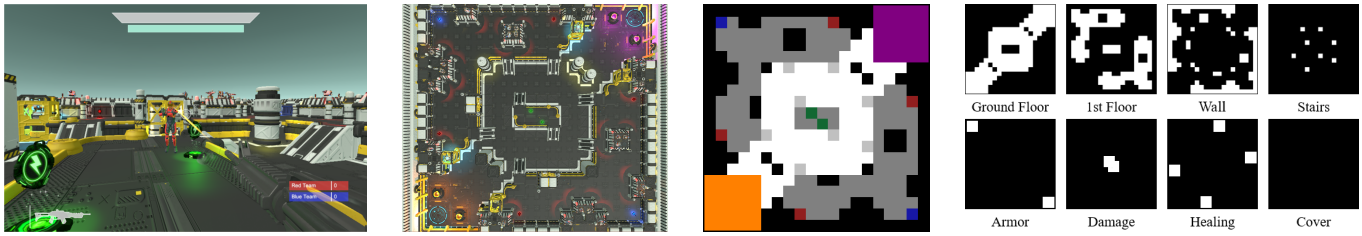
While PCG applications usually focus on one facet of games (usually levels), a key challenge along the path of fully autonomous game generation is the *orchestration* of the multitude of creative facets of a game. Based on the survey of [1], games consist of six facets (visuals, audio, narrative, levels, rules and gameplay) and designing content of one facet must account for properties of the other facets. Cook and Colton [17] describe a multi-faceted approach to generating arcade games, using static fitness functions on each facet separately. At each step of the process, each generator incorporates the best design of the other generators into the evaluation of their design. Without actual game simulations, however, it was difficult to achieve playable content. Using simulated playthroughs with artificial agents has shown promise for orchestrating multiple game facets, as each facet may affect the simulation differently. In the seminal work of Browne and Maire [14], both rulesets and board layouts were optimized based on metrics that processed the progress of gameplay between two agents with the same decision-making criteria. Similar to [14], this paper approaches orchestration by combining both levels and rules in the same genotype: therefore, facet generation is concurrent rather than via a generative pipeline favored by several game orchestration studies [1].

Simulated gameplay has been used to compute metrics for search-based PCG in several ways. For instance, [14] computed 57 metrics, many of which related to game progression: these included the duration of the game (in moves), the uncertainty of the game and its drama (i.e. how easy one player could reclaim the lead towards victory). When assessing game rules, [13] measured the learnability of the game by observing the average score of a population of controllers learning to play the game across all generations. In the genre of first-person shooters, Lanzi et al. [18] evolve levels towards score balance between players, while Gravina et al. [19] use the heatmaps of players' deaths from a simulation in order to create surprising weapons that break heatmap patterns. This paper extracts two simple but established metrics from the simulations: the score ratio at the end of the simulation, similar to [18], and the duration of the match, similar to [14].

Unlike simulation-based evaluations, gameplay outcomes in this paper are *predicted* by a convolutional neural network (CNN), which acts as a *surrogate model* [20]. Though famous for their classification performance in natural images, CNNs can learn to extract patterns from any type of spatially arranged input. Convolutional neural networks have been successfully applied to guide game playing agents in a large variety of games, most notably [21], [22], which has led to a surge of academic interest in deep learning for game playing [23]. Patterns extracted via deep learning can also be applied to generate game content, e.g. by predicting tile types in levels for *Super Mario Bros.* (Nintendo, 1985) [24] or resource locations for maps in *StarCraft II* (Blizzard, 2010) [25]. Often, algorithms apply the machine learned patterns directly to an initial design and use the output as the generated content, without evaluating its gameplay [26]. An interesting exception is [27], which describes a two-phased approach for creating *Super Mario Bros.* levels. In the training phase, a convolutional generative adversarial network is trained to learn patterns of existing levels by mimicking them. In the generation phase, a simulation-based algorithm searches the latent space of the generative network for playable levels with the desired gameplay. This approach is in many ways the inverse to the one described in this paper, where the CNN learns patterns in the latent representation but the generator applies changes on the original representation (e.g. the 2D map of the level).

## III. Game Framework

This paper uses first-person shooter games as a test case for mapping level structure and game parameters to gameplay metrics. This mapping is used as a surrogate to balance a one versus one deathmatch game between two players by adjusting




(a) 3D level: player's perspective (b) 3D level: top-down (c) Level sketch (d) CNN inputs

Fig. 1. The in-game 3D level, its representation as a level sketch, and its transformation into CNN inputs. In Fig. 1c, orange and purple areas are the bases of player 1 and respectively; black tiles are impassable walls; white and gray tiles are the ground and first floor; light gray tiles are stairs from the ground floor to the first floor; red tiles are healing locations, blue and turquoise tiles are armor and double damage powerups respectively.

the level, the players' character classes or both. The game used for these experiments (see a screenshot in Fig. 1a) is implemented in the *Unity 2017* game engine, and is based on an existing toolkit [28].

### A. Gameplay Facet

The players are assumed to be on an equal skill level, and each of them controls an avatar that belongs to a specific *character class*. This paper uses the same character class names and attributes from *Team Fortress 2* (TF2) as a frame of reference for the game parameters used in the experiments. While game parameters for character classes and their weapons are described below, an important parameter is the hit points (HP): if any player drops to 0 HP, they are killed and respawn at their starting location (base). Deathmatch games are competitive, and the player who scores more kills than their opponent(s) is the winner. A session in a deathmatch game finishes usually when the time limit expires or after a specific number of kills by one or both players. The framework in this paper considers matches to be complete when a total of 20 kills are scored; a time limit of 600 seconds is also in place, but results from matches that timed out are ignored.

### B. Character Class Parameters (Game Rules Facet)

The character class of each avatar is represented by eight parameters. Two of these parameters are specific to the character, i.e. hit points and movement speed, while the other six are characteristics of their weapon: damage (per shot), accuracy (i.e. the size of the cone in which bullets are fired), rate of fire, clip size, the number of bullets per shot and weapon range. As noted earlier, the inspiration for the class parameters is the TF2 game; experiments in this paper use parameter values from the game itself[1]. The only addition to TF2 parameters was that of range, to discern when AI agents should shoot. There are three options for range: "short", "medium" and "long".

### C. Level Facet

The levels in the game consist of a grid of $20 \times 20$ tiles. Each tile may be an impassable, tall wall or a passable tile on the ground or first floor. There are no tunnels or bridges. Stairs connect the ground floor with the first floor and players can drop from any ledge to go down from the first floor to the ground floor. Passable tiles can contain one of three types of powerups typically seen in shooter games: a *health pack* (increases HP up to a maximum), *armor* (offers additional HP which is depleted first) and a *damage boost* (player's bullets temporarily deal double damage). An example level is shown in Figure 1c: this simple representation will be used throughout the paper, while the level can be transformed into a 3D detailed mesh as shown in Fig. 1b and Fig. 1a. The spawn point of the first player (P1) is always in the bottom left corner, while the second player (P2) always spawns in the top right corner.

[1]https://wiki.teamfortress.com/wiki/Classes

## IV. Surrogate Model

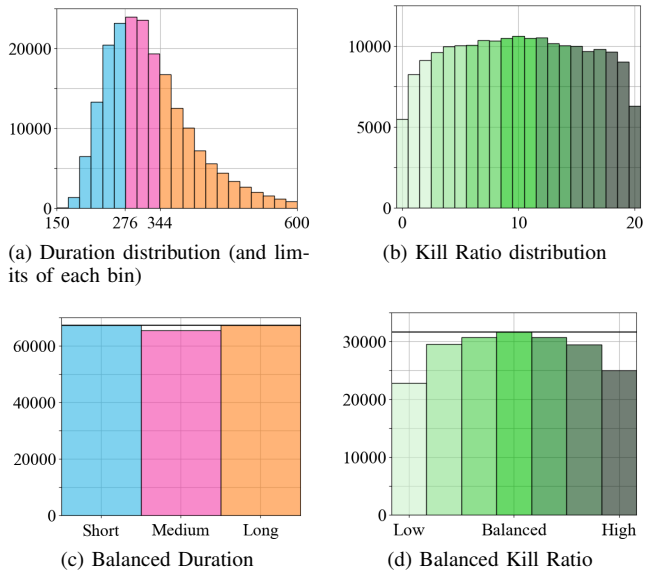
Various models for mapping aspects of the level, game rules and gameplay facets were explored in [4]–[6]. Results showed that the best mapping was obtained by a convolutional neural network (CNN), which informed the choices of this paper. Unlike earlier work, however, in this paper a separate CNN for each gameplay outcome (kill ratio and match duration) is trained, as this simplifies data balancing per outcome. Each CNN is trained with the same hyperparameters. This section describes the corpus used for training, the network architecture and the model's performance.

### A. Dataset Construction

In order to obtain the rich and expressive dataset required for deep learning, $10^5$ sets of levels and classes were procedurally generated and evaluated in simulated matches between artificial agents. A generated level and pair of character classes was simulated twice, swapping the classes of player 1 and player 2 in the second match. If either match did not result in a total of 20 kills within the time limit (600 seconds), it was ignored and a new level-class combination was generated. This brings the total size of the dataset to $2 \cdot 10^5$ data points (matches).

To simulate the gameplay, we use artificial agents controlled by behavior trees that were adapted from [29]. In descending order of priority, agents (a) search for health packs if their health is low, (b) pick up nearby powerups, (c) attack the opponent if one is in sight, and (d) search for distant powerups.

To generate the 8 character class parameters described in Section III-B, each parameter was normalized within $[0, 1]$ based on a predetermined value range per parameter (inspired by the values of TF2 classes). A random real value within $[0, 1]$ was assigned to each parameter except for range which was randomly assigned a "short", "medium" or "long" value. With two competing classes, this results in 16 parameters.




(a) Duration distribution (and limits of each bin)

(b) Kill Ratio distribution

(c) Balanced Duration

(d) Balanced Kill Ratio

Fig. 2. A visualization of the distribution of the original gameplay outcomes in (a), (b). The size of each bin is shown before oversampling in (c) and (d) and the black lines show the size of all bins after oversampling.

To generate the levels for the corpus, we use a constructive approach which combines random digging agents and generative grammars [30]. The final level consists of $20 \times 20$ tiles, broken into 16 cells of $5 \times 5$ tiles each (see the cell breakdown in Fig. 4). Two paths are created between the bottom-left and top-right corners of the level, first at the cell level and then at the tile level. Paths initially remove walls to add ground-floor tiles, but first-floor tiles are added via a random transformation of some walls followed by an iteration of cellular automata as per [30]. Among all possible stair locations (between a ground-floor and first-floor tile), one staircase is placed with a 20% chance. Each unreachable first floor tile is transformed into a wall, guaranteeing that all first-floor tiles are reachable. Finally, there is a 33% chance that a cell will have a powerup. This powerup is randomly assigned a type and randomly placed on a ground or first-floor tile in that cell.

### B. Gameplay Outcomes and Dataset Balancing

Two gameplay outcomes were computed from each simulated match: the kill ratio ($KR$) of the first player's kills to the total kills, and the duration of the match in seconds. The distribution of these two gameplay metrics is shown in Figure 2. $KR$ is almost uniformly distributed between matches with a clear advantage for P1 ($KR = 1$), a clear advantage for P2 ($KR = 0$) and balanced matches ($KR = 0.5$). Match durations range from 150 seconds to 600 seconds, but are skewed towards 300 seconds, with few matches lasting less than 200 seconds (2%) or more than 500 seconds (4%). Performing machine learning on unbalanced data might introduce a bias in the model. As such, a more uniformly distributed dataset for each outcome was obtained by oversampling the data with infrequent target values. The bins required for counting occurrences were created by artificially discretizing the outcome values in an ad-hoc manner.

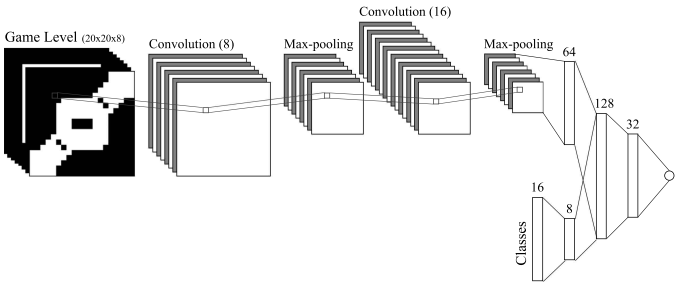



Fig. 3. The architecture of the CNN-based model, which combines the 2D channels of the level in Fig. 1 with the class parameters of both players as a vector. The single output of the model is the predicted gameplay outcome (match duration or kill ratio).

The duration targets ($t$) are normalized via min-max normalization and split into three equally sized bins with edges at $t = 0.28$ (276 sec) and $t = 0.43$ (344 sec). With this discretization, the first bin contains the lower third of the dataset's $t$ (shortest matches) and the third bin contains the upper third of the dataset's $t$ (longest matches). After oversampling for balancing bins in $t$, the training set and validation set sizes are respectively $182.7 \times 10^3$ and $20.2 \times 10^3$ (see Fig. 2c).

The $KR$ targets have 20 possible values, which are split into 7 bins. Treating a difference of one kill as insignificant, the discretized space is based on treating $KR = 0.5 \pm 0.05$ as one bin, with equally sized bins up and down in the range of $[0, 1]$. After oversampling for balancing bins in KR, the new training set and validation set sizes are respectively $199.1 \times 10^3$ and $22.1 \times 10^3$ (see Fig. 2d).

### C. Convolutional Neural Network Architecture

The general architecture of the network chosen for this task is similar to [4]–[6], although some hyperparameters were changed after preliminary experiments. The inputs of each CNN are the 16 character class parameters normalized to $[0, 1]$ and a set of 2D binary channels for the level, while the outputs of the CNN is the predicted gameplay outcome of a match simulation. The level input consists of 8 binary channels (one for walls, two for tile elevation, three for powerups, one for stairs and one for cover positions[2]), shown in Fig. 1d.

Each CNN has two separate information streams, one for the level and one for the pair of character classes. The level input is passed through two blocks of convolution and max-pooling, with 8 and 16 filters respectively. The convolutions are of size $5 \times 5$ (without zero-padding), and the end-result is a flat vector of 64 features for the level. The 16 parameters of the character classes are passed to a single fully-connected layer of 8 nodes, the output of which is concatenated to the flat feature vector of the level. Finally, this combined feature vector is connected to a fully connected layer of 128 nodes which connects to another layer of 32 nodes, which connects to one output that predicts the gameplay metric (kill ratio or duration). All nodes use an ELU activation function [31] and the output of each hidden layer is normalized via batch normalization [32]. The resulting architecture is shown in Figure 3.

[2]Cover positions are not used in the current levels, so all pixels for this channel are currently 0.

TABLE I
MEAN ABSOLUTE ERROR (MAE) AND THE $R^2$ VALUES FOR DURATION ($t$) AND SCORE ($KR$) PREDICTION ON THE VALIDATION SET. RESULTS ARE AVERAGED FROM 10 INDEPENDENT RUNS.

| Model | $MAE_{KR}$ | $MAE_t$ | $R^2_{KR}$ | $R^2_t$ |
|---|---|---|---|---|
| Linear Regression | 0.0959 | 0.0941 | 0.8316 | 0.4912 |
| Perceptron | 0.0856 | 0.0936 | 0.8563 | 0.4931 |
| MLP | 0.0846 | 0.0842 | 0.8562 | 0.5810 |
| CNN | 0.0673 | 0.0817 | 0.9118 | 0.6044 |

### D. Model Validation

In order to validate the performance of the CNN architecture, several baseline models were trained on the same tasks. For the sake of brevity, this paper reports the results of the best multi-layer perceptron (MLP), which has a hidden layer of 128 nodes, as well as linear regression (LR) and a perceptron. All networks were implemented in Keras [33] and trained with Adam [34] using 10% hold-out. Each model was trained for up to 100 epochs, using early stopping to prevent overfitting.

Modeling duration ($t$) and kill ratio (KR) are both regression tasks. Such tasks are typically evaluated via (a) the model's prediction error and (b) how much of the variance in the data is explained by the model. The former is computed by the mean absolute error, $MAE_t$ and $MAE_{KR}$ for duration and kill ratio respectively. The latter is computed by the $R^2$ metric (with typical ranges of $[0, 1]$) for these dimensions ($R^2_t$ and $R^2_{KR}$). Table I shows the average of these performance criteria based on 10 independent runs.

All models can fairly accurately predict the kill ratio, as $R^2_{KR} \in (0.83, 0.92)$. Interesting in this regard are the perceptron and the MLP as their kill ratio accuracy is very similar, but the seemingly small difference in $MAE_t$ has a big impact on how well they model duration, as reflected by the large difference in $R^2_t$. Perhaps this is an indication that, despite the data balancing, it is hard for the models to predict when the matches will be extremely long or short.

In general, the more complex models have a lower error and a higher explained variance than the simpler ones. The CNN has the lowest error on both tasks, though the difference in terms of $MAE_t$ with the MLP is small. In terms of KR prediction, on the other hand, there is a fair performance improvement of the CNN in terms of both metrics.

## V. FACET GENERATION

The main goal of this work is the generation of facets of games: specifically, parts of the level design and rules facet [1]. Following a search-based PCG approach [8], an Evolutionary Algorithm (EA) modifies a homogeneous initial population which is based on human designs. The surrogate model of Section IV is used to evaluate the evolving population, offering a prediction of the gameplay outcomes without computationally heavy simulations. While this paper tests three generators (one for character classes, one for levels, and one for levels and classes combined), the underlying EA is the same. The details of the EA, including a single-objective and a multi-objective variant, can be found in Section V-A; the representation and genetic operators for each facet generator are in Sections V-B and V-C and the orchestration process for evolving both levels and classes is in Section V-D.

### A. Evolutionary Algorithms

Following the search-based PCG paradigm [8], every EA in this paper consists of a population of individuals and assigns a fitness score to each individual; the best individuals based on this fitness score are preferred for mutation and recombination, and their offspring replace the least fit individuals of the previous population. Particularly for this problem, fitness is assigned based on the output of the surrogate model of Section IV, using the parameters of the evolving individual as input. The goal of every EA in this paper is to adjust the initial designs (levels, classes, or both) to bring them closer to a designer-specified target gameplay outcome.

The simplest way to measure proximity to a target gameplay outcome is to aggregate the distance between the outcomes of the current individual and each target value. For the single-objective evolutionary algorithm (SO-EA), the goal is to minimize the mean squared error between the model's predictions for individual $m$ and the desired outcomes:

$$F(\boldsymbol{m}) = \frac{1}{2}\big((p_t(\boldsymbol{m}) - d_t)^2 + (p_{KR}(\boldsymbol{m}) - d_{KR})^2\big) \quad (1)$$

where $\boldsymbol{m}$ is the individual being evaluated (i.e. a vector of class and/or level parameters); $\boldsymbol{p}(\boldsymbol{m}) = \{p_{KR}, p_t\}$ is the two-dimensional vector of gameplay outcomes as predicted by the surrogate model, i.e. predicted kill ratio ($p_{KR}$) and predicted duration ($p_t$); $\boldsymbol{d} = \{d_{KR}, d_t\}$ is the two-dimensional vector of the desired gameplay outcomes specified by the designer, i.e. desired kill ratio ($d_{KR}$) and desired duration ($d_t$).

SO-EA experiments in this paper employ a generational evolutionary algorithm with a population size of 100. The initial population consists of copies of a single initial seed provided by the designer. In each generation, the fittest 10% of the population is copied from the previous generation unchanged (applying elitism), while the remaining 90% is chosen via tournament selection of size 5, and then recombined and mutated. Each gene of each individual has a 20% chance of being mutated; each pair of individuals has a 80% chance of producing offspring via recombination. These values were chosen based on a grid search of at least 100 runs with each generator, based on overall best performance across facets. The mutation and recombination operators depends on the content generated and will be described in Sections V-B, V-C and V-D.

The fitness function described in Eq. (1) treats both gameplay outcomes as equal and attempts to optimize them simultaneously. There is no guarantee that the two gameplay outcomes are not conflicting, however: for instance, longer matches may be less balanced (assuming long duration and balanced kill ratio are the target outcomes). In order to test the effect of combining both gameplay outcomes into a single target, this paper compares the described single-objective algorithm with a multi-objective evolutionary algorithm (MO-EA) that aims to minimize the two components of Eq. (1) separately:

$$F_{KR}(\boldsymbol{m}) = (p_{KR}(\boldsymbol{m}) - d_{KR})^2 \quad (2)$$

$$F_t(\boldsymbol{m}) = (p_t(\boldsymbol{m}) - d_t)^2 \quad (3)$$

The MO-EA attempts to minimize both objectives of Eq. (2) and (3), using the popular NSGA-II algorithm [35] for creating the next generation. NSGA-II maintains a diverse Pareto front of non-dominated individuals and performs a binary tournament selection based on non-domination ranking and distance to other individuals. The other operators and hyperparameters (e.g. population size) of the MO-EA are the same as the SO-EA described above.

### B. Generating Classes

The character class generator operates on the sixteen parameters (eight per player class) described in Section III-B. The parameters are represented as floating values between 0 and 1, based on the same predetermined value range as in the dataset construction of Section IV-A. Each parameter has an equal chance of being mutated. Except for weapon range, class parameters are mutated by adding a random number sampled from a normal distribution ($\mu = 0$, $\sigma = 0.1$). The mutation of the weapon range randomly changes it into one of the other two possible values (long, medium, short). Values above 1 and below 0 are truncated to 1 and 0 respectively. Recombination is implemented as a standard one-point crossover.

### C. Generating Levels

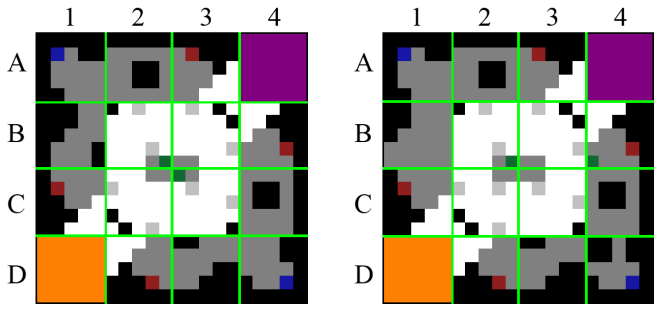
Following the methodology presented in [6], the level generator operates on a hierarchical representation of two layers. While the level consists of $20 \times 20$ tiles, the gene represents this as a $4 \times 4$ grid of cells that each contain $5 \times 5$ tiles (see Fig. 4). Recombination is implemented by randomly picking a cell from either parent at each position of the cell grid.

When applying mutation, each cell has a 20% chance of being mutated by one of the following variants: *Move Powerup* to another cell, *Grow Cell* or *Erode Cell* (in terms of either first-floor tiles or walls), *Place Stairs* next to a random first-floor tile, *Place Block* or *Dig Hole* (to add 'chunks' of tiles of the same elevation). A mutation example is shown in Fig. 4; see more details of the mutation operators in [6]. If a mutation is not applicable, (e.g. if a cell does not contain powerups for the first variant), the algorithm tries another mutation variant until a mutation is applied or all variants have been tested.

After mutation and recombination, a repair function ensures that every level is well-formed. Each level is analyzed in terms of traversability, enforcing the following constraints: (a) bases should always be reachable via ground-floor tiles, (b) each powerup must be reachable, (c) each first-floor tile should be connected to at least one stair, (d) there should be no holes in an area of first-floor tiles without a stair to climb out of the hole and (e) a stair should always lead to the first floor. A naive constructive algorithm repairs unreachable areas and places or removes stairs (without changing players' bases). If repair is impossible, the individual receives a fitness penalty.

### D. Orchestration Class and Level Generation

When generating both facets at the same time, the classes and the level are treated as two components of one individual's gene. Mutation has a 50% chance of applying the respective mutation operations to either the class or the level component. Recombination is implemented by swapping the level component between the two parents, resulting in two offspring with the classes of one parent and the level of the other parent.




(a) Parent (b) Offspring

Fig. 4. An example of a level and its offspring when only mutation is applied. The levels are broken into cells: in cell B1 the *Erode Cell* mutation removed some walls; in C2 the *Place Stairs* mutation added a stair; in C3 the *Move Powerup* mutation moved a double damage powerup to B4; in D4 the *Place Block* mutation has added a $3 \times 3$ 'chunk' of walls.

## VI. Case Study

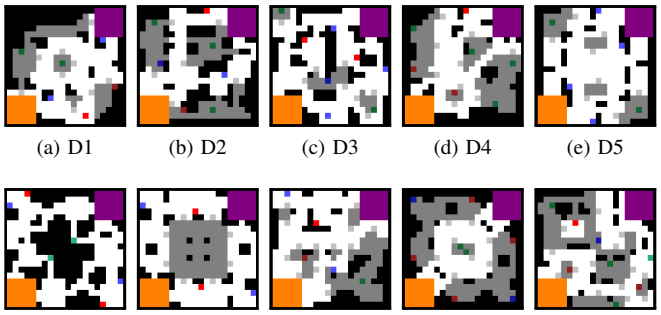
As a case study, this paper explores how a match between two archetypical classes can be balanced towards desired gameplay targets. This case study involves three experiments: adjusting a level given two character classes, adjusting two character classes given a specific level and adjusting both the level and the character classes at the same time.

All experiments in this paper have three desired gameplay outcomes: balanced matches (i.e. desired kill ratio $d_{KR} = 0.5$) of short, medium and long duration. The exact values for the desired durations ($d_t$) are respectively $0.14$, $0.36$, and $0.72$. These values are calculated from the midpoints of the bins in Fig. 2a and shown as colored diamonds in Fig. 6.

### A. Initial Seeds

Ten levels were designed to test how classes can be generated to match them or how the levels can be adapted to match specified or generated classes. The levels of Fig. 5 intentionally feature explicit level patterns such as arenas, choke points and flanking routes [36], and a varying degree of symmetry.

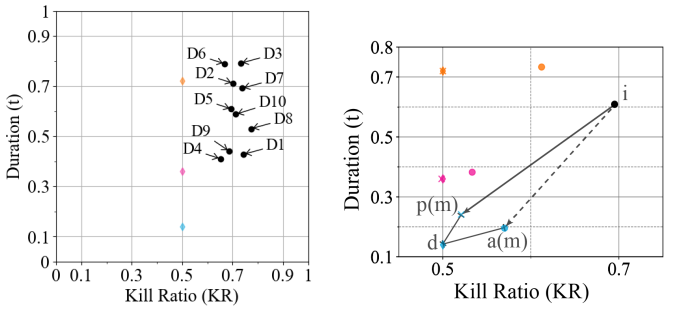
A number of archetypical classes [37] were considered, and their parameters were adapted to our framework based on the parameters in Team Fortress 2. We refer to these initial classes as TF2 classes in this paper. A preliminary experiment was performed on the ten levels of Fig. 5 to derive the simulation-based gameplay outcomes in pairwise matchups of the Scout, Heavy and Sniper classes. While the Sniper class dominated every opponent in every level (the kill ratio of the Sniper was over 0.90), the matchups between Scout for player 1 and Heavy for player 2 were imbalanced but diverse. Fig 6a shows the different kill ratios and match durations per level based on 30 simulated playthroughs. Fig 6a shows that while the Scout always has an advantage against the heavy class, this differs from level to level. Moreover, the match durations were almost evenly distributed between medium (40%) and long durations (60%). This pairing was selected for further



(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

(f) D6 (g) D7 (h) D8 (i) D9 (j) D10

Fig. 5. Levels used for evaluating evolved classes and as initial seeds for level generation.




(a) All initial seeds (b) Improvement via evolution

Fig. 6. Initial state of the game design of Scout vs Heavy in terms of the gameplay dimensions, and evolution towards short matches. The black dots indicate the different levels, the diamonds indicate the desired gameplay values for short (blue), medium (magenta) and long (orange) matches.

experiments since it was evident that changing the level itself could influence the player balance (unlike with the Sniper class), while the lack of short matches raised a challenge for the generator to tackle.

### B. Performance Metrics

The main goal of these experiments is to identify whether the surrogate-based facet generators can bring the different game elements closer to the intended gameplay outcomes. Towards this, the *improvement* metric in Eq. (4) calculates the actual gameplay outcomes of the evolved content and compares it to the actual gameplay outcomes of the original content (see Fig. 6b). The actual gameplay outcomes are calculated from 30 simulated playthroughs with AI agents. A second goal is to observe whether the predictive model matches the actual gameplay outcomes of evolved content. Towards this, the *error* metric in Eq. (5) compares the output of the surrogate model (which is used as part of the fitness calculation) with the actual gameplay outcome from 30 simulated playthroughs.

$$O(\boldsymbol{m}) = \frac{dist(\boldsymbol{d}, \boldsymbol{a}(\boldsymbol{m})) - dist(\boldsymbol{d}, \boldsymbol{i})}{dist(\boldsymbol{d}, \boldsymbol{i})} \quad (4)$$

$$E(\boldsymbol{m}) = dist(\boldsymbol{p}(\boldsymbol{m}), \boldsymbol{a}(\boldsymbol{m})) \quad (5)$$

where $dist(\boldsymbol{x}, \boldsymbol{y})$ is the Euclidean distance between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$; $\boldsymbol{a}(\boldsymbol{m}) = \{a_{KR}, a_t\}$ is the two-dimensional vector of actual gameplay outcomes averaged from 30 simulated playthroughs, i.e. actual kill ratio ($a_{KR}$) and actual duration ($a_t$); $\boldsymbol{i} = \{i_{KR}, i_t\}$ is the vector of actual gameplay outcomes of the initial class/level set, i.e. initial kill ratio ($i_{KR}$) and initial duration ($i_t$); other notations are described in Eq. (1).

Finally, it is relevant to understand how the content is adapted in different situations. To do this, we calculate the degree in which different parameters of the content change. This can include changes in class parameters (i.e. the genotype), changes in level appearance (i.e. pixel to pixel difference between levels) or level statistics (e.g. the number of healthpacks or stairs).

For single-objective evolution, the fittest individual in the end of the evolutionary process is tested in simulated playthroughs. Since MO-EA approaches produce a large set of Pareto-optimal individuals, in the interest of computational effort —and fairness in comparisons— only the best individual is tested based on the single-objective fitness of Eq. (1).

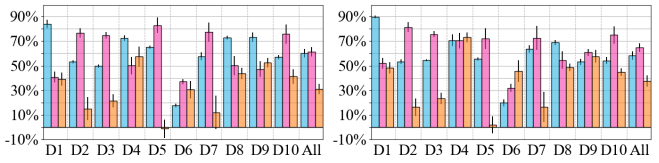
Throughout the experiments, significant differences are established on a statistical significance threshold $\alpha = 0.05$.

### C. Balancing Character Classes

As a first experiment, the surrogate-based generator adjusts the parameters of the two classes (Scout and Heavy) in all ten levels of Fig. 5. The evolved classes are collected from ten evolutionary runs per level and target gameplay outcomes (i.e. 300 evolved classes per EA); the actual gameplay outcomes of these classes are calculated from 30 simulated playthroughs.

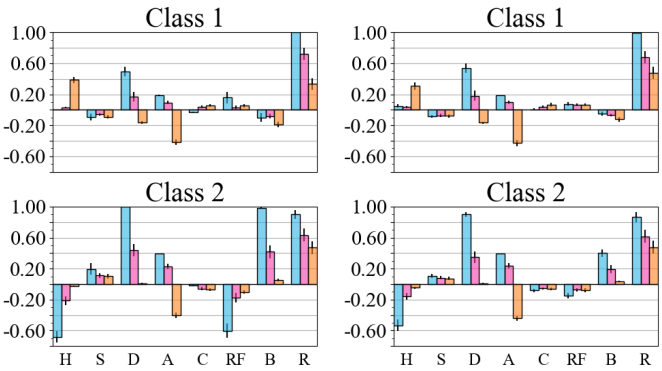
Figure 7 shows the percentage improvements over the initial classes on a per level basis and across all levels. It is evident that the evolved classes perform closer to the desired target in general, although the improvement is significantly higher in short or medium match targets than in long match targets for both SO-EA and MO-EA. On the one hand, for D4 the improvement is consistently high for all targets (over 50% for SO-EA and over 70% for MO-EA). On the other hand, long matches for D5, D2 and D7 are rarely improved in both SO-EA and MO-EA. For D2 and D7 the initial classes' match duration was very close to the desired long duration (see Fig. 6a) so the lack of improvement is to a degree expected. D5 has similar gameplay outcomes to D10 (see Fig. 6a), but class evolution in D10 manages to improve by at least 40% on long matches.

To identify how the classes are adjusted towards different targets, Fig. 8 shows the difference from the original class parameters. averaged across all ten levels. Trends seem to be consistent across SO-EA and MO-EA, while there are some interesting differences in how each class changes. For instance, in all cases both the Heavy and the Scout (which are short-range classes) see an increase in range which is however less pronounced in long matches. Similarly for both classes the accuracy increases in short matches and drops in long matches. Some of the parameter changes make sense considering the initial values for the two classes: for instance, the Heavy class has the most HP and lowest damage per bullet of all TF2 classes and thus it is understandable that in short matches its HP is reduced substantially (to reduce survivability) while damage is increased (to increase the threat to the opponent). These trends are consistent: the Scout class




(a) SO-EA (b) MO-EA

Fig. 7. **Class Adaptation:** Average improvement towards the desired kill ratio and duration of evolved classes, from 10 evolutionary runs per level of Fig. 5. The last column shows average improvement of all 100 runs per target duration; error bars show the 95% confidence interval.




(a) SO-EA (b) MO-EA

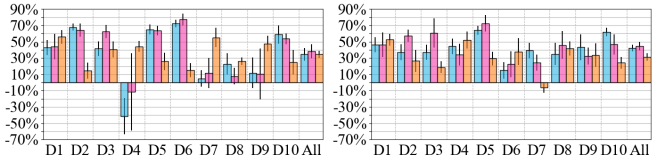
Fig. 8. **Class Adaptation:** Parameter changes of the two classes evolved for balanced matches of short, medium and long duration. Values show the difference of each parameter of the evolved class compared to the initial class (Scout for P1, Heavy for P2): hit points (H), speed (S), damage (D), accuracy (A), clip size (C), rate of fire (RF), bullets per shot (B) and range (R). Values are averaged from 100 runs; error bars show the 95% confidence interval.

(which had few HP to begin with) increases its HP in long matches to increase survivability. All changes are therefore intuitive, with long matches favoring classes with many hit points and low accuracy while short matches favor high-damage, long-range classes.

### *D. Balancing a Level*

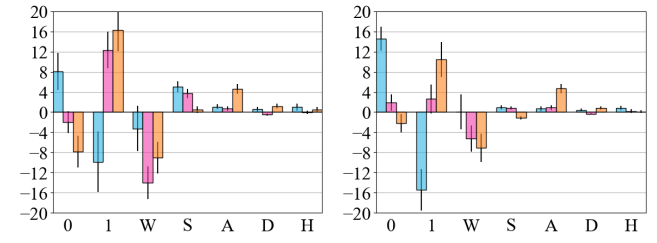
For the second experiment, the surrogate-based generator is used to evaluate an evolving level while the two classes competing in it remain the same (i.e. the Scout and Heavy TF2 classes). The initial seed for each experiment is one of the levels of Fig. 5, and the desired gameplay target is a balanced match of one of the three durations (for a total of 30 experiments of 10 evolutionary runs each). This results in 300 fittest levels for SO-EA and 300 fittest levels for MO-EA.

Figure 9 shows the percentage improvements over the initial levels. Unlike improvements in evolved classes in Fig. 7, patterns are much less consistent and in some cases the evolved levels are worse than the initial ones in terms of adherence to desired outcomes. Moreover, there are noticeable differences between SO-EA and MO-EA outcomes: where levels evolved from D4 and D9 are worse or equal to the initial one for SO-EA (with short or medium match targets), there are consistent improvements in all durations with MO-EA, except D7 for long matches. With D7 as an initial seed, improvements are negligible for short and medium match targets for SO-EA while the opposite is true for MO-EA. Levels evolved for short




(a) SO-EA (b) MO-EA

Fig. 9. **Level Adaptation:** Average improvement towards the desired kill ratio and duration of evolved classes, from 10 evolutionary runs per level of Fig. 5. The last column shows average improvement of all 100 runs per target duration; error bars show the 95% confidence interval.



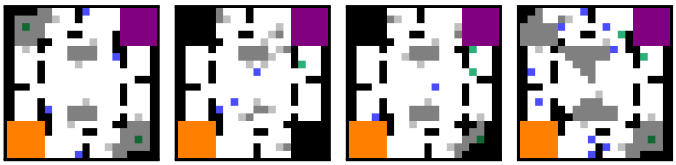

(a) SO-EA (b) MO-EA

Fig. 10. **Level Adaptation:** Changes in maps evolved for balanced matches of short, medium and long duration. Values show the difference in the number of tiles of the evolved levels compared to the initial level used as a seed: ground-floor tiles (0), first-floor tiles (1), walls (W), stairs (S), armor powerups (A), double damage powerups (D) and healthpacks (H). Values are averaged from 100 runs; error bars show the 95% confidence interval.

and medium match targets improve greatly over the initial D6 when evolved with SO-EA, but only slightly with MO-EA. Based on the overall improvements, evolving with MO-EA improves the levels more than SO-EA for all target durations, though the difference between the results on the long match targets are small.

To understand why the two EA approaches differ in terms of outcomes, Fig. 10 shows how the number of different tiles (e.g. healthpacks) change from the initial level, averaged across all ten initial seeds. It is evident that in both SO-EA and MO-EA the number of ground-floor tiles increases when targeting short matches, to the expense of first-floor tiles. Medium and long matches see an increase in first-floor tiles to the expense (primarily) of walls. This results in larger levels in terms of places that can be visited by the players. Interestingly, for SO-EA the smaller number of first-floor tiles in short matches does not result in a decrease in the number of stairs but rather the opposite. The reason will be evident in the next paragraph. Finally, while the number of healthpacks and double damage powerups per level does not fluctuate on average from the initial values, the number of armor powerups increases substantially when evolving levels for longer matches. Similar to the class changes in Fig. 8, where classes tailored to longer matches had more hit points (HP), armor powerups artificially increase players' HP and are favored for long matches to increase players' survivability.

In order to better understand how levels can be adjusted towards balanced matchups of different durations, the most improved levels across target durations are shown in Fig. 11 and 12. We observe that for short matches (Fig. 11b and 12b)



(a) Initial $\boldsymbol{i} = (0.70, 0.61)$ (b) Short $\boldsymbol{a} = (0.57, 0.20)$ $O = 82\%$ (c) Medium $\boldsymbol{a} = (0.53, 0.38)$ $O = 87\%$ (d) Long $\boldsymbol{a} = (0.61, 0.73)$ $O = 50\%$

Fig. 11. **Level Adaptation:** Most improved levels for D5, via SO-EA.

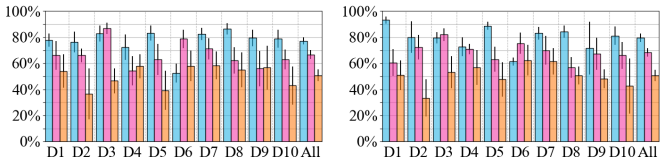
(a) Initial $\boldsymbol{i} = (0.65, 0.41)$ (b) Short $\boldsymbol{a} = (0.55, 0.19)$ $O = 77\%$ (c) Medium $\boldsymbol{a} = (0.52, 0.38)$ $O = 82\%$ (d) Long $\boldsymbol{a} = (0.53, 0.89)$ $O = 89\%$

Fig. 12. **Level Adaptation:** Most improved levels for D4, via MO-EA.

many of the first-floor areas are transformed into walls (top left and bottom right corners of Fig. 11b) or into ground-floor tiles (middle 'chunk' of Fig. 12b). However, there are still several small 'chunks' of first-floor tiles (of one or two tiles) which need to be connected via one or more stairs due to the constraints described in Section V-C; this explains the prevalence of stairs even when first-floor tiles are few as observed in Fig. 10. In terms of broader level structures, it is evident that levels adjusted towards short matches are sparser while levels evolved towards long durations are denser, with more complex paths between the two bases. Compare, for example, the straight diagonal line connecting the two bases in Fig. 12b with the two long paths in Fig. 12d, one of which requires passing through a large chunk of first-floor tiles. In terms of gameplay objects, we observe more double damage powerups in short matches (e.g. in Fig. 12b) and more armor powerups in long matches Fig. 11d). Level generation also attempts to correct for the imbalance between the classes: the Heavy class that spawns in the purple base has better access to powerups (e.g. double damage powerups in Fig. 12b, 12c and 12d) than the Scout in the orange base. The kill ratio of the Scout was much higher than the Heavy in the initial levels (e.g. $i_{KR} = 0.70$ in Fig. 11a), but this imbalanced distribution of powerups near one base improves the balance between classes.

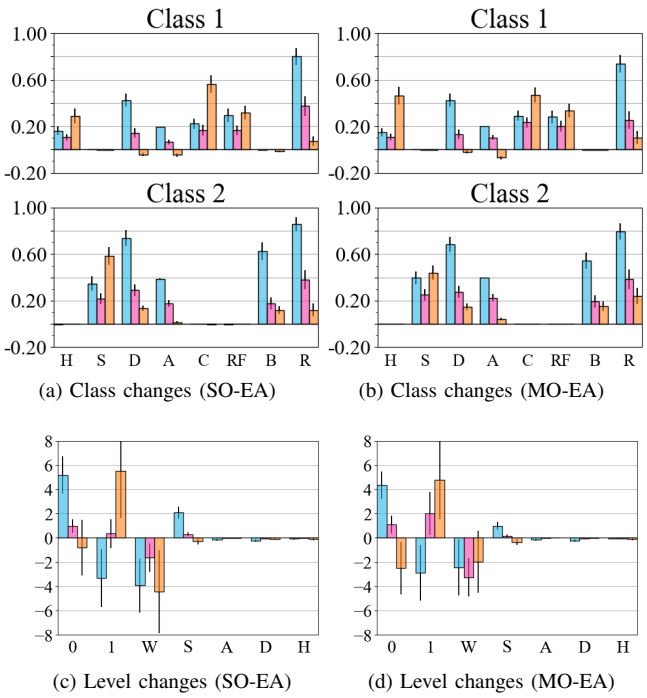
### *E. Balanced Matches via Rule and Level Orchestration*

A core question posed in this paper is how the generation of both facets (rules and levels) can be orchestrated via the surrogate model that maps them to gameplay outcomes. In this set of experiments the generative process adjusts both the initial level and the initial classes that compete in it. As in previous experiments, 10 evolutionary runs attempt to adjust an initial population of Scout and Heavy matchups in the same level, towards a desired gameplay outcome. With 10 initial




(a) SO-EA (b) MO-EA

Fig. 13. **Orchestration:** Average improvement towards the desired kill ratio and duration of evolved classes, from 10 evolutionary runs per level of Fig. 5. The last column shows average improvement of all 100 runs per target duration; error bars show the 95% confidence interval.




(a) Class changes (SO-EA) (b) Class changes (MO-EA)

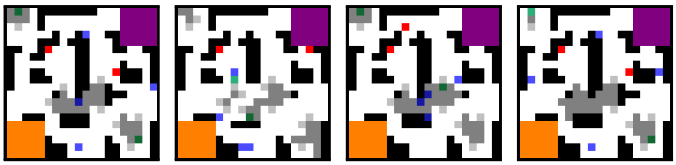
(c) Level changes (SO-EA) (d) Level changes (MO-EA)

Fig. 14. **Orchestration:** Changes in the two classes' parameters and in the levels evolved for balanced matches of short, medium and long duration. Values show the difference of each parameter of the evolved class compared to the initial class (Scout for P1, Heavy for P2) and the initial level used as a seed. Notations are the same as Fig. 8 and Fig. 10. Values are averaged from 100 runs; error bars show the 95% confidence interval.

levels and three different target gameplay outcomes, a total of 300 evolved levels and classes are collected per EA.

Figure 13 shows the improvement over the initial setup. It is evident that on average the improvements are higher when targeting short matches, in part because the initial matchups had longer durations (see Fig. 6a). This behavior is consistent with class generation experiments summarized in Fig. 7, although the improvements themselves are significantly higher when both levels and classes are adjusted, on average.

On the degree to which levels and classes are adjusted, Fig. 14 shows the parameter changes from the initial seeds for class parameters and levels' tile counts. It is clear that there are few substantial differences between SO-EA and MO-EA in terms of parameter changes or level changes. It is also evident that changes in terms of levels are much more subdued than in level-only generation (i.e. Fig. 10), although similar patterns prevail (more ground-floor tiles and stairs for short



(a) Initial $\boldsymbol{i} = (0.73, 0.79)$ (b) Short $\boldsymbol{a} = (0.51, 0.16)$ $O = 97\%$ (c) Medium $\boldsymbol{a} = (0.53, 0.34)$ $O = 93\%$ (d) Long $\boldsymbol{a} = (0.58, 0.75)$ $O = 64\%$

Fig. 15. **Orchestration:** Most improved levels for D3, evolved via SO-EA.

(a) Initial $\boldsymbol{i} = (0.74, 0.43)$ (b) Short $\boldsymbol{a} = (0.51, 0.14)$ $O = 98\%$ (c) Medium $\boldsymbol{a} = (0.52, 0.36)$ $O = 90\%$ (d) Long $\boldsymbol{a} = (0.52, 0.76)$ $O = 89\%$

Fig. 16. **Orchestration:** Most improved levels for D1, evolved via MO-EA.

matches, more first-floor tiles for long matches). However, the number of powerups does not change from its initial values in most cases: indeed, none of the three powerup counts changed in 79% of SO-EA experiments. The more subdued level adjustments are in part due to the crossover operators for the orchestrated approach which differ from level generation. Classes similarly do not change as much as when evolving on their own (i.e. Fig. 8). While there are no negative corrections (e.g. lowered HP for the Heavy class in Fig. 8), some familiar patterns remain, such as increased damage and range for short matches. Notably, four class parameters of 16 stay unchanged in at least 97% of SO-EA runs.

As indicative examples of how orchestrated evolution can improve matchups towards the intended gameplay outcomes, Fig. 15 shows the SO-EA results with the highest improvements for D3 and Fig. 15 shows the MO-EA results with the highest improvements for D1. D3 and D1 were chosen as they have the highest average improvements across all targets for SO-EA and MO-EA respectively. While familiar trends as in Section VI-D are observed in short matches (e.g. first-floor chunks made up of one tile and multiple stairs), it is also evident that changes in the level architecture are less pronounced and powerup placement does not particularly favor one player's base. Most of tweaks therefore are performed on the classes rather than on the level: indeed, both classes' damage is given a boost when targeting all durations (but especially for the short duration). Unlike the findings in Section VI-C, for short matches neither class evolves to match the Sniper archetype in TF2.

### *F. Summary*

Table II summarizes how the different experiments compare in terms of the distance between the desired ($\boldsymbol{d}$), the actual simulation-based ($\boldsymbol{a}$) and the predicted ($\boldsymbol{p}$) gameplay outcomes. The prediction error (i.e. the distance between predicted and actual outcomes) is significantly lower when both

TABLE II
BEHAVIORAL DIFFERENCES FROM THE ORIGINAL AND THE PREDICTED. RESULTS ARE AVERAGED FROM 300 EVOLUTIONARY RUNS ACROSS THE 10 LEVELS, AND THE 95% CONFIDENCE INTERVALS IS INCLUDED.

| Generator | $dist(\boldsymbol{a}, \boldsymbol{p})$ | $dist(\boldsymbol{a}, \boldsymbol{d})$ | $dist(\boldsymbol{p}, \boldsymbol{d})$ |
|---|---|---|---|
| SO-EA class | 0.120 ± 0.008 | 0.174 ± 0.012 | 0.063 ± 0.011 |
| MO-EA class | 0.117 ± 0.007 | 0.168 ± 0.012 | 0.066 ± 0.012 |
| SO-EA level | 0.182 ± 0.013 | 0.214 ± 0.015 | 0.043 ± 0.007 |
| MO-EA level | 0.167 ± 0.010 | 0.215 ± 0.015 | 0.055 ± 0.009 |
| SO-EA both | 0.107 ± 0.007 | 0.114 ± 0.008 | 0.012 ± 0.003 |
| MO-EA both | 0.102 ± 0.007 | 0.112 ± 0.008 | 0.020 ± 0.005 |

TABLE III
PHENOTYPICAL DISTANCE FROM THE INITIAL CLASSES (C1 FOR SCOUT AND C2 FOR HEAVY) AND THE INITIAL MAP. RESULTS ARE AVERAGED FROM 300 EVOLUTIONARY RUNS ACROSS THE 10 MAPS, AND THE 95% CONFIDENCE INTERVALS IS INCLUDED.

| Generator | C1 distance | C2 distance | tile difference |
|---|---|---|---|
| SO-EA class | 1.000 ± 0.042 | 1.319 ± 0.083 | — |
| MO-EA class | 0.978 ± 0.040 | 1.102 ± 0.058 | — |
| SO-EA map | — | — | 74.3 ± 3.5 |
| MO-EA map | — | — | 52.5 ± 2.8 |
| SO-EA both | 0.733 ± 0.043 | 0.865 ± 0.052 | 34.0 ± 3.1 |
| MO-EA both | 0.761 ± 0.042 | 0.825 ± 0.048 | 30.1 ± 2.5 |

level and classes are adjusted via MO-EA, while the highest errors are when only levels are adjusted. This is also true for the distance between desired and actual outcomes, which is closely related to the improvement metric of Eq. (4). Indeed, the average $dist(\boldsymbol{a}, \boldsymbol{d})$ is significantly lower (and the average improvement significantly higher) when both level and classes are adjusted. On the one hand, this is not surprising since the generator has multiple degrees of freedom and can counteract imbalances between classes by tweaking class parameters but also through level re-design. On the other hand, it is also evident that the predictive model can recognize and match tweaks in both facets with a fairly accurate estimate of the actual gameplay outcomes. The latter is an important finding which shows the benefit of orchestration both as a generative medium but also as a surrogate for gameplay simulations. Regarding the surrogate model itself, the $dist(\boldsymbol{p}, \boldsymbol{d})$ metric shows that the predictive model is understandably more "optimistic" than the actual simulations demonstrate, but still steers evolution in the right direction. An interesting observation is the low $dist(\boldsymbol{p}, \boldsymbol{d})$ for level adjustments which is not corroborated by the $dist(\boldsymbol{a}, \boldsymbol{d})$ metric: it seems that the large size of the level input and the complex nature of its representation is more likely to lead level evolution astray than other facets.

Table III compares the initial classes and levels with the final best outcomes in the different generative approaches. Unlike tile counts in Sections VI-D and VI-E, the tile difference metric is the pixel to pixel image difference[3] which is more granular. It is evident that orchestration makes fewer changes to either class or to the level, which is corroborated by Fig. 14.

[3]Pixel to pixel difference using the representations of e.g. Fig. 5 directly reflects the tiles on that specific location, including whether a healthpack is on the ground or the first floor.

It is also interesting that changes to the Heavy class are more prevalent for SO-EA approaches than changes to the Scout class. Meanwhile, level adjustments are smaller for MO-EA approaches (significantly so when evolving levels alone).

Generally, the differences between the single-objective approach and the multi-objective approach are surprisingly small: MO-EA is slightly ahead in terms of average improvement but the differences are not significant. The highest average improvement across all initial seeds and target durations is with MO-EA on both levels and classes ($O = 66.4\% \pm 2.5\%$). The real benefit of MO-EA is that the improvements (however marginal) over SO-EA are made with fewer changes to the original levels and/or classes.

## VII. Discussion

The case study in Section VI investigated how a specific matchup between archetypical character classes in shooter games (i.e. a fast short-range Scout class and a slow, inaccurate Heavy class) can be tailored to a more balanced gameplay outcome of a certain match duration. The experiments demonstrated that tweaks on both the character class parameters and the level can lead to a closer outcome to the desired one, while attempting to tweak character classes alone was prone to major design shifts (e.g. turning every class into a Sniper class for short matches). Moreover, it is evident that the improvements as a result of evolution and the accuracy of the surrogate model both vary depending on initial level and target duration. Further enhancements can thus be made on both fronts.

Despite a robust set of experiments with a total of 1800 evolutionary runs in Section VI, there are many possibilities for extending this work. An obvious limitation is that only one class matchup was used as an initial seed: evidently, a different matchup could be more difficult to adjust via evolution or could lead to less accurate predictions from the surrogate model. Indicatively, the Sniper class has over 90% kill ratio against all TF2 classes in all levels of Fig. 5; thus, it is expected that evolving levels alone would not suffice to balance matchups against the Sniper class. Another limitation is that only two gameplay outcomes are quantified, learned and targeted through surrogate-based evolutionary search. More nuanced gameplay metrics such as the entropy of players' movement could be used as alternative targets for evolution; combined with kill ratio balance, these metrics could push evolution towards balanced matchups in more than one way.

In terms of the evolved character classes and levels presented in Section VI, there is room for improvement. On the one hand, the evolved classes are often very different from the original ones, especially when evolving classes alone. This could be avoided in future work by having a secondary objective for minimizing distance from the inital seeds (see Table III for sample similarity metrics for classes and levels), as in the work of [38]. In terms of the evolved levels, it is clear that some architectural formations seem too chaotic, while the one-tile 'chunks' of first floor tiles serve little purpose and are visually jarring. Enhancements to the repair operators for evolved levels could improve their appearance (e.g. with cellular automata using a von Neumann neighborhood [30] to make corners and remove the smaller 'chunks') with the caveat of a more disparate mapping from genotype and phenotype.

For broader future work in terms of facet orchestration, alternative ways of generating levels and classes could be devised. The current generative approach highlighted in Section V-D evolves classes and levels simultaneously by combining them in the same genotype, similar to boards and rules in *Ludi* [14]. On the other hand, a generative pipeline [1] could start by adapting a level first and then adapt the classes for the best evolved level. Alternatively, the two facets could be decoupled in a bottom-up orchestration approach similar to a blackboard system [1] where classes could be generated separately, levels could be generated separately, and then the best combination among all candidate classes and levels (based on the model) would be chosen either to evolve further or as final result.

While MO-EA was introduced in this paper under the assumption that the two gameplay targets could be conflicting, the experiments in Section VI only evaluated the best individual of the Pareto front based on the fitness of Eq. (1) which combines the two targets. However, the Pareto front with distances to each gameplay target could be interesting to game designers working with a mixed-initiative interface [3]. Through this Pareto front visualization, designers could determine the best tradeoff between balance or desired duration based on their own needs. While MO-EA was in general marginally better than SO-EA (see Section VI-F), the potential for designer intervention is another point in its favor.

Despite this paper's focus on search-based PCG with a surrogate model, the general mapping between game facets and gameplay outcomes could be exploited in a generative adversarial network (GAN) setup. Following the PCGML paradigm [26], a conditional GAN could be trained to output levels and classes based on a desired gameplay outcome provided as input. A network similar to the surrogate model in this paper could judge not only the resulting gameplay, but also whether generated output is good enough to pass as a real design. Such an architecture would likely resemble the Auxiliary Classifier GAN used for natural image synthesis [39].

As mentioned earlier, the surrogate-model evaluation of gameplay can be integrated into a designer tool for immediate feedback during the design of a new level or new classes. As an example, the tool can assess the Kill Ratio of every possible pairing of archetypical TF2 classes in a designer's level and suggest the best class matchup for it. Alternatively, the model can *explain* to the designer [40] which sections of the level most impact the duration prediction and thus should be changed [41]. Finally, inspired by [3], evolution can produce suggestions that improve predicted balance or duration for a level or its classes as the latter is being designed.

## VIII. Conclusions

This paper has proposed a model for mapping multiple game facets (rules, levels, and gameplay) and a generative approach that uses this mapping to adjust human designs towards certain desired gameplay outcomes. Focusing on shooter games, experiments showed that classes, levels and their combination can be adjusted towards more balanced matchups of a specific duration. Using artificial evolution guided by a single

objective that combined both gameplay targets or a multi-objective approach that decoupled them, it was evident that an orchestration of both classes and levels was more efficient at improving the human designs towards the desired outcomes. The use of surrogate models for evaluating multiple game facets opens up a broad range of future research directions in procedural content generation and AI-assisted design.

## IX. Acknowledgements

This project has received funding from the European Union's Horizon 2020 programme under grant agreement No 787476.

## References

[1] A. Liapis, G. N. Yannakakis, M. J. Nelson, M. Preuss, and R. Bidarra, "Orchestrating game generation," *IEEE Transactions on Games*, vol. 11, no. 1, pp. 48–68, 2019.

[2] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbck, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genetic Programming and Evolvable Machines*, 2013.

[3] G. N. Yannakakis, A. Liapis, and C. Alexopoulos, "Mixed-initiative co-creativity," in *Proceedings of the Foundations of Digital Games Conference*, 2014.

[4] D. Karavolos, A. Liapis, and G. N. Yannakakis, "Learning the patterns of balance in a multi-player shooter game," in *Proceedings of the FDG workshop on Procedural Content Generation*, 2017.

[5] ——, "Pairing character classes in a deathmatch shooter game via a deep-learning surrogate model," in *Proceedings of the FDG Workshop on Procedural Content Generation*, 2018.

[6] ——, "Using a surrogate model of gameplay for automated level design," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2018.

[7] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[8] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, 2011.

[9] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Interactive evolution for the procedural generation of tracks in a high-end racing game," *Interface*, pp. 395–402, 2011.

[10] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Automatic content generation in the galactic arms race video game," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 4, pp. 245–263, 2009.

[11] A. Liapis, G. N. Yannakakis, and J. Togelius, "Adapting models of visual aesthetics for personalized content creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 3, pp. 213–228, 2012.

[12] ——, "Designer modeling for personalized game content creation tools," in *Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics*, 2013.

[13] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[14] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[15] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," in *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013.

[16] N. Sorenson, P. Pasquier, and S. DiPaola, "A generic approach to challenge modeling for the procedural creation of video game levels," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 229–244, 2011.

[17] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011, pp. 289–296.

[18] P. L. Lanzi, D. Loiacono, and R. Stucchi, "Evolving maps for match balancing in first person shooters," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2014.

[19] D. Gravina, A. Liapis, and G. N. Yannakakis, "Constrained surprise search for content generation," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2016.

[20] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[21] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[23] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Transactions on Games*, 2019, accepted.

[24] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via LSTMs," in *Proceedings of DiGRA & FDG*, 2016.

[25] S. Lee, A. Isaksen, C. Holmgard, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks," in *Proceedings of the AIIDE Workshop on Experimental AI in Games*, 2015.

[26] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (PCGML)," *IEEE Transactions on Games*, vol. 10, no. 3, 2018.

[27] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018.

[28] Opsive, "Ultimate first person shooter framework (UFPS)," https://legacy.opsive.com/assets/UFPS/, 2012.

[29] ——, "Deathmatch AI kit documentation," https://legacy.opsive.com/assets/DeathmatchAIKit/, 2016, accessed: 2018-02-02.

[30] N. Shaker, A. Liapis, J. Togelius, R. Lopes, and R. Bidarra, "Constructive generation methods for dungeons and levels," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016, pp. 31–55.

[31] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *Proceedings of the International Conference on Learning Represenations*, 2016.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning*, 2015.

[33] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[34] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 5 2015.

[35] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[36] K. Hullet and J. Whitehead, "Design patterns in FPS levels," in *Proceedings of the Foundations of Digital Games Conference*, 2010.

[37] R. Giusti, K. Hullett, and J. Whitehead, "Weapon design patterns in shooter games," in *Proceedings of the FDG workshop on on Design Patterns in Games*, 2012.

[38] D. Gravina and D. Loiacono, "Procedural weapons generation for unreal tournament III," in *Proceedings of the IEEE Conference on Games, Entertainment, Media*, 2015.

[39] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *Proceedings of the International Conference on Machine Learning*, 2017.

[40] J. Zhu, A. Liapis, S. Risi, R. Bidarra, and G. M. Youngblood, "Explainable AI for designers: A human-centered perspective on mixed-initiative co-creation," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2018.

[41] A. Liapis, D. Karavolos, K. Makantasis, K. Sfikas, and G. N. Yannakakis, "Fusing level and ruleset features for multimodal learning of gameplay outcomes," in *Proceedings of the IEEE Conference on Games*, 2019.