Journal of Applied Intelligent System (e-ISSN : 2502-9401 | p-ISSN : 2503-0493)
Vol. 5 No. 1, 2020, pp. 47 – 56

Automatic Power-up Items Placement on Shooter Game using Convolutional Neural Network

Alvin Satria Nugraha[1], **Abas Setiawan***[2], **and Wijanarto**[3]
Faculty of Computer Science,Universitas Dian Nuswantoro, Semarang, Indonesia
E-mail address: 111201609431@mhs.dinus.ac.id[1], *abas.setiawan@dsn.dinus.ac.id**[2], *wijanarto@dsn.dinus.ac.id*[3]
**Corresponding author*

Abstract - A shooter game is a popular game genre with various components. To make a shooter game more attractive, some power-ups items can support players to achieve their goals. Power-ups items provide more power to players, some of which include ammo, extra lives, and invulnerability. The location of power-ups items should be in a special place so that it neither too easy to find nor too difficult to find. Item placement could be done manually by a human or a technical artist. It will need a relatively long time and high cost. In this paper, we try to mimic technical artist vision when placing an item. Visual images have been collected by scanning spatially the forest terrain by using a virtual camera on top. Each image data comply with the item placement rules according to the Tomb Raider and Uncharted 4 games. Convolutional Neural Network (CNN) is used to find out which images can be occupied by power-up items or not. From several experimental scenarios, the use of the Global Average Pooling layer is proven to produce a model that is not overfitting. The best CNN models are developed and got an accuracy of 90.5% with an architecture that includes the Global Average Pooling layer. That model is applied to the new forest terrain so that power-up items can automatically be placed in an appropriate location.

Keywords - shooter game, item placement, forest terrain, CNN, Global Average Pooling

1. INTRODUCTION

Nowadays, the shooter game is a popular game genre that can be played from a third-person or first-person perspective. In the shooter game, there are supporting objects for players to achieve their goal called items. The items are game objects with different functions according to the shape, size, and location. There are two types of items, namely functional items and aesthetic items. Functional items can change the state behavior of the player in gameplay. Therefore, the player can play an important role to make the game progress better, while aesthetic items only change the appearance of the player avatar in the game [1].

One of the functional items in the shooter game is power-up items. In a shooter game, power-up items can provide more powerful abilities to players, some of which include ammo, extra lives, and invulnerability [2]. The placement of this item is not randomly placed somewhere in the 3D game world or in a place that is easily found by the player. These items must be in a strategic location from the player walkable path. Thus, the players are still feel challenged in solving problems in the game. Conventionally, the items were placed manually by technical artists [3].

A technical artist who places these items must understand what type of items should be put in the right place. This can be very troublesome for a technical artist because the right place

can vary in location and shape. One of the challenging game world for a technical artist to put the power-up items is the forest terrain. In forest terrain, there some hidden locations or locations with shapes that are difficult to distinguish to determine whether this place is suitable to put items or not. The placement of power-up items on the forest terrain is also implemented in well-known games such as Star Wars Battlefront 2, Tomb Raider Reboot, Uncharted 4, and Ghost Recon Breakpoints.

While the job of placing these items always assigned to the technical artist, it might need a long time and more costly to finish the game levels [4]. Because at different game levels there will be different shapes of forest terrain. Besides that, in the process of developing a game, we must be able to minimize time and produce interesting levels [5]. However, the placement of these items should not interfere with the main objective of the level and should be able to support the player to achieve that main objective [6].

A few studies have focused on automatic or procedural item placement. The rules-based procedural method has been demonstrated for items in RPG games [7]. There are two types of algorithms, rule-based randomized and fully randomized for procedurally generating items in the game. In comparing the performance of the two algorithms, the rule-based randomized algorithm is proven to be better in implementation.

Some other research is determining the correct location of the item placement automatically. It is based on the path pattern of the player activity to reach a goal in the platformer game [8]. A level of the platformer game has illustrated as a grid where some items are placed in the form of coins. The starting and ending positions of the item placement are also determined along with the path the player can reach. Then, it looks for the closest distance between the placement path and the final position of the grid. If there are no coins in a grid with a certain priority, the coins will be moved to the next grid position.

The study which is closest to this work may be conducted by the weapons placement generator in First Person Shooter (FPS) game [3]. In this study, the weapons in the FPS Game will be classified into the placement category which is divided based on several things such as name, description, weapon consequences, and level patterns of weapons use. In this case, the placement of weapons conducted by the technical artist is very crucial. It because the placement of weapons in the right place by a technical artist can help players to choose the right weapon.

In this study, we focus on the placement of power-up items in the Shooter Game with a forest terrain environment as background captured visually. The visual data is obtained by scanning the existing forest terrain into pieces of an image. From the pieces of an image, a Convolutional Neural Network (CNN) is applied to classify whether the power-up items are suitable to be placed in there or not. In this way, the placement of the power-up item can be done automatically without having placed by the technical artist manually.

2. RESEARCH METHOD

The methodology of automatic item placement consists of several stages shown in Fig. 1. First, generating the forest terrain using procedural terrain generator then scanning a suitable area with a camera. Second, getting the image spatial data from capturing every part of the forest terrain by a camera-like game object. Third, Collecting and labeling the image datasets manually. Forth, configuring CNN architecture. Fifth, training the dataset using CNN to find the best model. Sixth, selecting the best model and placing the item based on that model automatically.

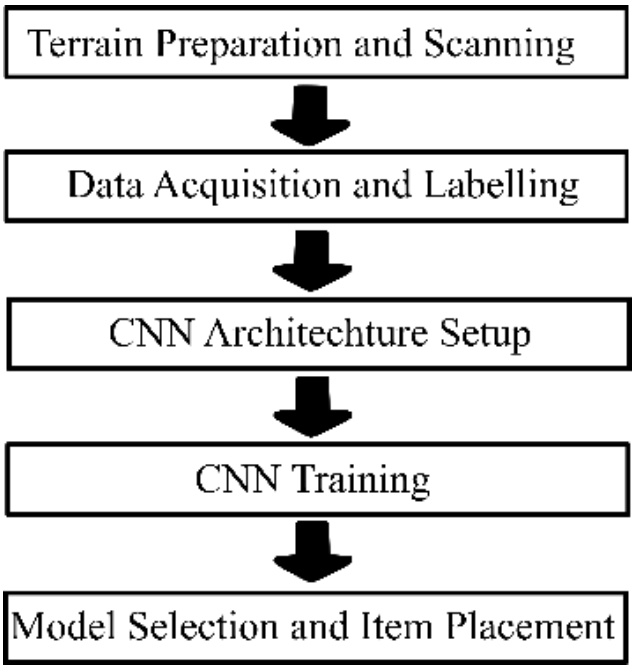

Figure 1. The Methodology of Automatic Item Placement

2.1 Forest Terrain Preparation and Scanning

Unity game engine proposed to implement the entire game system. The terrain can be created manually or with additional plugins in Unity. Gaia Terrain Generator was chosen as an additional plugin because it can automatically generate terrain levels with only a few configurations settings by the developer. Thus, the process of forming a terrain will be easier than having to create a terrain from scratch. In this case, forest terrain has been chosen as a terrain model. It needs several forest terrain samples to be scanned and to obtain cutout images of the captured forest terrain. These forest terrains are required for the formation of the datasets. The number of generated forest terrains is 32. Each forest terrain has a resolution size of 2048×2048 pixels.

The process of scanning the forest terrain to produce captured cutout image data is by positioning the camera-like game object on the top of the terrain. The camera object position must be located at a certain distance so that the terrain is filled with the camera field of view. To provide desired results, the process of scanning each forest terrain is carried out into 8 horizontal and vertical paths. In Fig. 2, the red line shows the horizontal path and the blue line shows the vertical path with the camera facing down. The forest terrain then applied with the x, y, and z positions of 0. While the camera field of view configured at 15 with initial positions x = 33, y = 125, and z = 19 along with rotation only on the x-axis = 90.

After the initial camera conditions have been prepared, then we need a method to obtain each captured cutout forest terrain image. The method is translating the camera position horizontally and vertically to capture each terrain area. Switching the camera position conducted with the offset of the captured area. Because the camera movement is horizontally or vertically changing, so one forest terrain can produce 64 pieces of captured image data.

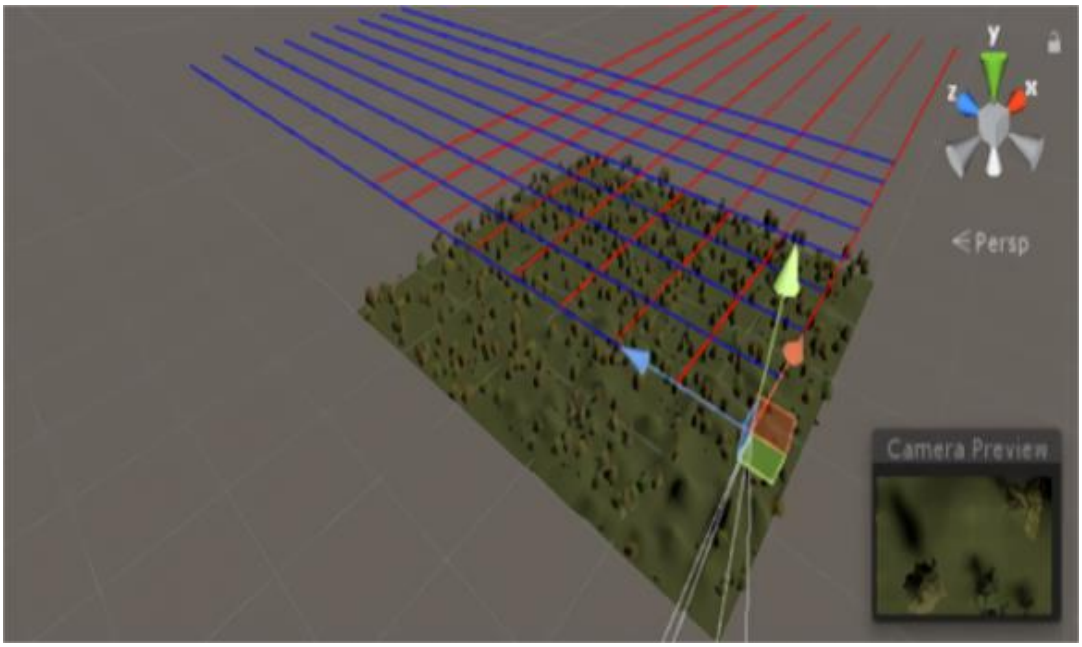

Figure 2. Forest Terrain Scanning Path

2.2 Data Acquisition and Labelling

Labeling the images data conducted after each required data has been collected. There are two labels on each image, namely positive and negative images. Positive images are cutout images of the forest terrain that contains a suitable place which is the right place to put power-up items. In contrast, negative images are cutout images of the forest terrain containing an unsuitable place for the power-up items to be placed.

Positive images regulation follows the considerations in Shadow of Tomb Raider and Uncharted 4 games. Table 1 show the rules for placing power-up items inspired by the Shadow of Tomb Raider and Uncharted 4 games from the player or forward point of view.

Table 1. The Item Placement Rules from a Forward Point of View

No	Item Placement Rule
1	Item place under a tree inside the forest and between the rocks.
2	Item place on the rocky ground at the end of the forest.
3	Item place inside a shallow puddle.
4	Item place on the rocky plateau at the cliff.

However, Table 1 contains a rule that is viewed not from the top of the terrain. Because the camera object position is the top and facing down, so the rules in Table 1 are adjusted to the rules shown in Table 2.

Table 2. Rules for positive captured image labeling from the top viewpoint

No	Item Placement Rule
1	The captured image was so many trees and little rocky ground at side.
2	The captured image was a few trees and rocky ground at the end of the forest.
3	The captured image was so many trees and some shallow puddles.
4	The captured image was a cliff edge dominating.

Positive images are collected measly compared to negative images. From 32 forest terrain can produce a total of 2048 captured images. Each image has dimensions of 128×128. Because the positive images collected by 552 then selected negative images also 552. Thus, the total data are 1104 images. Afterward, the data augmentation was performed by rotating the image by 90 and 180 degrees. Consequently, the number of images increased to become 3312 images collected. It is divided into 2184 train data, 728 validation data, and 400 test data.

2.3 CNN Architecture Setup

After the image datasets have been collected, then it is trained using CNN. CNN algorithm is commonly used for visual classification and recognition tasks in various fields [9]. The CNN ability is processing visual information inspired by the human biological brain system [10]. There are several benefits of CNN when applied to games, such as increasing game interaction [11], detecting glitch [12], controlling NPC behavior [13], recognizing game objects [14], and improving gameplay [15].

Because each cutout data of forest terrain is a colored image, so the number of channels in the input layer is three related to the image color channels, namely red, green, and blue [16]. The proposed CNN architecture consists of three convolution layers, three max-pooling layers [17], one Flatten or Global Average Pooling layer [18], one fully connected layer, and one output layer. The Rectified Linear Unit (ReLU) activation function [19] also applied to the convolution layer and fully connected layer. Meanwhile, the output layer uses the sigmoid activation function.

Convolution Layer 1 is composed of a 5×5 kernel with the same padding, 64 Feature Map, and a ReLU activation function. Then it followed by Max Pooling Layer 1 with the pooling size of 2×2 and valid padding. Convolution Layer 2 is composed of a 3×3 kernel with the same padding, 256 Feature Map, and ReLU activation function. Then it is followed by Max Pooling Layer 2 which has the same configuration as Max Pooling Layer 1. Convolution Layer 3 and Max Pooling 3 are the same as convolution configuration Layer 2 and Max Pooling 2. The flatten layer or Global Average Pooling and the number of units in the fully connected layer will be explained later in Section 3.

In classic CNN, the flatten layer is designed by performing feature maps transformations on a single vector. Instead of doing this, the average computation applied for each feature map and then transformed into a single vector. This is the idea of composing the Global Average Pooling layer [18]. The advantage of Global Average Pooling can form a category confidence map where the map features correspond to the class category. Besides, the Global Average Pooling layer avoids overfitting because there is no parameter optimization process.

The last pooling layer in this architecture contains 256 feature maps, with each dimension size of 16×16. Let f_k represent the activation map, where $k \in \{1, \dots, 256\}$. Global Average Pooling layer reduces the size of the preceding layer to 256 units by taking the average of each feature map. Alternatively, when using a flatten layer, the size of the preceding layer becomes 65536 units.

2.4 CNN Training

The training process consists of two stages, namely forward propagation and backpropagation. The forward propagation is the feed-forward calculation from the convolution layer 1 to the output layer. The feed-forward means the output from the previous layer will become the input for the next layer, and so on until the output from the last layer. The calculation involving parameters that have been previously initialized with random values. After forward propagation, the results from the last output layer are compared with the desired output or label so that a loss value will be generated. Because the last output layer consists single unit and it uses a sigmoid activation function, the loss calculation is solved by the binary Cross Entropy Error, as in

$$\mathcal{L} = -\big(y\ log(\hat{y}) + (1-y) log(1-\hat{y})\big), \qquad (1)$$

where $\hat{y}$ are the desired output value and y is the predicted output value.

The backpropagation process is to calculate the error gradient of the loss function to all the parameters by finding the partial derivative of the function. It is conducted from the output layer back to the input layer. This process intends to readjust each parameter based on the loss value obtained during the forward propagation. The next step is to update the parameters of θ. Parameters updates using Adam's optimization [20] at timestep t calculated as

$$g_t = \nabla_\theta f_t(\theta_{t-1}), \qquad (2)$$

$$\hat{m}_t = \beta_2 \cdot m_{t-1} + (1-\beta_1) \cdot g_t/(1-\beta_1^t), \qquad (3)$$

$$\hat{v}_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2/(1-\beta_2^t), \qquad (4)$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}, \qquad (5)$$

where $\beta_1, \beta_2 \in [0,1]$ are exponential decay rates, α is learning rate, $\hat{m}_t$ bias-corrected first-moment estimation, and $\hat{v}_t$ bias-corrected second raw moment estimated. The update biased first moment $\hat{m}_t$ and second raw moment $\hat{v}_t$ estimate calculated by the gradient errors ∇_θ with related to the stochastic objective function $f_t(\theta_{t-1})$.

Besides that, the whole training was carried out in a mini-batch fashion with a batch size of 32. The process of feedforward propagation, backpropagation, and weight updating with Adam was carried out repeatedly to minimize the loss function. Thus, the network will be able to find the optimal parameters so that it will be able to distinguish which cutout image of the forest terrain is positive or negative.

3. RESULTS AND DISCUSSION

3.1 Experimental Scenarios

The experiment in this study is evaluating the CNN architecture with three convolution layers and three pooling layers as described in Section 2. There are different experimental

scenarios which affected the architecture. First, the number of units in a fully connected layer. Second, the use of the Global Average Pooling layer instead of the Flatten layer. Third, different learning rate hyperparameters in Adam optimization. Table 3 shows the details of the experimental scenario.

Table 3. Experimental Scenarios Configuration

Id	FC Layer	Flatten Layer	GAP Layer	Learning Rate
1	256	False	True	0.0005
2	256	False	True	0.0001
3	256	True	False	0.0005
4	256	True	False	0.0001
5	482	False	True	0.0005
6	482	False	True	0.0001
7	482	True	False	0.0005
8	482	True	False	0.0001

3.2 Experimental Result

CNN training scenarios implementation in Table III is trained by a GPU Nvidia GTX 1070. In this study, we analyze the effect of the different number of units on the Fully Connected Layer. Besides, the use of the flatten layer and Global Average Pooling will also be considered. The last one is analyzing two variants of learning rate values. The training involving the train and validation dataset with a total sample of 2912 data. This training process is carried out with 20 epochs. Fig. 4 depict the results of the loss values for each experimental scenario during the training.

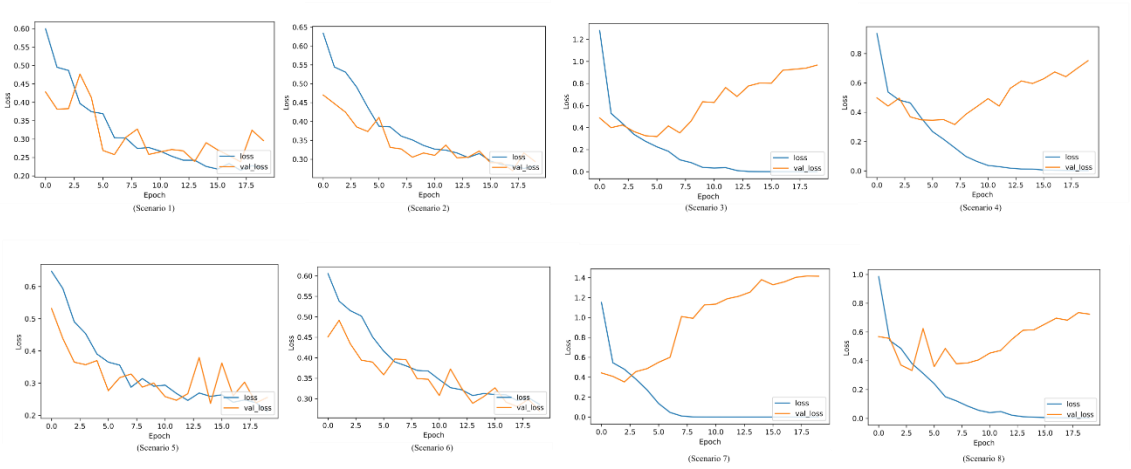

Figure 3. Loss values during the training

The different setting of a fully connected layer is not too significant in minimizing the loss value. Similarly, the learning rate of 0.0005 or 0.0001 has no significant effect, but if it less or more that that value, the training process became unstable. The validation loss results always try to decrease when CNN applies Global Average Pooling as presented in scenarios 1, 2, 5, and 6. Other scenarios that did not apply Global Average Pooling, make the training unstable and tend

to increase the loss value when the training reach about a quarter of the total epoch passed. It has an impact on increasing accuracy. It means the network can generalize the patterns during the training process. The result of the validation accuracy in each test scenario during the training process is shown in Fig. 4.

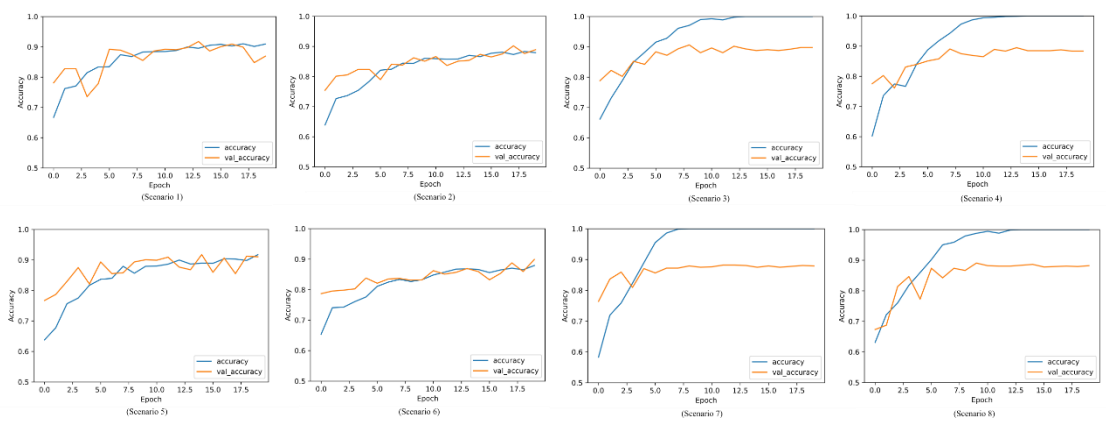

Figure 4. Accuracy values during the training

The consequence of the network fails to reduce loss in every scenario that does not apply Global Average Pooling is the result of validation accuracies are overfitting. The training accuracy has a good shape, but the validation accuracy tends to be flat starting from the fifth epoch. When the training process complete, each scenario model will be saved at local storage. The final evaluation is applying the test data of 400 samples for each stored model. Table IV depicts the final accuracy achieved for each scenario model.

Table 4. Accuracy Result

Id	1	2	3	4	5	6	7	8
Accuracy	88	86	80	78.2	90.5	85.7	75.5	79

3.3 Model Selection and Power-up Items Placement

The best model with the highest accuracy is the fifth scenario as shown in Table IV. The best model must be converted into a file format that can be understood by the Unity game engine. We use the Unity Barracuda plugin to become an inference engine. One of the supported formats in Unity Barracuda is Open Neural Network Exchange (ONNX). Before the best model is converted into an ONNX file format, it must be saved as a hierarchical data format.

When the best model becomes an ONNX file format then it stores as a Unity game asset. Unity can automatically detect the file and display the model description in the inspector window. The model cannot be used directly. It must be linked using the Unity script with the support of the C# programming language. Then, with a similar mechanism as forest terrain scanning described in Section 2, a cutout image of forest terrain was obtained. The cutout image and the spatial point is temporarily stored in a specific directory. The spatial point means the midpoint of the image data captured by the camera because power-up items will be placed with a certain offset radius.

A cutout image then applies the previously selected ONNX model to classify whether power-up items can be placed at the spatial point or not by using a forward propagation calculation. It was conducted linearly from the first cutout image to the last cutout image. In this case, we use a third-person shooter game perspective as a game simulation. Fig. 5 shows the power-up items which have been placed in its rightful position. Fig. 5 shows a power-up Item that has been successfully placed in the middle of the forest and the reachable puddle. The power-up items may appear hidden but it still can be reached by the player.

Figure 5. Location After Items Placed in Game

4. CONCLUSION

Power-up items placement in shooter game is more time consuming and high cost to handle manually by technical artists. We proposed the approach which tries to mimic a human artist when he finds a suitable area to place an item visually. A CNN can be used to classify whether certain locations on the forest terrain are suitable to place power-up items or not based on a captured cutout image of the forest terrain. The captured cutout image of the forest terrain obtained from a camera that is placed on top of the forest terrain and moved it vertically and horizontally. There are eight experimental scenarios with different CNN architectures to find the best classification model. The similarities of the eight scenarios are in the convolution and pooling layers. After experimenting with eight different CNN architectural scenarios, there is a high accuracy result if the CNN architecture implements Global Average Pooling.

The fifth experimental scenario gets the best accuracy results, namely 90.5. This accuracy value greatly affects how the system will correctly classify whether the forest terrain image pieces are suitable for placing power-up items. The training model from the fifth scenario then applied to the Unity game engine. Then each image captured by the camera will be classified by the fifth scenario model by storing the spatial location of each image. The power-up items can be placed automatically at the spatial location point, where the spatial location point is in the middle of the classified image. Therefore, using CNN to place power-up items on forest terrain can accelerate and reduce costs in game development.

REFERENCES

[1] M. McCaffrey, *The Evolution and Social Impact of Video Game Economics* , vol. 36, no. 3. Lexington Books, 2020.
[2] S. Rogers, *Level Up!: The Guide to Great Video Game Design*, 1st Editio. Wiley, 2010.
[3] R. Giusti, K. Hullett, and J. Whitehead, “Weapon design patterns in shooter games,” *ACM*

Int. Conf. Proceeding Ser., 2012.

[4] J. Taylor and I. Parberry, "Randomness + structure = clutter: A procedural object placement generator," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6972 LNCS, pp. 424–427, 2011.

[5] T. Le Dang, "Level Designing in Game Engine," 2017.

[6] E. Andersen, Y. E. Liu, R. Snider, R. Szeto, S. Cooper, and Z. Popović, "On the harmfulness of secondary game objectives," in *Proceedings of the 6th International Conference on the Foundations of Digital Games, FDG 2011*, 2011, pp. 30–37.

[7] C. K. On, N. W. Foong, J. Teo, A. A. A. Ibrahim, and T. T. Guan, "Rule-based procedural generation of item in Role-Playing Game," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 7, no. 5, pp. 1735–1741, 2017.

[8] A. Sarkar, V. Sriram, R. Padte, J. Cao, and S. Cooper, "Desire path-inspired procedural placement of coins in a platformer game," *CEUR Workshop Proc.*, vol. 2282, 2018.

[9] W. Rawat and Z. Wang, "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review," *Neural Comput.*, vol. 29, no. 9, pp. 2352–2449, Sep. 2017.

[10] G. W. Lindsay, "Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future," *J. Cogn. Neurosci.*, pp. 1–15, Feb. 2020.

[11] D.-S. Tran, N.-H. Ho, H.-J. Yang, E.-T. Baek, S.-H. Kim, and G. Lee, "Real-Time Hand Gesture Spotting and Recognition Using RGB-D Camera and 3D Convolutional Neural Network," *Appl. Sci.*, vol. 10, no. 2, p. 722, Jan. 2020.

[12] C. Garc, K. Tollmar, and L. Gissl, "Using Deep Convolutional Neural Networks to Detect Rendered Glitches in Video Games," in *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-20) Using*, 2020.

[13] E. S. Soares and V. Bulitko, "Deep Variational Autoencoders for NPC Behaviour Classification," in *2019 IEEE Conference on Games (CoG)*, 2019, vol. 2019-Augus, pp. 1–4.

[14] V. Reno, N. Mosca, R. Marani, M. Nitti, T. D'Orazio, and E. Stella, "Convolutional Neural Networks Based Ball Detection in Tennis Games," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, vol. 2018-June, pp. 1839–18396.

[15] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep Learning for Video Game Playing," *IEEE Trans. Games*, vol. 12, no. 1, pp. 1–20, Mar. 2020.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[17] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6354 LNCS, no. PART 3, 2010, pp. 92–101.

[18] M. Lin, Q. Chen, and S. Yan, "Network In Network (paper)," *arXiv Prepr.*, p. 10, Dec. 2013.

[19] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical Evaluation of Rectified Activations in Convolutional Network," May 2015.

[20] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, Dec. 2015.