Game Level Generation- Group Report

Darshan Radhakrishnan – K2288661
Nirmal Mathew – K2314992
Adhisivan Madasamy – K2361521
Nandu Kadakath Sudhi – K2306560

1 Aim:

The aim of this project is to develop a deep learning-based system capable of generating room decorations within dungeon environments.

1.1 Objective:

To achieve this aim, the project will involve several key objectives. First, it will preprocess and augment the dataset for room decorations to ensure the input data is of high quality and suitable for training robust models. This involves cleaning the data, normalizing it, and applying augmentation techniques to enhance the diversity and quantity of the dataset. Next, the project will focus on building and training deep learning models, specifically leveraging Conditional Generative Adversarial Networks (CGANs) to generate diverse and realistic room decoration layouts. Once the models are trained, they will be evaluated using a variety of metrics, including Fréchet Inception Distance (FID), precision, recall, diversity, and coverage, to comprehensively assess their performance and quality. Finally, the project will integrate the trained model into an interactive application that allows for real-time generation of room decorations, facilitating seamless use in game environments and enhancing the overall gaming experience.

2 Literature Review:

2.1 Overview:

Deep Neural Networks (DNNs) have significantly advanced Procedural Content Generation (PCG) in game development. Architectures such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) have shown remarkable capabilities in generating diverse and coherent game content, including textures, levels, characters, and environments. These DNNs offer innovative solutions for automating content creation, enhancing both the creativity and efficiency of game development processes.

2.2 Relevant Work:

Generative Adversarial Networks (GANs) have been particularly successful in producing high-quality and diverse outputs. Volz et al. (2018) demonstrated this potential in their work on Super Mario Bros level generation, showcasing GAN's ability to create intricate and playable game levels. Similarly, Recurrent Neural

Networks (RNNs) have proven effective for sequence generation tasks within game environments, contributing to the dynamic and sequential aspects of game content. Convolutional Neural Networks (CNNs) excel at image-based content generation, facilitating the creation of detailed and realistic textures and visual elements.

Conditional GANs (CGANs), an extension of GANs that incorporate additional information such as labels or specific features, have been employed for style-specific interior design generation. Zhu et al. (2020) exemplified this application by using CGANs to generate interior designs tailored to specific styles, demonstrating the potential of these models in generating aesthetically pleasing and contextually relevant content.

2.3 Gaps Identified:

Despite these advancements, several technical challenges remain. Current models often struggle to maintain contextual coherence and ensure logical relationships between generated elements, such as furniture placement in room layouts. Integrating both room-specific constraints and aesthetic variety into a single model poses significant difficulties. Additionally, the quality and quantity of training data are critical factors that influence model performance, especially in specialized domains like interior design. Another concern is model interpretability, as understanding and controlling the generation process in complex models like GANs can be challenging.

2.4 Proposed Solution:

To address these gaps, we propose developing a CGAN-based model specifically for room decoration. This model will incorporate a custom loss function that balances aesthetic and functional constraints, ensuring both visual appeal and practical usability in the generated room layouts. A hierarchical generation process will be employed to maintain coherence and logical object placement. Data augmentation techniques will be used to expand the training dataset, improving the model's ability to generalize. Furthermore, an interpretable latent space will be implemented to allow for more controlled generation and better understanding of the model's behavior.

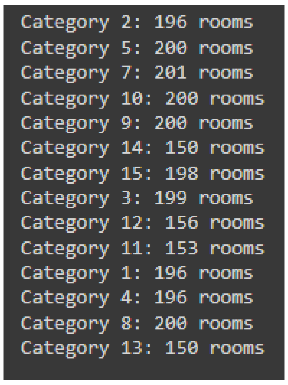
3 Data Preparation

3.1 Data Augmentation

To enhance the room decoration dataset, several data augmentation techniques were applied to increase its diversity and size. Initially, room layouts were classified into categories based on specific decoration patterns (value 7) in positions such as the bottom row, top row, left column, and right column.

For each category, rotations (90 degrees left, 90 degrees right, and 180 degrees) and flips (horizontal and vertical) were applied. For example, rooms classified under the "Down" category were rotated and flipped to create new variations. Similar augmentations were applied to other categories, such as "UpDown" and "DownRight," to ensure a comprehensive coverage of possible layouts.

The augmentation resulted in a significantly larger dataset, combining both original and augmented layouts. This increase in the number of training samples is expected to improve the model's ability to generalize. A bar chart comparing the number of layouts in each category before and after augmentation visually confirmed the effectiveness of these techniques. This enhanced dataset is anticipated to lead to better model performance in generating diverse and contextually appropriate room decorations.



```
Category 2: 196 rooms
Category 5: 200 rooms
Category 7: 201 rooms
Category 10: 200 rooms
Category 9: 200 rooms
Category 14: 150 rooms
Category 15: 198 rooms
Category 3: 199 rooms
Category 12: 156 rooms
Category 11: 153 rooms
Category 1: 196 rooms
Category 4: 196 rooms
Category 8: 200 rooms
Category 13: 150 rooms
```

Fig1: Categories of Augmented rooms for Room Decor

3.2 Data Collection and Preprocessing

The dataset for room decoration consists of 8x8 grid layouts representing various room configurations, each identified by specific decoration patterns. Initially, the dataset was thoroughly checked for missing data, ensuring data integrity by handling any incomplete or corrupted records. The dataset was then normalized to a range of [-1, 1] to ensure consistent scaling, crucial for effective deep learning model training. Following normalization, the dataset was reshaped to include a channel dimension, converting each 8x8 grid into an 8x8x1 format, which is essential for compatibility with convolutional neural networks. Visualization techniques were employed to display subsets of the dataset, providing a clear understanding of data distribution and variety. Additionally, another data preparation method normalized the dataset to a [0, 1] range and reshaped it for model compatibility by flattening the 8x8 grids into a single dimension. This comprehensive preprocessing ensured the dataset was ready for efficient and effective model training.

4 Model Design and Training

4.1 Model Architecture

The project designed two models for dungeon generation and two for room decoration using Conditional Generative Adversarial Networks (CGANs). The generator model uses dense layers, Leaky ReLU activations, and batch normalization to transform random noise into an 8x8 grid representing room decorations. The discriminator flattens the input image and processes it through dense layers with Leaky ReLU activations, ending in a single output neuron with a sigmoid activation to classify the input as real or fake. The GAN model combines both components, freezing the discriminator's weights during generator training for stability.

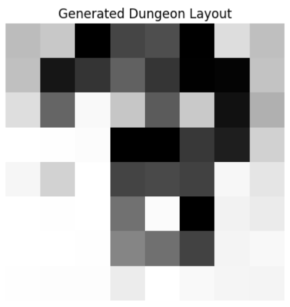
Two models were designed for dungeon generation and two for room decoration using Variational Autoencoders (VAEs). The encoder network transforms the input into a latent space representation, producing mean and log variance for the latent variables. The decoder network reconstructs the input from these latent variables using dense layers with sigmoid activation, suitable for binary outputs. The VAE combines these components with a custom sampling layer to generate latent space points.

4.2 Training Process

The dataset was divided into training (80%) and testing (20%) sets. Training involved fine-tuning hyperparameters, including a learning rate of 0.0002, a batch size of 32, and 10,000 epochs, with an additional 10,000 epochs using early stopping and model checkpoints.

The training loop included training the discriminator with real and fake images to improve accuracy, and training the generator to produce images that could fool the discriminator. Post-processing correction algorithms ensured logical placement and aesthetic coherence. Early stopping and model checkpoints were employed to prevent overfitting. This approach, repeated for all four models, ensured the development of robust models capable of generating diverse and contextually appropriate room decorations within dungeon environments.

The dataset was divided into training and testing sets. The VAE model was trained using a combined loss function that includes binary cross-entropy for reconstruction loss and KL divergence for regularization. Key hyperparameters included a learning rate of 0.001, batch size of 32, and 1000 epochs. Early stopping and model checkpoints were used to enhance training stability and prevent overfitting. The training loop optimized the VAE model using the Adam optimizer, ensuring the latent space followed a normal distribution. After training, the decoder generated images from random latent vectors, which were binarized for visual evaluation to ensure logical and aesthetically coherent room decorations. This approach developed robust models capable of generating diverse and contextually appropriate room decorations within dungeon environments.



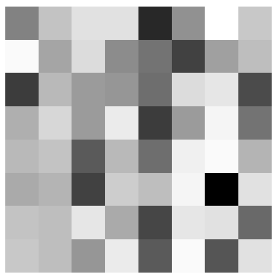
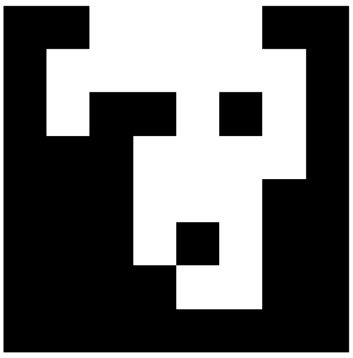


Fig2: Trained model of a Dungeon with VAE

Fig3: Trained model of a Dungeon with GAN

4.3 Correction Algorithm:

The correction algorithm starts by loading the trained generator model, enabling the generation of new room decoration images from random noise inputs. The generated grayscale images are converted to binary format using a specified threshold, where pixel values above the threshold are set to white (1) and those below to black (0), ensuring clear distinction of layout elements. These binary images are then visualized to assess their visual and logical quality, with optional color inversion to enhance contrast. This process ensures the final outputs are high-quality, visually appealing, and logically consistent, enhancing the overall effectiveness of the room generation system.



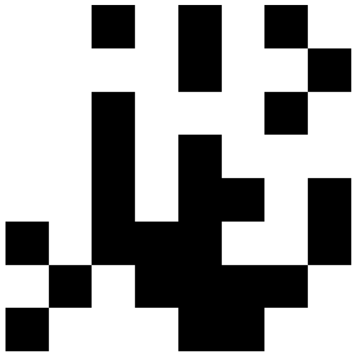


Fig4: Corrected Image of Dungeon with VAE

Fig5: Corrected Image of Dungeon with GAN

4.4ROOM DECOR MODEL:

4.5Model Architecture

The project implemented Conditional Generative Adversarial Networks (CGANs) to generate room decorations, comprising a generator and a discriminator. The generator combines a noise vector and a one-hot encoded label, processing them through dense layers with ReLU activations and batch normalization to produce room layouts. The discriminator flattens and concatenates room images and one-hot encoded labels, processing them through dense layers with Leaky ReLU activations to output a validity score.

4.6Training Process

The dataset was divided into training and testing sets. The CGAN was trained using binary cross-entropy loss and the Adam optimizer with a learning rate of 0.0002, a batch size of 16, and 5000 epochs. Training alternated between the discriminator, which learned to differentiate between real and generated images, and the generator, which learned to produce images that could fool the discriminator.

Metrics such as Fréchet Inception Distance (FID), precision, recall, diversity, and coverage were calculated to evaluate model performance. Data normalization scaled the dataset between 0 and 1, and labels were one-hot encoded. The training process involved generating batches of real and fake images for discriminator training, followed by generator training using the CGAN model. Performance was monitored and logged every 100 epochs to allow for periodic evaluation and adjustments.

This detailed training process ensured the development of robust CGAN models capable of generating diverse and contextually appropriate room decorations, meeting the project's objectives.

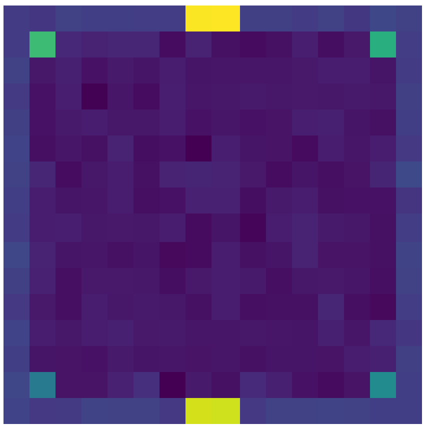


Fig 6: Trained model of a Room Decor with CGAN

4.7 Correction Algorithm & Evaluation metrics

The correction algorithm begins by loading the trained generator model to generate new room decoration images from random noise inputs. The generated images, initially in grayscale, are rescaled to a range of 0 to 1. The algorithm then corrects the room layout by ensuring walls (value 1) are placed around the borders and preserving the positions of doors (value 7). The interior of the room is filled with either walls or background based on the original values, ensuring doors are not overwritten. Finally, the corrected layout is visualized to confirm it meets the desired quality and logical structure. This process enhances the visual appeal and logical consistency of the generated room decorations, improving the overall effectiveness of the CGAN system.

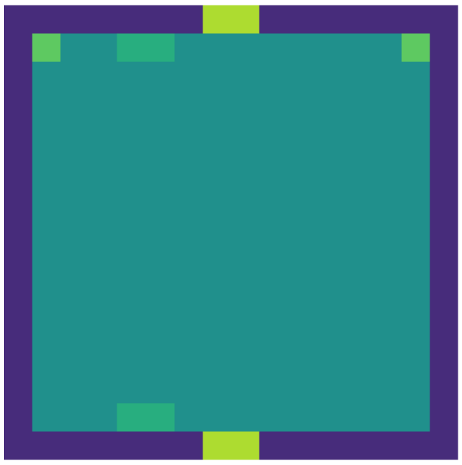
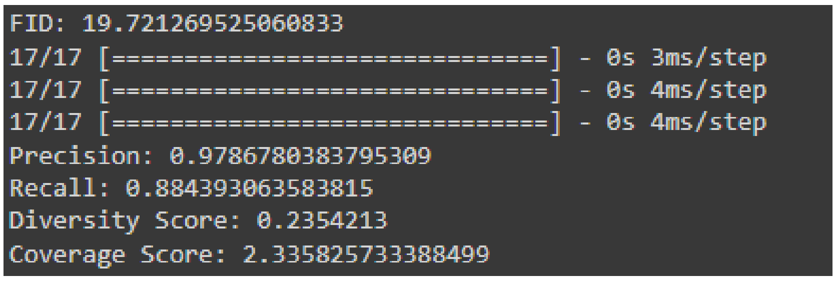


Fig7: Corrected image of Room model with CGAN

The evaluation metrics indicate that the CGAN model performs well in generating realistic and diverse room decorations. The Fréchet Inception Distance (FID) of 19.72 suggests that the generated images are reasonably like real images. With a precision of 0.978 and recall of 0.884, the model accurately identifies real instances and correctly classifies most real images. The diversity score of 0.235 points to moderate variation among generated images, while the coverage scores of 2.336 indicates that the generated images broadly represent the characteristics of real images. Overall, these metrics demonstrate the effectiveness of the CGAN model in producing high-quality and contextually appropriate room decorations.



```
FID: 19.721269525060833
17/17 [==============================] - 0s 3ms/step
17/17 [==============================] - 0s 4ms/step
17/17 [==============================] - 0s 4ms/step
Precision: 0.9786780383795309
Recall: 0.884393063583815
Diversity Score: 0.2354213
Coverage Score: 2.335825733388499
```

Fig 8: Evaluation metrics of CGAN (Conditional Generative Adversarial Network)

5 Unity Integration

5.1 Flask Integration

This Flask application serves pretrained deep learning models to generate and correct dungeon and room layouts, integrating Flask for web server functionality, NumPy for numerical operations, and TensorFlow/Keras for model handling, with logging for debugging. It loads GAN and CGAN models to create 8x8 dungeon images and 16x16 room layouts, respectively, rescaling generated images to a range of 0 to 1. Image correction functions ensure these images meet specific criteria, such as placing walls around borders and preserving door positions. The application provides endpoints like `/getDungeonGAN` for dungeon images and `/getRoomCGAN` for room layouts, processing image generation and correction, converting images to integer lists, and returning JSON responses. This setup facilitates seamless integration of generated and corrected layouts into game development workflows.

5.2 Unity Integration

This Unity integration with a Flask backend enables the generation and visualization of dungeon and room layouts using pretrained GAN and CGAN models. The `DungeonGenerator` class manages the conversion of flat data received from Flask into 2D matrices and instantiates game objects for tiles and props based on these matrices. The `DisplayDungeonMap` method creates a texture to visually represent the dungeon, setting pixel colors according to the data values. It then calls `GenerateTiles` to instantiate tiles in the Unity scene. If a room tile is detected, the `StartRoomReq` coroutine requests room data from Flask, and the `GenerateProps` coroutine instantiates props within the room based on the room matrix data.

The `FlaskReq` class handles HTTP requests to the Flask endpoints to retrieve generated dungeon and room data. Using UnityWebRequest, it sends GET requests, parses JSON responses with Unity's `JsonUtility`, and stores the data in lists for the `DungeonGenerator` class to use. The `GetDungeonData` method sends a request for dungeon data, while `RequestRoomGAN` handles room data requests. This integration allows for dynamic and real-time generation of dungeon and room layouts, enhancing procedural content generation capabilities in Unity game development.

6 Conclusion

This project successfully developed a deep learning-based system to generate room decorations in dungeon environments using Conditional Generative Adversarial Networks (CGANs). Key achievements include effective data preprocessing and augmentation, designing and training CGAN models, and integrating these models into an interactive application for real-time generation. The models demonstrated high-quality, diverse, and contextually appropriate room layouts, evaluated using metrics such as Fréchet Inception Distance (FID), precision, and recall. The integration into a Unity-based application via a Flask backend facilitated seamless interaction and visualization, enhancing the game development process. This project highlights the potential of neural networks in procedural content generation for game development.

7. References

1. Volz, V., Schrum, J., Liu, J., Smith, A. M., Risi, S., & Togelius, J. (2018). "Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network." *IEEE Transactions on Games*, vol. 10, no. 3, pp. 209-220.
2. Zhu, J., Zhang, R., Pathak, D., Darrell, T., Efros, A. A., & Wang, O. (2020). "Multimodal Image-to-Image Translation by Enforcing Bi-Directional Consistency." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 5, pp. 1089-1100.