ゲームプレイ特徴のクラスタリングによるローグライクゲームの動的難易度調整

阿保 達也[1,a)]　松原 仁[2,b)]

概要：コンピュータゲーム開発において、レベルデザイン、つまり難易度の調整は非常に重要なフェーズであるとともに、手間がかかりそれ自体が難しい。本研究では、ローグライクゲームのプレイ特徴を抽出、クラスタリングすることで、その結果を動的難易度調整に適用し、プレイ内容に応じて実際に難易度を動的に調整することを目的とする。

Dynamic difficulty adjustment in roguelike game using clustering of gameplay features

Tatsuya Abo[1,a)]　Hitoshi Matsubara[2,b)]

Abstract: In computer game development, level design, that is, adjustment of the difficulty level, is a very important phase, and takes time and effort itself. The purpose of this research is to extract and cluster the play characteristics of roguelike games, apply the results to dynamic difficulty adjustment, and adjust the difficulty dynamically according to the play contents.

1. はじめに

1.1 背景

コンピュータゲームにおいて、難易度の適切な調整はプレイヤーが感じる没入感や楽しさに大きく関わることがわかっている [1]。しかし、多くのプレイヤーが楽しめる難易度にデザインするのは困難であり、調整にも時間がかかる。

1.2 目的

本研究では、クラスタリング手法を用いて、コンピュータゲームで動的な難易度調整を行う仕組みを提案する。動的難易度調整での動的というのは、「ゲームプレイに応じて自動的に」という意味である。事前に集めたデータに応じて難易度を変化させることで、自動的な難易度調整を行う。本研究では、動的難易度調整をゲーム内に導入することによって「ゲームにおけるレベルデザインの手間の緩和」と「プレイヤーの操作に合わせたゲームの実現」を図ることを目的とする。

2. 関連研究

2.1 動的難易度調整の例

動的難易度調整研究の例として、「フローチャンネルを利用したゲームステージのオンライン自動生成 [2]」が挙げられる。この研究は、ニューラルネットワークでの学習によりプレイングスキルを判別し、フローチャンネルに入るような難易度のステージ断片を生成する。それにより、動的難易度調整のシステムを実現している。このシステムでは、おおむねプレイヤーのスキルに合わせたステージを生成していたが、難易度が高いステージが多めに生成されている。上記の研究でのフローとはミハイ・チクセントミハイによって提唱されたものである [3]。

[1] 公立はこだて未来大学大学院システム情報科学研究科
Graduate School of System Information Science, Future University Hakodate
[2] 公立はこだて未来大学
Future University Hakodate
[a)] g2118001@fun.ac.jp
[b)] matsubar@fun.ac.jp

2.2 k-means++

本研究では、クラスタリング手法として k-means++を適用する。k-means++とは、David Arthur らによって提唱された、k-means の初期値選択に改良を加えた手法である [4]。

3. 実験環境

本研究で使用するゲームは、ゲームエンジンである「Unity」を用い作成する。クラスタリングの実装には Python のオープンソースライブラリである「scikit-learn」を使用する。

3.1 ローグライク

本研究では、「ローグライク」を対象とするゲームジャンルとする。「ローグライク」とは、ダンジョン探索を行うロールプレイングゲームのジャンルの一つである。実装するローグライクは「Berlin Interpretation[5]」に基づいて、ローグライクを構成する一般的なルールを適用する。

図 1 ローグライク

3.1.1 ローグライクのフィールド

本研究でのローグライクのフィールドは、複数階層で構成されていて、進めるだけ進むことができる。また、フィールドは「部屋」「道」「壁」で構成されている。「部屋」「道」は移動可能であり、「壁」は移動、侵入不可能である。「部屋」の中に「階段」があり、プレイヤーが「階段」に到達することで次の階層に進む。本研究では、9 部屋作成し、隣り合う部屋は道を作成する。

4. 提案手法

4.1 動的難易度調整システム

本研究では、動的難易度調整を行うシステムを「要素登録フェーズ」「ログ収集フェーズ」「クラスタリングフェーズ」「動的難易度調整フェーズ」の 4 つで構成する。

4.2 要素登録フェーズ

難易度に関わる、動的変更を行いたいゲーム内の要素を登録し、また、ログとして収集するデータを登録する。動的変更を行いたいゲーム内の要素は、0 から 1 の値で定義する。

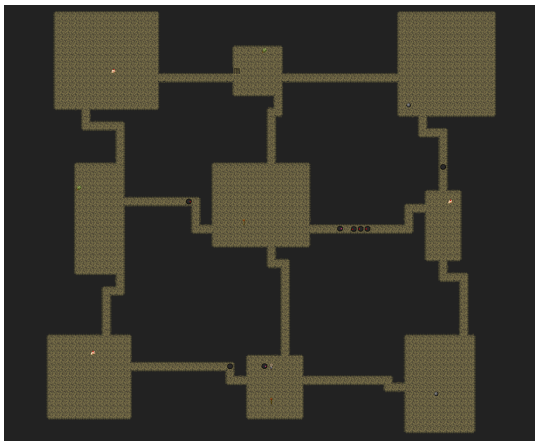

図 2 ローグライクのフィールド

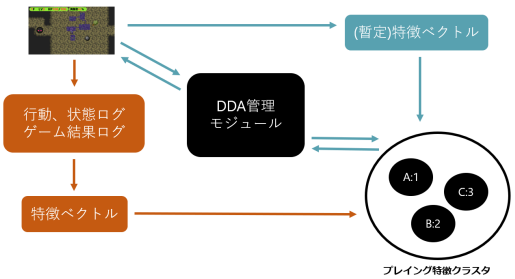

図 3 動的難易度調整の流れ

4.3 ログ収集フェーズ

ゲームプレイ中にログを収集する。収集するログとして、「行動」「状態」「結果」のログの 3 種類で定義する。

4.3.1 行動ログ

ゲームの入力のログである。各ターン 0 か 1 の値で保存する。

4.3.2 状態ログ

ゲームの状態 (パラメータ等) のログである。0〜1 の値で保存する。

4.3.3 結果ログ

ゲームの結果 (評価等) のログである。0〜1 の値で正規化する。

4.4 クラスタリングフェーズ

ログ収集フェーズで得たログを特徴ベクトル、平均ラベル値ベクトルに変換する。それぞれの特徴ベクトルを平均ラベル値ベクトルでラベル付けをし、その後、特徴ベクトルをクラスタリングする。手法として k-means++を使用する。

4.4.1 特徴ベクトル

各ターン毎の行動ログを行動ベクトル a、各ターン毎の状態ログを状態ベクトル b として扱い、要素ベクトル e を

以下のように定義する。ここで、n_1 は行動ログとして登録した要素数、n_2 は状態ログとして登録した要素数、T は転置記号、τ を各階層のターン数とする。

$$e_{t_1} = (a_1, a_2, \cdots, a_{n_1}, b_1, b_2, \cdots, b_{n_2})^T_{t_1} \quad (t_1 = 1, 2, \cdots, \tau) \tag{1}$$

そして、特徴ベクトル x を以下のように定義する。ここで、w は平均化間隔とする。

$$x_{t_2} = \frac{1}{w} \sum_{i=t_2}^{t_2+w} e_i \quad (t_2 = 1, 2, \cdots, \tau - w) \tag{2}$$

4.4.2 平均ラベル値ベクトル

各ターン毎の結果ログを結果ベクトル r として定義し、ラベル値ベクトル l を以下のように定義する。ここで m は結果ログとして登録した要素数とする。

$$l_{t_1} = (r_1, r_2, \cdots, r_m)^T_{t_1} \tag{3}$$

そして、平均ラベル値ベクトル y を以下のように定義する。

$$y_{t_2} = \frac{1}{w} \sum_{i=t_2}^{t_2+w} l_i \tag{4}$$

4.4.3 クラスタの優劣

各クラスタに属する、各特徴ベクトルにラベル付けした平均ラベル値ベクトル (4) のユークリッドノルムの平均値をプレイングスキル p として定義し、これをクラスタの優劣として扱う。ここで、ν をクラスタ数、ω をクラスタ、$|\omega|$ をクラスタに含まれる要素数とする。

$$p_i = \frac{1}{|\omega_i|} \sum_{y_{t_2} \in \omega_i} \|y_{t_2}\| \quad (i = 1, 2, \cdots, v) \tag{5}$$

4.5 動的難易度調整フェーズ

動的難易度調整フェーズは、以下の手順で行う。

(1) ゲームプレイ中に特徴ベクトルを計算
(2) 特徴ベクトルとクラスタリングフェーズで計算したクラスタの重心との距離比較で一番近いクラスタを計算
(3) そのターンの距離が近いクラスタのプレイスキル値を保持
(4) 前のターンのプレイスキル値と今のターンのプレイスキル値を比較し、今のターンのプレイスキル値のほうが前のターンのプレイスキル値より高ければ、要素登録フェーズで登録した動的変更を行いたいゲーム内の要素の値を上昇させ、逆であれば下降させる。

5. 環境設定

ゲームプレイのログを収集するにあたってルールベースの AI を用いて計 50 ゲームプレイさせた。ルールベース AI のアルゴリズムを Algorithm 1 に示す。

```
初期化;
while 体力 > 0 do
    if アイテム使用判定 then
        アイテムを使用;
    else
        if 部屋にいる then
            if 部屋に階段がある then
                階段に向かって進む;
            else
                if 敵が周囲 5 マス内にいる＆敵が同じ部屋にいる then
                    if 敵が周囲 1 マス内にいる then
                        敵を攻撃する;
                    else
                        敵に向かって進む;
                    end
                else
                    部屋の出口に向かって進む;
                end
            end
        else
            if 敵が周囲 1 マス内にいる then
                敵を攻撃する;
            else
                道の出口に向かって進む;
            end
        end
    end
end
```

Algorithm 1: ルールベース AI のアルゴリズム

5.1 動的変更を行うゲーム内の要素

(1) 各部屋の敵数
(2) 各部屋のアイテム数

5.2 実験で使用するログ

5.2.1 行動ログ

(1) 上下左右斜め移動 (8 種)
(2) 攻撃
(3) 待機
(4) アイテム使用 (4 種)

5.2.2 状態ログ

(1) 体力
(2) 満腹度

5.2.3 結果ログ

(1) 到達階層数

5.3 学習に適したクラスタ数、平均化間隔の選定

k-means++でのクラスタリングを行うにあたり、クラスタ数によってクラスタ分類のパフォーマンスが大きく異なることが予想される。そこでクラスタ内誤差平方和 (SSE) の分析、シルエット分析によってクラスタ数の検証を行う。また、本論文 4.4.1 節で述べたように、特徴ベクトルを形成するにあたり、前後の時系列で平均化する。この平均化間隔によって、SSE とシルエットスコアの違いが出るかどうかも検証する。

5.3.1 クラスタ内誤差平方和 (SSE) の分析

それぞれのクラスタ数で、クラスタ内誤差平方和 (SSE) を計算する。SSE とは、下式で示される、各クラスタに属するベクトルとそのクラスタの重心の距離の和である。ここで、S を SSE、c をクラスタの重心とする。

$$S = \sum_{i=1}^{\nu} \sum_{x_{t_2} \in \omega_i} \|x_{t_2} - c_i\| \tag{6}$$

また、平均化間隔によって形成されるベクトルの数が異なる。平均化間隔が短いほど多くベクトルが形成され SSE の値が大きくなるので、SSE を全クラスタに含まれる特徴ベクトル数で割った値、MSSE を定義する。ここで、M を MSSE、Ω を全クラスタに含まれる要素の集合とし、$|\Omega|$ を全クラスタに含まれる要素数とする。

$$M = \frac{S}{|\Omega|} \tag{7}$$

平均化間隔 2 と 30 のクラスタ数毎 MSSE を図 4 に示す。ここで、MSSE を SSE を全クラスタの特徴ベクトル数で割ったものと定義する。また、図 4 の値の差分を図 5 に示す。

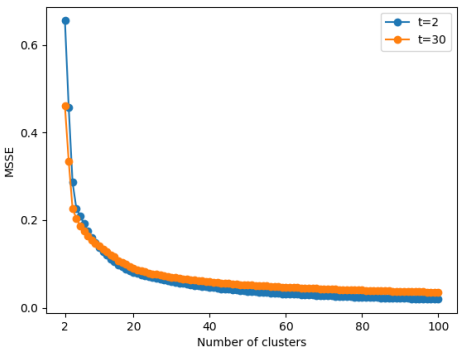

図 4 平均化間隔 2、30 のクラスタ数毎 MSSE

図 5 から、クラスタ数間の SSE の変動が大きく、かつその後変動幅が少なくなっているのは、クラスタ数 5-6 間である。すなわち、平均化間隔 2、もしくは 30 の場合、クラスタ数 6 で計算するとうまく特徴ベクトルをクラスタリングできると推測される。また、クラスタ数 2〜100 の平均

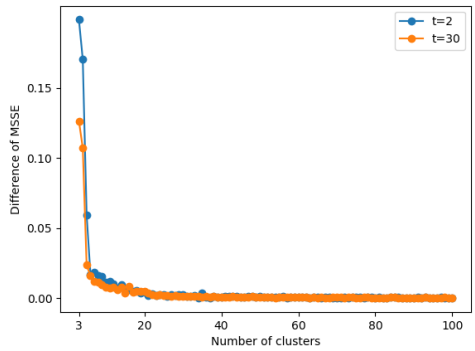

図 5 平均化間隔 2、30 のクラスタ数毎 MSSE の差分

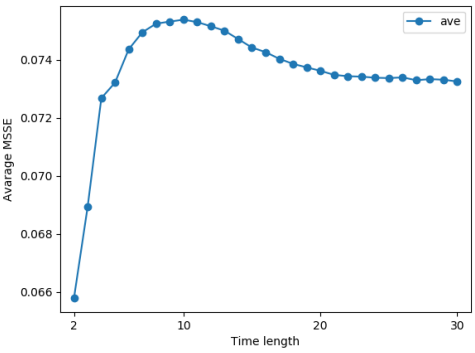

図 6 平均化間隔毎の MSSE の平均

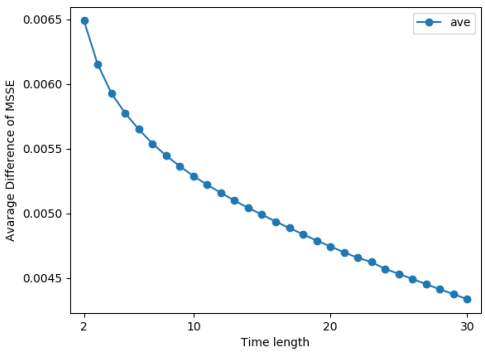

図 7 平均化間隔毎の MSSE の差分の平均

化間隔毎の MSSE の平均とその差分を図 6、図 7 に示す。

図 6 から、MSSE が高い平均化間隔として、7〜13 ターンが挙げられる。また、図 7 からは、平均化間隔が短いほど MSSE が収束しにくいことが推測される。

5.3.2 シルエット分析

平均化間隔 2、30 のクラスタ数毎のシルエットスコアを図 8 に示す。

図 8 から、クラスタ数 3〜8 でのシルエットスコアが高いので、クラスタ数 3〜8 で計算すると分類の精度が高い。また、クラスタ数 2〜100 の平均化間隔毎のシルエットス

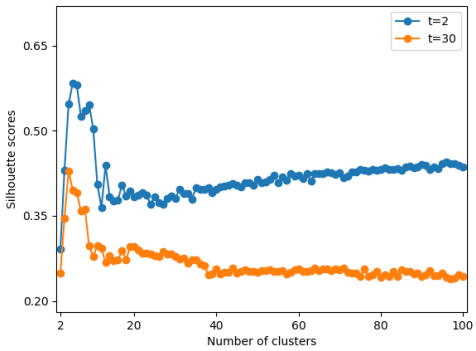

図 8　平均化間隔 2、30 のクラスタ数毎のシルエットスコア

コアの平均を図 9 に示す。

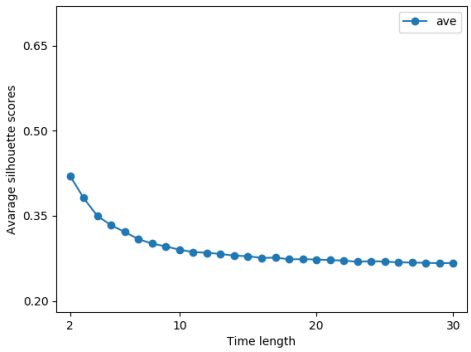

図 9　平均化間隔毎のシルエットスコアの平均

図 9 からは、平均化間隔が短いほど平均シルエットスコアが高く、クラスタリングの精度が高いことがわかる。

5.3.3　SSE 分析とシルエット分析を踏まえた選定

上記の SSE 分析とシルエット分析を踏まえ、クラスタ数 6、平均化間隔 5 で実験を行う。平均化間隔は、図 7 では、短いほど分類の精度がいい。しかし、平均化間隔を増やすことで、特徴ベクトルに前後の時系列の情報を含めることができると仮定し、図 6 から高い値になる直前の値を選択した。

6.　実験と結果

環境設定で定めた条件で提案手法を用いてプレイングに応じた難易度調整が行われているか調べる。

6.1　重心と特徴ベクトルの距離変化の比較

動的難易度適用前と適用後の環境で 1 プレイ中のクラスタの重心との距離の推移を調べる。全クラスタの重心とゲームプレイ中に得られる特徴ベクトルとの距離の総和の 1 例として図 10、図 11 に示す。

図 10、図 11 のから得られる、難易度調整適用前と難易度調整適用後の「距離変化量の平均」を各 5 ゲームで計算

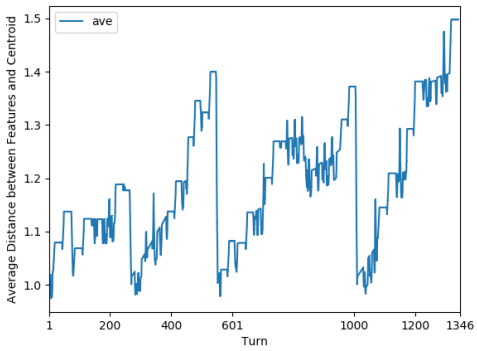

図 10　固定難易度の距離変化

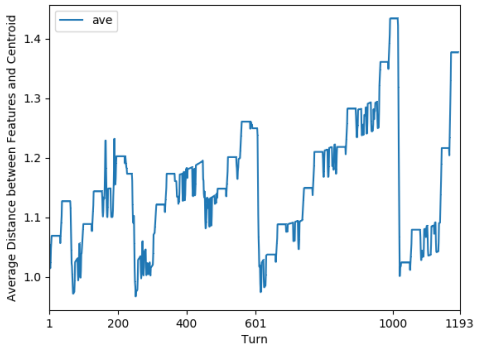

図 11　動的難易度調整時の距離変化

した。その結果を比較した結果、難易度調整適用前のターン間平均は 0.006239、難易度調整適用後のターン間平均は 0.005319 となった。難易度調整適用後ではプレイ中の特徴ベクトルの移動が平均的に小さく、もしくは移動そのものが少なくなっている。この結果と、ルールベース AI をデータ収集に使用していることを考慮すると、難易度調整によりプレイングスキルの推定がされていることが示唆される。

6.2　プレイングスキル (ゲーム結果) の比較

動的難易度適用前と適用後の環境でのプレイングスキル (5) の比較を行う。各 150 ゲームでのプレイングスキルの平均と分散を表 6.2 に示す。

表 1　固定難易度と動的難易度でのプレイングスキルの平均と分散

難易度	平均	分散
固定	3.36	2.04
動的	3.19	1.81

この結果により、動的難易度調整時のプレイングスキルのばらつきが抑えられている。つまりすぐにゲームが終了するケースや長くゲームが続くケースが少なくなっていることが示唆される。

7. まとめと今後の展望

本研究では、ローグライクゲームのプレイ特徴を抽出、クラスタリングすることで、その結果を動的難易度調整に適用し、プレイ内容に応じて実際に難易度を動的に調整するシステムを導入し、実際に難易度調整が行われているかを検証した。部分的に動的難易度調整が行われることが確認されたが、今回はルールベース AI でのプレイングデータを用いて実験検証したので、今後は人に対しての実験を行い、人に対しても難易度の調整がなされるのかを検証し、また、データ量を増やしさらに考察を深めたい。

参考文献

[1] Chen, J. (2007). Flow in games (and everything else). Communications of the ACM, 50(4), 31-34.

[2] 清水智行, 橋山智訓, 江崎朋人, 市野順子, & 田野俊一. (2011). フローチャンネルを利用したゲームステージのオンライン自動生成. In 日本知能情報ファジィ学会 ファジィ システム シンポジウム 講演論文集 第 27 回ファジィ システムシンポジウム (pp. 127-127). 日本知能情報ファジィ学会.

[3] Czikszentmihalyi, M. (1990). Flow: The psychology of optimal experience. New York: Harper & Row. Perennial.

[4] Arthur, D., & Vassilvitskii, S. (2007, January). k-means++: The advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (pp. 1027-1035). Society for Industrial and Applied Mathematics.

[5] Berlin Interpretation - RogueBasin, `http://www.roguebasin.com/index.php?title=Berlin_Interpretation`, (参照 2019-10-08)

[6] Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. Communications in Statistics-theory and Methods, 3(1), 1-27.

