Learning the Patterns of Balance in a Multi-Player Shooter Game

Daniel Karavolos
Institute of Digital Games
University of Malta
Msida, Malta
daniel.karavolos@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
Msida, Malta
antonios.liapis@um.edu.mt

Georgios Yannakakis
Institute of Digital Games
University of Malta
Msida, Malta
georgios.yannakakis@um.edu.mt

ABSTRACT

A particular challenge of the game design process is when the designer is requested to orchestrate dissimilar elements of games such as visuals, audio, narrative and rules to achieve a specific play experience. Within the domain of adversarial first person shooter games, for instance, a designer must be able to comprehend the differences between the weapons available in the game, and appropriately craft a game level to take advantage of strengths and weaknesses of those weapons. As an initial study towards computationally orchestrating dissimilar content generators in games, this paper presents a computational model which can classify a matchup of a team-based shooter game as balanced or as favoring one or the other team. The computational model uses convolutional neural networks to learn how game balance is affected by the level, represented as an image, and each team's weapon parameters. The model was trained on a corpus of over 50,000 simulated games with artificial agents on a diverse set of levels created by 39 different generators. The results show that the fusion of levels, when processed by a convolutional neural network, and weapon parameters yields an accuracy far above the baseline but also improves accuracy compared to artificial neural networks or models which use partial information, such as only the weapon or only the level as input.

CCS CONCEPTS

•Computing methodologies → Supervised learning by classification; •Applied computing → Computer games;

KEYWORDS

Automated Playtesting, Deep Learning, Shooter Games, Level Patterns, Procedural Content Generation, Game Balancing

ACM Reference format:
Daniel Karavolos, Antonios Liapis, and Georgios Yannakakis. 2017. Learning the Patterns of Balance in a Multi-Player Shooter Game. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 10 pages.
DOI: 10.1145/3102071.3110568

FDG'17, Hyannis, MA, USA

978-1-4503-5319-9/17/08…$15.00
DOI: 10.1145/3102071.3110568

1 INTRODUCTION

Despite its long history in the commercial game sector, procedural content generation (PCG) has only recently started to focus on assessing the quality of the artifacts it produces. With recent academic interest in PCG over the last 10 years, the issue of "sufficient" or "improved" quality of artifacts has become important, in part due to the need for academics to benchmark their problems and provide evidence of improvement. Quality can be assessed and ensured in many ways, e.g. via numerical evaluations of one or more game properties used as gradients for a search-based generator [32], via a set of constraints for what is an acceptable level in constraint-based PCG [29], or via a set of expressive and high-quality modular components that can be combined in grammar-based PCG [6]. Such attempts not only provide solid baselines to test PCG on, but also provide game design insight on how the design space for viable games is shaped by observing and generating edge cases [30].

Historically, the most common application of PCG in commercial games has been level or gameworld generation. This allows — purportedly— infinite replayability; it also allows the game rules (i.e. how a player can interact with the level, winning and losing conditions, etc.) to be carefully tuned by the design team and gradually learned by the player. Level generation is also the most researched type of PCG, although other attempts at generating or tweaking the game's ruleset are worthy of note. Rule generation can go as far as editing the source code of the underlying game engine [5], choosing from a preset list of e.g. movement or collision rules [31], or fine-tuning small parts of the game such as the parameters of weapons in a shooter game [11]. Whether generating a game level or tweaking a rule set, however, it is impossible to test one without the other. Taking shooter games as an example, generating a sniping weapon [9] with a long range and slow reload time for a game where all levels are narrow, curving corridors would give the weapon a clear disadvantage; an open plain would give it a clear advantage. Similarly, generating a shooter level where players can use sniper rifles affects the patterns that need to be included, e.g. sniper positions [12]. The architecture of a level, the patterns of weapon's parameters, but also other factors such as visuals or audio cues from the environment are orchestrated by human or computational designers to create the intended play experience [24], including the game outcomes in terms of challenge, playtime and balance. Therefore, when assessing such dimensions of game quality, it is necessary to consider more than one type of content: e.g. not only the rules of the game but also the layout of its levels.

This paper introduces a computational model which has learned to classify shooter game matches as balanced or as favoring one or the other team. The proposed approach takes advantage of the recent advances in deep learning to exploit patterns stored visually in the map of the game, and combine them with parameters of the

weapons used by each team to accurately predict a match's balance. In order to provide a large and diverse dataset for the model to learn from, over 50,000 simulated playthroughs by artificial agents were produced. In the test bed introduced in this study, teams of three artificial agents, using one weapon type per team, attempted to score more kills than the other team. The arena in which combat took place had large objects to block line of sight and small objects to take cover in. In order for the model to learn a broad set of level patterns, 39 generators were used to create a diverse set of levels in which the matches were simulated. Results show that while weapons had quite clear imbalances in power level which were easy to learn, combining those learned patterns with levels allowed the computational model to increase its accuracy above all other tested approaches. By fusing levels stored simply as an image and weapons stored numerically, the learned model can be used to generate new balanced levels for a specific weapon pair, to finetune weapons so that the weaker weapons have a higher chance of being useful, and to provide real-time feedback to human designers without the need for computationally heavy simulations.

2 RELATED WORK

This section describes related attempts at algorithmically assessing and improving a game's balance and provides a brief background on deep learning.

2.1 Game Balancing

Among the most important dimensions of a game's quality is that of *balance* — or symmetry as referred to in [1]. At its core, balance in a two-player adversarial game is when a player has an equal chance of winning against another player of the same skill level. Balance is often ensured via symmetrical rules and levels; for instance, in chess both players' pawns or rooks have the same movement rules, while the board itself and the placement of pawns is fully symmetrical. Several formulas have been proposed to assess balance in terms of gameplay, e.g. [25], or in terms of observable level patterns, e.g. [23]. When it comes to gameplay balance, it is imperative that the game is actually *played* in order to derive such balance metrics: while there are instances where human players are involved in this evaluation [4], the most common approach is to use simulations with artificial agents to create synthetic play traces. The playtraces can be processed in many ways, for instance measuring the changes in who is leading [2], an even ratio in terms of players' assets [18] etc. For the shooter genre in particular, a popular metric is the entropy of the kills distribution in a deathmatch game [17]; this metric has been used to evolve levels [3] as well as weapons [10, 11, 17] towards balanced artificial agents' match-ups. Simulations take up considerable computational resources, especially when using commercial games such as *Unreal Tournament III* (Epic Games 2007) which can only be sped up to a degree. A neural network that assesses the balance of a match through visual inspection can make balance evaluations in real-time and considerably speed up search-based generation of balanced game levels or weapon parameters.

2.2 Deep Learning

In the last few years, the domain of machine learning has received considerable attention due to the success of deep learning approaches. Deep learning originates from the process of training layers of a neural network sequentially, fitting their input into a progressively smaller representation and thus responding to increasingly higher level patterns. This process, called layer-wise pretraining, allowed for much deeper networks to be trained. Due to developments in the field and an increase in computing power, layer-wise pretraining is no longer necessary. However, the architectures of deep neural networks still exploit the property of finding higher level features by composing them of lower-level ones [19].

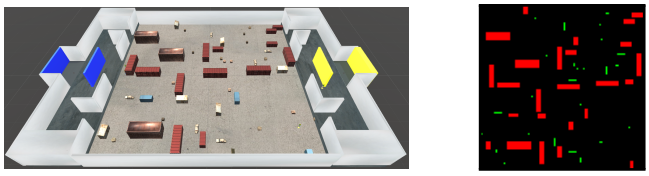
Deep learning has had significant success in representation learning and computer vision. Deep learning has also been favored in recently developed computationally creative software, as they can be used to combine the painting styles of different artists via style transfer [8, 26], to evolve sculptures by improving the classification error with a target physical object [20], or to design furniture [7].

In digital games, deep learning has been mostly focusing on artificial agent control, e.g. for playing the Go board game [28], different Atari games [27], and the Doom shooter game [14]; the latter two used the screen output as input to the deep network. The success of deep learning in computer vision has also been exploited for the visual identification of evolving spaceships [21], using the error of the deep learner as a measure of novelty for a computational designer. Finally, the ability of deep learning to find patterns of high-quality game levels was used to "auto-correct" miscreated platformer levels to become playable [13]. These early but significant successes of deep learning in games and beyond inspired us to use deep learning to visually inspect shooter levels in order to assess their balance.

3 THE DEATHMATCH TESTBED

This paper introduces a machine learning approach that can aid a computational designer in creating a balanced battle environment for a pair of weapons used by opposing teams. Specifically, this paper focuses on the genre of first person shooters and one of its most common contests: team deathmatch, i.e. trying to get more kills than the opposing team. In this study, the match consists of two teams of 3 players. Each player has unlimited lives, but a match ends after a total of 20 kills. The balance is measured by the ratio of kills of each team. A match is balanced when it resulted in a draw, or when difference in kills between the two teams was marginal.

For the purpose of this paper, an arena level was created in the *Unity 3D* game engine (Unity 2005). The deathmatch level consists of three areas, two spawn areas on opposite edges of the level and



(a) In-game view (b) Map

Figure 1: Example level, with team bases on the left (first team) and right (second team) of the arena, and the arena's map representation used as input for machine learning.

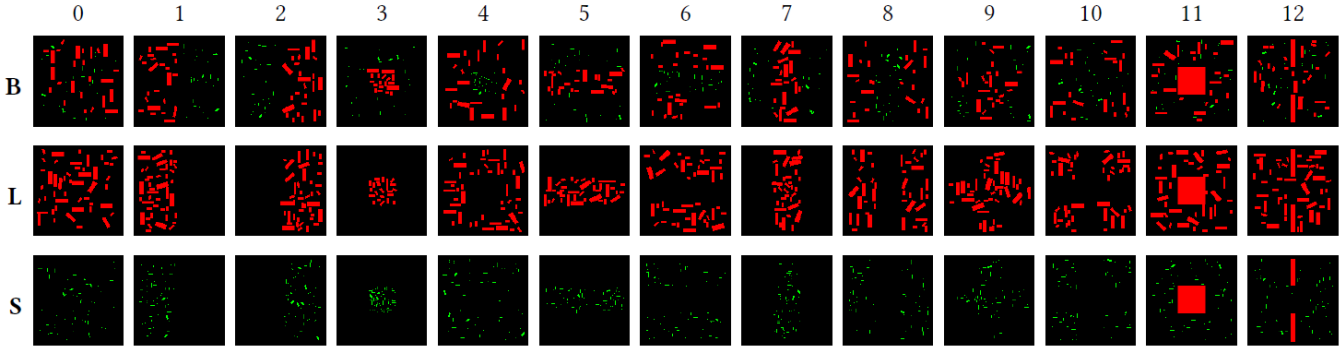


Figure 2: Example maps of each level generator, with both types of objects (B), only large objects (L) and only small objects (S).

a square *arena* in the middle, where the fighting will take place (see Fig. 1a). At the start of the game, two teams of 3 players start at their respective spawn areas, or *bases*. Every time a player is killed, they re-spawn at their team's base. Each base has three paths to the central arena, a ledge and an invisible wall that blocks players and projectiles in the arena from passing through, respectively; this prevents a team being pinned down inside their base. The central arena is a flat square area, bordered by walls, and the only area that is procedurally filled with objects. Such objects are a variety of 'block' objects found in modern warehouses, and provide cover from enemy fire. Two types of object are identified: small objects which come up to chest height and allow for players to shoot opponents while taking (some) cover from them, and large objects which can completely block line of fire and line of sight from enemies. Large objects and small objects have fundamentally different tactical properties, and so are represented differently in the system and handled differently by the artificial intelligence controlling the players of each team.

In order for the level to be used as input for machine learning, it is represented as a 100 by 100 pixel image (see Fig. 1b), ignoring the teams' bases as they are symmetrical and no combat occurs within them. Each pixel value of this representation determines whether that tile is occupied[1] by a large object (red pixel), a small object (green pixel) or is empty (black pixel). The chosen image resolution allows for even the smallest objects to be 'captured'.

3.1 Level Generation

To generate sufficient data for effective machine learning, a large number of diverse game levels must be created. For that purpose, 39 different generators were designed to create a broad range of levels. The generators primarily influence the spatial distribution of a set of large and small objects: due to concerns of objects colliding with each other, every generator places a total of 50 objects. Some generators place 50 large objects and no small objects (**L** row in Fig. 2), some place 50 small objects and no large objects (**S** row in Fig. 2), and some place 25 large and 25 small objects (**B** row in Fig. 2). The spatial distribution of objects, as shown in Fig. 2, includes placing objects (large or small) on one half of the level, in the center of the level, along a central row or column, or in the four corners of the level. As an addition, six generators feature a designer-defined level pattern which is not randomized: either a large impassable block in the center of the level (column 11 in Fig. 2), or a central choke point formed by two long walls (column 12 in Fig. 2). While generators determine the object's coordinates, rotation, and type, the complete level is stored as a 100 by 100 pixel image where each object usually takes up more than one pixel.

[1]Whether a tile is occupied by an object is determined by performing a downwards raycast at the center of the tile.

3.2 Playtest Data Collection

As a supervised learning approach, the model requires a set of training data to learn a model of balance. Ideally, this would be data collected from actual players but this is not feasible due to the volume of data needed for deep learning. Therefore, as an initial approach, we simulate player behavior with artificial agents.

The artificial agents are controlled by the *Shooter AI* plugin library (Squared55 2015); the artificial agents form two teams of three agents each. Each team starts at its respective 'base' (and agents re-spawn there if they die) and compete inside a generated arena as shown in Fig. 1. In terms of behavior, the agents wander around the map towards the enemy base. When they spot an opponent, they attempt to find cover and shoot the opponent from this cover position. If the opponent is not killed within a limited time frame (e.g. if the opponent is also in cover) the agent will try to move to another cover position. If no cover is found, agents switch to a chasing behavior, trying to kill the opponent from close range. Agents always aim for the head unless they carry a projectile weapon (which is aimed at the hips in order to take advantage of splash damage). For the sake of simplicity, all members of one team have the same weapon which has unlimited ammo; however, the weapon's clip size affects how many shots can be fired before the agents need to reload (reload time is one of the weapon parameters).

For experiments in this paper, five weapons were adapted from those of the *Shooter AI* library and tested: the shotgun, the rifle, the sub-machine gun (SMG), the sniper rifle and the rocket launcher. These weapons have very different patterns of use [12] and weapon characteristics, such as the high damage of the sniper rifle, the slow bullet speed of the rocket launcher, or the short range of the shotgun. In order to derive the desired mapping between a game level and a weapon pairing, each generated level was tested 25 times, once for each weapon combination (including matches where both teams had the same weapon). Simulations lasted until a total of 20

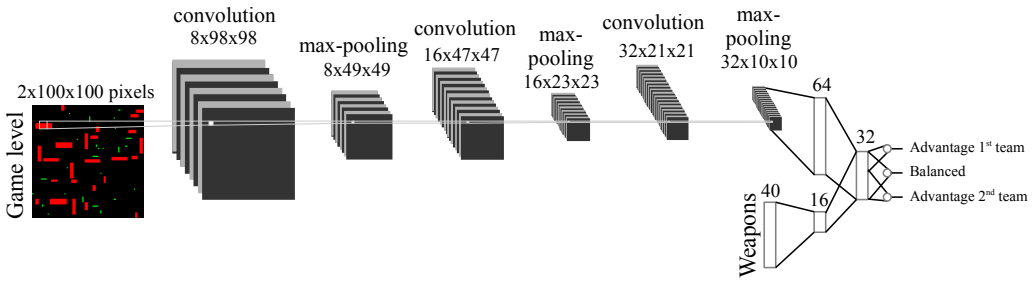


Figure 3: CNN architecture for level and weapon input parameters.

kills were scored, and balance between the teams was calculated based on the number of kills of the first team (the team to the left of Fig.1b) over 20 kills. A ratio of 50% indicates a perfectly balanced (i.e. tied) match-up, a ratio over 60% indicates that the weapon pairing and the level favor the first team, while a ratio of 40% or lower indicates that the second team was favored. Since kill ratios between 40% and 60% amount to a couple of kills for one or the other team, these match-ups are still considered balanced. These three labels were stored for supervised learning.

3.3 Machine Learning

This paper uses a convolutional neural network to predict game balance based on the weapons of each team and the layout of the level. The output of the network is a probability distribution over the three classes: balanced match, 1st team advantage (left in Fig. 1b), 2nd team advantage. The network has three output nodes, which are processed via a soft-max function to convert them into three real values between $[0, 1]$ that sum up to 1; the highest of the three values determines which class the match-up belongs to.

Each weapon is represented by 20 parameters, including damage per bullet, number of bullets per volley, explosion size, reload time, inaccuracy, etc. All members of the same team use the same weapon, therefore the total weapon parameters used as inputs are 40 (20 per team). Weapon parameters are normalized to $[0, 1]$ through min-max normalization across the five weapons. For instance, bullet damage is highest for the rocket launcher (40 hit points or HP), lowest for the SMG and shotgun (4 HP) and above average for the sniper rifle (30 HP); this parameter is converted into 1 for the rocket launcher, 0 for shotgun and SMG and 0.72 for sniper rifle.

As shown in both Fig. 1b and Fig. 2, levels are stored in a 100 by 100 pixel image with three colors (red, green, black) which represent the type of objects in the level (large, small, and no objects respectively). When used by the network, each pixel is represented by two binary values (10 for red, 01 for green, 00 for empty); this allows the network to clearly differentiate between a large object which blocks line of sight, a small object which provides cover, and the lack of either of them. The network therefore uses $2 \times 100 \times 100$ binary inputs (0 or 1) to represent the level. As noted earlier, the inputs describe only the generated arena and ignore the teams' hand-designed bases as no combat is allowed in those areas.

In recent years, convolutional neural networks (CNNs) have become the dominant machine learning approach to image processing due to their success in image classification tasks [19]. Convolutional layers can be understood as a set of filters that are moved over the input to detect certain features; the first layer of these filters often learns to detect edges or spots of different color. Higher layers learn compositions of lower-level features; generally, the more convolutional layers a network has, the more complex features it can detect. Due to the location invariance of these filters, a CNN typically needs fewer parameters than a fully-connected artificial neural network (ANN).

Based on extensive preliminary experimentation with different network architectures, learning rates and activation functions, the best performing CNN architecture used for reported experiments in Section 4 is shown in Fig. 3. The CNN uses three convolutional layers, each with several filters of 3 by 3 pixels applied on the previous layer (i.e. the original image for the first convolution). Since these filters ignore edges, the resolution of the output is 2 less than that of the input. The first convolutional layer outputs 8 different feature maps of size 98×98 out of the original $2 \times 100 \times 100$ pixel image; each filter ideally detects different patterns of the input. Each of these feature maps is downsampled to half its dimensions through *max-pooling*, which outputs the maximum value of a 2 by 2 region of the convolution's output. Max-pooling is applied after each convolution, ultimately producing an output volume of $32 \times 10 \times 10$, which is then converted via a fully-connected ANN layer into 64 values. The 40 weapon parameters are mapped via a fully-connected ANN layer into 16 values. The 64 outputs from the level and the 16 outputs from the weapons are concatenated and passed to a fully-connected layer of 32 nodes which connect to three output nodes that predict the probability that the input belongs to one of the three classes. All nodes in the network use a rectified linear unit (ReLU) as their activation function (which applies element-wise non-linearity), except for the output layer which uses a soft-max function, as described above.

4 EXPERIMENTS

This section discusses the training data collected from artificial gameplays, analyzes how different networks learn this data, and demonstrates how the computational model handles different weapons and level patterns.

4.1 Training Data

As noted in Section 3, training data was collected from generated levels played by teams of artificial agents which for simplicity use

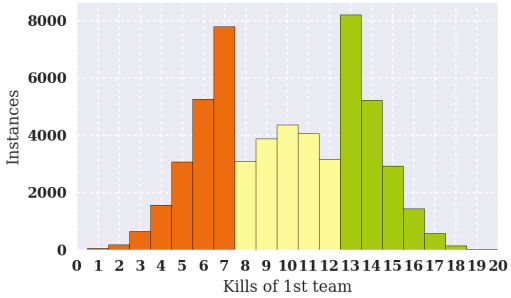
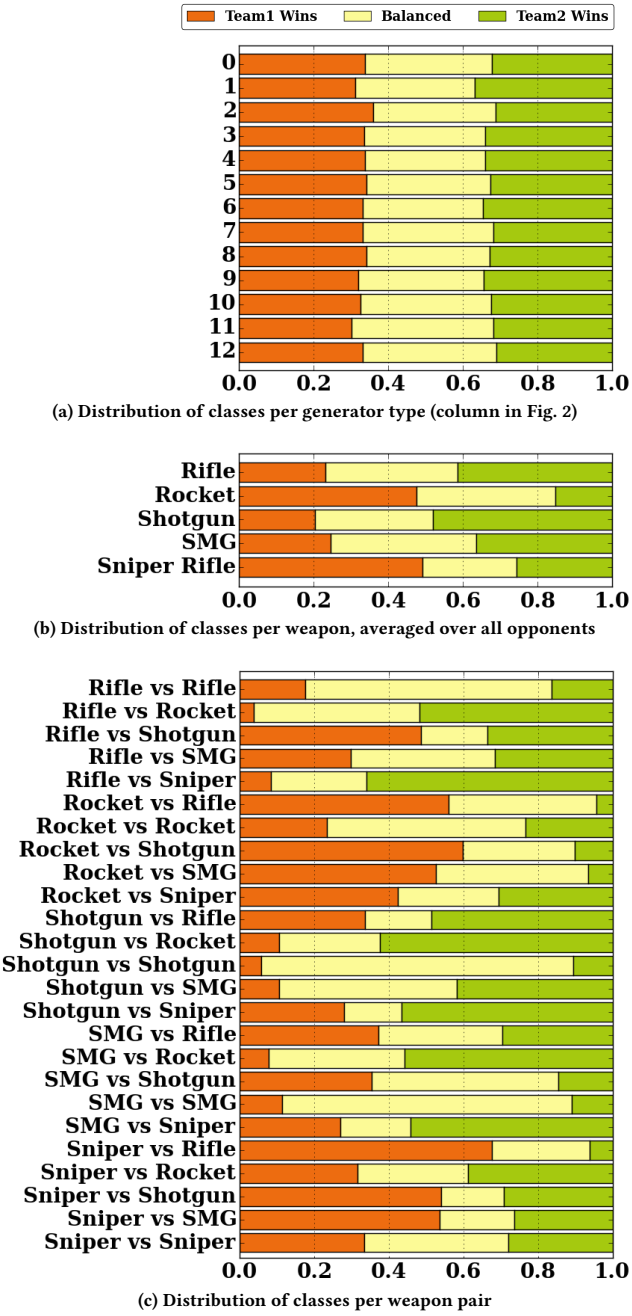


Figure 4: Distribution of kills of the first team (left in Fig. 1b) in the pruned training set.

the same weapon within each team. The 39 generators presented in Fig. 2 generated 100 levels each. These 3900 levels were playtested for all weapon pairings (i.e. 25 matches per level) resulting in $97.5{\times}10^3$ data points. Matches that did not finish before a timeout (less than 0.5% of the data) were removed. An inspection of the data showed that roughly two thirds of the matches classified as balanced. In order to prevent a machine-learned bias towards the most common class, data was pruned so that each class had the same size as the least common class; this was done by removing at random data points from more popular classes. At the same time, it was ensured that data per level generator and per weapon pairing were roughly equally common in the data set. Each of the five weapons is used in 18% to 22% of all data points, and similarly each of the 39 generators was used in 2.2% to 3% of all data points. This resulted in roughly 17×10^3 data points per class. Taking a look at the raw data (before classification) in terms of the first team's kill ratio in the pruned dataset, we can see in Fig. 4 that it is quite symmetric. Moreover, it is clear that before removing roughly half of the balanced matches, kill ratio followed a normal distribution.

In order to gain some insights into the patterns of the training data, we analyze them on a per generator, per weapon and per weapon pairing basis. For generators, we aggregate among all generators in the same column, as they constitute the folds on which we train on: from Fig. 5a we observe minor discrepancies, with all folds having an almost equal distribution of classes and only slight advantages to the first team (e.g. generators of column 2) or the second team (e.g. generators of column 1). There are more clear differences when looking at weapons used by the first team, in Fig. 5b. Both the sniper rifle and the rocket launcher severely favor the team using them (with 49% and 48% of matches belonging to 1st team advantage class respectively); the rocket launcher also has far fewer instances of 2nd team advantage. The opposite is true for the other weapons, which tend towards advantage to the team that does not use them or, at best, balanced matches. Looking at each weapon pairing individually in Fig. 5c, differences become even more clear in terms of class imbalances. Indicatively, the rifle is severely handicapped against the rocket launcher and the sniper rifle, winning only 4% and 8% against each respectively; even so, the rifle versus rocket launcher match-up has more balanced instances than the rifle versus sniper rifle match-up (indicating that those pairings are inherently different). Even when both teams use the



(a) Distribution of classes per generator type (column in Fig. 2)

(b) Distribution of classes per weapon, averaged over all opponents

(c) Distribution of classes per weapon pair

Figure 5: Ratio of each class in the pruned training set.

same weapon, the class distributions are quite uneven: for SMG versus SMG, 78% of match-ups are balanced while for sniper rifle versus sniper rifle only 39% of match-ups are balanced.

Based on the above analysis and the outlook of Fig. 5a, it would seem that the patterns of the levels play a minor, if any, role in the matches' balance. However, this conclusion is mostly due to averaging factors; when looking at the impact of a generator to a weapon pairing, differences become far more evident. For the

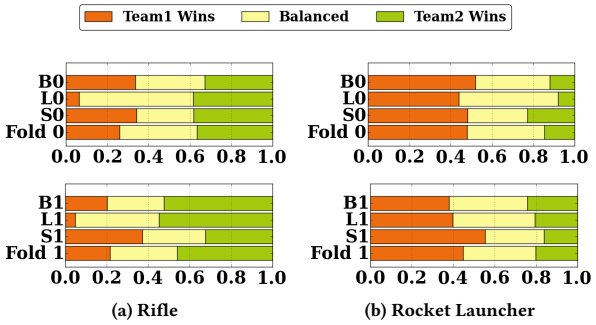


Figure 6: Distribution of classes per weapon for one generator or one fold.

sake of brevity, only two example folds will be examined in detail (column 0 and column 1 in Fig. 2), for two very different weapons: the rifle and the rocket launcher. Fig. 6a shows how classes are distributed when the first team uses a rifle, per generator (B0, L0, S0 and their average as fold 0; B1, L1, S1 and their average as fold 1); Fig. 6b shows the same information but for the rocket launcher. The first observation is that the ratio of small objects versus large objects plays a role regardless of their distribution: S0 and L0 are very different in terms of 1st team advantage instances for the rifle (although less so for the rocket launcher), and L1 and S1 are very different for both the rifle and the rocket launcher (S1 favoring both). The second observation is that each fold results in different class distributions. Fold 0 has a fairer distribution among the three classes for the rifle, while for fold 1 — which has many barriers on the side of the team using the rifle — the 2nd team has a clear advantage in 43% of instances. For the rocket launcher, differences when averaging across folds are less obvious; the rocket launcher is a consistently powerful weapon less sensitive to level differences.

4.2 Training Results

Several neural network architectures, topologies, and activation functions were considered and tested. For the purposes of brevity, this paper focuses on the best performing convolutional network (CNN) architecture, and compares it with the best performing fully-connected network (ANN) and the best performing single layer perceptron. All networks were trained according to a 13-fold cross-validation scheme, where the data in each fold coincides with the generators in the columns of Fig. 2. More specifically, machine learning used the data and ground truth of 11 folds to train on, used one fold as a control for stopping training, and tested the final model on the last fold to derive the validation errors reported here. The networks were trained on the cross-entropy loss, which measures the divergence of the predicted probability distribution with respect to the true class distribution and has several benefits compared to mean-squared error in classification problems [16]. Training was stopped after 5 epochs without improvement on the validation set.

As discussed in Section 3.3, the best CNN architecture is displayed in Fig. 3; it processes the level's image through three pairs of convolution and sub-sampling layers followed by a hidden layer that aggregates the different feature maps, and fuses it with the weapon parameters which are reduced via a hidden layer to 16 outputs, finally producing a probability distribution over the three classes of balance. Many different topologies (including one or two hidden layers) were also tested for ANNs; the best ANN discovered combines the level and weapon parameters (normalized to $[0, 1]$) into $20,040$ inputs ($2 \times 100 \times 100$ image pixels and 40 weapon parameters) and passes it through a single hidden layer of 256 nodes and again to three output nodes. Finally, the perceptron simply connects the $20,040$ inputs to the three output nodes.

Table 1: Mean accuracy and 95% confidence intervals of machine learning from 13-fold cross-validation.

Network	Epochs	Training Accuracy	Validation Accuracy
CNN	24	62%±0.5%	64%±0.2%
ANN	25	54%±11%	50%±2.2%
Perceptron	14	50%±2.2%	48%±1.1%
CNN levels only	12	36%±1.6%	36%±1.6%
CNN weapons only	36	53%±0.5%	55%±1.6%
ANN levels only	14	39%±0.5%	38%±1.1%
ANN weapons only	30	53%±0.6%	56%±1.1%

Table 1 shows the results of the learning process. A baseline of random guesswork would yield an accuracy of 33% among the evenly sampled classes. The perceptron improves on the random baseline by 15%, followed closely by the ANN which classifies 50% of the test data correctly. The CNN is clearly the best network with an average accuracy of 64% on the test data. Interestingly, the ANN tends towards quickly overfitting to the training set (54% accuracy) while the CNN does not do so.

4.2.1 Baselines. In order to determine the benefit of using both the level and the weapons as inputs for balance prediction, the same CNN and ANN topologies were trained using only one of those input modalities, setting the other inputs to zero. When level inputs are zero, the CNN essentially becomes a weapons-only fully-connected artificial neural network, although with a different topology than the tested ANN.

As can be seen from Table 1, training on one input modality without the other makes the two networks (ANN and CNN) perform much more similarly. Especially for the levels-only case, this is surprising since one would expect that the benefits of applying convolutions, such as weight sharing, would still hold. Training only on the levels yields a much poorer test accuracy than training on both inputs, resulting in a classification rate of 38% and 36% for the ANN and the CNN respectively, just slightly better than random. Observing Fig. 5a, this should not come as a surprise, as levels on their own give very similar ratios in terms of classes, regardless of the features (types of pixels or their distributions) they contain. Only when paired with specific weapons (as shown in Fig. 6) do level patterns show clear trends towards one class in the training set, and can be trained to give non-random predictions.

It seems that the weapons by themselves contain more useful information than the levels, as the drop in accuracy is much smaller.

Table 2: Average confusion matrix of the 13 CNNs and 13 ANNs on their respective validation sets.

	Predicted class		
Actual class	Team 2 wins	Balanced	Team 1 wins
Trained CNNs			
Team 2 wins	0.74	0.17	0.09
Balanced	0.26	0.46	0.29
Team 1 wins	0.09	0.17	0.74
Trained ANNs			
Team 2 wins	0.38	0.58	0.04
Balanced	0.15	0.75	0.10
Team 1 wins	0.07	0.60	0.33

In fact, training solely on the weapons results in a better test accuracy for the ANN, equal or marginally higher than that of the 'CNN' architecture using weapons only (i.e. ignoring convolutions). The distribution of outcomes per weapon pair are much more diverse than those of the generators (see Fig. 5c). Therefore, learning the distribution per weapon pair and predicting the most common class given can yield better results than doing this based on the generators. However, depending on the generator, the outcome for a specific weapon can actually be quite different (see Fig. 6). If the network can identify the features that correspond to these different outcomes, it can become more accurate than based on the weapon pair alone. As can be expected, the CNN is better at identifying these weapon-level relationships than the ANN.

As another baseline, we trained networks based on the generator rather than the image of the level itself, and based on the weapon pairing rather than on the weapons' individual parameters. Using a variant of one-hot encoding to determine which generator created the level in the dataset, this reduces the size of the input vector and removes the computer vision task of processing a full level image. The generators were converted into two one-hot vectors based on the rows and columns of Fig.2, i.e. one vector that encodes which type of objects it spawns and one vector that encodes in which area the objects are spawned. The weapons are encoded as one of the 25 possible pairings. These three vectors are concatenated into one vector of size 41 (25 inputs for weapons, 13 inputs for columns and 3 inputs for rows of Fig. 2); this input always contains three values of 1 (which weapon pairing, which column, which row) and the rest are 0. Given the limited input size, only fully-connected networks of one layer with a layer size smaller than the number of inputs were considered. The best ANN for this task has a hidden layer of size 16 and a test accuracy of 57%, while the perceptron has a test accuracy of 56%. The task becomes simpler with a smaller input size for the fully-connected networks, as both networks improve their accuracy from Table 1 and do not overfit to the training data. However, neither network becomes more accurate than the CNN.

4.3 Test Results and Analysis

For these machine learning experiments, we have used 13-fold cross validation by splitting the data based on the levels' generators in columns of Fig. 2. A test accuracy on the entire dataset can be calculated by aggregating all 13 test folds (collected during training and cross-validation). We can then partition the results on a per class, per generator type, and per weapon or per weapon pair basis to get different perspectives and insights.

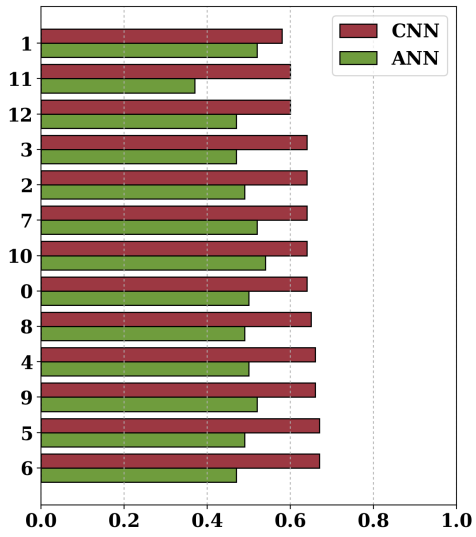


Figure 7: Test accuracy of the networks categorized per generator type, sorted by the accuracy of the CNN.

4.3.1 Results per Class. Table 2 has the confusion matrix of the ANN and CNN, which shows that the ANN has formed a general bias towards the balanced class. Approximately 60% of the time the ANN predicts that the match-up is balanced, regardless of what it actually is. The confusion matrix of the CNN shows that it does not have the same bias towards 'balanced' as the ANN; in fact it struggles to correctly classify the balanced class. Instead, it is very accurate in predicting either advantages. While the ANN correctly classifies 75% of the balanced class, the CNN correctly classifies 74% of both imbalanced classes. A positive note is that both networks have a low probability of misclassifying a match which is imbalanced in favor of one team as one that is imbalanced towards the opposite team.

4.3.2 Results per Generator. Figure 7 shows the accuracy per generator type; each type includes three generators that place different types of objects at the same areas of the level (and coincides with the folds used for cross-validation).

Firstly, it is obvious that the CNN has the highest accuracy for every generator type. Its lowest accuracies are with columns 1, 11 and 12 of Fig. 2. It makes sense that levels of type 11 and 12 are hard to predict, as these levels differ the most from the rest of the corpus due to the designer-placed objects (central block, choke point).

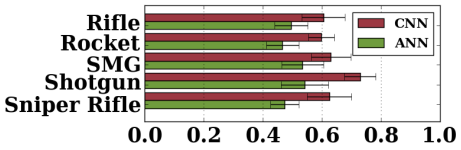
Levels of type 11 are also the most difficult for the ANN to predict accurately; however, type 1 levels are among the easiest to predict. While based on the confusion matrix one would expect that type 1 levels are predominantly balanced, based on the analysis in Fig. 6 this is not the case. It is likely that the left-only placement of small and large objects in S1 and L1 is easier for the ANN to detect compared to the complex patterns of e.g. type 11.

Table 3: Distribution of classes in the ground truth and predicted by the different networks, per B, L, S generator type.

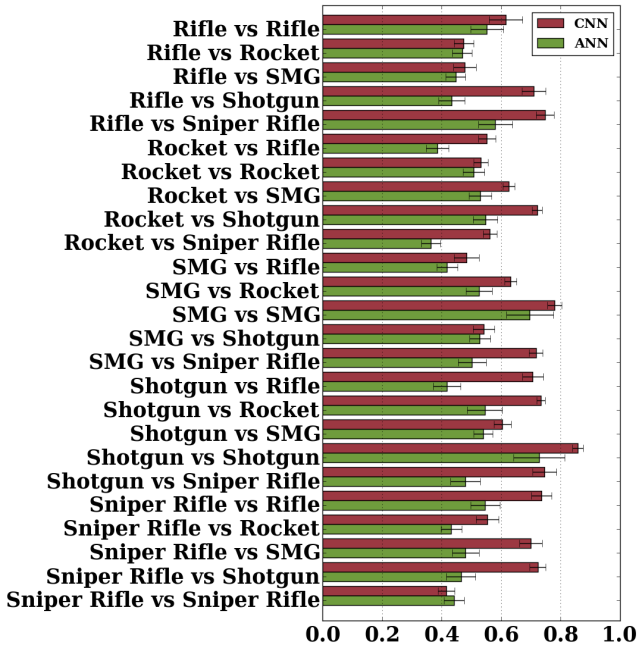
Predictor	Advantage 1st	Balanced	Advantage 2nd	Accuracy
B generators				
Ground Truth	**0.32**	**0.34**	**0.34**	**100%**
CNN	0.38	0.23	0.40	60%±1.6%
ANN	0.00	0.90	0.10	36%±2.6%
Perceptron	0.34	0.29	0.37	47%±2.3%
L generators				
Ground Truth	**0.30**	**0.41**	**0.29**	**100%**
CNN	0.31	0.43	0.26	61%±3.7%
ANN	0.03	0.87	0.10	43%±4.4%
Perceptron	0.34	0.29	0.37	37%±4.3%
S generators				
Ground Truth	**0.37**	**0.27**	**0.36**	**100%**
CNN	0.42	0.17	0.41	70%±3.5%
ANN	0.40	0.22	0.38	66%±4.1%
Perceptron	0.39	0.13	0.48	57%±3.2%

Since the class distributions of the training data when observed along generator types gives indistinct patterns (as in Fig. 5a), assessing accuracy based on the type of obstacles in the level may be more insightful. Assessing accuracy on all levels of the same row (**B**, **L**, **S**), Table 3 shows the distribution of classes in the training data and as predicted during cross-validation by the networks. The table reveals clear biases in the patterns found by the ANN, especially on generators with large objects (**B** and **L**), towards the balanced class. Astonishingly, while 32% of levels created by **B** generators had an advantage towards the 1st team, the ANN predicts that only 0.04% of total instances belong to this class. For **B**, the perceptron surprisingly has the closest distribution to the ground truth, although its predictions are not always accurate (accuracy of 47% compared to 60% of CNN). Based on Table 3, it is clear that the different networks have a bias towards specific classes according to the type of pixels (red and green for **B**, red for **L**, green for **S**) in the image input. Looking into whether those biases lead to correct predictions, on the other hand, we find that the accuracy for all networks improves when levels contain only small objects (**S**). This is likely because in such levels weapons are less affected by blocked lines of sight which cause combat at shorter ranges; this would benefit traditionally weak weapons such as the shotgun. Interestingly, the ANN has a lower accuracy than the perceptron, and a clear bias towards the balanced class in levels of type **B**. We suspect that the ANN does not handle two types of objects (large and small) in the same level well, although the perceptron does not suffer from that. It is therefore more likely that the ANN has overfitted to patterns in other levels (**L** and **S**) which are easier to predict.

4.3.3 Results per Weapon. Grouping the data based on the weapon used by the first team, the CNN also has the highest accuracy in all weapons (significantly higher in 3 of 5) as shown in Figure 8a. All three networks find the shotgun the easiest weapon to predict, which is not surprising as the weapon is consistently



(a) Accuracy averaged per weapon of 1st team.



(b) Accuracy per weapon pairing.

Figure 8: Test accuracy on matchups split by weapons. Error bars indicate the 95% confidence interval.

bad against all weapons in most levels (see Fig. 5b). The second worst weapon, the SMG, is also easy for all networks to predict. Interestingly, for the most powerful weapons (rocket launcher and sniper rifle) the accuracy of the ANN suffers the most; this can be traced to its bias towards classifying most match-ups as balanced (see Table 2) which is not often the case when those two weapons are used; especially for the sniper rifle, only 25% of the ground truth data belong to the balanced class.

When looking at the accuracies for each weapon pair, as in Fig. 8b, clearer patterns can be observed. The CNN has higher accuracies in 24 of 25 weapon pairs, and significantly higher in 17. Both CNN and ANN have the highest accuracy when predicting Shotgun vs. Shotgun (86% and 73% for CNN and ANN respectively), followed by SMG vs. SMG (78% and 69% for CNN and ANN respectively). As shown in Fig. 5c, both of these generally poor weapons when paired against each other lead to predominantly balanced matches (84% for Shotgun vs. Shotgun, 78% for SMG vs. SMG), the largest ratios for the balanced class among the weapon pairings. This explains the high accuracy of the ANN for those pairings, as it tends to predict balanced classes overall. Other weapon pairings with high accuracy for the CNN have a low accuracy for the ANN, most

notably Shotgun vs. Sniper Rifle, Sniper Rifle vs. Shotgun, Rifle vs. Shotgun and Shotgun vs. Rifle; again unsurprisingly, these four pairings have the lowest ratio of balanced match-ups (15%, 17%, 18% and 18%) which gives more predictive power to the CNN which leans towards predicting imbalanced classes rather than the ANN which leans towards the balanced class. As a final note, the only weapon pairing that the ANN is more accurate than the CNN is the Sniper Rifle vs. Sniper Rifle; this is an interesting case as it is not straightforward as to why. Sniper Rifle vs. Sniper Rifle match-ups do not particularly lean towards balanced (39% of instances), therefore the likelihood that ANN's bias towards balanced classes could be only partly an explanation. Based on qualitative evaluations of class distribution on a per generator basis, this weapon pairing is highly inconsistent (e.g. very rarely gives advantage for the first team for B1 and L1 and almost exclusively has balanced instances for L6 and S12; this erratic behavior seems to confuse the CNN more than the ANN, leading to its poor performance.

5 DISCUSSION

Based on the quantifiable results of the machine learning task described in this paper, CNNs are particularly capable of discovering patterns between the level architecture and weapon parameters. While the ANN achieves an accuracy well beyond random, it hardly performs better than a perceptron and, in fact, is outperformed by a network that only receives the weapons of both teams. While using the level only as an input to either the CNN or the ANN does not give them a high predictive capability, when combined with weapon information it contributes to accurately predicting cases where the level architecture affects the relative balance of weapons. The CNN architecture processes the level and discovers higher-level patterns than merely those in the pixel image used by the ANN. The compact and information-rich outputs of the convolutions are better combined with the patterns found in weapons' parameters to classify with 64% accuracy on average between balanced matchups or matchups which favor one team or the other. Another positive result is that misclassifications when the matchup was imbalanced towards one team were not often "catastrophic", i.e. did not predict that the matchup was imbalanced towards the other team.

The ability of the model to fairly accurately predict whether a level and weapon combination will result in a balanced matchup without the need to playtest it can enhance generative processes immensely. Obviously, such a predictive model can replace simulations in a simulation-based evolutionary process such as [3, 4], significantly lowering computation time needed to find new solutions. Based on the current model, however, classifying content into three classes may not be sufficient to provide the gradient towards better solutions that evolution can use as a fitness. Instead, the learned classes can be used in conjunction with simulation-based evaluations, as a first step. For instance, if evolving levels for a balanced matchup between a sniper rifle and an SMG, the predictive model can identify which of the levels are predicted to belong to the balanced class, and only those levels are then simulated to derive a more granular fitness for evolution to follow. The levels that are unbalanced in the above scenario can either be thrown away (e.g. regenerated or given the lowest fitness) or can be further evolved in the hopes of creating balanced ones using e.g. a feasible-infeasible 2-population genetic algorithm [15] as in [23]. Other applications of the model for level design can include a human designer. Since the model receives a raw image as input, it is not bound by the generators used in this study; it can provide input to a human designer by classifying images created manually either through image editing software or directly in Unity by placing objects manually (similar to how generators of type 11 and 12 were created). Since processing an image and weapon pairing by the trained model is computationally lightweight, it can provide feedback to designers in real-time while they are changing the level. As a designer aid, it could even suggest alternatives to the human creator (by adding or moving some objects and evaluating if the match-up is now classified as balanced), in a similar way to *Sentient Sketchbook* [22].

Since the model uses both the weapon pair and the level as input, it can also find the right combination of weapons per team for a provided level. This works in the same way as using the model for level generation (or critique) except now the level input is fixed and the weapon inputs are tested with the 25 combinations of existing weapons in the game. Beyond choosing a balanced set of weapons among those provided, however, the model can be used to generate new weapons or variations of existing weapons by fine-tuning their parameters (e.g. damage, clip size) to further improve the balance of the matches. As an example, the generally worse performance of the SMG can be improved by tweaking its different weapon parameters (via exhaustive or evolutionary search) and testing the modified weapon against all other weapons in a diverse set of game levels until a modified SMG receives a sufficiently increased ratio of predicted balanced instances compared to the original.

It should be noted, however, that the model was trained on artificial data produced by agents' playthrough in the many game levels generated. There is a downside to training a model based on simulated play tests. Unnatural, or even erratic, behavior of artificial agents can seep into the model through the training data. While it would have been better to use data gathered from human matches, the vast volume of data (almost 10^5 games before pruning) required to train the model would need a large and active community of players. Instead, the current trained model could be fine-tuned with sparse matches of human players; since it has learned most level and weapon patterns, it would be easier to fine-tune some associations rather than learn from scratch.

Another interesting way to include human play would be to use human-designed levels. This can be done in two ways: use human designed levels instead of generated ones to run simulations for the training data, and using snapshots of the level design process as the training data itself. For the former, the CNN would likely be able to learn popular level design patterns favored by expert designers, such as choke points and flanking routes [12], and the results of simulations may also more closely match results of real match-ups in popular games. For the latter, the system can assume that the designer attempts to balance out a team-deathmatch game through level design, so the initial level design could be assumed to be worse than future iterations based on design tweaks. By taking multiple snapshots of a designer's process (for instance, a snapshot of the level every 1 minute during design time) gives not only more data for the model to learn from[2], but also a direction towards which

[2]Unlike generated levels, human designed levels are also time-consuming to create, and *good* levels designed by experts are sparse. The generated levels are arguably

types of levels are more acceptable — the later snapshots versus the first snapshots. This information can be used for pre-training the convolutional layers alone, to find desirable level patterns before combining them with weapons to predict game balance.

6 CONCLUSIONS

We have demonstrated how deep learning can be applied to predict the outcome of a match in a simplified 3 versus 3 multiplayer deathmatch shooter game. Using data from simulated playtests, networks were trained to predict whether one of two teams would win by a large margin or whether the match was balanced, based on the level layout as an image and the weapon parameters of both teams. Among the tested architectures, a CNN with three layers of convolutions achieved the highest accuracy, far above the many baselines and alternatives tested. It was shown that the weapons were the most influential input, but also that level design patterns learned from convolutions improved the accuracy.

While we admit that creating balanced multiplayer levels does not solely depend on the score distribution between the two teams, we do think that taking into account the effect of players' weapons or inventories can be a valuable contribution to level generation and design. Moreover, being able to predict the balance of a match through inspecting the map directly can also speed up automatic playtesting. Direct applications of the trained model include level generation to balance a matchup between an imbalanced set of weapons, automatic tuning of weapon parameters to balance weapons on a multitude of levels, and human designer feedback on the balance of a level as it is being designed.

ACKNOWLEDGMENTS

This work has received funding from the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665) and the Horizon 2020 project CrossCult (project no: 693150).

REFERENCES

[1] Staffan Bjork and Jussi Holopainen. 2004. *Patterns in Game Design.* Charles River Media.
[2] Cameron Browne and Frédéric Maire. 2010. Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (2010), 1–16.
[3] William Cachia, Antonios Liapis, and Georgios N. Yannakakis. 2015. Multi-Level Evolution of Shooter Levels. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference.*
[4] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi. 2011. Evolving interesting maps for a first person shooter. In *Proceedings of the Applications of evolutionary computation.*
[5] Michael Cook, Simon Colton, Azalea Raad, and Jeremy Gow. 2013. Mechanic Miner: Reflection-Driven Game Mechanic Discovery and Level Design. In *Proceedings of Applications of Evolutionary Computation.* 284–293.
[6] Joris Dormans and Sander C. J. Bakkes. 2011. Generating missions and spaces for adaptable play experiences. *IEEE Transactions on Computational Intelligence and AI in Games. Special Issue on Procedural Content Generation* 3, 3 (2011), 216–228.
[7] Alexey Dosovitskiy, J. Springenberg, and Thomas Brox. 2015. Learning to Generate Chairs with Convolutional Neural Networks. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition.*
[8] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A Neural Algorithm of Artistic Style. *Computer Vision and Pattern Recognition* (2015).
[9] Robert Giusti, Kenneth Hullett, and Jim Whitehead. 2012. Weapon Design Patterns in Shooter Games. In *Proceedings of the FDG workshop on on Design Patterns in Games.*

also not as good as those created by expert human designers, replacing quality with volume and diversity for the sake of the machine learning process.

[10] Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. 2016. Constrained Surprise Search for Content Generation. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG).*
[11] Daniele Gravina and Daniele Loiacono. 2015. Procedural Weapons Generation for Unreal Tournament III. In *Proceedings of the IEEE Conference on Games, Entertainment, Media.*
[12] Ken Hullet and Jim Whitehead. 2010. Design Patterns in FPS Levels. In *Proceedings of the Foundations of Digital Games Conference.*
[13] Rishabh Jain, Aaron Isaksen, Christoffer Holmgard, and Julian Togelius. 2016. Autoencoders for Level Generation, Repair, and Recognition. In *Proceedings of the ICCC Workshop on Computational Creativity and Games.*
[14] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. 2016. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. In *Proceedings of the Computational Intelligence in Games Conference.*
[15] Steven Orla Kimbrough, Gary J. Koehler, Ming Lu, and David Harlan Wood. 2008. On a Feasible-Infeasible Two-Population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research* 190, 2 (2008), 310–327.
[16] Douglas M. Kline and Victor L. Berardi. 2005. Revisiting squared-error and cross-entropy functions for training neural network classifiers. *Neural Computing & Applications* 14, 4 (2005), 310–318. DOI:http://dx.doi.org/10.1007/s00521-005-0467-y
[17] Pier Luca Lanzi, Daniele Loiacono, and Riccardo Stucchi. 2014. Evolving maps for match balancing in first person shooters. In *Proceedings of the IEEE Conference on Computational Intelligence and Games.*
[18] Raul Lara-Cabrera, Carlos Cotta, and Antonio J Fernández-Leiva. 2013. A procedural balanced map generator with self-adaptive complexity for the real-time strategy game planet wars. In *Proceedings of the Applications of evolutionary computation.*
[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (5 2015), 436–444.
[20] Joel Lehman, Sebastian Risi, and Jeff Clune. 2016. Creative Generation of 3D Objects with Deep Learning and Innovation Engines. In *Proceedings of the International Conference on Computational Creativity.*
[21] Antonios Liapis, Héctor P. Martínez, Julian Togelius, and Georgios N. Yannakakis. 2013. Transforming Exploratory Creativity with DeLeNoX. In *Proceedings of the International Conference on Computational Creativity.*
[22] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-Aided Game Level Authoring. In *Proceedings of the Conference on the Foundations of Digital Games.* 213–220.
[23] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Towards a Generic Method of Evaluating Game Levels. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference.*
[24] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2014. Computational Game Creativity. In *Proceedings of the Fifth International Conference on Computational Creativity.*
[25] Beyer Marlene, Agureikin Aleksandr, Anokhin Alexander, Laenger Christoph, Nolte Felix, Winterberg Jonas, Renka Marcel, Rieger Martin, Pflanzl Nicolas, Preuss Mike, and Volz Vanessa. 2016. An Integrated Process for Game Balancing. In *Proceedings of the IEEE Conference on Computational Intelligence and Games.*
[26] Graeme McCaig, Steve DiPaola, and Liane Gabora. 2016. Deep Convolutional Networks as Models of Generalization and Blending Within Visual Creativity. In *Proceedings of the International Conference on Computational Creativity.*
[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[28] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2016), 484–503.
[29] Adam M. Smith and Michael Mateas. 2011. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 187–200.
[30] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis. 2011. What is procedural content generation? Mario on the borderline. In *Proceedings of the FDG Workshop on Procedural Content Generation.*
[31] Julian Togelius and Juergen Schmidhuber. 2008. An Experiment in Automatic Game Design. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games.*
[32] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011).