プログラムとデータの並行開発時の互換性とその形式化

長谷川 勇[†1]

一般的なゲーム開発においては，ランタイム・ツール・データの開発が並行して行われる．この場合，データ量が大きいこと，チーム内で複数のバージョンを利用することが多いことから，互換性は重要な課題である．本論文では単純な互換性以外の，いくつかの機能はデフォルト動作にフォールバックするが正しく動作する，相互前方互換という互換性を提案し，それらの形式化を述べる．

Compatibility and Its Formalization for Parallel Development of Program and Data

Isamu Hasegawa[†1]

Runtime, Tooling, and Data are developed concurrently during generic game development. In this case, the amount of data is huge, and more than one versions are often used in a team. Therefore the compatibility is an important problem. We proposed a compatibility "bidirectional forward compatibility", that means the version is compatible excepts some functionalities are falled back to their default behavior. And we explained a formalization about "bidirectional forward compatibility".

1. はじめに

一般的なゲーム開発においては，以下の 3 種類の開発が同時に並行して進行する．

- ゲームを実行するランタイムプログラムの開発
- ランタイムが動作するために読み込むデータ
- データの開発ツール

これらが並行して行われる際，互換性が大きな問題となる．これは以下が理由である．

(1) データの数量が膨大である
(2) 開発チーム内でバージョンが一致しない

(1) は，ランタイムやツールが開発中であり更新が頻繁に行われるため，膨大なデータを作り直すのは現実的でないためである．

(2) は，開発中のビルドがすべて正しく動作すると想定するのは現実的ではなく，あるバージョンにおいて正しく動作しない機能がある場合，その機能を利用するメンバはその機能が動作したバージョンで開発を続行するためである．

なお，この問題はゲーム開発において特有ではなく，以下の条件を満たせば，様々なツールで発生しうる．

- データの編集にツールを必要とする (この場合個々のデータ量が大きいことが多い)．
- 属性が完全には一致しない複数の集団でツールを利用する．

たとえば，静的解析ツールを複数の開発チームに提供する場合，チーム間では解析ツールのバージョンが異なることは多々あり，ルールの数が多くなると，上記の問題が発生しうる．

以降では，これらの互換性の問題に関して，はじめに定性的な分類を示し，その形式化を考える．

2. 定性的な分類と指針

2.1 互換性の定義

互換性に関しては，まず以下の後方互換および前方互換を考える．

- 後方互換: バージョン $v1$ からバージョン $v2$ にバージョンアップした際，バージョン $v1$ のツールで生成したデータは，バージョン $v1, v2$ のランタイムで同一の動作をする．
- 前方互換: バージョン $v1$ からバージョン $v2$ にバージョンアップした際，バージョン $v2$ のツールで生成したデータは，バージョン $v2$ で追加された機能を除き，バージョン $v1, v2$ のランタイムで同一の動作をする．

ここで，上述の通り，チーム内では複数のバージョンが混在する可能性がある．このため，$v1$ から $v2$ に

†1 株式会社スクウェア・エニックス
SQUARE ENIX CO., LTD.

バージョンアップした際に「$v1$ から $v2$ への前方・後方互換」だけではなく,「$v2$ から $v1$ への前方・後方互換」も考えなければならない．

理想的には $v1, v2$ 相互に後方互換性を持つことが望ましいが，開発中は機能追加を行うことから現実的ではない．一方，相互に前方互換性を維持できれば，ある程度開発は行うことができる．このため，互換性を以下の 3 種類に分類する．

- 相互後方互換: $v1$ から $v2$ および $v2$ から $v1$ に後方互換性を持つ状態
- 相互前方互換: $v1$ から $v2$ および $v2$ から $v1$ に前方互換性を持つ状態
- 実行不能: クラッシュなど正常に動作しない状態．

2.2 変更の分類と指針

ここでは一例として，木構造のデータを仮定し，データフォーマットの変更の分類を考える．

- データ項目の追加
- データ項目の削除
- データ項目の ID 変更
- データ項目の (親または子要素への) 移動
- データ項目の単位変更
- データ項目のデフォルト値の変更

詳細は省略するが，これらの変更においていくつかルールを定めることで，相互前方互換を保つことができる．

2.3 形式化の試行

前節で，ルールを定めることで相互前方互換を保つことができると述べたが，それらのルールを厳密にすることと，前節で一例として述べた木構造以外にも一般化するため，分類の形式化を考える．

ゲーム開発においては，ツールとランタイムで扱うデータ形式が異なり，ツールで扱う編集時データをランタイムで実行する実行時データにコンバートすることが多い．しかし，ここでは問題を単純化するため，ツールとランタイムが扱うデータ形式は同じであり，ツールでロードできるならばランタイムで相互前方互換 (相互後方互換ではないことに注意) と仮定する．また，データにはエディタ上でのホワイトスペースやレイアウト情報といった，ランタイムに影響のないデータ項目は含まれないものとする．

はじめに単純な互換性を考える．

- ツールのバージョン x, y をそれぞれ T_x, T_y と呼び，T_x, T_y で保存したデータ全体の集合を $\mathbb{D}_x$, $\mathbb{D}_y$ とする．
- ランタイムのバージョン x, y をそれぞれ R_x, R_y と呼ぶ．これらは R_x, R_y は $\mathbb{D}_x$, $\mathbb{D}_y$ の要素であるデータを読み込み実行する．

このとき，T_x, T_y に対して，単純な互換性は以下で表現できる．

- ${}^{\exists}d_x \in \mathbb{D}_x$ が T_y でロードできない．
- ${}^{\forall}d_x \in \mathbb{D}_x$ を T_y でロードできる．
- ${}^{\exists}d_x \in \mathbb{D}_x$ が R_y で実行できない．
- ${}^{\forall}d_x \in \mathbb{D}_x$ を R_y で実行できる．

次に，前章で定義した相互前方互換のため，写像 $D_{x\to y} : \mathbb{D}_x \to \mathbb{D}_y$ を考える．データ d_{x1} を T_y でロードしセーブしなおしたデータが d_{y1} であるとして，以下で定義する．

$$D_{x\to y}(d_{x1}) = d_{y1}(d_{x1} \in \mathbb{D}_x)$$

なお，d_{x1} が T_y でロードできない場合の値は $\perp$ とする．

ここで定義した写像 $D_{x\to y}$ は互換性を表しており，相互前方互換性に関して以下で表現できる．

- 相互後方互換: $D_{x\to y}$ が全射 $\wedge$ $D_{y\to x}$ が全射
- 相互前方互換: ${}^{\forall}d_x \in \mathbb{D}_x : D_{x\to y}(d_x) \neq \perp \wedge {}^{\forall}d_y \in \mathbb{D}_y : D_{y\to x}(d_y) \neq \perp$
- 実行不能: ${}^{\exists}d \in \mathbb{D}_x : D_{x\to y}(d) = \perp$

さらにこの写像を用いることで変更の分類も可能である．以下にいくつか例を示す．

- $D_{x\to y}$ が単射でない，すなわち ${}^{\exists}d_{x1}, {}^{\exists}d_{x2} : d_{x1} \neq d_{x2} \wedge D_{x\to y}(d_{x1}) = D_{x\to y}(d_{x2})$ ならばデータ項目の削除と考えることができる．
- 一方 $D_{x\to y}$ が全射でないならば，データ項目の追加と考えることができる
- $D_{x\to y}$ が全単射で，かつ，${}^{\exists}d : d \neq D_{x\to y}(d)$ ならば，データ項目の ID 変更と考えることができる．

これらを組み合わせることで，データ変更の種類と，互換性の有無を表現できる．たとえば，上記の ID 変更は相互後方互換の条件も満たしている，つまりツールやランタイム内部で必要な ID 変換が行われていると考えることができる．

これらの条件を順に調べていくことで一般化が可能であると考える．

3. おわりに

本論文では，木構造のデータフォーマット変更について，その一部の形式化を述べた．今後はその他のフォーマット変更や，木構造以外のデータフォーマットについても検討したい．ワークショップでは，互換性の分類・形式化の妥当性について議論したい．