シミュレーションモデルの追加拡張が可能な新しいデジタルカーリングシステムの構築

上原嘉織[1,a)]　伊藤毅志[1,b)]

概要：近年カーリングのストーンの物理的挙動のメカニズムに迫るべく，ストーンの精緻な計測が行われるようになっている．そのような観測データを参考に比較的現実に近いカーリングシミュレーションモデルが提案されている．当研究室で開発しているデジタルカーリングにおいては，カーリング AI 開発，対戦用システムとして，実際のカーリングに近づいた新しいシミュレーションモデルをいち早く取り入れる必要があるが，一方で従来システムにおいて新しいシミュレーションモデルを取り入れる拡張は設計上困難であった．本研究ではシステムの設計を見直し，シミュレーションモデルの継続的な追加拡張に対応し，更に氷面の動的変化への対応も可能とした，新しいデジタルカーリングシステムの構築を行った．

Developing New Digital Curling System that is Extendable with Additional Simulation Models

Kaoru Uehara[1,a)]　Takeshi Ito[1,b)]

Abstract: In recent years, precise measurements of stones have been carried out in order to understand the mechanism of the physical behavior of stones in curling. A curling simulation model that is relatively close to reality has been proposed based on the observed data obtained in such studies. The Digital Curling system that have been developed in our laboratory for curling AI development and competition needs to quickly incorporate new simulation models that are closer to actual curling. On the other hand, it has been difficult to extend conventional systems to incorporate new simulation models by design. In this study, a new Digital Curling system was developed to accommodate continuous additions to the simulation model and to respond to dynamic changes in the ice surface.

1. はじめに

デジタルカーリングは当研究室の北清らが開発し [1]，後に森らによって改良された [2]，主にカーリング AI 同士の対戦を行うためのカーリングシミュレーションシステムである．以下では北清らのシステムを第一世代システム，森らのシステムを第二世代システムと呼ぶ．当研究室はこのシステムを用い，より強いカーリング AI の開発のために過去 8 年間にわたって 16 回のデジタルカーリング AI 大会を実施してきた．

[1] 電気通信大学
The University of Electro-Communications, Chofu, Tokyo 182-8585, Japan
[a)] uehara.digitalcurling@gmail.com
[b)] ito@cs.uec.ac.jp

デジタルカーリングプロジェクトの直接的な目的は強いカーリング AI を開発することだが，これは強い AI によるカーリング選手への支援を一つの動機としている．そして，そのためにはデジタルカーリングのカーリングシミュレーションをできる限り現実のカーリングに近づける必要がある．現実のカーリングに近い環境において最適化された AI によって現実のカーリングの戦術，戦略について分析し，選手の競技力向上を図る．これがデジタルカーリングプロジェクトの一大目標である．

この目標に対し，従来システムには根本的な問題点があった．システム内で実行されるカーリングシミュレーションが現実のカーリングの実測データに則していないという点である．これは，カーリングにおけるストーンの挙動のメカニズムが未解明であるばかりか，第 1 世代開発当

初はストーンの軌跡の実測データすら得られなかったことから，シミュレータにおけるストーンの挙動はあくまで仮のものとして簡易な式でおかれたためである．

我々はシステムのシミュレーションが実測値に則していない問題に対処すべく，先行研究で計測されたデータに基づいて新しいシミュレーションモデルである Friction - Curl Velocity モデル（FCV モデル）を提案した [3]．このモデルでは動摩擦係数とストーンの軌跡の曲がり（カール）の観点でストーンの挙動が計測データに近づいた．

この研究を受け，我々は作成した FCV モデルを第二世代システムに組み込み，システムのシミュレーションを現実に近づける改善を試みたが，第二世代システムはシミュレーションモデルの変更を想定していない設計であった．そして，そのような第二世代システムを改修するより新しいシステムを構築すべきと判断した．

そこで，本研究では新しいシミュレーションモデルを継続的に組み込んでいける新しいデジタルカーリングシステムの構築を行った．新システムは FCV モデルの組み込みのみに限らず，今後も随時新モデルを組み込み，拡張していけるものを想定して構築された．なぜなら，FCV モデルでは現実のカーリングにおける実験事実に再現できないものがあるばかりか，近年カーリングにおいて今まで以上に精緻な観測が行われるようになっていることから [4]，今後はより現実のカーリングに近いシミュレーションモデルが提案されることが予想されるためである．以下，本研究で構築した新しいデジタルカーリングシステムは第三世代システムと呼ぶ．

2. デジタルカーリング

本節では特に指定しない限り全ての世代のデジタルカーリングシステムに共通する事項を述べる．

2.1 基本機能

デジタルカーリングの基本となる機能は思考エンジン同士の対戦である．対戦サーバープロセスに対し接続した 2 つのカーリング思考エンジンプロセス（クライアント）の間でカーリングの試合のシミュレーションが行われる．通信方式は TCP/IP（第一・第三世代）あるいはプロセス間通信（第二世代）を使用し，専用のプロトコルで通信を行う．

大まかな通信の流れを**図 1** に示す．まず思考エンジンがサーバーに接続し，両思考エンジンの接続後，試合設定がサーバーから送信され，両思考エンジンが応答し，試合が始まる．試合が始まった後はサーバーが両思考エンジンに毎ターン試合状況を送信し，手番の思考エンジンは自身が採る行動をサーバーに送信する．サーバーでは思考エンジンから送信された行動をもとに 1 ターンのシミュレーションを行い，更新された試合状況をまた両思考エンジンに送信する．このようにして試合が進み，試合が終了したら試合終了通知がサーバーから両思考エンジンに送信され，接続が解除され通信が終了する．

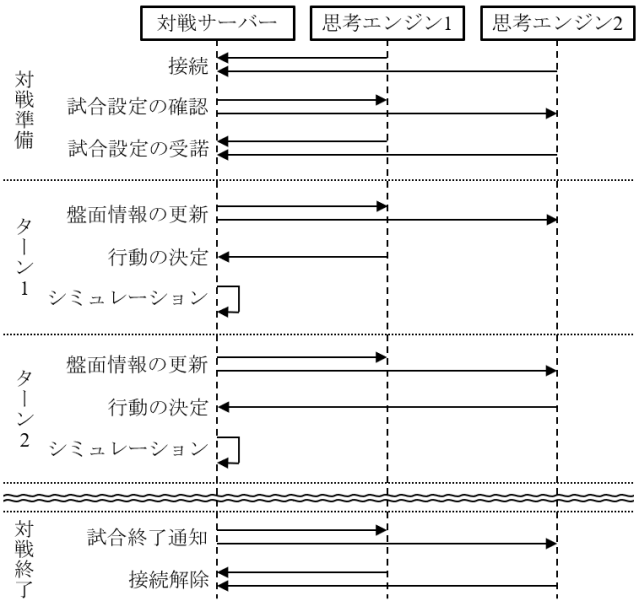

図 1 デジタルカーリングの対戦における通信

2.2 ライブラリ

対戦サーバーは内部でカーリングのシミュレーションを行う．一方，思考エンジンも通常先読みのためにシミュレーションを行うが，先読み精度を重視する場合シミュレーション処理はサーバーと同じものにすべきである．そこでデジタルカーリングではサーバーが行うカーリングシミュレーションをライブラリとしてまとめて提供している．これにより思考エンジン制作者はライブラリをリンクするだけでサーバーと同じシミュレーションを行える．

まとめると，デジタルカーリングのライブラリはカーリングシミュレーションに関連する機能を提供し，主に対戦サーバーと思考エンジンから参照されることを想定している．

2.3 構成要素

デジタルカーリングが提供する主な要素を次に示す．

- ライブラリ
- 対戦サーバー
- GUI

ライブラリと対戦サーバーについては前述の通りである．GUI は対戦ログ再生（全世代共通）や人間用対戦クライアント（第一世代のみ）の機能があり，主にカーリングの試合内容を人間にとって分かりやすく表示する役割がある（**図 2**）．

これらはすべて C++で記述されている．また，これら以外にも思考エンジン作成のためのサンプルプログラムや各種マニュアルが提供される．

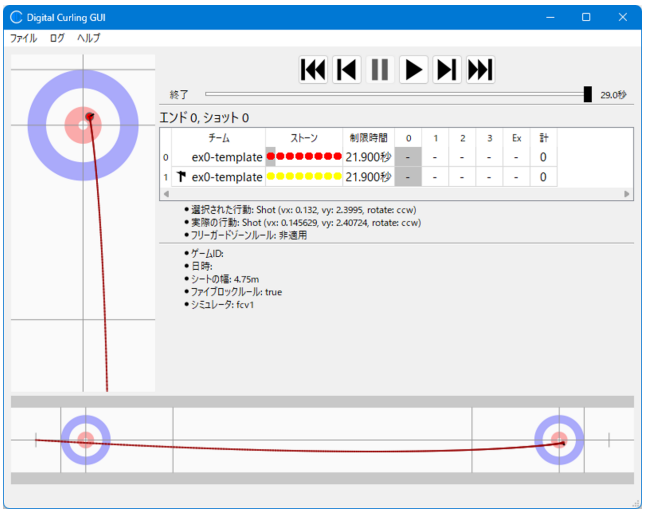

図 2 デジタルカーリング GUI（第三世代システム）

3. システムへのシミュレーションモデルの追加

第三世代システムは従来システムへのシミュレーションモデルの追加が困難であったために開発された．ここではこの点について従来システムにおける問題点と第三世代システムでの解決方法について説明する．

3.1 従来システムにおける問題点

3.1.1 第一世代システム

第一世代システムのライブラリにおいて中心となる，シミュレーションを行い試合を 1 ターン進める関数 `Simulation()` のシグネチャを**図 3** に示す．第一引数 `pGameState`（`GAMESTATE` 型）は試合状況を表す構造体で，具体的には現在のターン数，試合のエンド数，スコア，手番，ストーンの位置を格納する．関数の実行後にはこの構造体の内容が次のターンの情報に更新される．第二引数 `Shot`（`SHOTVEC` 型）はショットの初速と回転方向を表す．第三引数 `Rand` はショット前に初速（第二引数 `Shot` に含まれる）に加えられる乱数の大きさについてのパラメータである．これは現実のカーリングで人間によってショットが行われることに起因するショットの不確定性（ひいてはカーリングのゲームとしての不確定性）を表現するための機能である．第四引数 `lpResShot` は実際に行われたショット，つまり第二引数 `Shot` 内の初速に対し乱数が加えられたものが返る．第五引数 `LoopCount` はシミュレーションの最大ステップ数を指定する．

この関数には物理シミュレーションの挙動を制御する引数が存在しない．これは意図的なもので，関数内で行うシミュレーションの内容を特定しないことで，関数内部の挙動を自由に変更することができ，使用するシミュレーションモデルを別のものに差し替えることが可能になっていた．

一方で，シミュレーションモデルはそのままに摩擦係数

```
int Simulation(
  GAMESTATE *pGameState,
  SHOTVEC Shot,
  float Rand,
  SHOTVEC *lpResShot,
  int LoopCount);
```

図 3 第一世代システムのシミュレーション関数

```
class Simulator {
public:
  Simulator(
    float friction,
    float friction_stone);

  int Simulation(
    GameState* const game_state,
    ShotVec shot_vec,
    float random_x,
    float random_y,
    ShotVec* const run_shot,
    float *trajectory,
    size_t traj_size);

  // ...
};
```

図 4 第二世代システムのシミュレーション関数（クラス ISimulator のメンバの一部は省略されている）

等のシミュレーションの動作に関連するパラメータを変更するような場合は，ライブラリのソースコードを編集し，再ビルドする必要があった．これは物理シミュレーションのパラメータを頻繁に変更する場合問題になる．そして，現実のカーリングでは氷面の状態が変化しストーンの滑りが変化することを考えると，物理シミュレーションの摩擦等のパラメータは実行時に変更できるようにすべきである．そして，第二世代システムではこの点が改善された．

3.1.2 第二世代システム

第二世代システムにおいてシミュレーションを行う関数 `Simulation()` のシグネチャを**図 4** に示す．第一世代システムからシグネチャは変更されているが，試合を 1 ターン進めるという基本的な機能は変更されていない．

第一世代システムの反省を受け，物理シミュレーションに関するパラメータはクラス `Simulator` のコンストラクタの引数として指定できるようになっている．コンストラクタの第一引数 `friction` は氷面-ストーン間の摩擦，第二引数 `friction_stone` はストーン-ストーン間の摩擦についてのパラメータである（なお，`friction_stone` は実装上のミスにより動作上無視されていたことが分かっている）．

第二世代システムでクラス `Simulator` のコンストラクタの引数として明示されているシミュレーションのパラメータは，第一世代システムから引き続き採用されていた

```
1  void ApplyMove(
2    GameSetting const& setting,
3    ISimulator& simulator,
4    IPlayer& player,
5    GameState& state,
6    Move& move,
7    std::chrono::milliseconds const&
8        thinking_time_used,
9    ApplyMoveResult* result,
10   std::function<void(ISimulator const&)>
11       on_step);
```

図 5 第三世代システムのシミュレーション関数

簡易モデルにおけるパラメータである．このように，第二世代システムは使用するモデルに特化した作りになっており，シミュレーションモデル自体の変更が想定されていなかった．従って，第二世代システムのライブラリは破壊的変更（ライブラリにリンクするアプリケーションの動作が保証されない変更）無しでは内部で使うシミュレーションモデルの変更ができない．

3.2 新システム（第三世代システム）における解決

第一，第二世代の問題点を踏まえると，シミュレーションモデルを継続的に追加できるようにするためには，シミュレーションモデル自体の変更を可能としつつ，実行時にシミュレーションのパラメータを変更できるような仕組みが必要と考えられる．第一世代システムは前者のみに，第二世代システムは後者のみに対応していた．そして，第三世代システムではこの両者への対応を行った．以下では，それぞれへの対応について説明する．

3.2.1 シミュレーションモデルの変更

第三世代システムにおけるライブラリのシミュレーションモデルの変更方法について述べる前に，このライブラリの主要部分であるシミュレーションを行う関数について述べる．

3.2.1.1 シミュレーション関数

第三世代システムにおいて試合のシミュレーションを行う関数 `ApplyMove()` のシグネチャを**図 5** に示す．この関数は第一，第二世代における関数 `Simulation()` と同じように試合を 1 ターン進める．第一引数 `setting`（`GameSetting` 型）は試合中に変化しない試合設定（何エンドゲームか，追加ルールを適用するか等）を示す．第二引数 `simulator`（`ISimulator` 型）は氷面とストーンの物理法則を記述する．第三引数 `player`（`IPlayer` 型）は現実のカーリングで人間のプレイヤーがストーンを投げることに起因するショットの不確定性について記述する．第四引数 `state`（`GameState` 型）は現在の試合状態を格納し，関数呼び出しが返ると次のターンの状態に更新される．第五引数 `move`（`Move` 型）は手番側が行う行動（ショットもしくはコンシード）であり，ショットの場合ではストーンの初速度と回転方向が格納される．そのほか，第六引数 `thinking_time_used` はこのターンに消費した思考時間，第七引数 `result` は関数の動作結果の詳細，第八引数 `on_step` はストーンの軌跡を得るために使用されるコールバックである．

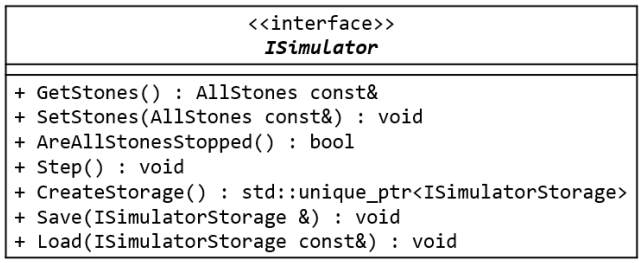

図 6 インターフェース ISimulator（メンバ関数の一部は省略されている）

3.2.1.2 インターフェースによるモデル変更への対応

関数 `ApplyMove()` は試合を 1 ターン進めるが，そのために必要なストーンの物理シミュレーション（滑走，カール，衝突など）についての処理は `ISimulator`（**図 6**）に一任されている．具体的には関数 `ApplyMove()` は引数として渡された `ISimulator` 型の変数 `simulator` のメンバ関数を呼び出すことでシミュレーションを行う．これは大まかに次のようなものである．

(1) `SetStones()` を呼び出し，盤面上のすべてのストーンの位置と速度を設定する．発射対象のストーンは発射時の初速度で発射位置に，それ以外の盤面上のストーンは停止した状態で対応する位置に配置される．
(2) `AreAllStonesStopped()` の返り値が `true` になるまで，つまりすべてのストーンが停止するまで，`Step()` を繰り返し呼び出し，シミュレータ内部での時間を進める．
(3) 全ストーンの停止後，`GetStones()` を呼び出し，ストーンの配置を得る．ここで得たストーンの配置に基づいて得点計算が行われる．

なお，実際には `GetStones()` と `SetStones()` は上述以外にも呼び出され，エリア外に出たストーンの除去が行われるが，ここでの説明は省略している．

ここで重要なことは `ISimulator` がオブジェクト指向におけるインターフェースである（つまり上記メンバ関数はすべて純粋仮想関数である）ため，実際に呼び出される関数は実行時に渡される `simulator` の実行時型情報（RTTI）に依ることである．これにより，使用する物理シミュレーションモデルを実行時に変更することが可能である．

インターフェース `ISimulator` の実装上の条件は上記の処理が成立するもの，つまり，「ある速度で投げたストーンがしばらくすると停止する」というもののみである．この条件を満たしさえすれば他のストーンの物理的性質（摩擦，カール，衝突についての性質）を問うことは無い．従って，

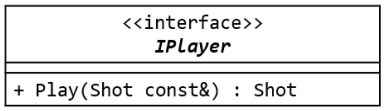

図 7 インターフェース IPlayer（メンバ関数の一部は省略されている）

ISimulator はカーリングとして現実的なシミュレーションモデルであればどのようなものであっても対応可能であると考える．

IPlayer（**図 7**）も ISimulator 同様にインターフェースであるが，こちらは現実のカーリングで人間のプレイヤーがショットを行うことに起因するストーンの初速のブレや初速最大値の制限を担当する．

IPlayer の主要なメンバ関数は Play() のみである．ApplyMove() は第三引数として渡された player の Play() を呼び出す．Play() の引数としては ApplyMove() の第五引数 move がそのまま渡される．この move は手番側が行いたい理想的なショットを意味するもので，Play() では理想のショットに対しプレイヤーによってランダムなブレが加えられ，実際に行われるショットに変換される．そして変換後のショットが ISimulator に入力され，物理シミュレーションが行われる．

メンバ関数 Play() は純粋仮想関数であるため，こちらも実行時に使用されるアルゴリズムを変更することができる．

Play() の実装上の条件は特に無く，入力された理想的なショットに対しどのような値を返しても構わない．従って，IPlayer はカーリングのプレイヤーのショットのブレを再現する現実的なモデルならばすべて対応可能であると考える．

まとめると，第三世代システムは ISimulator や IPlayer の実装を行うことで新しいシミュレーションモデルに対応できる．そして，現実的なモデルであればあらゆるモデルに対応可能と考える．

3.2.1.3 再利用性の向上

第三世代システムでは，従来システムで一緒くたに処理されていたカーリングのシミュレーションが，ルール部分（ApplyMove()，GameSetting，GameState が担当する），物理部分（ISimulator が担当する），プレイヤー部分（IPlayer が担当する）の 3 つに分けて処理されると言える．これは各部分の再利用性の向上というメリットをもたらしている．

例えば，新しい ISimulator の実装として新しい物理シミュレーションモデルを追加し，そのモデルを使う場合も，ルール部分とプレイヤー部分は元々利用していたもの継続して利用できる．また，現実のカーリングでは数年ごとに細かいルールが変更されているが，例えばルール変更によって，試合状況を表す GameState に新しいフラグ変数が必要になり，ApplyMove() の内の処理が一部変更になったとしても，物理部分とプレイヤー部分はそのまま利用できる．

3.2.2 シミュレーション関連パラメータの実行時変更

第三世代システムではルール部分，物理部分，プレイヤー部分のそれぞれで実行時にパラメータの指定を行える．以下ではそれぞれの指定方法について説明する．

3.2.2.1 ルール設定の変更

ルールに関する設定パラメータは ApplyMove() の第一引数 GameSetting が格納しており，メンバ変数の値を指定することで適用するルールを指定できる．具体的には GameSetting を介して，試合のエンド数，シートの横幅，ファイブロックルールの有無，各思考エンジンの持ち時間を指定することができ，ApplyMove() はその内容に従って試合ルールの適用を行う．

3.2.2.2 物理シミュレーション設定の変更

物理シミュレーション，つまり ISimulator に関する設定は ISimulator の生成時に行う．ISimulator は ISimulatorFactory のメンバ関数 CreateSimulator() で生成される．ISimulatorFactory も ISimulator 同様インターフェースであり，従って CreateSimulator() は純粋仮想関数である．これはオブジェクト指向におけるデザインパターンの Abstract Factory パターンを踏襲している．

ISimulator に対するパラメータ設定については具体例で説明を行う．**図 8** は我々が作成した FCV モデルを実装した SimulatorFCV1 とそれを生成する SimulatorFCV1Factory についてのクラス図である．SimulatorFCV1 ではステップ毎の経過時間を変更できるが，これは SimulatorFCV1Factory のメンバ seconds_per_frame を設定した後，CreateSimulator() を呼び出すことで行う．これによってステップ毎の経過時間を変更した SimulatorFCV1 を利用することができる．このように，ISimulator のパラメータの設定は ISimulatorFactory を介して行う．

ISimulatorFactory は構造化されたデータフォーマットである JSON との相互変換が可能である．JSON のキー"type"でシミュレーションモデルを指定することができ（FCV モデルを実装した SimulatorFCV1 の場合"fcv1"），"seconds_per_frame"といった対応するパラメータも設定可能である．

3.2.2.3 プレイヤー設定の変更

ショットのランダム性については IPlayer の管轄だが，パラメータの設定は IPlayerFactory による生成時に行う．これは ISimulator，ISimulatorFactory と同様であるため説明は省略する．

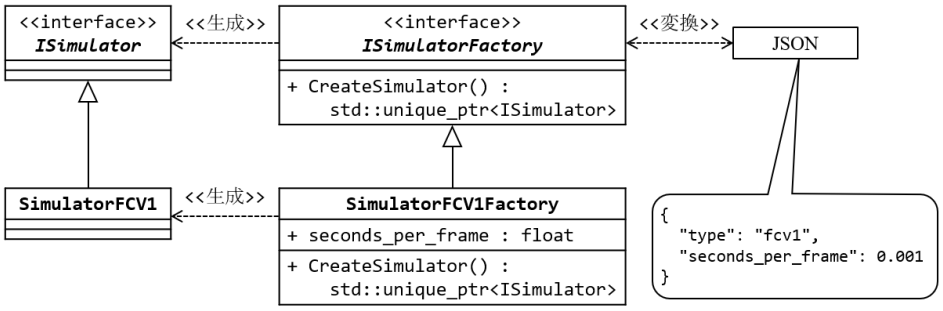

図 8 ISimulatorFactory による ISimulator の生成（メンバの一部は省略されている）

4. 新システム（第三世代システム）における追加・変更点

4.1 シミュレーションモデルの継続的追加拡張への対応

第三世代システムではシミュレーションモデルの継続的な追加拡張への対応を行った．これについては 3 節で述べた．

現状では ISimulator として，我々が昨年作成した FCV モデル [3] が実装されている（図 8）．また，IPlayer として，ショットの初速ベクトルの大きさと角度にそれぞれ正規分布に従う乱数を加える正規分布乱数プレイヤー，入力された理想的なショットを加工しない恒等写像プレイヤーが実装されている．正規分布乱数プレイヤーは人間のプレイヤーを想定した実装であり，例えば対戦サーバー内のシミュレーションに用いる．一方，恒等写像プレイヤーは用いることでショットにブレが生じない確定的なシミュレーションが行えるため，思考エンジンの先読み等に向いている．

新しいシミュレーションモデルの追加（ISimulator, IPlayer の実装の追加）は我々ライブラリ開発側だけでなく，ライブラリにリンクして使用する側からでも可能である．そのため，例えばシミュレーションの高速化のために思考エンジン開発者側で独自に ISimulator として高速な物理シミュレーションモデルを実装し，ApplyMove() に渡して使用することが可能である．

4.2 氷面の動的変化への対応

現実のカーリングではストーンの通過やスウィープによってストーンの滑りが試合中にも動的に変化することが知られている．第三世代システムはこのような氷面の動的な変化に対応している．これは氷面の局所的な変化も含む．この対応のため次に示す 2 つの機能を実装した．

- 物理シミュレーション状態の保存：例えば木探索を行う場合を考える．ある盤面 A の先の盤面をシミュレーションする場合，盤面 A での候補手から一手選んで盤面をシミュレーションした後，一度盤面 A に戻ってきて，更に候補手から他の一手を選びシミュレーションを行う．この際，盤面の状態をあらかじめ別の場所に保存し，シミュレーションを行った後に元通りに復元する操作が行われる．ISimulator にはこのような盤面情報の保存のために CreateStorage(), Save(), Load() というメンバ関数が定義されている．これら関数とその他の構造体を併用することで任意の時点でシミュレーション状態の保存，復元が可能である．
- 通信プロトコルへの軌跡データの追加：従来システムでは通信プロトコルにはストーンの軌跡データ（各時刻に対する位置データ）は含まれていなかった．従来システムのシミュレータではストーンの初速が決まれば軌跡は一意に定まるため，サーバーはプレイヤーによるブレが加えられた後の初速をクライアントに送信すれば，クライアント側でシミュレーションを行い，軌跡を復元できたためである．一方，物理シミュレーションモデルに氷面の動的変化を許容するとストーンの初速に対し軌跡は一意には定まらなくなる．そこで，第三世代システムの通信プロトコルには軌跡データを含めている．

4.2.1 現状の実装

現状では氷面の動的変化を再現する物理シミュレーションモデルは実装されていない．この機能は将来的な実装のために追加された．

4.3 通信プロトコルの変更

第三世代システムではシミュレーション設定を変更することができるが，サーバー内でどのようなシミュレーション設定で試合が行われるかは試合の前に思考エンジンも知っておく必要がある．そこで，通信プロトコルとしてシミュレーション設定は含める必要がある．また，前節の氷面の動的変化への対応に伴って，ストーンの軌跡データも通信プロトコルに含めることになった．このような通信プロトコルに含めるデータの追加は今後も起こりうるが，一方で，従来システムのテキストを空白と改行で区切ったシンプルな通信プロトコルでは互換性を維持した拡張が難しいという問題があった．

そこで第三世代システムの通信プロトコルは JSON ベースのものに変更した．JSON は辞書として構造化されてい

表 1 デジタルカーリングシステムの機能

機能	第一世代	第二世代	第三世代
シミュレーションモデルの変更	対応	非対応	対応
シミュレーション設定の実行時変更	非対応	対応	対応
ミックスダブルスルール	非対応	対応	非対応（将来的に対応予定）
プレイヤー毎の設定変更	非対応	対応	対応
氷面の動的変化	非対応	非対応	対応
スウィーピング	非対応	非対応	非対応
通信プロトコル	v1.0	v1.1	v2.0（1.x と互換性無し）
通信方式	TCP/IP	プロセス間通信	TCP/IP
ログ再生 GUI	対応	部分的に対応	対応
対戦中の試合状況の GUI 表示	対応	非対応	非対応（将来的に対応予定）
人間用 GUI 対戦クライアント	対応	非対応	非対応（将来的に対応予定）

るため，通信プロトコルの互換性を維持した拡張も容易である．また，JSON は人間にとって可読性が高く，メジャーなフォーマットであるため多くのライブラリが提供されていて解析も容易という利点がある．一方で，通信プロトコルの変更によって第二世代システム以前の思考エンジンとは通信ができなくなってしまった．

なお，ライブラリ内のほぼすべての構造体，クラスは JSON との相互変換機能をサポートしている．従って，ライブラリを用いることで通信プロトコルの解析が容易に行える．

4.4 マルチプラットフォームへの対応

第三世代システムは Windows，Linux といった複数のプラットフォームに対応している．このために CMake[*1] という C/C++用のマルチプラットフォームに対応したビルドツールを用いた．CMake を導入したことで異なるプラットフォーム間で共通のプロジェクトファイルを使用でき，また，異なるプラットフォームであっても同じコマンドを用いてビルドすることが可能となっている．そのため，マルチプラットフォームに対応しながらも管理はそこまで複雑化していない．

5. システムの機能のまとめ

各世代のシステムの機能をまとめて**表 1** に示す．

表 1 の項目「プレイヤー毎の設定変更」は第二世代システムで導入されたもので，カーリングの通常ルールではエンドごとにチームを構成する 4 人が 2 投ずつ投げるが，このプレイヤーごとの技量の差を再現するための機能である．第三世代システムでもこの機能に対応している．

従来システムで対応していたが第三世代システムでは対応できていないこととして，ミックスダブルスルール，対戦中の試合状況の GUI 表示，人間用 GUI 対戦クライアントの機能がある．これらは順次対応していく予定である．

現実のカーリングでプレイヤーが行うスウィーピングについては従来システム同様に対応していない．スウィーピングの効果については未だに不明な点も多く，どのような実装が適当であるか判断しがたいためである．

6. おわりに

本研究では従来のデジタルカーリングシステムで問題となった新しいシミュレーションモデルの導入についての問題を解決するため，従来システムの問題を分析し，その問題を解決した新しいデジタルカーリングシステムを構築した．新システムはオブジェクト指向のインターフェースを用いることで現実的なあらゆるモデルに対応できるように設計された．また，システムには将来的な氷面の動的変化への対応を見据えた機能が実装された．

今後は，まず本システムを用いたデジタルカーリング AI 大会を実施する予定である．また，従来システムにあった現状の第三世代システムで不足している機能の実装も順次進めていく．更に，取得されつつある現実のカーリングの観測データを分析し，より現実のカーリングに近いシミュレーションモデルを作成し，第三世代システムへの導入を進めていく．これにより，現実のカーリングに近い環境で動作するカーリング AI の開発に貢献していく．

参考文献

[1] 北清勇磨，伊藤毅志：カーリングの戦略を支援するシステムの提案と構築，The 18th Game Programing Workshop 2013 (2013).

[2] 森健太郎，伊藤毅志：条件の変更にロバストなデジタルカーリングの改良，情報処理学会論文誌，Vol.60，No.11，pp.2085-2092 (2019).

[3] 上原 嘉織，伊藤 毅志：実環境データに近づけるデジタルカーリングのシミュレータの改良，情報処理学会ゲーム情報学研究会，GI-46(18)，pp.1-8 (2021).

[4] Murata, J.: Study of curling mechanism by precision kinematic measurements of curling stone’s motion. *Sci. Rep.* **12**, 15047 (2022).

[*1] https://cmake.org/