F-060

2D アクションゲームにおける GAN による多様なステージ生成手法の検討
A method of various level generation using GAN in the 2D video game

高田 宗一郎[1)]　清 雄一[1)]　田原 康之[1)]　大須賀 昭彦[1)]
Soichiro Takata　Yuichi Sei　Yasuyuki Tahara　Akihiko Ohsuga

1　はじめに

Procedural Content Generation (PCG) とよばれる，ゲームコンテンツの自動作成技術の研究が，ビデオゲームが誕生して以降さかんに行われている．ゲームコンテンツの自動生成に関する研究の中で最も多いのがゲームの環境やステージを生成する研究である．ステージの自動生成はゲームの難易度や面白さに直結する部分であるが，人手による生成はコストが大きいため長年にわたって自動生成の研究が行われてきた．最近ではアルゴリズムによる自動生成ではなく深層学習や強化学習が用いられる事例が増えてきている [1].

ゲームステージの自動生成において，その多様性の確保は重要な課題である．AI の評価，学習を行うシミュレーション環境への適用を考えても，ステージが多様であることは AI の汎化性能の評価および向上のために必要であると考えられる．加えて，生成したステージがゲームの制約を遵守し，プレイおよび攻略が可能であることも当然必要であり，ゲームステージの自動生成においてはこの 2 つを両立させることが必要となる．

本研究では，簡易的な 2D アクションゲームの環境である General Video Game AI Framework[2] の Zelda 環境を用いて，深層生成モデルにより生成ステージのプレイアビリティと多様性を両立したステージ生成手法を検討する．畳み込みと Self-Attention を組み合わせたモデルを用いて GAN の学習を行うことによりステージを生成し，生成されたステージのプレイ可能割合とその多様性を評価した．生成ステージ間の距離を拡大する制約を Generator の損失関数として追加して GAN の学習を行うことで，本研究で導入した多様度を評価する指標のもとで，生成されるステージの多様性が定量的に向上することが示された．

2　関連研究

2.1　Generative Adversarial Network(GAN)

深層学習による生成モデルの 1 つに，敵対的生成ネットワーク（GAN）がある．GAN は Goodfellow ら [3] が 2014 年に提案して以降，主に画像の生成モデルとして多くの研究が行われており，画像生成においては著しい成果をあげている生成モデルである．最も基本的な GAN では，データの生成を行うニューラルネットワークである Generator に加え，Generator により生成されたデータと実データを判別するネットワークである Discriminator を組み合わせ，交互に学習を行うことで，Generator の生成品質を向上させる．具体的には，式 (1) の形で定式化された価値関数 $V(D,G)$ によるミニマックスゲームを通して Generator$G$ が学習される．

$$\min_D \max_G V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log D(G(z))] \quad (1)$$

1）　電気通信大学 大学院情報理工学研究科
Graduate School of Informatics and Engineering, The University of Electro-Communications

近年ではその生成品質や学習の安定性，編集性能の向上など様々な研究が行われている．生成品質向上の研究では，Zhang ら [4] は，Generator と Discriminator の両方に，入力の大域的な特徴をとらえることのできる Self-Attention モジュールを導入することで，それまでの主に畳み込み層による GAN では困難だった，画像領域内で離れた情報が矛盾しないような高精細な画像生成が可能となったことを示した．

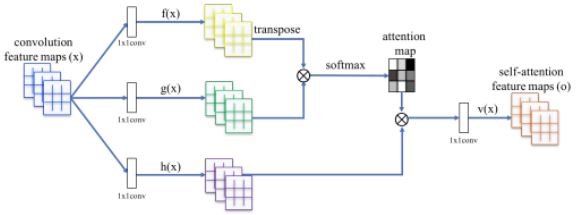

図 1　Self-Attention モジュールのアーキテクチャ．[4] より引用

2.2　Procedural Content Generation(PCG)

ビデオゲームのコンテンツの自動生成は，Procedural Content Generation と呼ばれ，40 年以上研究が行われており，様々なゲームコンテンツがアルゴリズムによって生成されてきた．PCG の分野の中でも，ゲームステージの生成は最もポピュラーなものであり，特に 2D のアクションゲームである Super Mario Bros. のような横スクロールのアクションゲームやローグライクゲームと呼ばれる探索型のアクションゲームなどが多く研究の対象として用いられている [5].

近年，画像生成や文章生成等で深層学習が著しい成果をあげていることから，PCG の分野においてもその有用性が期待され，ゲームコンテンツの生成に深層学習が用いられる研究が盛んにおこなわれており [1]，ステージ生成においても，深層生成モデルや深層強化学習が用いられる研究は多く存在する [6, 7, 8, 9]．特に，GAN を用いたステージ生成の研究が成果をあげている．Volz ら [10] は DCGAN[11] を用いて Super Mario Bros. のステージ生成を行い，GAN の潜在変数を進化計算により最適化することで指定した目的を満たすステージを生成する手法を提案した．Torrado ら [12] は，Generator と Discriminator に畳み込みによるモデルではなく，Self-Attention をベースとしたモデルを用いて本研究と同じ GVGAI Framework の Zelda 環境でステージ生成を行っている．Self-Attention Map にステージ上の各タイルの数の情報を結合して条件つき生成を行うことで，従来の畳み込みをベースとしたモデルによる生成よりも生成したステージのプレイアビリティが向上し，重複率も低くなることを示した．また，生成データを学習途中で GAN の正解データとして組み込むことで，5 つというごく少量のデータセットによる学習でも 47%のプレイアビリティと 60.3%の重複率を達成したことが報告されて

いる．我々の研究 [13] では，この研究から着想を得て，データセットの拡張を行い，畳み込み層と Self-Attention 層を両方用いたモデルによりステージの生成を行うことで，[12] で報告されたプレイアビリティを上回る成果を達成した．本研究では [13] の手法をベースとし，学習過程で損失関数による多様化の制約を課すことにより生成したステージの多様性を向上させることを目指す．

3　環境について

ステージ生成を行う環境として，本稿では 2D アクションゲームの環境である GVGAI Framework の Zelda 環境を用いた．この環境ではプレイヤーはグリッド状のステージ上を動き，ステージ上に配置された鍵オブジェクトを取得し，ゴールのマスに到達することでクリアとなる．ステージ上には敵オブジェクトが存在することがあり，ゲームクリアの為には避けるかプレイヤーの攻撃によって倒さなければならない．この環境ではステージは壁のマス，床のマス，敵のマスなど全 8 種のマスから構成されているとみなすことができる．本研究では，生成モデルは各マスについて，8 種類のタイルから 1 種類のタイルを選択することによりステージを生成する．ただしゲームの仕様上 Valid なステージであるためには以下のような制約が守られる必要がある．

- 「プレイヤー」マスはただ一つ存在する．
- 「鍵」マスはただ一つ存在する．
- 「ゴール」マスはただ一つ存在する．
- 「プレイヤー」マスから「鍵」マスおよび「ゴール」マスに到達可能である．
- ステージの上下左右は「壁」マスに囲まれている．

したがって，Zelda 環境のステージ生成においては，上記の制約をすべて満たしたステージを生成する必要がある．本研究では，ステージのサイズは縦 12 マス，横 16 マスとした．

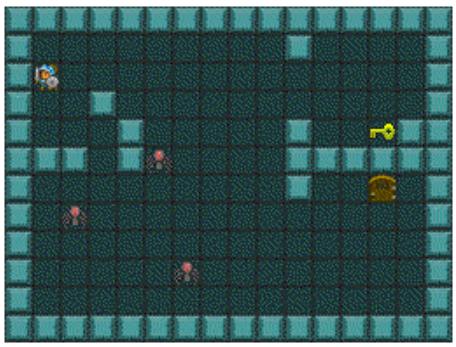

図 2　Zelda 環境のステージの例

4　提案手法

4.1　ベースモデル

我々の以前の研究 [13] では，GAN の Generator と Discriminator 両方に畳み込み層と Self-Attention モジュールを用いたモデルを採用することで，ステージの大域的な依存情報を効率的にとらえることにより生成ルールを守り，プレイアビリティの高いステージ生成に成功している．本研究でもこのモデルを用いる．Generator および Discriminator のアーキテクチャの概略図を図 3，4 に示す．

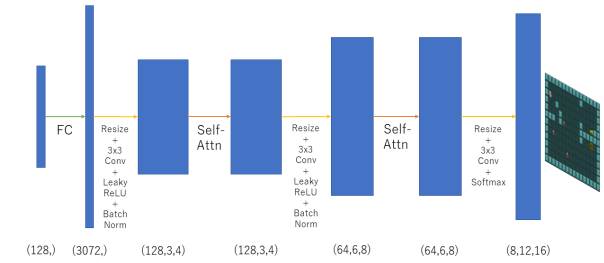

図 3　Generator の概略図

図 4　Discriminator の概略図

学習時には，Goodfellow ら [3] が提案した損失関数に，Generator の勾配消失が起こりにくいような工夫 [14] を施した以下の損失関数を用いる．

$$L_G = -\mathbb{E}_{z\sim p(z)}[\log D(G(z))] \tag{2}$$

$$L_D = -(\mathbb{E}_{x\sim p_{data}(x)}[\log(D(x))] + \mathbb{E}_{z\sim p(z)}[(1-\log(D(G(z))))]) \tag{3}$$

Generator の出力をステージに変換する際はチャンネル方向に Softmax 関数を適用し，最も値の大きいチャンネルに対応するタイルを選択する．Generator と Discriminator 両方に Self-Attention モジュールを導入することで，大域的な依存関係を考慮したステージの生成および判別がしやすくなることが期待される．今回の Zelda 環境においては，特定のオブジェクトが正確に 1 つ存在しなければならないという制約が守られやすくなる（≒プレイ可能なステージが生成されやすくなる）ことが期待できる．

4.2　提案手法

本研究では生成ステージの多様性を向上するために，損失関数を変更することにより GAN の学習過程において制約を加える．Bontrager らの研究 [7] では，Generator の生成ステージの多様性を向上させるため，出力のミニバッチ内のランダムなペア同士の L2 距離の平均値を最大化するようにネットワークの重みの更新を行っていた．本研究ではこの手法を GAN の学習過程に組み込み，学習過程でモデルに対し生成するステージの多様性が向上するような制約を加えている．具体的には，式

(2) の Generator の損失関数を以下の式 (4) に置き換える．

$$L_G = -\left(\mathbb{E}_{z\sim p(z)}[\log D(G(z))] + \lambda_{div} L_{div}\right) \quad (4)$$

$$L_{div} = \sum_{i=0}^{N-1} d(G(z_i), G(z_{i+1})) \quad (5)$$

ここで，λ_{div} は定数，関数 d は距離による損失関数であり，L1 損失または L2 損失 (平均二乗誤差) を用いる．このように Generator の学習時にミニバッチ内のデータ間の距離を拡大する制約を加えることで，Generator がより多様なデータを生成しやすくなることを期待する．

4.3 生成データによるデータセットの拡張

データセットに対しても，より多様性を持たせるため Torrado らの研究 [12] 同様に生成データをデータセットに追加する処理を行っている．本研究では具体的には学習の過程で，学習 1 エポックごとに 300 個のステージ生成を行い，生成したステージのうち 3 節で示したプレイ可能な条件を満たしたステージについてのみ，ランダムなデータセット内のデータと入れ替える処理を行う．

5 実験

5.1 データセットの準備

収集が比較的容易な人間の顔や風景などの写真に比べ，基本的に人間が制作する必要があるゲーム内コンテンツは数が少ないため，一般的に Procedural Content Generation 分野においてこのデータセットの準備は課題となっている．数が少ない偏ったデータでの GAN の学習は，一般にも Discriminator の過学習が原因でうまくいかないことが知られている．本研究では，ステージは縦 12 マス，横 16 マス，各マスに対し 8 種類の候補があることから，ステージを 8 チャンネル縦 12 ピクセル横 16 ピクセルの画像，すなわちサイズ (8,12,16) のテンソルとして表現する．GVGAI の Zelda 環境においては，デフォルトで 5 種類のステージが用意されているが，本研究ではデータセットの拡張を行い，データ数を増やす．具体的には，各ステージに対して以下のような水増しを行う．

- 垂直方向，水平方向の反転 (4 倍)
- ステージの各マスに対し，確率 0.05 でタイルの置き換えを行い新しいステージを生成する (250 倍)

以上のデータ拡張を行うことで，データ数を 5×4×250 = 5000 個とし，GAN の学習を行う．

5.2 評価手法

評価には生成されたステージの見た目による評価に加え，プレイアビリティについて定量的な評価を行う．具体的には学習したモデルに 1000 個のステージを生成させ，それらのうちどのくらいの割合のステージが 3 節で示したプレイ可能なステージの条件を満たしているかの定量評価を行う．評価に使用するモデルは，学習過程で前述したプレイアビリティの評価指標が最も高かった時点でのモデルを使用する．また，本研究では生成したステージの多様性を定量的に評価するため，以下のような指標を導入する．

Level Similarity

生成したステージ群のすべてのペアに対し，同じ位置にあるタイルが同種である割合を算出し，平均したもの．

Duplicate Rate

生成したステージ群のすべてのペアに対し，上記の類似度が 90%を超えたペアの割合

5.3 パラメータ設定

GAN の学習に用いたパラメータは表 1 の通りである．

表 1 実験における各パラメータ

ジェネレータの入力変数の次元数	128
バッチサイズ	64
学習ステップ数	5×10^4
最適化手法	Adam
Generator の学習率	1.0×10^{-4}
Discriminator の学習率	1.0×10^{-4}
損失 L_{div} の重み係数 λ_{div}（L1）	100.0
損失 L_{div} の重み係数 λ_{div}（L2）	50.0

6 結果

提案手法を評価するため，損失関数 (2)，(3) により学習を行ったベースライン手法と，Generator 側の損失関数を式 (4) の形に変更した提案手法とで，各評価指標の定量的な比較を行った．提案手法においては，多様性向上のための距離による損失関数 d を L1 損失にした場合，L2 損失にした場合それぞれで比較を行った．比較の結果を表 2 に示す．同じ設定であったとしてもモデルの学習 1 回ごとの各指標のばらつきが大きいため，本論文では各設定で 5 つのモデルに学習を行わせ，各モデルについて各評価指標の最小値と平均値，最大値を算出している．

また，ベースライン手法，提案手法 (L1 損失,L2 損失) それぞれにより生成されたステージの例を図 5,6,7 に示す．

表 2 提案手法とベースライン手法の各指標の値の比較 (%,min/mean/max)

	ベースライン	提案手法,L1 損失	提案手法,L2 損失
Playablity↑	75.4/**83.9/88.6**	76.8/78.8/81.1	**76.9**/82.3/86.7
Similarity↓	80.8/83.3/85.6	**68.1/69.4/71.4**	69.0/72.1/75.4
Duplicate Rate↓	14.7/23.6/32.1	**6.0/8.0/11.4**	6.1/9.2/14.8

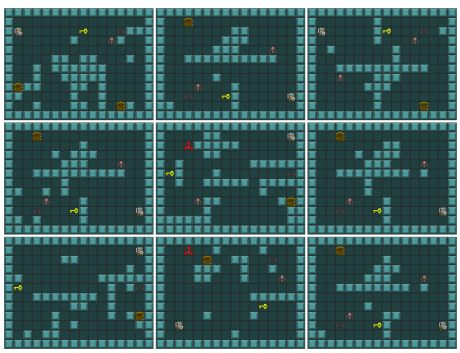

図 5 ベースライン手法により生成されたステージ (9 個)

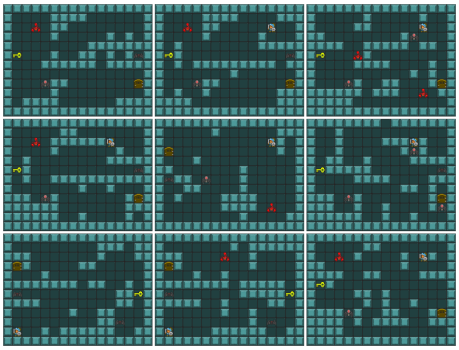

図 6　提案手法 (L1 損失) により生成されたステージ (9 個)

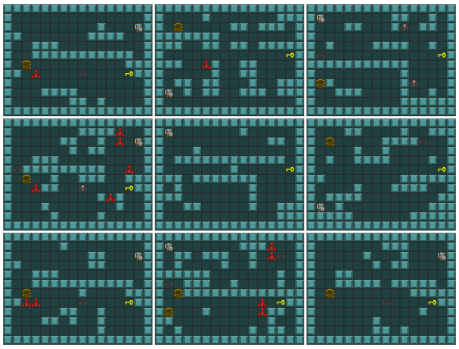

図 7　提案手法 (L2 損失) により生成されたステージ (9 個)

7　考察

実験では表 2 のとおり，提案手法による学習を行ったモデルの方が今回用いた指標の上では，定量的に多様性が向上していることが示された．また，図 5,6,7 を比較すると，ベースライン手法により生成されたステージに比べ，提案手法により生成されたステージは全体的に壁マスや敵オブジェクトの数が多くなっており，これにより定量的に多様性が向上しているものと考えられる．一方で，プレイヤー，鍵，ゴールなどのただ一つ存在するべきマスについては特定の場所に生成されやすい傾向がある．これらのマスはデータセットの水増し時に場所が変更されないため，オリジナルのデータの数が少ない本環境ではデータセットの時点で配置のパターンが少なく，ある程度固定化されてしまっているものと考えられる．この課題の解決は生成データ間の距離を拡大する本手法では難しく，データセット拡張の際の工夫や，モデル構造の工夫などが必要だと考えられる．

また，今回は λ_{div} を 100.0 および 50.0 として実験を行った．この場合 L_{div} はオリジナルの Generator のロスの数倍程度のスケールとなるが，多様性を向上させつつプレイ可能性を落とさずに学習を行うにはこの程度の値が良いようである．この値が小さいと多様性は低くなり，大きいと多様性は大きくなるがプレイ可能性が低下するということが実験時にみられ，今回の手法では多様性とプレイ可能性の間にトレードオフのような関係があることが分かった．

8　まとめ

本研究では GVGAI の Zelda 環境の GAN によるステージ生成において，畳み込みと Self-Attention を用いたモデルで生成したステージ間の距離を拡大する制約を加えて GAN の学習を行うことで，ランダムノイズ入力から既存手法よりも多様なステージを生成できることを示した．一方で，オリジナルのデータセットが少ないことから特定のオブジェクトの配置が偏るという問題がみられた．一般にステージのデータセットを偏りなく十分に用意することは困難であるため，少量のデータにより全体的な特徴を学習しつつ，生成時に外部からの入力によりオブジェクトの配置を制御可能であるような生成モデルの獲得が今後の課題である．また，横スクロールアクションのような別のドメインやより大きなステージへの適用についても検討していく．

謝辞

本研究は JSPS 科研費 JP21H03496， JP22K12157 の助成を受けたものです．

参考文献

[1] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius. Deep learning for procedural content generation. *CoRR*, Vol. abs/2010.04548, , 2020.

[2] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms, 2018.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[4] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2018.

[5] Noor ShakerJulian TogeliusMark J. Nelson. *Procedural Content Generation in Games*. SpringerLink, 2016.

[6] Linus Gisslén, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, and Konrad Tollmar. Adversarial reinforcement learning for procedural content generation. *CoRR*, Vol. abs/2103.04847, , 2021.

[7] Philip Bontrager and Julian Togelius. Learning to Generate Levels From Nothing. *arXiv:2002.05259 [cs]*, August 2021. arXiv: 2002.05259.

[8] Tianye Shu, Jialin Liu, and Georgios N. Yannakakis. Experience-Driven PCG via Reinforcement Learning: A Super Mario Bros Study. *arXiv:2106.15877 [cs]*, July 2021. arXiv: 2106.15877.

[9] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. PCGRL: Procedural Content Generation via Reinforcement Learning. *arXiv:2001.09212 [cs, stat]*, August 2020. arXiv: 2001.09212.

[10] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam M. Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. *CoRR*, Vol. abs/1805.00728, , 2018.

[11] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[12] Ruben Rodriguez Torrado, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, and Julian Togelius. Bootstrapping conditional gans for video game level generation. *CoRR*, Vol. abs/1910.01603, , 2019.

[13] 高田宗一郎, 清雄一, 田原康之, 大須賀昭彦. 2d アクションゲームにおけるプレイ可能なステージを生成しやすい生成モデル. 電子情報通信学会 「人工知能と知識処理」研究会 (AI), 2022.

[14] Martin Arjovsky Soumith Chintala, Emily Denton and Michael Mathieu. soumith/ganhacks. https://github.com/soumith/ganhacks, 2016.