

ArchiMate によるゲームアーキテクチャの実装実験

北野不凡, 山本修一郎
名古屋国際工科専門職大学

愛知県名古屋市中村区名駅 4-27-1

Experimental implementation of game architecture with ArchiMate

Kitano Masaru, Shuichiro Yamamoto

IPUT in Nagoya
4-27-1, Meieki, Nakamura-ku, Nagoya Aichi Japan

概要

筆者らによる「カジュアルゲームにおけるゲーム要素の分解・整理 ArchiMate で表現する提案」[1]においてカジュアルゲームを構成するゲーム要素やそれらを包含するコンポーネントの定義を提案した。

本稿では、分解、整理したコンポーネントやそれに含まれるゲーム要素を用いて実際にゲームを実装することの可否を確認する。具体的には、実装を実験するゲームは上記論文の「隕石避けゲーム」として以下の点を検証する。

1. 上記論文に従ってコンポーネントをクラス形式で実装できる。
2. 1 で実装したクラスを用いて上記論文で示した「隕石避けゲーム」を実装できる。

Abstract

In my March 2022 paper, "Decomposition and Organization of Game Elements in Casual Games: A Proposal for Representation in ArchiMate" [1], I proposed definitions of game elements that constitute casual games and the components that encompass them. In this paper, we confirm the feasibility of actually implementing a game using the decomposed and organized components and the game elements included in them. The game to be tested is the "Meteorite Avoidance Game" of the above paper, and the following points are verified: 1.

1. It is possible to implement the components in the form of classes according to the above paper.
2. Be able to implement the "Meteorite Dodging Game" from the above paper using the classes implemented in 1.

1. はじめに

2022 年 3 月に発表した「カジュアルゲームにおけるゲーム要素の分解・整理 ArchiMate で表現する提案」[1]では、カジュアルゲームのアーキテクチャを ArchiMate で表現し、ゲーム内容の把握をより正確にする方法を明らかにした。

しかし、この提案はゲームの実装には言及しておらず、実際にその ArchiMate 図から実装が可能かについては検証していなかった。

本発表ではこの点を補い、[1]で例に挙げた「隕石避けゲーム」をゲームエンジン Unity で実装を試みる。

まず[1]で挙げたコンポーネント層に沿って C# のクラスを設計する。

先に実装したクラスを用いてゲームそのものを実装する。

以下では、まず 2 節で関連研究を説明する。次いで、3 節でコンポーネント層をクラスに再設計する。

4 節では、具体的な事例への適用例を説明する。5 節で、考察を述べ、6 節でまとめと今後の課題を述べる。

2. 関連研究

以下では関連研究について説明する。

- 2.1 「カジュアルゲームにおけるゲーム要素の分解・整理 ArchiMate で表現する提案」[1]
ゲームを表現するにあたり、フローチャートや状態遷移図ではなくアーキテクチャーで表現し、その手段として ArchiMate を使用することを提案した。
- 2.2 ArchiMate によるゲームメカニクス表記ルールの提案[2]
[1]においてゲームアーキテクチャを ArchiMate で表現する事を提案したがその表記法を定めていないことによる表記揺れを防ぐために表記ルールを提案した。
- 2.3 ArchiMate

ArchiMate は、The Open Group が標準化する EA モデリング言語である[2].ArchiMate は 2002 年から 2004 年までオランダの産官学連携プロジェクトで開発された.その後 The Open Group で EA モデリング言語として標準化されることになり、2009 年に ArchiMate1.0 が公開された。最新版は 2019 年末に公開された ArchiMate3.1 である[3].

3. コンポーネント層のクラス設計

3.1 コンポーネントとクラス

図 1 に[1]で示したコンポーネント層を再掲する.ここには、階層構造の中に、以下のようなコンポーネントを定義している.

- GameManager
- InGame

InGame 内に以下のコンポーネントを包含してい

る.このコンポーネントはゲームの基本的なメカニクスを記述する.ゲーム内で使用するさまざまな共通機能は以下のコンポーネントに記述し随時利用する.

•制限管理

制限時間や残機数などゲームオーバーやゲームクリアに関連するデータを扱う.

•バリア管理

移動する敵や障害物、地形など自機以外のゲームオブジェクトに関連するデータを扱う.

•自機管理

自機やプレイヤーキャラクタに関するデータを扱う.ここには操作に関連する入力機能も含まれる.これらを Unity の C#によるクラスで表現する.このオブジェクト図を図 2 に示す.

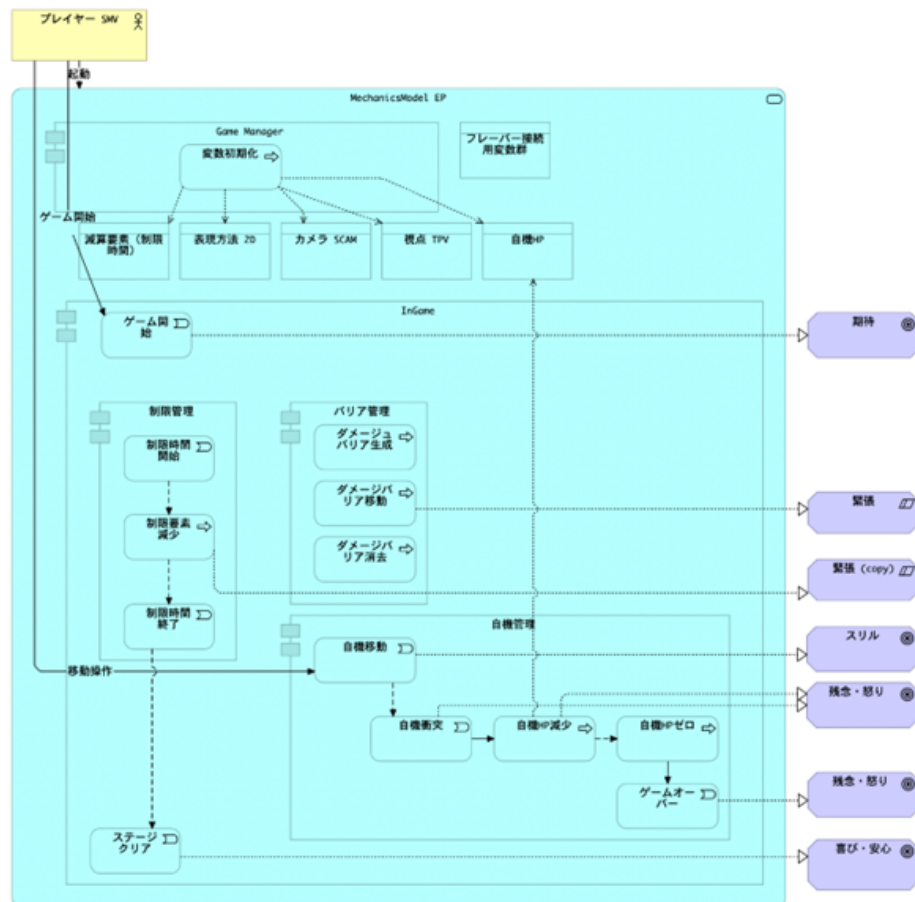


図 1 「隕石避けゲーム」のコンポーネント層

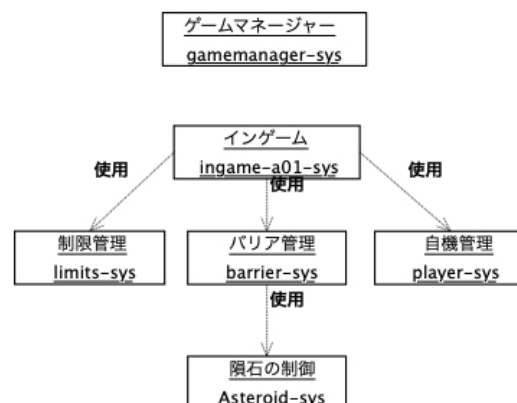


図 2 実装クラスのオブジェクト図

3.2 研究仮説

RQ1:[1]で示したコンポーネント層をクラスに変換できる

RQ2:RQ1 で実装したクラスを用いて「隕石避けゲーム」が実装できる。

4. 具体例

以下では、サンプルとして単純なメカニクスをもったゲームを実装する。

4.1 ゲーム内容

図 3 は[2]で示した当該ゲームのメカニクスをArchiMate で記述したものである。

これは制限時間中ゲームをプレイし、画面上部から降ってくる隕石を避け続けるという内容である。

自機は左右に動かすことができるが画面左右端にまでくると、動けなくなる。

プレイヤーが隕石を避けることに失敗し、自機が隕石と衝突するとゲームオーバー。制限時間まで耐えればクリアという内容となる。

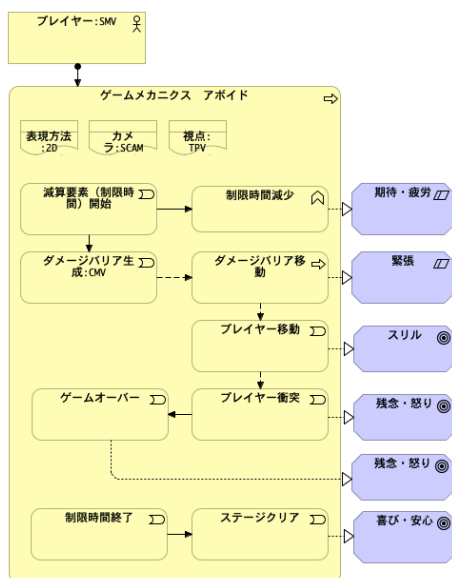


図 3 「隕石避けゲーム」のメカニクス

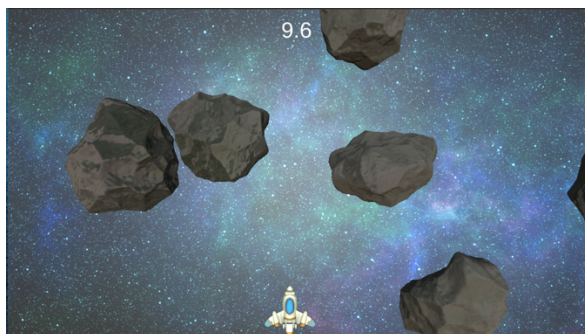


図 4 実装した「隕石避けゲーム」の画面

4.2 コンポーネント

図 1 に示すように、4 つのコンポーネントを定義した。

・GameManager

ゲームの基本的情報(カメラの位置や角度など)を記述する。

ゲームの状態遷移や進行を管理する。

・InGame

ゲームプレイのメカニクスを記述する。

下記 3 つのコンポーネントを内包する。

・制限管理

ゲームにおける一定の範囲を超えると生じられるイベントなどを管理する。

・バリア管理

ゲームにおける「敵」や「障害物」の動きや発生、消滅を管理する。

・自機管理

ゲームにおける自機やプレイヤーキャラクターを管理する。

4.3 クラス

4.2 で述べたコンポーネントを、以下のようなクラスとして実装する。

・GameManager

付録1にソースの一部を示した。ゲーム全般の設定を行う。付録1 ではカメラの設定を変数で設定する事でゲームの画面の準備を行い、状態機械を設定している。

・InGame

付録2 にソースの一部を示した。状態機械に沿ってインナークラスの Update メソッドの中で各種処理を行う。InGame のインナークラスとして、以下のようなクラスを実装した。

・LimitManage

・BarrierManage

・PlayerManager

・GameEconomics

その他、バリア要素(隕石)にアタッチするクラスとして Asteroid を実装した。

4.4 設定する主な変数

[2]においてゲームの初期設定の定義を行った。これに沿って主たるゲーム要素に対して属性を定義した。それらの属性や変数の具体的な名前と定義するクラスを定める。一例としてカメラを設定する変数を示す(表 1)。

表 1-1 カメラ変数 CameraType

変数名	CameraType	GameManegre-SYS
型	int	
値	0	SCAM 固定カメラ
値	1	FCAM 追従カメラ
値	2	DCAM 移動カメラ

表 1-2 カメラ変数 CameraView

変数名	CameraView	GameManegre-SYS
型	int	
値	0	TPV: トップビュー
値	1	SDV: サイドビュー
値	2	QTV: クォータービュー
値	3	TDV: 第三者視点
値	4	FTV: 一人称視点

表 1-3 カメラ変数 CameraPresen

変数名	CameraPresen	GameManegre
型	int	
値	0	3DI: アイソメトリック
値	1	3DP: パースペクティブ

今回の「隕石避けゲーム」の場合、図 4 のように画面下に自機、画面上に HUD、画面上から隕石が降る。よって、カメラ関係の変数は以下のように設定する。

CameraType:0 固定カメラ

CameraView:0 トップビュー

CameraPresen:0 アイソメトリック

となる。その他、当該ゲームに登場する要素としてゲームクリアを判定するとして制限付き加算要素、隕石として複数のダメージバリア、HUD 表示用に可変文字列がある。

4.5 設定する主なメソッド

図 1 に沿って実装するメソッドの主なものを表 2 に示す。

表 2 実装する主なメソッド

メソッド名	クラス名	用途
start	InGame	初期化
update	状態遷移	ゲームプレイ本体
Update	BarrierManager	隕石生成
Update	Asteroid	隕石消去
Update	PlayerManager	自機移動入力/描画
Update	LimitManager	制限時間判定

5. 考察

5.1 仮説の検証

本稿は[1]で示したコンポーネント層が実装可能かを判断する実装実験である。

【RQ1 の検証】具体例から[1]で示したコンポーネント層をクラスに変換できる事が可能と考える。

【RQ2 の検証】具体例で示したように RQ1 で実装したクラスを用いて「隕石避けゲーム」が実装できる。

5.2 留意点

今回実装したクラスはカプセル化のために、インナークラスを採用した。しかし、インナークラスのコンストラクタや Start メソッドにおいて prefab ゲームオブジェクトの Find や Instantiate の使用が許されていないことが判明した。そのため Update メソッド内でこれらの処理を行ったため実行速度が大きく低下してしまった。Unity での設計は開発言語の仕様に制限されるため、今後はより基本的な実験を経て実装や設計を行う必要がある。そのため次の機会を捉えて継承を用いたクラスの再設計をおこないたい。

「隕石避けゲーム」は最小限のメカニクスを実装したのみであって、ゲームをよりエキサイティングにするための諸要素は組み込まれていない。それらの要素を基本メカニクスの「アボイド」の中に組み込んでいく事が実用性を構築していく上でハードルとなる。これは今後の実験で改善していきたい。

5.3 限界

(1) 実験の限界

本稿の実験はコンポーネント実装のごく基本的な部分のみであるため、この試みが実用性を得られるか不透明である。

(2) 拡張性の限界

[1]の時点での設計では用途の拡大などについて考慮しておらず、純粋に「隕石避けゲーム」のための設計となっている。そのため、メカニクス層とその動作を下支えするコンポーネントが同一コンポーネントに収納されていた。これに沿って本稿では低水準なクラスを InGame クラスのインナークラスとして実装したが、当システムを他のメカニクスにも拡張していくと考えた場合、メカニクス毎にインナークラスの内容が異なる可能性があり、メンテナンス性に問題がある。そのため次稿以降ではクラス設計を考え直し、外部クラスとして実装することが望ましいと考えられる。

(3) フレーバー層の欠如

この段階でフレーバー層をサポートするコンポーネントの設計及びクラスの実装が行われていないため統一的なビジュアルやサウンドの投入が実装不可能な状態である。

次回実装実験では、考察で述べたクラス設計の変更および、フレーバー層の実装も含めてメカニク

スを拡張したものの実装も行いたいと考えている。



図 5 フレーバー層の概念図

6. まとめ

本研究の目的は、以下の通りである。

- ・ゲーム企画の表現法の標準的手法の作成
ArchiMate を使用するアプローチを採用する。
- ・ゲーム開発の容易化

数多く存在するゲームの中から共通要素を抽出し、機能別に分解・整理を行いゲーム開発未経験者にも複雑なゲームの開発が行えることを目指す。

そのための基礎研究として「隕石避けゲーム」を最初の例に挙げ、最も基本的なゲーム要素から抽出し、その組み合わせでゲーム開発が行えるアプローチを採用する。

本稿の結果は、以下の通りである。

ArchiMate3.0 の仕様[5]を用いてゲームのアーキテクチャを定義し、コンポーネント毎にクラスを設計した。その設計を Unity の C# によって実装し「隕石避けゲーム」のインゲーム部分を実装することができた。

今後の方向性として以下のような拡張を考慮中である。

- ・ゲーム開始から終了までの画面遷移
- ・他メカニクスに対応したクラス設計
- ・未定義のゲーム要素の追加
- ・フレーバー層のクラス設計と実装

また、これらを実現するゲーム要素の提案と、実装を充実させゲーム開発やプレイ体験の少ないユーザーが様々なゲーム企画を実現できる環境を目指していく。

参考文献

- [1] 北野不凡, 山本修一郎「カジュアルゲームにおけるゲーム要素の分解・整理 ArchiMate で表現する提案」2022
- [2] 北野不凡, 山本修一郎「ArchiMate によるゲームメカニクス表記ルール」の提案」2023
- [3] 北野不凡, 山本修一郎「ゲーム要素の分類におけるフレーバー層追加の提案」2023
- [4] 山本修一郎, 現代エンタープライズ・アーキテクチャ概論 - ArchiMate 入門, デザインエッグ社, 2016
- [5] TOG, ArchiMate3.0, <https://pubs.opengroup.org/architecture/archimate30-doc/>

付録1 GameManager コーディング例 (一部)

```
////////////////////////////////////
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameManager : MonoBehaviour
{
    //ステートマシン
    public enum StateMachine { gwait,gplay,gover,gclear }
    public static StateMachine gsm;
    int CameraType = 0;//固定カメラ
    int CameraView = 0;//トップビュー
    int CameraPresen = 0;//アイソメトリック レンダリング
    int ScreenSizeX = 1920;//ターゲット画面横サイズ
    int ScreenSizeY = 1080;//ターゲット画面縦サイズ Y 軸
    int DLightX = 0;//ディレクショナルライト位置 X
    int DLightY = 0;//ディレクショナルライト位置 Z
    int DLightZ = -14;//ディレクショナルライト位置 Y(高さ)
    int DLightA = 0;//ディレクショナルライト角度 下向き
    // Start is called before the first frame update
    void Start ()
    {
        //GameStart
        CameraSetting();
        gsm = StateMachine.gwait;//InGame の内部でゲームスタート
    }
}
```

付録2 InGame コーディング例 (一部)

```
////////////////////////////////////
public class InGame : MonoBehaviour {
    public static int Life = 1; //生存確認フラグ 1:生存中
    public static int Sclear = 0; //ステージクリアフラグ 0:未クリア 1:クリア

    // Start is called before the first frame upda
    void Start () {
        Debug.Log ("InGame Start");
        this.gameObject.AddComponent<LimitManage> ();
        this.gameObject.AddComponent<BarrierManage> ();
        this.gameObject.AddComponent<PlayerManage> ();
        this.gameObject.AddComponent<GameEconomics> ();
    }

    // Update is called once per frame
    void Update () {

    }

    /// /// インナークラス ///
    //制限時間などの管理
    private class LimitManage : MonoBehaviour {

        private int TUpDown = 1; //Time Up/Down Switch 0:NoUse 1:Up 2:Down
        private float countup = 0.0f; // 制限時間初期値
        private float timeLimit = 10.0f; //タイムリミット
        GameObject DispTime;
        GameObject DispStat;

        private void Start () {
            this.DispTime = GameObject.Find ("timeText2");
            this.DispStat = GameObject.Find ("timeText1");
        }
        private void Update () {
            if (TUpDown == 1) {
                countup += Time.deltaTime; //時間をプラスする
            } else {
                countup -= Time.deltaTime; //時間をマイナスする
            }
            if (InGame.Sclear == 0) {

```

```
        DispTime.GetComponent<TextMeshProUGUI> ().text = countup.ToString ("f1");//時間を表示する
    }
    if (countup >= timeLimit) {
        InGame.Sclear = 1;
        DispStat.GetComponent<TextMeshProUGUI> ().text = "StageClear !";
    }
}
}

//バリアの発生などの管理
private class BarrierManage : MonoBehaviour {
    private float GenIntarval = 1.0f;
    private float PlayTime = 0.0f;
    private int rndX;
    private string rndXS;
    private float rndXF;

    private void Start () {
    }

    private void Update () {
        GameObject ast = GameObject.Find ("Rock1");
        //PlayTime += Time.deltaTime; //時間をプラスする
        if (Random.Range (1,40)==30) {
            rndXF = Random.Range (-7.0f, 18.0f);
            Instantiate (ast, new Vector3 (rndXF, 6.0f, -1.5f), Quaternion.identity);
        }
    }
}

//プレイヤーのキー入力と移動の管理
private class PlayerManage : MonoBehaviour {
    private void Start () {
    }

    private void Update () {
        //キーボード入力チェック
        GameObject Player = GameObject.Find ("Player");

        // 左に移動
        if (Input.GetKey (KeyCode.LeftArrow)) {
            Player.transform.Translate (0.1f, 0.0f, 0.0f);
        }
        // 右に移動
        if (Input.GetKey (KeyCode.RightArrow)) {
            Player.transform.Translate (-0.1f, 0.0f, 0.0f);
        }
    }
}
}
```