第7回講義：結論・デザインパターン

第7回講義：結論・検証とMachinationsデザインパターン

1. Chapter 8：結論と検証
    1.1 ゲームの構造的特性のまとめ
    1.2 検証方法
    1.3 教育での活用
    1.4 プロトタイプ構築
    1.5 業界からの評価
    1.6 本論文で扱わなかった領域
    1.7 今後の研究方向
2. 付録A：Machinationsリファレンス
    2.1 リソース接続の種類
    2.2 状態接続の種類
    2.3 ノードの種類
    2.4 アクティベーションモード
    2.5 プル/プッシュモード
    2.6 ゲートタイプ
3. 付録B：Machinationsデザインパターン（13パターン）
    3.1 エンジン型パターン
    3.2 フリクション型パターン
    3.3 エスカレーション型パターン
    3.4 その他のパターン
    3.5 パターン一覧表
4. 付録C：ゼルダ風レベルのレシピ
    4.1 概要
    4.2 生成ステップ
    4.3 生成プロセスの全体像
5. 全講義のまとめ

第7回講義：結論・検証とMachinationsデザインパターン

> **対応範囲:** Chapter 8（結論と検証）+ 付録A（Machinations概要）+ 付録B（デザインパターン）+ 付録C（ゼルダ風レベルのレシピ）

---

1. Chapter 8：結論と検証

1.1 ゲームの構造的特性のまとめ

本論文で提案した2つのフレームワークは、ゲームの構造的特性を異なる角度から形式化する。

フレームワーク	モデル化の対象	主な分析対象
Machinations	フィードバック構造と内部経済	エマージェンス（創発）
Mission/Space	レベルデザインと進行構造	プログレッション（進行）

両者を統合することで、エマージェンスとプログレッションの**橋渡し**が可能になる。ゲームの経済システム（Machinations）とレベル構造（Mission/Space）を同時にモデル化し、より完全なゲーム設計を実現する。

2つのフレームワークの統合

2つのフレームワークの統合

1.2 検証方法

研究成果は以下の4つの方法で検証された。

検証方法	内容
ソフトウェア実装	Machinationsツール（Webベースシミュレータ）とLudoscope（レベル生成ツール）の開発
コミュニティ発表	GDC（Game Developers Conference）等の業界カンファレンスでの発表とフィードバック
査読付き論文	学術誌・国際会議での発表と査読プロセスによる評価
教育現場での実践	Hogeschool van Amsterdam（アムステルダム応用科学大学）での授業導入

1.3 教育での活用

Hogeschool van Amsterdamのゲーム開発コースにおいて、Machinationsを活用したカリキュラムが実施された。学生はダイアグラムを用いてゲーム経済を設計・シミュレーションし、理論と実践の接続を体験した。形式的フレームワークは、設計意図の**共有**と**議論**を促進する教育ツールとして有効であることが示された。

1.4 プロトタイプ構築

LKE（Lock-Key-Emergence）実験ゲームが構築された。これはMission/SpaceとMachinationsの統合を実証するプロトタイプであり、ロック＆キー構造（プログレッション）にエマージェントなメカニクスを組み込んだ実験的ゲームである。

1.5 業界からの評価

ゲームデザインの権威であるErnest Adamsをはじめとする業界専門家から肯定的な評価を受けた。特にMachinationsフレームワークは、ゲームバランスの分析・調整における実用性が認められている。

1.6 本論文で扱わなかった領域

以下の領域は本論文の範囲外として意図的に省略されている。

- **物理シミュレーション** – 物理エンジンによるリアルタイム挙動
- **操作性（コントロール）** – インターフェースとプレイヤー操作
- **社会的インタラクション** – プレイヤー間のメタ的コミュニケーション
- **失敗処理** – エラーハンドリングやリカバリーメカニズム

1.7 今後の研究方向

#	方向性	概要
1	**Machinationsの拡張**	より複雑な経済構造、非数値リソース、時間依存メカニクスへの対応
2	**Mission/Spaceの拡張**	非線形ナラティブ、マルチパスレベルデザインの表現力強化
3	**より多くのゲーム分析**	多様なジャンルへのフレームワーク適用と事例蓄積
4	**手続き的生成の改善**	Ludoscopeの書き換え規則の洗練と生成品質の向上
5	**適応型ゲームの発展**	プレイヤーの行動に応じて動的に調整されるゲームメカニクス
6	**教育ツールの改善**	より直感的なインターフェースと教育カリキュラムへの統合

> **重要ポイント** - 2つのフレームワークの統合により、エマージェンスとプログレッションの両面からゲームを形式的に設計できる - 検証はソフトウェア・学術・業界・教育の4方向から多角的に行われた - 物理・操作・社会的インタラクションなどは今後の課題として残されている

---

2. 付録A：Machinationsリファレンス

2.1 リソース接続の種類

接続	表記	機能
リソースフロー	実線矢印 →	リソースの物理的な移動経路を表す

リソース接続にはフローレート（流量）を指定でき、1ステップあたりに移動するリソース数を制御する。

2.2 状態接続の種類

種類	表記	機能
ラベル修飾子（Label Modifier）	点線矢印 → ラベル	リソース接続の流量（ラベル値）を動的に変更する
ノード修飾子（Node Modifier）	点線矢印 → ノード	ノードの内部パラメータを変更する
トリガー（Trigger）	点線矢印 + *	条件成立時に対象ノードのアクションを1回発火させる
アクティベーター（Activator）	点線矢印 + ◆	ノードの有効/無効を条件に応じて切り替える

2.3 ノードの種類

ノード	記号	機能	特徴
プール（Pool）	○	リソースの蓄積・保持	現在のリソース量を保持する基本ノード
ソース（Source）	△	リソースの無限生成	リソースを生み出す。枯渇しない
ドレイン（Drain）	▽	リソースの消費・除去	リソースをゲームから取り除く

ノード	記号	機能	特徴
コンバーター（Converter）	▷	リソースの種類変換	入力リソースを消費し、別の種類を出力
ゲート（Gate）	◇	リソースの条件分岐	確率的または決定的に分配
トレーダー（Trader）	⇄	プレイヤー間のリソース交換	双方向のリソース移動
エンドコンディション（End Condition）	◎	ゲーム終了条件の判定	条件成立でゲーム終了を宣言
レジスター（Register）	□	数値の計算・保持	数式に基づく値の算出（リソースは通過しない）

2.4 アクティベーションモード

モード	記号	動作タイミング
自動（Automatic）	*	毎タイムステップで自動実行
インタラクティブ（Interactive）	ダブルライン	プレイヤーの操作で起動
受動（Passive）	—	他ノードからの要求で起動
開始時（Start）	S	ゲーム開始時に1回だけ実行

2.5 プル/プッシュモード

モード	方向	説明
プル（Pull）	下流 → 上流	下流ノードが上流からリソースを引き寄せる（需要主導）
プッシュ（Push）	上流 → 下流	

モード	方向	説明
		上流ノードが下流へリソースを送り出す（供給主導）

2.6 ゲートタイプ

タイプ	動作	用途
確率的ゲート（Probabilistic）	各出力に確率を割り当て、ランダムに分配	サイコロ、ランダムドロップ
決定的ゲート（Deterministic）	固定の順序またはルールに基づき分配	均等分配、優先順位制御

> **重要ポイント** - Machinationsは8種類のノード、2種類の接続（リソース・状態）、4つのアクティベーションモードで構成される - 状態接続の4種類（ラベル修飾子・ノード修飾子・トリガー・アクティベーター）がフィードバック構造の鍵となる - プル/プッシュモードの選択により、リソース不足時の挙動が変わる

---

3. 付録B：Machinationsデザインパターン（13パターン）

3.1 エンジン型パターン

B.1 スタティックエンジン（Static Engine）

- **タイプ：** エンジン
- **意図：** 固定レートでリソースを生成する最も基本的なエンジン。ソースから一定量のリソースがプールに流れ続ける。

- **適用場面：**
    - ターンごとに固定収入を与えるシステム
    - 時間経過で一定量回復するHP/MPシステム
    - ボードゲームの基本的な資源供給
- **ゲーム例：** Descent（ターンごとの固定アクション）、バックギャモン（サイコロによる固定的移動権の供給）

スタティックエンジンのダイアグラム

スタティックエンジンのダイアグラム

B.2 ダイナミックエンジン（Dynamic Engine）

- **タイプ：** エンジン
- **意図：** フィードバックにより生産率が動的に変化するエンジン。投資によってエンジンの出力が向上し、正のフィードバックループを形成する。
- **適用場面：**
    - リソースを再投資して生産力を高めるシステム
    - 「金で金を生む」経済構造
    - プレイヤーの成長が加速する仕組み
- **ゲーム例：** StarCraft（ワーカーを追加して鉱物採掘速度を向上）、カタンの開拓者たち（開拓地の拡大による資源増加）

ダイナミックエンジンのダイアグラム

ダイナミックエンジンのダイアグラム

B.3 コンバーターエンジン（Converter Engine）

- **タイプ：** エンジン
- **意図：** あるリソースを別のリソースに変換して活用するエンジン。変換レートとタイミングがゲームの中核的な戦略要素になる。
- **適用場面：**
    - 原材料を加工品に変換する生産チェーン
    - 通貨交換や資源トレードの仕組み

    - クラフトシステム
- **ゲーム例：** Power Grid（燃料→電力の変換が経済の中心）、Elite（貿易による商品変換で利益を得る）

B.4 エンジンビルディング（Engine Building）

- **タイプ：** エンジン
- **意図：** ゲームプレイの大部分がエンジンの構築と最適化に費やされるパターン。プレイヤーは初期の小さなエンジンを徐々に拡大・改良する。
- **適用場面：**
    - 都市建設・文明発展ゲームの中核メカニクス
    - デッキ構築ゲームにおけるカードエンジンの組み立て
    - 工場建設シミュレーション
- **ゲーム例：** SimCity（都市インフラの構築と最適化）、Puerto Rico（建物による生産エンジンの構築）、カタン（開拓地ネットワークの拡大）

3.2 フリクション型パターン

B.5 スタティックフリクション（Static Friction）

- **タイプ：** フリクション
- **意図：** ドレインが固定レートでリソースを自動的に消費する。リソースの蓄積を抑制し、プレイヤーに継続的な供給の必要性を課す。
- **適用場面：**
    - 毎ターンの固定維持費（食料、燃料など）
    - 時間経過によるリソース減少（腐敗、劣化）
    - 固定コストの経営要素
- **ゲーム例：** Caesar III（市民の食料消費）、Monopoly（固定資産税、修繕費）

B.6 ダイナミックフリクション（Dynamic Friction）

- **タイプ：** フリクション
- **意図：** 消費率がゲーム状態に応じて動的に変化するフリクション。典型的な負のフィードバックループを形成し、成長に伴いコストも増大する。
- **適用場面：**
    - 領土拡大に比例して増加する維持費
    - 軍隊の規模に応じた兵站コスト
    - レベルアップに伴う経験値要求量の増加
- **ゲーム例：** Civilizationシリーズ（都市の規模が大きくなるほど不満や維持費が増大する経済構造）

B.7 アトリション（Attrition）

- **タイプ：** フリクション
- **意図：** プレイヤーが互いのリソースを攻撃・破壊する、マルチプレイヤー環境における破壊的フィードバック。消耗戦を生み出す。
- **適用場面：**
    - 直接攻撃によるリソース破壊
    - 対戦カードゲームの除去メカニクス
    - 領土の奪い合い
- **ゲーム例：** Magic: The Gathering（クリーチャーの戦闘による相互破壊）、Space Hulk（ユニットの消耗戦）

B.8 ストッピングメカニズム（Stopping Mechanism）

- **タイプ：** フリクション
- **意図：** 特定のメカニズムの効果が使用するたびに減少し、支配的戦略の出現を抑制する。同じ行動の繰り返しが不利になるよう設計される。
- **適用場面：**
    - 需要と供給による価格変動
    - 繰り返し使用による効果減衰
    - 独占防止メカニズム

- **ゲーム例：** Power Grid（燃料市場：購入するほど価格が上昇し、同一燃料への依存を抑制）

3.3 エスカレーション型パターン

B.11 エスカレーティング・コンプリケーション（Escalating Complications）

- **タイプ：** エスカレーション
- **意図：** ゲームの進行に伴い、外部からの脅威や障害が段階的に増大する。プレイヤーに適応と成長を要求する。
- **適用場面：**
    - 敵の出現頻度や強さが段階的に上昇するシステム
    - 時間経過で難易度が上がるサバイバル要素
    - ウェーブ制の敵襲システム
- **ゲーム例：** Space Invaders（エイリアンの速度が段階的に上昇）、Pac-Man（ゴーストの挙動が面ごとに高度化）

B.12 エスカレーティング・コンプレキシティ（Escalating Complexity）

- **タイプ：** エスカレーション
- **意図：** 増大する複雑さにプレイヤーが対抗するゲーム。正のフィードバックにより複雑さが蓄積し、最終的にプレイヤーが対処しきれなくなって敗北する。
- **適用場面：**
    - 蓄積型パズルゲーム
    - 管理対象が増え続けるシステム
    - 「いつまで耐えられるか」を競うエンドレスモード
- **ゲーム例：** Tetris（ブロックの蓄積により操作空間が減少し、速度も上昇）

3.4 その他のパターン

B.9 マルチプルフィードバック（Multiple Feedback）

- **タイプ：** その他
- **意図：** 1つのアクションが異なるシグネチャ（正/負）を持つ複数のフィードバックループを同時に発動させる。複雑な戦略的判断を生み出す。
- **適用場面：**
    - 攻撃行動が複数の結果を同時にもたらすシステム
    - リスクとリターンが複合的に絡み合う判断
    - 1つの選択が複数のリソースに異なる影響を与える構造
- **ゲーム例：** Risk（攻撃が「領土獲得（正）」「兵力増加（正）」「カード獲得（正）」の3つの正のフィードバックを同時に発動）

B.10 トレード（Trade）

- **タイプ：** その他
- **意図：** プレイヤー間でリソースを交換するメカニズム。負の建設的フィードバックを形成し、資源の偏りを平準化しつつ交渉による戦略性を加える。
- **適用場面：**
    - プレイヤー間の直接交渉による資源交換
    - 市場を介した売買システム
    - 外交と同盟による資源共有
- **ゲーム例：** カタンの開拓者たち（プレイヤー間の自由交渉による資源交換）、Civilization III（外交による技術・資源の取引）

B.13 プレイスタイル強化（Playing Style Reinforcement）

- **タイプ：** その他
- **意図：** 遅い正のフィードバックによってプレイヤーのプレイスタイルに適応し、その選好を強化する。プレイヤーごとに異なる体験を生み出すRPG的要素。
- **適用場面：**
    - 使用頻度に応じてスキルが成長するシステム
    - プレイ傾向に基づく動的な難易度調整
    - キャラクターの特化・差別化メカニズム

- **ゲーム例：** Blood Bowl（チームの経験による特化）、The Elder Scrolls IV: Oblivion（使用したスキルが優先的に成長）、Caylus（戦略の方向性が強化される恩恵システム）

3.5 パターン一覧表

#	パターン名	タイプ	主なフィードバック	代表例
B.1	スタティックエンジン	エンジン	なし（固定）	Descent
B.2	ダイナミックエンジン	エンジン	正のフィードバック	StarCraft
B.3	コンバーターエンジン	エンジン	変換チェーン	Power Grid
B.4	エンジンビルディング	エンジン	正のフィードバック	SimCity
B.5	スタティックフリクション	フリクション	なし（固定消費）	Caesar III
B.6	ダイナミックフリクション	フリクション	負のフィードバック	Civilization
B.7	アトリション	フリクション	相互負のフィードバック	M:TG
B.8	ストッピングメカニズム	フリクション	負のフィードバック	Power Grid
B.9	マルチプルフィードバック	その他	複合フィードバック	Risk
B.10	トレード	その他	負の建設的フィードバック	カタン
B.11	エスカレーティング・コンプリケーション	エスカレーション	正のフィードバック	Space Invaders

#	パターン名	タイプ	主なフィードバック	代表例
B.12	エスカレーティング・コンプレキシティ	エスカレーション	正のフィードバック	Tetris
B.13	プレイスタイル強化	その他	遅い正のフィードバック	Oblivion

> **重要ポイント** - 13のデザインパターンは**エンジン型（4つ）**、**フリクション型（4つ）**、**エスカレーション型（2つ）**、**その他（3つ）**に分類される - エンジンはリソースの**生成・拡大**、フリクションはリソースの**消費・抑制**を担う - 優れたゲーム経済は、エンジンとフリクションの**バランス**によって成立する - 1つのゲームが複数のパターンを組み合わせて使用するのが一般的である

---

4. 付録C：ゼルダ風レベルのレシピ

4.1 概要

Ludoscopeを使い、書き換え規則（Rewriting Rules）を段階的に適用することで、ゼルダ風のダンジョンレベルを自動生成する。ミッショングラフの生成からスペース（空間）への変換まで、8つのステップで構成される。

4.2 生成ステップ

ステップ1：基本ミッション生成

入口（Entrance）とゴール（Goal）を設定し、ボス・ミニボス・ガーディアンを配置する。さらにアクト（Act）を拡張してミッションの基本構造を構築する。

ステップ2：ロック追加

書き換え規則を1〜3回適用し、ミッショングラフにロック（Lock）と対応する鍵（Key）を追加する。これによりプログレッション構造が生まれる。

ステップ3：ロックの複雑化

以下の操作でロック構造をより複雑にする。 - **鍵の複製：** 1つの鍵を複数のパーツに分割 - **ロックの分散：** ロックを複数の場所に分散配置 - **鍵パーツの追加：** 鍵を集めるための追加課題を生成

ステップ4：スペースへの変換

ミッショングラフ（抽象的な課題の順序）をスペースグラフ（具体的な部屋の配置）に変換する。入口、ロック、ゲーム要素が空間上に配置される。

ステップ5：ロックの移動

ロックを入口方向に6〜11回移動させる。これにより、プレイヤーが鍵を見つける前にロックに遭遇する構造（先にロックを発見→鍵を探索→戻って解除）が形成される。

ステップ6：スペースの拡張

追加の部屋を5〜9回配置し、ダンジョンの空間的な広がりと複雑さを増す。探索要素や寄り道が生まれる。

ステップ7：ロックの整理

ロック同士が直接接続されている不自然な箇所を解消し、各ロックの間に適切な空間を確保する。

ステップ8：ショートカットの作成

鍵を発見した後の移動を効率化するショートカット（近道）を配置する。バックトラッキングの冗長性を軽減し、プレイヤー体験を向上させる。

4.3 生成プロセスの全体像

```
ステップ1　基本ミッション生成
　　↓
ステップ2　ロック追加（1–3回）
　　↓
ステップ3　ロックの複雑化
　　↓
ステップ4　スペースへの変換
　　↓
```

```
ステップ5　ロックの移動（6–11回）
　　↓
ステップ6　スペースの拡張（5–9回）
　　↓
ステップ7　ロックの整理
　　↓
ステップ8　ショートカットの作成
　　↓
 完成：ゼルダ風ダンジョン
```

各ステップで書き換え規則が自動的に適用され、パラメータの範囲内でランダム性を持たせることで、毎回異なるダンジョンが生成される。

ゼルダ風レベル生成プロセス

ゼルダ風レベル生成プロセス

> **重要ポイント** - ゼルダ風レベル生成は**ミッション→スペース**の変換を段階的に行う8ステップの手続き的プロセスである - ロックの移動（ステップ5）が「鍵を探す動機付け」を生み、探索の面白さを構造的に保証する - パラメータの範囲を変えることで、同じレシピから多様なダンジョンを自動生成できる - Ludoscopeの書き換え規則は、Mission/Spaceフレームワークの実用的な実装である

---

5. 全講義のまとめ

本講義シリーズ（全7回）を通じて、Joris Dormansの博士論文に基づくゲームデザインの形式的アプローチを体系的に学んだ。第1回ではゲームデザイン理論の必要性とエマージェンス／プログレッションという2つの基本構造を導入し、第2回では既存のゲームデザイン理論を批判的にレビューした。第3回でMachinationsフレームワークの基本要素を習得し、第4回ではフィードバック構造とゲームバランスの分析手法を深めた。第5回でMission/Spaceフレームワークによるレベルデザインの形式化を学び、第6回ではLudoscopeを用いた手続き的レベル生成の実践に取り組んだ。そして本講義（第7回）で、研究の結論と検証、付録に収録された13のデザインパターン、およびゼルダ風レベルの自動生成レシピを総括した。

Machinationsは「エンジン」「フリクション」「エスカレーション」などのデザインパターンとして体系化され、ゲーム経済の設計・分析における共通語彙を提供する。一方、Mission/Spaceはレベルデザインのプログレッション構造を形式化し、Ludoscopeによる手続き的生成を実現した。これら2つのフレームワークを統合することで、ゲームの経済システムとレベル構造を一貫した視点から設計・評価できるようになる。

ゲームデザインは依然として創造的な営みであり、形式的フレームワークがデザイナーの直感や経験を置き換えるものではない。しかし、MachinationsとMission/Spaceが提供する**共通言語**と**分析ツール**は、設計意図の明確化、チーム内のコミュニケーション向上、そしてバランス調整の効率化に大きく貢献する。今後の研究によるフレームワークの拡張と、教育・実務への一層の普及が期待される。