エマージェンスの工学：ゲームデザインの応用理論

エマージェンスの工学：ゲームデザインの応用理論

Engineering Emergence: Applied Theory for Game Design

Joris Dormans (2012) に基づく日本語教科書

目次 - 第1回：導入とルール（Chapter 1-2） - 第2回：ゲームデザイン理論（Chapter 3） - 第3回：Machinationsフレームワーク（Chapter 4） - 第4回：Mission/Spaceフレームワーク（Chapter 5） - 第5回：プログレッションとエマージェンスの統合（Chapter 6） - 第6回：プロトタイピング実践（Machinationsツールとゲーム設計の検証） - 第7回：ゲームの自動生成（Chapter 7） - 第8回：結論・デザインパターン（Chapter 8 + 付録）

第1回講義：導入 —— ゲームデザインの理論的基盤

> **対応範囲:** Chapter 1（導入）+ Chapter 2（ルール、表現、リアリズム）

---

1. Chapter 1：導入

1.1 ゲームデザイン理論の必要性

ゲーム産業は映画産業に匹敵する規模へと成長したが、その設計手法は依然として個人の経験や直感に頼る部分が大きい。建築や工業デザインのように、ゲームデザインにも**体系的・形式的な理論基盤**が必要とされている。理論的フレームワークがあれば、設計の意図を正確に伝達し、反復的な改善を効率化できる。

1.2 エマージェンスとプログレッション

ゲームの構造は大きく2つの型に分類できる。

構造	特徴	代表例
エマージェンス（創発）	少数のルールの組み合わせから、デザイナーが予期しない複雑な振る舞いが生まれる	チェス、囲碁、SimCity
プログレッション（進行）	デザイナーが事前に設計した一連の課題をプレイヤーが順番にクリアしていく	アドベンチャーゲーム

エマージェンスとプログレッションの比較
エマージェンスとプログレッションの比較

多くの現代ゲームはこの2つの構造を**組み合わせて**使用している。例えば、RPGではレベルの進行（プログレッション）と戦闘の戦略性（エマージェンス）が共存する。

1.3 本論文の2つのフレームワーク

本論文では、ゲームの構造を分析・設計するための2つの形式的フレームワークを提案する。

Machinations（マシネーションズ） - ゲームメカニクスと**内部経済**をモデル化する動的ダイアグラム - リソースの流れ、フィードバックループ、ランダム性を視覚的に表現 - エマージェントな構造の分析に特に有効

Mission/Space（ミッション／スペース） - レベルデザインと進行構造をモデル化 - 空間的構造（マップ）とミッション構造（課題の順序）の関係を記述 - プログレッション構造の分析に特に有効

1.4 研究の問い

> **ゲームの構造的特性をどのように形式化し、設計プロセスに活用できるか？**

1.5 論文の構成

- **Chapter 1:** 導入と研究の背景
- **Chapter 2:** ルール、表現、リアリズム —— ゲームの基本概念
- **Chapter 3:** 既存のゲームデザイン理論のレビュー
- **Chapter 4:** 内部経済の概念とMachinationsフレームワークの基礎
- **Chapter 5:** Machinationsの詳細な設計パターン
- **Chapter 6:** フィードバック構造とバランス調整
- **Chapter 7:** Mission/Spaceフレームワーク
- **Chapter 8:** レベルデザインへの応用
- **Chapter 9:** 結論と今後の展望

> **重要ポイント** - ゲームデザインには経験則だけでなく、形式的な理論基盤が求められている - ゲーム構造は**エマージェンス**（創発）と**プログレッション**（進行）の2軸で整理できる - Machinationsは内部経済、Mission/Spaceはレベルデザインをそれぞれモデル化するフレームワークである

---

2. Chapter 2：ルール、表現、リアリズム

2.1 ゲームとシミュレーションの関係

ゲームとシミュレーションはどちらも「現実の一部を抽象化したモデル」であるが、その目的は異なる。

- **シミュレーション:** 現実世界の現象を可能な限り忠実にモデル化し、予測や訓練に用いる
- **ゲーム:** ルールと楽しさ（プレイヤー体験）を追求し、必ずしも現実に忠実である必要はない

2.2 ルールの重要性

ゲームは本質的に**ルールの体系**である。ルールが存在しなければ「遊び（play）」はあっても「ゲーム」にはならない。ルールはプレイヤーの行動に制約を課し、意味のある選択を生み出す。

2.3 Juulによるゲームの定義

Jesper Juulはゲームを以下の**6つの特徴**で定義した。

1. **ルールベースのシステム** —— 明確なルールに基づいて動作する
2. **可変で定量化可能な結果** —— 結果が数値や状態として測定できる
3. **異なる結果への価値付与** —— 勝利・敗北など、結果に優劣がある
4. **プレイヤーの努力** —— 結果を左右するためにプレイヤーが行動する
5. **結果へのプレイヤーの愛着** —— プレイヤーが結果を気にかける
6. **交渉可能な結果** —— 同じゲームでも現実世界への影響は任意に設定できる（賭けの有無など）

2.4 抽象化とリアリズムのバランス

ゲームデザインでは**抽象化**（ルールの簡素化）と**リアリズム**（現実への忠実さ）のバランスが常に問題となる。

- 抽象化が過ぎると、プレイヤーはゲーム世界に没入しにくくなる
- リアリズムを追求しすぎると、ルールが複雑化し、楽しさが損なわれる

デザイナーは「どこを抽象化し、どこをリアルにするか」を意識的に選択する必要がある。

2.5 デジタルゲームの特性

コンピュータがルールの処理を自動化することで、デジタルゲームは以下の利点を得る。

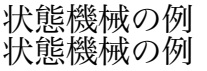
- **複雑なルール処理の自動化:** プレイヤーがルールを暗記する必要がない
- **リアルタイム処理:** ボードゲームでは困難な同時進行・連続的な変化が可能
- **大量の状態管理:** 膨大な数のゲームオブジェクトを同時に追跡できる
- **ルールの隠蔽:** プレイヤーに対してルールの一部を意図的に隠すことが可能

2.6 ペトリネットと状態機械

ゲームのルールを形式的に表現する代表的な手法として、**状態機械**と**ペトリネット**がある。

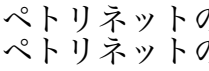
状態機械（State Machine）

- 有限個の**状態**と、状態間の**遷移**で構成される
- ある時点でシステムは1つの状態にのみ存在する
- 入力や条件に基づいて状態が遷移する
- ゲームの進行状態（メニュー → プレイ中 → ゲームオーバーなど）の記述に適している


状態機械の例
状態機械の例

ペトリネット（Petri Net）

- **プレース**（場所）、**トランジション**（遷移）、**トークン**（資源）で構成される
- トークンの流れによってシステムの振る舞いをモデル化する
- **並行処理の表現に優れる** —— 複数のプロセスが同時に進行する様子を自然に記述できる
- Machinationsフレームワークの理論的基盤となっている


ペトリネットの基本構造
ペトリネットの基本構造

要素	状態機械	ペトリネット
基本単位	状態と遷移	プレース、トランジション、トークン
並行処理	困難（状態の組み合わせ爆発）	自然に表現可能
適用場面	ゲームの進行フロー	リソースの流れ、並行するプロセス

> **重要ポイント** - ゲームは本質的に**ルールの体系**であり、シミュレーションとは目的が異なる - Juulの6特徴は、何がゲームで何がゲームでないかを判別する基準となる - デジタルゲームはコンピュータによるルール処理の自動化が最大の特性 - **ペトリネット**は並行処理の表現に優れ、Machinationsフレームワークの基礎となる - **状態機械**はゲームの進行フローの記述に適する

まとめ

第1回講義では、ゲームデザインの理論的基盤としてエマージェンスとプログレッションの2つの構造を学び、ルールの形式的表現手法（状態機械・ペトリネット）を確認した。次回以降、これらの基礎の上に構築されるMachinationsフレームワークとMission/Spaceフレームワークを段階的に学んでいく。

第2回講義：ゲームデザイン理論

> **対応範囲:** Chapter 3（ゲームデザイン理論）

---

1. 既存のゲームデザイン理論のレビュー

ゲームデザインに関する理論は多くの研究者・実務者によって提案されてきた。本章では、主要な理論とフレームワークを整理し、それぞれの強みと限界を明らかにする。

---

2. MDAフレームワーク

MDA（Mechanics, Dynamics, Aesthetics） はHunicke、LeBlanc、Zubekが提唱した、ゲームデザインの分析と設計のためのフレームワークである。

3つのレイヤー

レイヤー	内容	対応する視点
Mechanics（メカニクス）	ゲームのルール、データ表現、アルゴリズム	デザイナーが直接設計する要素
Dynamics（ダイナミクス）	ルールが実行されたときの振る舞い、プレイヤーの行動とシステムの相互作用	ルールの「動き」
Aesthetics（エステティクス）	プレイヤーが感じる感情的体験（楽しさ、緊張感、達成感など）	プレイヤーの主観的体験

デザイナーとプレイヤーの視点の違い

```
デザイナー → Mechanics → Dynamics → Aesthetics
プレイヤー → Aesthetics → Dynamics → Mechanics
```

- **デザイナー**はメカニクスから設計を始め、それがどのようなダイナミクスを生み、最終的にどのようなエステティクスをプレイヤーに届けるかを考える
- **プレイヤー**はまずエステティクス（体験・感情）を受け取り、そこからダイナミクス、メカニクスを理解していく

MDAフレームワーク
MDAフレームワーク

> **重要ポイント** - MDAはゲームを3つの抽象度で分析する枠組みである - デザイナーとプレイヤーでは、ゲームを認知する方向が**逆**になる - メカニクスの小さな変更がダイナミクスとエステティクスに大きな影響を及ぼしうる

---

3. フォーマルデザインツール（Doug Church）

Doug Churchは、ゲーム設計の議論を明確にするための「フォーマルデザインツール」として以下の3つの概念を提唱した。

3.1 意図（Intention）

プレイヤーが**計画を立て、目標を設定し、それを実行する**能力。ゲームがプレイヤーに対して十分な情報と選択肢を提供しているかが鍵となる。

3.2 知覚可能な結果（Perceivable Consequence）

プレイヤーの行動がもたらす**結果の可視性**。行動の結果がプレイヤーに明確に伝わらなければ、意味のある選択は成立しない。

3.3 ストーリー（Story）

ゲームプレイを通じて**プレイヤー自身が生み出す物語**。あらかじめ用意された脚本ではなく、プレイヤーの選択と行動の結果として紡がれる体験を指す。

> **重要ポイント** - 「意図」と「知覚可能な結果」がセットで機能することで、意味のあるゲームプレイが生まれる - プレイヤーが自分自身のストーリーを生成できることが、優れたゲームデザインの条件の一つ

---

4. エマージェントゲームの設計原則

Harvey SmithとRandy Smithは、エマージェントなゲーム（Deus Exなど）の設計から得られた実践的な知見を整理した。

主な原則： - プレイヤーに**複数の解法**を提供する - ゲームシステムの要素が**互いに組み合わせ可能**であるようにする - プレイヤーの**創造的な問題解決**を許容し、報いる - 予期しないプレイを「バグ」ではなく「フィーチャー」として受け入れる設計思想

> **重要ポイント** - エマージェンスを活かすには、システム要素の**組み合わせ可能性**が重要 - デザイナーの想定外のプレイが生まれることは、エマージェントデザインの成功を示す

---

5. ゲームデザインパターン（Bjork & Holopainen）

Staffan BjorkとJussi Holopainenは、建築分野のパターン言語（Christopher Alexander）に着想を得て、ゲームデザインにおける**パターン言語**を構築した。

特徴

- **半形式的（semi-formal）** な記述 —— 自然言語による説明と構造化されたテンプレートの組み合わせ
- **850以上のパターン**を体系化（例：「パワーアップ」「ライフ」「フォグ・オブ・ウォー」など）
- 各パターンには以下の要素が含まれる：
  - パターン名
  - 説明
  - 他のパターンとの関係（インスタンス化、変調など）

利点と限界

利点	限界
ゲーム要素を共通言語で議論できる	パターン数が多く全体像の把握が困難
既存ゲームの分析に有用	パターン間の関係が複雑
新しいデザインの発想に活用できる	定量的な分析には不向き

> **重要ポイント** - ゲームデザインパターンは設計要素を共通言語化する試みである - 850以上のパターンが体系化されているが、その多さ自体が運用上の課題でもある

---

6. その他の重要な理論家

Crawford

ゲームにおける**インタラクティビティ**の重要性を強調。ゲームを他のメディアと区別する本質は、プレイヤーとシステムの間の双方向的なやり取りにあるとした。

Costikyan

「ゲームにはプレイヤーが管理する**リソース**が不可欠である」と主張。意思決定の核心は常にリソースの配分にあるとした。

Koster

「楽しさの正体は**パターン認識と学習**である」と理論化。プレイヤーはゲームの中にパターンを見出し、それを習得する過程で楽しさを感じる。パターンが枯渇すると飽きが来る。

> **重要ポイント** - Crawford: インタラクティビティこそがゲームの本質 - Costikyan: リソース管理が意思決定の核 - Koster: 楽しさ = パターン認識と学習

---

7. 4つの経済機能

ゲームの内部経済を構成する基本的な機能は以下の4つに整理できる。これらはMachinationsフレームワークの基礎となる概念である。

機能	役割	例
ソース（Source）	リソースを生成する	モンスターがゴールドをドロップ、時間経過で体力回復
ドレイン（Drain）	リソースを消費・除去する	弾薬の消費、体力の減少
コンバーター（Converter）	あるリソースを別のリソースに変換する	鉱石を武器に加工、経験値をレベルに変換
トレーダー（Trader）	プレイヤー間でリソースを交換する	ショップでの売買、プレイヤー間の取引

4つの経済機能
4つの経済機能

これらの機能の組み合わせによって、ゲーム内のリソースの流れ（内部経済）が構成される。

> **重要ポイント** - 内部経済の基本要素は**ソース・ドレイン・コンバーター・トレーダー**の4つ - あらゆるゲームの経済システムはこの4機能の組み合わせで記述できる - これらはMachinationsフレームワークの構成要素に直接対応する

---

8. Jesse Schellの「レンズ」

Jesse Schellは著書『The Art of Game Design』において、ゲームデザインを多角的に分析するための「**レンズ**」という概念を提唱した。異なるレンズを通してゲームを見ることで、様々な側面に気づくことができる。

代表的なレンズの例： - **感情のレンズ** —— プレイヤーにどのような感情を喚起したいか - **サプライズのレンズ** —— プレイヤーの予想を裏切る要素はあるか - **楽しさのレンズ** —— なぜこれが楽しいのか、楽しさを増やすにはどうするか - **フローのレンズ** —— プレイヤーが没頭できる難易度のバランスは取れているか - **チャレンジのレンズ** —— 適切な挑戦を提供しているか - **好奇心のレンズ** —— プレイヤーの知りたいという欲求を刺激しているか - **内因的価値のレンズ** —— ゲーム内のリソースや報酬にプレイヤーは価値を感じるか

> **重要ポイント** - レンズはゲームを**異なる視点**から分析するための思考ツールである - 単一の理論ではなく、複数のレンズを組み合わせることで多面的な分析が可能になる

---

9. フィードバック構造の重要性

ゲームメカニクスにおいて、**フィードバック構造**はゲーム体験の質を決定する最も重要な要素の一つである。

正のフィードバック（Positive Feedback）

- 強い者をさらに強くし、弱い者をさらに弱くする
- **不安定化**の方向に作用する
- ゲームを決着に向かわせる効果がある

例：モノポリー
物件を多く持つ → 収入が増える → さらに物件を買える → さらに収入が増える

利点: ゲームが膠着せず、決着がつきやすい **欠点:** 一度差がつくと逆転が困難（ラバーバンド効果がない場合）

負のフィードバック（Negative Feedback）

- バランスを取り戻す方向に作用する
- **安定化**の効果がある
- 劣勢のプレイヤーに逆転の機会を与える

例：マリオカート
順位が低い → 強力なアイテムが出やすくなる → 逆転のチャンスが増える

利点: 接戦を生み出し、最後までプレイヤーの関心を維持する **欠点:** スキルの差が反映されにくくなる場合がある

フィードバックの設計指針

正と負のフィードバック
正と負のフィードバック

特性	正のフィードバック	負のフィードバック
効果	不安定化（差を拡大）	安定化（差を縮小）
ゲームの展開	早期に決着	長期的な接戦
プレイヤー体験	優位者は快感、劣位者は不満	全員が最後まで楽しめる

特性	正のフィードバック	負のフィードバック
設計上の注意	逆転メカニクスとの併用	スキル反映とのバランス

多くの優れたゲームでは、正と負のフィードバックを**意図的に組み合わせて**、テンションの波とバランスを両立させている。

> **重要ポイント** - **正のフィードバック**は不安定化を促し、ゲームを決着に導く - **負のフィードバック**は安定化を促し、接戦を生み出す - 両者の適切な組み合わせが、優れたゲームバランスの鍵となる - フィードバック構造の分析は、Machinationsフレームワークの核心的な機能の一つ

まとめ

第2回講義では、MDAフレームワークを中心に、主要なゲームデザイン理論を概観した。ゲームの内部経済を構成する4つの基本機能（ソース、ドレイン、コンバーター、トレーダー）と、フィードバック構造（正と負）の概念は、次回以降のMachinationsフレームワークの学習に直結する重要な基礎知識である。

理論・概念	提唱者	核心
MDAフレームワーク	Hunicke, LeBlanc, Zubek	メカニクス → ダイナミクス → エステティクス
フォーマルデザインツール	Doug Church	意図・知覚可能な結果・ストーリー
デザインパターン	Bjork & Holopainen	850以上のパターンによる共通言語化
4つの経済機能	—	ソース・ドレイン・コンバーター・トレーダー
フィードバック構造	—	正（不安定化）と負（安定化）

第3回講義：Machinationsフレームワーク（Chapter 4）

1. Machinationsとは

Machinationsは、ゲームの**内部経済（Internal Economy）**をモデル化するための動的ダイアグラムフレームワークである。リソースの生成・消費・変換・流通をビジュアルに表現し、ゲームバランスの分析やシミュレーションを可能にする。

2. 基本要素（ノード）

要素	記号	機能
プール（Pool）	丸 ○	リソースを蓄積・保持する
ソース（Source）	上向き三角 △	リソースを無限に生成する
ドレイン（Drain）	下向き三角 ▽	リソースを消費・除去する
コンバーター（Converter）	横向き三角 ▷	あるリソースを別の種類に変換する
ゲート（Gate）	ひし形 ◇	リソースの流れを条件分岐する
トレーダー（Trader）	—	プレイヤー間でリソースを交換する

> **重要ポイント：** 各ノードの役割を正確に理解することが、ダイアグラム設計の基礎となる。プールは「貯蔵」、ソースは「生成」、ドレインは「消費」、コンバーターは「変換」と覚える。

3. 接続の種類

3.1 リソース接続（実線矢印 →）

リソースの**物理的な流れ**を表す。ソースからプールへ、プールからドレインへなど、実際にリソースが移動する経路。

3.2 状態接続（点線矢印 ⟶）

情報の流れを表す。リソースそのものは移動しないが、あるノードの状態が別のノードの挙動に影響を与える。

状態接続には以下の4種類がある：

修飾子	機能
ラベル修飾子（Label Modifier）	リソース接続の流量を変更する
ノード修飾子（Node Modifier）	ノードの内部状態を変更する
トリガー（Trigger）	条件を満たしたときにアクションを発火する
アクティベーター（Activator）	ノードの有効/無効を切り替える

> **重要ポイント：** リソース接続は「モノの流れ」、状態接続は「情報の流れ」である。状態接続を使いこなすことで、条件分岐やフィードバックなど複雑なゲームメカニクスを表現できる。

---

4. アクティベーションモード

ノードがいつ動作するかを決定するモード。

モード	動作	記号表記
自動（Automatic）	毎ステップで自動的に実行される	*
インタラクティブ（Interactive）	プレイヤーが手動で起動する	ダブルライン
受動（Passive）	他のノードから起動される	—

> **重要ポイント：** 自動モードはシステム駆動のメカニクス（毎ターンの収入など）、インタラクティブモードはプレイヤーの意思決定（カードを引く、アイテムを使うなど）に使う。

---

5. プル/プッシュモード

リソースの流れる方向の制御方式。

- **プルモード（Pull）：** 下流のノードが上流からリソースを「引っ張る」
  - 例：プレイヤーが山札からカードを引く
- **プッシュモード（Push）：** 上流のノードが下流にリソースを「押し出す」
  - 例：ソースが自動的にリソースを生成して送り出す

> **重要ポイント：** プルは「需要主導」、プッシュは「供給主導」である。リソースが不足したときの挙動が異なるため、ゲームメカニクスの性質に合わせて適切に選択する。

---

6. ゲートの種類

ゲートはリソースの分配方法を決定する。

6.1 確率的ゲート（Probabilistic Gate）

確率に基づいてリソースの行き先を決定する。サイコロの出目やランダムイベントの表現に使用。

- 例：50%の確率でルートAへ、50%でルートBへ

6.2 決定的ゲート（Deterministic Gate）

あらかじめ定められた条件・パターンに基づいてリソースを分配する。

- 例：3回に1回はルートAへ、それ以外はルートBへ

> **重要ポイント：** 確率的ゲートはランダム性のあるメカニクス、決定的ゲートは規則的なメカニクスに対応する。ゲームのランダム性の度合いを制御する重要な要素。

---

7. フィードバック構造の分析

ゲームバランスの核心となるフィードバックループの分析。

7.1 正のフィードバック（Positive Feedback）

成長を加速させるループ。「勝っている者がさらに勝つ」構造。

- 例：モノポリーの不動産投資 — 物件を持つ → 家賃収入が増える → さらに物件を買える
- 効果：ゲームを収束させる（勝敗の決着を早める）
- リスク：格差が開きすぎてゲームが退屈になる可能性

7.2 負のフィードバック（Negative Feedback）

バランスを維持するループ。「勝っている者にハンデが付く」構造。

- 例：リスクの軍隊分配 — 領土が多い → 防衛が分散する → 弱体化する
- 効果：ゲームを長引かせる（プレイヤー間の格差を縮める）
- リスク：ゲームが膠着して終わらなくなる可能性

> **重要ポイント：** 正のフィードバックは収束を生み、負のフィードバックは均衡を生む。良いゲームデザインは両者を適切に組み合わせる。

---

8. フィードバックプロファイルの7つの特性

フィードバックループを分析する際の7つの観点：

#	特性	説明
1	**極性（Polarity）**	正のフィードバックか、負のフィードバックか
2	**強度（Strength）**	効果が弱いか、強いか
3	**持続性（Duration）**	一時的か、永続的か
4	**速度（Speed）**	効果の発現が速いか、遅いか
5	**投資の有無（Investment）**	プレイヤーの投資を必要とするか
6		リソースを増やすか、減らすか

#	特性	説明
	構築的/破壊的（Constructive/ Destructive）	
7	**直接的/間接的（Direct/ Indirect）**	フィードバックが直接的か、他の要素を経由するか

> **重要ポイント：** これら7つの特性を用いることで、フィードバックループを多角的に分析でき、ゲームバランスの調整ポイントを特定できる。

---

9. SimWarの例

SimWarは2人対戦の戦略ゲームであり、Machinationsフレームワークの実践的な分析対象として取り上げられる。

- 両プレイヤーがリソース（兵力）を生成・配分し、相手と戦闘する
- 正のフィードバック（領土獲得 → 兵力増加）と負のフィードバック（戦線拡大 → 防衛力分散）が共存
- Machinationsダイアグラムでリソースフローを可視化し、バランスの偏りを発見できる

> **重要ポイント：** SimWarの分析を通じて、Machinationsが実際のゲーム分析にどのように活用できるかを理解する。ダイアグラムを描くことで、直感ではわかりにくいバランスの問題を発見できる。

まとめ

- Machinationsはゲームの内部経済をビジュアルに分析するツール
- 6種類のノード、2種類の接続、3種類のアクティベーションモードが基本構成要素
- フィードバック構造の理解がゲームバランス設計の鍵
- 7つのフィードバック特性を用いて体系的に分析する

第4回講義：Mission/Spaceフレームワーク（Chapter 5）

1. レベルデザインの2つの構造

レベルデザインを分析するために、2つの独立したグラフ構造を用いる。

- **ミッション（Mission）：** プレイヤーが達成すべきタスクの構造をグラフで表現する。「何をするか」の設計。
- **スペース（Space）：** ゲーム空間の幾何学的構成をグラフで表現する。「どこでするか」の設計。

この2つを分離して考えることで、レベルデザインの本質を明確に分析できる。

> **重要ポイント：** ミッションは「タスクの論理構造」、スペースは「空間の物理構造」である。同じミッションでも異なるスペースに配置でき、逆もまた然りである。この分離がレベルデザインの柔軟性を生む。

---

2. ミッショングラフ

2.1 構成要素

- **ノード：** 個々のタスク（戦闘、パズル、鍵の獲得、ボス戦など）
- **エッジ：** タスク間の依存関係

2.2 接続タイプ

タイプ	意味	例
弱い要求（Weak Requirement）	推奨される順序。従わなくても進行可能	「先に回復アイテムを取ることを推奨」
強い要求（Hard Requirement）	必須の順序。前提タスクの完了が必要	「ボスキーがないとボス部屋に入れない」
抑制（Inhibition）	ロック。特定条件を満たすまで進行を阻止	「赤い扉は赤い鍵で開く」

> **重要ポイント：** 強い要求と抑制の違いに注意。強い要求は「Aを完了しないとBに進めない」という順序制約であり、抑制は「特定のアイテムや条件がロックを解除する」というメカニズムである。

---

3. スペースグラフ

3.1 構成要素

- **ノード：** 場所（部屋、エリア、ゾーンなど）
- **エッジ：** 通路（場所間の接続）

3.2 通路の種類

- **双方向通路：** 行き来が自由にできる（通常の扉や廊下）
- **一方通行通路：** 片方向のみ移動可能（崖から飛び降りる、一方通行ゲートなど）

> **重要ポイント：** スペースグラフは純粋に空間のトポロジー（接続関係）を表し、距離や面積といったメトリクスは含まない。空間の「つながり方」に焦点を当てる。

---

4. ミッションからスペースへのマッピング

同一のミッション構造を異なるスペースに配置できる。

4.1 直線的マッピング

- タスクの順序とスペースの配置が一致
- 一本道のレベル構成
- わかりやすいが、探索の自由度が低い

4.2 非直線的マッピング

- タスクの順序とスペースの配置が必ずしも一致しない
- プレイヤーが空間を行き来しながらタスクを達成
- 探索の自由度が高く、リプレイ性も向上

> **重要ポイント：** マッピングの方法がプレイヤー体験を大きく左右する。直線的マッピングはナラティブ主導のゲームに、非直線的マッピングは探索主導のゲームに適する。

---

5. ロックと鍵のメカニズム

レベルデザインにおける進行制御の基本構造。

5.1 ロック（Lock）

プレイヤーの進行を阻止するバリア。対応する鍵がなければ通過できない。

5.2 鍵（Key）

ロックを解除するための手段。以下のように多様な形態をとる。

鍵の種類	例
物理的な鍵	ドアの鍵、スイッチ、ボタン
能力（Ability）	二段ジャンプ、爆弾、フックショット
情報（Knowledge）	パスワード、暗号の解答、NPC からの情報

> **重要ポイント：** ロックと鍵は単なる「扉と鍵アイテム」に限定されない。能力の獲得や情報の入手もロック解除として機能する。メトロイドヴァニア系ゲームでは能力型の鍵が中心的な役割を果たす。

---

6. ゼルダの伝説「森の神殿」の分析

複雑なレベルデザインの実例分析。

6.1 構造の特徴

- 複数のロックと鍵が入れ子状に配置されている
- プレイヤーは神殿内を何度も行き来する（バックトラッキング）
- 小鍵・ボスキー・特殊アイテムなど複数の鍵タイプが共存

6.2 ミッションとスペースの関係

- **ミッショングラフ：** 鍵の獲得 → ロック解除 → 次の鍵の獲得…という連鎖構造
- **スペースグラフ：** 中央ホールを起点に複数の翼が分岐するハブ型構造
- ミッション上は線形に見えるが、スペース上では非線形に見える

6.3 バックトラッキングの設計

- ショートカットの配置により、バックトラッキングの煩わしさを軽減
- 新たな能力獲得後に以前の場所を再訪する意味を持たせている

> **重要ポイント：** 森の神殿はミッションの線形性とスペースの非線形性を巧みに組み合わせた好例。バックトラッキングを退屈にさせないための工夫（ショートカット、新発見）がレベルデザインの質を決める。

---

7. コンストラクション

ミッショングラフからスペースグラフを体系的に生成する手法。

7.1 タイトなコンストラクション（Tight Construction）

- ミッションとスペースが密接に対応
- 各タスクが固有の場所に配置される
- 無駄のないコンパクトなレベルになる

7.2 ルーズなコンストラクション（Loose Construction）

- ミッションとスペースの対応が緩やか
- 余分な部屋や通路が追加される
- 探索の余地が広がり、空間が豊かになる

> **重要ポイント：** タイトなコンストラクションは効率的だが窮屈になりがち。ルーズなコンストラクションは冗長だが探索の楽しさが増す。ゲームの目的に応じて使い分ける。

---

8. ミッション/スペースの形態分析（Morphology）

レベルの空間構造を類型化したもの。

形態	構造	特徴
コリドー型（Corridor）	一本道の直線構造	強いナラティブ誘導。進行方向が明確
ハブ型（Hub）	中央から放射状に分岐	プレイヤーに選択肢を提供。順序の自由度が高い
ループ型（Loop）	環状の経路構造	バックトラッキングを自然に促す。探索の効率が良い
メトロイドヴァニア型	能力獲得による段階的な探索拡大	能力型の鍵が中心。再訪による新発見が設計の核

各形態の使い分け

- **コリドー型：** ストーリー重視のゲーム（アンチャーテッドなど）
- **ハブ型：** 選択自由度の高いゲーム（ゼルダのダンジョンなど）
- **ループ型：** 効率的な探索が求められるゲーム（ダークソウルなど）
- **メトロイドヴァニア型：** 長期的な探索が核となるゲーム（ホロウナイトなど）

> **重要ポイント：** レベルの形態はプレイヤー体験を根本的に決定する。コリドーは「導かれる体験」、ハブは「選ぶ体験」、ループは「巡る体験」、メトロイドヴァニアは「開拓する体験」を生む。

まとめ

- レベルデザインはミッション（タスク構造）とスペース（空間構造）に分離して分析する
- ロックと鍵のメカニズムが進行制御の基本となる
- ミッションからスペースへのマッピング方法がプレイヤー体験を決定する
- コンストラクション手法（タイト/ルース）でスペースの密度を調整する
- レベルの形態（コリドー/ハブ/ループ/メトロイドヴァニア）は目的に応じて選択する

第5回：プログレッションとエマージェンスの統合

概要

ゲームデザインにおいて、**プログレッション**（設計者が用意した固定的な進行）と**エマージェンス**（ルールの相互作用から生まれる創発的なプレイ）は対立する概念ではなく、統合して用いることで優れたゲーム体験を実現できる。本章では、その統合手法を多角的に分析する。

---

1. エマージェンスとプログレッションの対比と融合

特性	エマージェンス	プログレッション
構造	少数のルールから複雑な挙動が生まれる	設計者が段階的に体験を配置する
プレイヤーの自由度	高い	低い（ガイドされる）
リプレイ性	高い	低い傾向
物語制御	困難	容易

多くの現代ゲームは両者を**ハイブリッド**で採用している。プログレッションで大枠の進行を制御し、各局面でエマージェントなゲームプレイを提供する。

> **重要ポイント:** プログレッションとエマージェンスは二項対立ではなく、スペクトラム上に位置する。優れたゲームは両者を目的に応じて使い分け、組み合わせる。

---

2. おもちゃ → 遊び場 → ゲーム

ゲームの構造化には段階がある。

おもちゃ→遊び場→ゲームの連続体
おもちゃ→遊び場→ゲームの連続体

おもちゃ（Toy）

- **定義:** 目標のない自由な操作対象
- **例**: SimCity（サンドボックスモード）。プレイヤーは街を自由に作るが、勝利条件は存在しない
- ルールは存在するが、ゴールがない

遊び場（Playground）

- **定義:** おもちゃに空間的・構造的な文脈を与えたもの
- 複数のおもちゃが組み合わさり、相互作用が生まれる
- 探索や実験が主な楽しみ

ゲーム（Game）

- **定義:** 明確な目標（ゴール）とルールが設定された構造体
- 勝敗や達成条件が定義される
- プレイヤーの行動が評価される

> **重要ポイント:** おもちゃ→遊び場→ゲームは、自由度と構造のトレードオフを示す連続体である。デザイナーはプレイヤーにまず「おもちゃ」として操作を覚えさせ、段階的にゲームへ導くことができる。

---

3. 構造化された学習曲線

プレイヤーを複雑なメカニクスに段階的に導入する設計手法。

原則

1. **一度に一つ**のメカニクスを導入する
2. 新しいメカニクスを**安全な環境**で練習させる
3. 既存メカニクスとの**組み合わせ**を徐々に要求する
4. **難易度を段階的に上昇**させる

Half-Life 2 の例

Half-Life 2 は構造化された学習曲線の模範例である。

Half-Life 2の段階的導入

- **バール → ピストル → SMG → グレネード → 重力銃** と順に導入
- 各武器の導入時には、その武器が**最適解となる状況**を配置
- 重力銃は最初に物理オブジェクト操作として「おもちゃ」的に紹介され、後にメイン武器として機能する
- 物理エンジンの活用がプログレッション（段階的導入）とエマージェンス（自由な物理操作）を融合

> **重要ポイント:** 優れた学習曲線は、プレイヤーが「教わっている」と感じないよう設計される。環境自体が教師となり、メカニクスの理解を自然に促す。

---

4. 経済構築ゲーム

Caesarシリーズの分析

Caesar（シーザー）シリーズは、経済構築ゲームにおけるプログレッションの好例である。

Caesarの経済メカニクス構造

- **段階的な経済の複雑化:**
  - 初期ミッション: 住宅と基本的な食料供給のみ
  - 中期ミッション: 水道、宗教施設、娯楽施設が追加
  - 後期ミッション: 貿易、軍事、複数の資源チェーンが必要
- **ミッションが経済メカニクスのアンロックを制御:**
  - 各ミッションで利用可能な建築物が制限される
  - プレイヤーは限定された要素で経済の基礎を学ぶ
  - 後のミッションで要素が増え、既習のメカニクスの上に新しい層が積まれる
- **内部経済の相互依存:**
  - 住民の満足度 → 人口増加 → 税収 → 建設投資 というフィードバックループ
  - 各ループが段階的に開放される

> **重要ポイント:** 経済構築ゲームでは、プログレッション（ミッション構造）がエマージェンス（経済の創発的挙動）の複雑さを制御するゲートとして機能する。

---

5. ミッションとスペースのミスマッチ

ミッション設計とゲーム空間が完全に一致しない場合、プレイヤーに選択の自由を与える手法。

ダイアログツリー

ダイアログツリーの構造例

- 会話内で選択肢を提示し、**分岐する物語進行**を実現
- 各選択がゲーム状態（敵対関係、情報取得、アイテム獲得など）に影響
- プログレッションの中にプレイヤーの**エージェンシー（主体性）**を埋め込む

Deus Ex の例

- 一つのミッションに対して**複数の解法**が存在:
  - **戦闘:** 正面から敵を排除
  - **ステルス:** 見つからずに目的を達成
  - **ハッキング:** セキュリティを電子的に突破
  - **社会的:** 会話や交渉で問題を解決
- ゲーム空間（スペース）が複数の経路を物理的に提供
- ミッション目標は固定だが、達成手段がエマージェント

> **重要ポイント:** ミッション（プログレッション）とスペースの間にあえて「遊び」を持たせることで、線形的な進行の中にもエマージェントな体験を実現できる。

---

6. メカニクスによるプログレッション制御

ロックと鍵のメカニズム

ゲーム空間の進行をメカニクスで制御する古典的パターン。

ロックと鍵のMachinations表現

Machinationsによる表現

- **鍵（Key）**：プレイヤーが獲得するリソースまたは能力
- **ロック（Lock）**：鍵がなければ通過できないゲート
- Machinationsダイアグラムでは、リソースの有無を条件としたゲートで表現

森の神殿のスペースグラフ拡張

ゼルダの伝説シリーズの「森の神殿」をMachinationsで拡張する例:

森の神殿のスペースグラフ
森の神殿のスペースグラフ

- スペースグラフの各部屋をノードで表現
- ロックされた扉をゲートメカニクスで表現
- 鍵の取得がリソースフローとして可視化される

複数鍵メカニズム

- 1つのロックに対して**複数の鍵**が必要な場合
- AND条件: すべての鍵が必要
- OR条件: いずれか1つの鍵で開放可能
- これにより非線形な探索パターンが生まれる

ボムリングメカニズム

- **爆弾**と**ブーメラン**の連携による複合メカニクス
- ブーメランで遠距離の爆弾を起動
- 単独では機能しない能力の**組み合わせ**がロック解除条件となる
- プレイヤーに複数メカニクスの統合的理解を要求

> **重要ポイント**: ロックと鍵のメカニズムは、空間的プログレッションをメカニクスのリソースフローとして形式的にモデル化できる。複合鍵メカニズムは非線形な探索を実現する。

---

7. ロックと鍵のフィードバックメカニズム

動的フリクション

進行に伴って**難易度を動的に調整**するメカニズム。

動的フリクションのパターン
動的フリクションのパターン

- プレイヤーの進行度合いに応じて「摩擦（フリクション）」を増減させる
- 進みすぎたプレイヤーには高い抵抗を、遅れたプレイヤーには低い抵抗を提供
- ゲームバランスを自動的に維持

ダイナミックエンジンパターンの適用

- 第4章で学んだ**ダイナミックエンジン**のパターンをプログレッション制御に応用
- リソース生成率と消費率をプレイヤーの進行状態に連動させる
- フィードバックループにより、プログレッションの速度が自己調整される

> **重要ポイント:** 動的フリクションは、プログレッション（固定的な進行）にエマージェンス（動的な調整）を組み込む具体的手法である。ダイナミックエンジンパターンがその基盤となる。

---

8. プログレスをリソースとして扱う

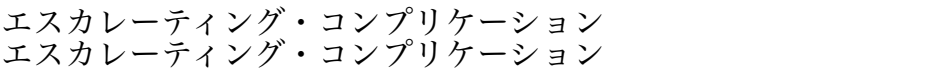
進行そのものをMachinationsの**リソース**として表現し、メカニクスに組み込む手法。

Warhammer Fantasy Roleplay の進行トラッカー

- キャラクターの成長を進行トラッカーとしてモデル化
- 経験値（XP）をリソースとして蓄積し、閾値を超えると新しい能力が解放
- 進行自体がゲーム内経済の一部として機能

エスカレーティング・コンプリケーション

時間経過とともに障害が増大するパターン。



- **Pac-Man:** ゴーストの速度が段階的に上昇、パワーペレットの効果時間が短縮
- **Space Invaders:** エイリアンの数が減るほど移動速度が上昇
- 同じメカニクスの中で**パラメータが変化**することで難易度が上昇
- プレイヤーの行動（エマージェンス）は変わらないが、環境が厳しくなる

エスカレーティング・コンプレキシティ

時間経過とともにシステムの複雑さ自体が増大するパターン。

- **Tetris:** ブロックの落下速度が上昇するだけでなく、積み上がったブロックが状態の複雑さを増す
- プレイヤーが管理すべき**状態空間**が拡大していく
- エスカレーティング・コンプリケーションとの違い: パラメータではなく**構造的な複雑さ**が増大

Seasonsゲームの色魔法経済

- 季節ごとに利用可能な**色付きリソース（魔法エネルギー）**が変化
- プレイヤーは季節の変化を先読みしてリソース管理を行う
- 進行（季節の推移）がリソース供給を制御する
- プログレッション（季節サイクル）とエマージェンス（リソース管理戦略）の統合

> **重要ポイント:** 進行をリソースとしてモデル化すると、時間経過による難易度上昇を形式的に設計・分析できる。エスカレーティング・コンプリケーション（パラメータ変化）とエスカレーティング・コンプレキシティ（構造的複雑化）は異なるアプローチである。

まとめ

手法	プログレッション側	エマージェンス側
学習曲線	メカニクスの段階的導入	各段階での自由なプレイ
経済構築	ミッションによるアンロック	経済の創発的挙動
ロックと鍵	空間進行の制御	鍵の取得順序の自由度
動的フリクション	進行の速度制御	フィードバックによる自己調整
エスカレーション	時間経過による変化	プレイヤーの適応行動

プログレッションとエマージェンスの統合は、ゲームデザインの中核的な課題であり、両者のバランスがプレイヤー体験の質を大きく左右する。

第6回講義：プロトタイピング実践 ── Machinationsツールとゲーム設計の検証

> **対応範囲:** プロトタイピング手法、Machinationsシミュレーション、Ludoscopeレベル設計、デザインパターンの実践的活用

6.1 プロトタイピングの重要性

ゲームデザインにおけるプロトタイピングの位置づけ

ゲームデザインは紙上では完璧に見えても、実際にプレイすると想定外の問題が頻出する。プロトタイピングは、このギャップを早期に発見し修正するための最も効果的な手法である。

紙上のアイデアと実際のプレイ体験のギャップ：

- メカニクスの相互作用は、動かしてみるまで予測が困難
- プレイヤーの感情的反応（フラストレーション、達成感）は机上では評価できない
- 数値バランスの問題は、実際のシミュレーションなしには見えない
- フィードバックループの長期的影響は、直感に反する結果をもたらすことが多い

早期プロトタイプによるデザイン検証の価値

- **コスト削減：** 問題の発見が遅れるほど修正コストは指数関数的に増大する
- **リスク低減：** 根本的なメカニクスの欠陥を開発初期に排除できる
- **コミュニケーション：** チーム内でデザイン意図を共有する具体的な手段となる
- **創造性の促進：** 実際に動くものを触ることで、新たなアイデアが生まれる

反復的デザインプロセス

ゲームデザインは直線的なプロセスではなく、以下のサイクルを繰り返す反復作業である。

1. **アイデア** —— コンセプトとメカニクスの構想
2. **プロトタイプ** —— 検証可能な形での実装
3. **テスト** —— プレイテストによる問題の発見
4. **改善** —— フィードバックに基づく設計の修正

このサイクルを高速に回すことが、優れたゲームデザインの鍵である。

> **重要ポイント** プロトタイピングは「作ってみること」ではなく「検証すること」が目的である。各プロトタイプには明確な検証目標を設定し、その結果に基づいてデザインを進化させる。

6.2 Machinationsツールによるシミュレーション

Machinationsソフトウェアの概要

Machinationsは、ゲームの内部経済をシミュレーションするためのビジュアルツールである。リソースの流れ、フィードバックループ、確率的要素を含むダイアグラムを作成し、実際に動かして挙動を観察できる。

ダイアグラムの作成手順

ステップ1：ノードの配置

ノード種別	役割	使用例
プール（Pool）	リソースを貯蔵する	プレイヤーの所持金、HP
ソース（Source）	リソースを生成する	給料の支払い、アイテムドロップ
ドレイン（Drain）	リソースを消費・除去する	購入、ダメージ
コンバーター（Converter）	リソースを変換する	素材→装備
ゲート（Gate）	リソースの流れを分岐する	確率的分配

ステップ2：接続の設定

- **リソース接続（Resource Connection）：** ノード間のリソースの流れを定義
- **状態接続（State Connection）：** あるノードの状態が別のノードの振る舞いに影響を与える

ステップ3：パラメータの調整

- **流量（Flow Rate）：** 1ステップあたりのリソース移動量
- **容量（Capacity）：** プールが保持できるリソースの上限
- **確率（Probability）：** ゲートでの分岐確率

ステップ4：シミュレーションの実行と観察

ダイアグラムを実行すると、リソースが各ノード間を流れる様子をリアルタイムで観察できる。時間経過に伴うリソース量の変化をグラフで確認し、バランスの問題を特定する。

動的ダイアグラムの利点

静的な設計書と異なり、Machinationsのダイアグラムは**実際のゲームのように動く**。これにより：

- 長期的なバランス崩壊を事前に発見できる
- フィードバックループの実際の影響を可視化できる
- パラメータ変更の効果を即座に確認できる

バランス調整のワークフロー

1. パラメータを変えてシミュレーションを繰り返す
2. フィードバックループの影響を可視化する
3. 問題のあるメカニクスを早期発見する
4. グラフの推移から安定性・収束性を判断する

実践例1：モノポリー経済のシミュレーション

モノポリーの経済構造をMachinationsでモデル化する。

- **プール：** プレイヤーの所持金
- **ソース：** GOマスを通過するたびに受け取る給料（$200）
- **ドレイン：** 他プレイヤーへの家賃支払い
- **正のフィードバックループ：** 不動産を購入→家賃収入が増加→さらに不動産を購入

シミュレーションを実行すると、正のフィードバックにより資産格差が急速に拡大する様子が可視化される。これがモノポリーの「富者がさらに富む」メカニクスの本質である。

実践例2：StarCraftの資源経済

リアルタイムストラテジーゲームの資源経済をモデル化する。

- **ダイナミックエンジン：** ワーカーがミネラルを採掘し、資源プールに蓄積
- **トレードオフ：** 資源をワーカー増産に使うか、軍事ユニットに使うか

- **投資のジレンマ：** ワーカーへの投資は長期的な資源生産を増やすが、短期的な軍事力は低下する

Machinationsでこのトレードオフをシミュレーションすることで、最適な投資バランスを探索できる。

> **重要ポイント** Machinationsシミュレーションの真価は、直感では見えないフィードバック構造の長期的影響を可視化できることにある。パラメータを少し変えるだけで結果が劇的に変わるケースこそ、シミュレーションで検証すべきポイントである。

6.3 Ludoscopeによるレベル設計

Ludoscopeツールの概要

Ludoscopeは、ゲームレベルの設計と生成を支援するツールである。文法ベースの変換規則（レシピ）を用いて、抽象的なミッション構造から具体的なレベルレイアウトを段階的に生成する。

レシピベースの生成

Ludoscopeでは、レベル生成のプロセスを**レシピ**（変換規則の集合）として定義する。

- **文法の定義：** ミッション要素の生成規則を形式文法として記述
- **ルールの適用：** 抽象的な構造を段階的に具体化
- **制約の設定：** 生成されるレベルが満たすべき条件を指定

混合主導型アプローチ

Ludoscopeの特徴は、自動生成とデザイナーの手動編集を組み合わせた**混合主導型**（**Mixed-Initiative**）アプローチにある。

- デザイナーが高レベルの構造を指定し、詳細を自動生成に委ねる
- 生成結果をデザイナーが確認・修正し、再度生成を実行する
- 創造性と効率性を両立する

ミッショングラフの視覚化と編集

ミッショングラフは、プレイヤーが達成すべきタスクの依存関係を表現する有向グラフである。Ludoscopeでは、このグラフを視覚的に編集し、タスクの順序や分岐を設計できる。

スペースグラフの自動生成

ミッショングラフからスペースグラフ（レベルの空間的構造）への変換を自動化する。ミッションの依存関係を満たしながら、空間的に整合性のあるレベルレイアウトを生成する。

デッドロック検出

Ludoscopeには、生成されたレベルの**進行不能状態（デッドロック）**を自動検出する機能がある。

- 鍵が到達不能な位置に配置されていないか
- ロックの解除順序に矛盾がないか
- プレイヤーが必ずクリアできる経路が存在するか

実践例：ゼルダ風ダンジョンの生成

ステップ1：基本ミッション構造の定義

ダンジョンのミッションを「鍵の取得→ロックの解除→ボス戦」の基本パターンとして定義する。

ステップ2：ロックと鍵の配置

文法規則を用いて、鍵とロックのペアを配置する。複数の鍵が必要なロックや、オプションの宝箱なども生成規則に含める。

ステップ3：空間への変換

ミッショングラフをスペースグラフに変換し、部屋と通路のレイアウトを生成する。空間的な制約（部屋の重複禁止、通路の接続性）を満たすように配置する。

ステップ4：テストプレイとイテレーション

生成されたダンジョンをデッドロック検出で検証し、問題があればレシピを修正して再生成する。

> **重要ポイント** Ludoscopeの混合主導型アプローチは、デザイナーの創造的意図を保ちながら大量のレベルバリエーションを効率的に生成できる。デッドロック検出により、進行不能なレベルを自動的に排除できることが大きな利点である。

6.4 ゲームプロトタイプの構築

LKE（Lock-Key-Emergence）実験ゲームの事例

LKEは、ロックと鍵のメカニクスにエマージェントな要素を統合した実験的ゲームである。

- **設計フェーズ：** Machinationsでリソース経済とフィードバック構造を設計
- **実装フェーズ：** シミュレーション結果に基づいてゲームを実装
- **テストフェーズ：** プレイテストで実際の体験を検証
- **改善フェーズ：** テスト結果をMachinationsモデルにフィードバック

このサイクルにより、理論的な設計と実際のプレイ体験のギャップを効率的に埋めることができる。

プロトタイプの種類

種類	手法	主な検証対象
ペーパープロトタイプ	物理的なカード・ボード・トークンでテスト	基本的なメカニクスの面白さ
デジタルプロトタイプ	Machinationsシミュレーション	経済バランスとフィードバック構造
プレイアブルプロトタイプ	実際に遊べるゲームとして実装	総合的なプレイ体験

各段階で何を検証するか

ペーパープロトタイプ： - ルールは理解しやすいか - 意味のある選択肢があるか - 基本的なゲームループは楽しいか

Machinationsプロトタイプ： - リソース経済は安定しているか - フィードバックループは適切に機能しているか - 極端な戦略でバランスが崩壊しないか

プレイアブルプロトタイプ： - 操作感は快適か - 難易度曲線は適切か - プレイヤーは意図したとおりの体験をしているか

> **重要ポイント** プロトタイプの種類ごとに検証すべき項目が異なる。低コストなプロトタイプ（ペーパー、Machinations）で基本設計を固めてから、コストの高いプレイアブルプロトタイプに進むことで、開発全体の効率を最大化できる。

6.5 デザインパターンの実践的活用

13のMachinationsデザインパターンの実装ガイド

Machinationsのデザインパターンは、ゲーム経済の構築に再利用可能な設計テンプレートである。これらのパターンを理解し適切に組み合わせることで、意図したプレイ体験を効率的に実現できる。

パターンの選択方法

ゲームの目標から逆算してパターンを選択する。

エンジン系パターン ―― リソース生産の基盤を作る： - ダイナミックエンジン：自動的にリソースを生産する構造 - エンジンビルディング：プレイヤーが生産力を構築する楽しさ - コンバーターエンジン：リソースの変換による戦略的選択

フリクション系パターン ―― バランスと緊張感を生む： - スタティックフリクション：一定量のリソース消費 - ダイナミックフリクション：状況に応じた消費量の変化 - アトリション：プレイヤー間のリソース奪い合い

エスカレーション系パターン ―― 難易度曲線を設計する： - エスカレーション：時間経過に伴う難易度上昇 - アームズレース：プレイヤー間の能力競争 - スローサイクル：長期的な戦略の波

パターンの組み合わせ例

Civilization型（ダイナミックエンジン＋ダイナミックフリクション）： - 都市がリソースを自動生産（エンジン） - 維持費が都市数に比例して増加（フリクション） - 拡張と維持のバランスが戦略の核となる

RTS型（エンジンビルディング＋アトリション）： - ワーカーと施設で生産基盤を構築（エンジン） - 軍事ユニットで敵の生産基盤を破壊（アトリション） - 経済力と軍事力のバランスが勝敗を分ける

経済ゲーム型（コンバーターエンジン＋トレード）： - 原材料を加工品に変換して価値を高める（コンバーター） - プレイヤー間の取引で市場が形成される（トレード） - 変換効率と市場の読みが成功の鍵

アンチパターン：避けるべき設計

アンチパターン	問題	対策
制御不能な正のフィードバックのみ	先行者が必ず勝つ	負のフィードバックを追加
フリクションなしのエンジン	リソースが際限なく蓄積	消費・維持コストを導入
プレイヤー選択肢のない決定的システム	最適解が一つしかない	複数の有効戦略を設計

> **重要ポイント** デザインパターンは個別に使うよりも組み合わせて使うことで真価を発揮する。「エンジン＋フリクション」の基本構造を軸に、ゲームの特性に合わせてパターンを追加していくのが効果的なアプローチである。

6.6 ワークショップ：ゲーム経済の設計演習

演習1：シンプルなリソース経済の設計

目標： Machinationsの基本操作を習得し、リソースの流れを理解する。

1. ソース→プール→ドレインの基本構造を作る
2. 流量を調整し、プールのリソースが安定するバランスを見つける
3. 流入量と流出量の比率がゲーム体験に与える影響を観察する

演習2：フィードバックループの追加

目標： フィードバック構造の効果を体験的に理解する。

1. 演習1のモデルに正のフィードバック（投資→成長）を追加する
2. シミュレーションでリソースの指数関数的増加を確認する
3. 負のフィードバック（フリクション）を追加して安定化させる
4. 正負のフィードバックのバランスを調整し、適切な成長曲線を設計する

演習3：マルチプレイヤー経済

目標： プレイヤー間の相互作用をモデル化する。

1. 2人のプレイヤーのリソースプールを作成する
2. アトリションパターン（相互のリソース消費）を追加する
3. トレードメカニズム（リソースの交換）を導入する
4. 対称・非対称の開始条件でシミュレーションし、公平性を検証する

演習4：完全なゲーム経済の構築

目標： 複数のデザインパターンを統合したゲームシステムを設計する。

1. ゲームのコンセプトを定義する（ジャンル、テーマ、目標）
2. 必要なデザインパターンを選択し、Machinationsで実装する
3. シミュレーションを実行し、バランスの問題を特定する
4. パラメータを調整し、意図したプレイ体験が実現されるまで反復する

> **重要ポイント** 演習では「正解を見つける」ことではなく「パラメータ変更がゲーム体験にどう影響するか」を体験的に理解することが重要である。シミュレーション結果を観察し、なぜそうなるかを考察する姿勢を持つこと。

第6回まとめ

テーマ	ツール/手法	主な成果
早期検証	反復的デザインプロセス	問題の早期発見とコスト削減
経済シミュレーション	Machinations	フィードバック構造の可視化とバランス調整
レベル設計	Ludoscope	ミッション/スペースの自動生成とデッドロック検出
プロトタイプ段階	ペーパー/デジタル/プレイアブル	段階に応じた検証項目の最適化
パターン活用	13のデザインパターン	再利用可能な設計テンプレートの組み合わせ
実践演習	ワークショップ	パラメータ調整の体験的理解

プロトタイピングは、ゲームデザインにおける理論と実践の橋渡しである。Machinationsによるシミュレーションとデザインパターンの活用により、直感だけに頼らない、検証に基づくゲーム設計が可能になる。

第7回：ゲームの自動生成

概要

ゲームデザインの各側面（ミッション、スペース、メカニクス）を**形式的なモデル**として扱い、モデル変換やプロシージャル生成によってゲームコンテンツを自動生成する手法を学ぶ。形式文法、書き換えシステム、グラフ文法などの理論的基盤から、実際のプロシージャルコンテンツ生成や適応型ゲームへの応用までを扱う。

---

1. モデル駆動工学のゲームデザインへの応用

モデル駆動工学（Model Driven Engineering: MDE）

ソフトウェア工学の手法で、**抽象モデルからコードやシステムを自動生成**するアプローチ。

モデル駆動工学の概念図

- **モデル変換**: あるモデル（例: ミッション構造）から別のモデル（例: 空間レイアウト）を生成する
- ゲームデザインの3つの側面をモデルとして扱う:
  - **ミッションモデル**: プレイヤーが達成すべきタスクの構造
  - **スペースモデル**: ゲーム世界の空間的構成
  - **メカニクスモデル**: ゲームの内部経済とルール（Machinationsダイアグラム）
- これらのモデル間の**変換規則**を定義することで、一貫性のあるゲームコンテンツを自動生成できる

> **重要ポイント**: モデル駆動工学をゲームデザインに適用することで、ミッション→スペース→メカニクスの変換を形式的・自動的に行える。各モデルは独立して設計・検証可能である。

---

2. 形式文法（Formal Grammars）

ゲーム構造の生成に用いる数学的基盤。

基本要素

要素	説明	例
終端記号	それ以上書き換えられない記号	`fight`, `key`, `lock`
非終端記号	書き換え可能な記号	`Mission`, `Challenge`
開始記号	生成の出発点となる非終端記号	`Start`
書き換え規則		

要素	説明	例
	非終端記号を別の記号列に変換する規則	`Mission → Challenge lock Mission`

チョムスキー階層

形式文法の表現力による分類体系。

チョムスキー階層
チョムスキー階層

タイプ	名称	特徴	用途例
タイプ3	正規文法	最も制限的。右辺は終端記号＋非終端記号1つ	正規表現、単純なパターン
タイプ2	文脈自由文法	左辺は非終端記号1つ	プログラミング言語の構文解析
タイプ1	文脈依存文法	周囲の文脈に応じた書き換え	自然言語の一部
タイプ0	無制限文法	制約なし。最も表現力が高い	チューリングマシンと等価

ミッション記述のための文法

```
Start     → Mission
Mission   → fight Boss
Mission   → fight Mission
Mission   → key lock Mission
Boss      → miniboss | finalboss
```

この文法から「fight → key → lock → fight → fight → finalboss」のようなミッション列を生成できる。

> **重要ポイント:** 形式文法を用いることで、ミッション構造の生成ルールを厳密に定義できる。チョムスキー階層はゲーム構造の複雑さに応じた文法の選択指針となる。タイプ2（文脈自由文法）がゲーム生成で最もよく使われる。

---

3. 書き換えシステム（Rewrite Systems）

形式文法を一般化した変換システム。

正規形（Normal Form）

- 書き換えシステムの標準的な表現形式
- 規則の適用方法を統一し、分析を容易にする

合流性（Confluence）

合流性の概念図

- **定義:** 異なる順序で規則を適用しても、最終的に**同じ結果に到達**する性質
- 合流性がある書き換えシステムでは、規則の適用順序に依存せず一意な結果が得られる
- ゲーム生成において、合流性は**結果の予測可能性**を保証する

停止性（Termination）

- **定義:** 書き換えが**有限ステップ**で終了する性質
- 停止しないシステムは無限ループに陥る
- ゲーム生成では停止性がなければ生成プロセスが完了しない

> **重要ポイント:** 合流性と停止性は書き換えシステムの健全性を保証する重要な性質である。ゲーム生成に用いる文法は、これらの性質を満たすよう設計すべきである。

---

4. グラフ文法（Graph Grammars）

文字列ではなく**グラフ構造**を書き換える文法。

グラフ文法の書き換え規則例

基本概念

- 書き換え規則の左辺と右辺がともに**グラフ**（ノードとエッジの集合）
- 左辺のパターンにマッチする部分グラフを、右辺のグラフで置換する
- ゲームのミッショングラフやスペースグラフの生成に適用

ミッションからスペースへの変換

1. ミッション構造をグラフとして表現
2. グラフ文法の規則を適用し、各ミッション要素を空間的な構造に変換
3. 結果として、ミッションに対応したゲーム空間のレイアウトが生成される

> **重要ポイント:** グラフ文法は、ゲームのミッション構造やスペース構造のように、線形ではない複雑な関係性を持つ構造の生成と変換に適している。

---

5. シェイプ文法（Shape Grammars）

幾何学的な形状を対象とする文法。

シェイプ文法による空間生成

プリミティブ

- **点**: 位置の指定
- **線分**: 壁、通路、境界
- **四角形**: 部屋、エリア
- その他の幾何学的図形

空間レイアウトの生成

- 書き換え規則が幾何学的変換（移動、回転、拡大縮小）を含む
- 建築設計やゲームのレベル生成に応用
- 例: 「四角形 → 四角形を4分割」「線分 → 線分＋ドア」

> **重要ポイント**: シェイプ文法は、グラフ文法がトポロジー（接続関係）を扱うのに対し、幾何学（実際の形状と配置）を扱う。ゲームレベルの物理的なレイアウト生成に有効である。

---

6. ロックと鍵の変換例

ミッション構造からロック＆キーを含む空間構造を自動生成する具体例。

6つの書き換え規則

ロック＆キー変換規則
ロック＆キー変換規則

規則	入力	出力	説明
規則1	直線ミッション	部屋の連鎖	基本的な部屋の連結
規則2	鍵の取得タスク	分岐路＋鍵部屋	鍵を取得するための脇道を生成
規則3	ロックされた進行	ロック付きドア	鍵が必要な扉を配置
規則4	分岐ミッション	分岐する通路	複数経路の空間構造
規則5	ボス戦	大部屋＋入口制限	ボス部屋の生成
規則6	オプション要素	隠し通路	任意探索の空間構造

変換プロセス

1. ミッション文法でミッション構造を生成
2. グラフ文法でミッション→空間の変換規則を適用
3. シェイプ文法で空間→幾何学的レイアウトに変換
4. 最終的な物理的レベルが生成される

> **重要ポイント:** ミッション→スペースの変換は、複数段階の文法変換として形式化できる。6つの基本規則の組み合わせでも、多様なレベル構造を生成可能である。

---

7. スペースの生成

ミッション構造から空間的構成を自動生成する手法。

- ミッショングラフの各ノードを**部屋や区画**に対応付ける
- エッジを**通路やドア**に変換する
- 制約条件（部屋サイズ、通路の長さ、鍵とロックの整合性）を満たすよう配置を最適化
- トポロジー（接続関係）から幾何学（物理配置）への変換が必要

> **重要ポイント:** スペース生成はミッション構造との一貫性を保ちながら、物理的に実現可能なレイアウトを生成する必要がある。トポロジーと幾何学の両方を考慮しなければならない。

---

8. メカニクスの生成

Machinationsダイアグラムの書き換え規則

メカニクス生成の書き換え規則
メカニクス生成の書き換え規則

- Machinationsの基本パターン（エンジン、ドレイン、コンバータなど）を書き換え規則として定義
- ゲームのジャンルや目的に応じた**テンプレート規則**を用意
- 規則の適用により、バランスの取れた内部経済を自動生成
- 生成後にシミュレーションで**バランスを検証**できる

> **重要ポイント:** メカニクスの生成は、Machinationsのパターンライブラリと書き換え規則を組み合わせることで実現できる。生成と検証をサイクルで回すことが実用上重要である。

---

9. プロシージャルコンテンツ生成

アルゴリズムによるゲームコンテンツの自動生成。

代表的な事例

ゲーム	生成対象	手法
Rogue (1980)	ダンジョンレイアウト	ランダム生成の先駆け
Diablo	ダンジョン、アイテム	ランダム＋テンプレート
Torchlight	ダンジョン階層	タイルベースの接続
Minecraft	地形全体	パーリンノイズ＋バイオーム規則

アクションアドベンチャーへの応用

- ロック＆キー構造を含むダンジョンの自動生成
- ミッション文法 → グラフ文法 → シェイプ文法のパイプライン
- ゼルダ風のダンジョンをプロシージャルに生成する研究

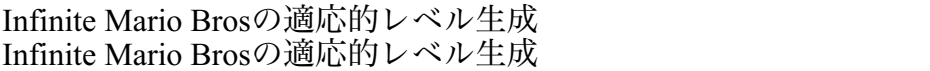
> **重要ポイント:** プロシージャルコンテンツ生成は、リプレイ性の向上とコンテンツ制作コストの削減を両立する。ただし、品質制御が課題であり、生成物の検証メカニズムが不可欠である。

---

10. 適応型ゲーム

プレイヤーの行動に基づいてコンテンツを動的に生成・調整するゲーム。

Infinite Mario Bros


Infinite Mario Brosの適応的レベル生成
Infinite Mario Brosの適応的レベル生成

- **プレイヤーモデル**に基づくレベル生成:
  - プレイヤーの行動パターン（ジャンプ頻度、敵との戦闘傾向など）を分析
  - プレイヤーの好みに合ったレベルを動的に生成
- 探索好きなプレイヤーには広い空間を、アクション好きには敵の多いレベルを提供
- 機械学習によるプレイヤーモデリング

Infinite Boulder Dash

- Boulder Dashのルールに基づく無限レベル生成
- パズル要素の難易度をプレイヤーのスキルに適応
- 文法ベースの生成と評価関数の組み合わせ

> **重要ポイント:** 適応型ゲームは、プロシージャル生成にプレイヤーモデルを組み合わせることで、個々のプレイヤーに最適化された体験を提供する。生成アルゴリズムとプレイヤー分析の統合が鍵となる。

---

11. 自動化デザインツール

混合主導型アプローチ（Mixed-Initiative）

- **人間のデザイナー**と**アルゴリズム**が協働するアプローチ
- 完全自動生成ではなく、デザイナーの意図を尊重しつつ自動化で支援
- デザイナーが方向性を決め、アルゴリズムが詳細を埋める

Ludoscope ツール


Ludoscopeの概念図

Ludoscopeの概念図

- ゲームレベルの**混合主導型**生成ツール
- **レシピ（Recipe）**: 文法規則の適用順序を定義した自動生成プラン
- **マニュアル編集**: デザイナーが生成結果を手動で調整
- 生成パイプライン:
  1. ミッション文法でミッション構造を生成
  2. グラフ変換でスペース構造に変換
  3. シェイプ文法で幾何学的レイアウトに変換
  4. 各段階でデザイナーが介入・修正可能
- レシピとマニュアル編集を自由に切り替えながらレベルを制作できる

> **重要ポイント**: 混合主導型アプローチは、完全自動生成の品質問題と完全手動設計のコスト問題の両方を解決する。Ludoscopeは、本書で学んだミッション・スペース・メカニクスのモデルを統合的に扱うツールの実例である。

まとめ

手法	入力	出力	特徴
形式文法	書き換え規則	ミッション列	理論的基盤が明確
グラフ文法	グラフ書き換え規則	ミッション/スペースグラフ	非線形構造に対応
シェイプ文法	幾何学的規則	空間レイアウト	物理的配置を生成
プロシージャル生成	アルゴリズム＋パラメータ	ゲームコンテンツ全般	リプレイ性向上
適応型生成	プレイヤーモデル＋生成規則	個別最適化コンテンツ	パーソナライズ
混合主導型	デザイナー入力＋生成規則	高品質レベル	効率と品質の両立

ゲーム生成の各手法は、ゲームデザインの形式的モデル（ミッション、スペース、メカニクス）を基盤としている。モデルを厳密に定義することで、自動生成・変換・検証が可能になる。

第8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
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

ノード	記号	機能	特徴
			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	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

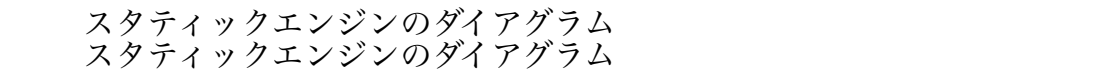
---

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）

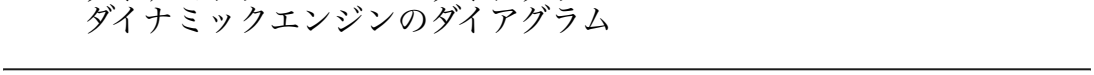
- **タイプ：** エンジン
- **意図：** 固定レートでリソースを生成する最も基本的なエンジン。ソースから一定量のリソースがプールに流れ続ける。
- **適用場面：**
  - ターンごとに固定収入を与えるシステム
  - 時間経過で一定量回復するHP/MPシステム
  - ボードゲームの基本的な資源供給
- **ゲーム例：** Descent（ターンごとの固定アクション）、バックギャモン（サイコロによる固定的移動権の供給）



スタティックエンジンのダイアグラム

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

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

#	パターン名	タイプ	主なフィードバック	代表例
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
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。