修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工学研究科　情報・ネットワーク工学専攻　博士前期課程		
氏　　名	森　健太郎	学籍番号	1631151
論 文 題 目	**条件にロバストなデジタルカーリングの改良**		

要　旨

「デジタルカーリング」は，カーリングの戦略を議論することを目的として 2014 年に当研究室の北清らによって開発されたカーリングプラットフォームである．このシステムを広く公開したことでカーリングの戦略が統一された場で議論されるようになり，カーリング AI の発展に大いに寄与してきた．しかし,「パラメータの変更が考慮されておらず，ルールの変更に柔軟に対応できないこと」「プレイヤの技量差や氷の状態のような，実際のカーリングでは重要なパラメータの変更が容易でないこと」などが問題点として指摘されていた．

本研究では，従来のデジタルカーリングシステムの問題点を踏まえ，ルールや物理シミュレーションのパラメータの変更に対して柔軟に対応できる改良システムを新たに構築した．従来システムでは散逸していて変更が難しかったパラメータをひとまとめにし，ゲーム進行に必要な関数群もまとめたゲーム進行クラスを定義した．これにより，ルールだけでなく物理シミュレーションに必要なパラメータ等の変更もゲームサーバから行えるようになった．また従来システムで表現されていなかったプレイヤによる技量差を，乱数の大きさやショットベクトルの最大値などといった形で表現できるようにした．

本システムを用いて GPW 杯 2018 大会を，通常ルールに加えカーリングの変則ルールであるミックスダブルス部門の競技を実施して，ルールの変更に対して柔軟な対応が可能であることを示した．変則ルールへの対応や技量差の導入を行うことにより，現実に近いより複雑な条件に対応したゲームを再現することを可能とした．これにより，改良システムがカーリング AI 研究のより一層の発展に寄与することが期待できる．また本研究室で別途開発されているカーリングの学習支援システムへ本システムを利用することで，様々なパラメータの変更に対応できる実用性の高い支援への応用も期待される．応用の一例として，ショットのウェートの上限を変えた状況でカーリング AI によるショット選択の実験を行ったところ，条件によって違うショットの選択肢が提示されることを確認した．

平成 30 年度　修士論文

条件にロバストなデジタルカーリングの改良

学　籍　番　号　　1631151

氏　　名　　森 健太郎

電気通信大学　情報理工学研究科
情報・ネットワーク工学専攻　コンピュータサイエンスコース

主任指導教員　伊藤　毅志　准教授
指導教員　小林　聡　教授

2019 年 3 月 14 日

目次

1. 序論 5

2. 関連研究 6

2.1 ロボカップリーグ 6

2.2 Computational Pool 7

2.3 デジタルカーリング 8

2.3.1 デジタルカーリングシステム 8

2.3.2 デジタルカーリングを用いた学習支援 9

3. 従来のデジタルカーリングシステム 10

3.1 システムの概要 10

3.2 物理シミュレータ 11

3.2.1 概要 11

3.2.2 ショットのシミュレーション 12

3.2.3 乱数 14

3.2.4 ライブラリのインターフェース 15

3.3 ゲームサーバ 16

3.3.1 概要 16

3.3.2 試合条件の設定 17

3.4 通信プロトコル 18

3.4.1 概要 18

3.4.2 コマンドの詳細 18

3.5 問題点 20

3.5.1 システム全体の問題点 20

3.5.2 物理シミュレータ 20

3.5.3 ゲームサーバ 22

3.5.4 通信プロトコル 22

4. システムの改良 23

4.1 システムの概要 23

4.2 物理シミュレータ 24

4.2.1 概要 24

4.2.2 ショットのシミュレーション 24

4.2.3 乱数生成器 25

4.2.4 ライブラリのインターフェース 26

4.3 ゲームサーバ 29

4.3.1 概要 29

4.3.2 試合進行クラス 29

4.3.3 プレイヤ毎の技量差の表現 32

4.3.4 設定ファイル 33
4.3.5 ミックスダブルスルールへの対応 35
4.4 通信プロトコル 36
4.5 改良システムの動作例 38
4.5.1 通常ルールにおける動作例 38
4.5.2 ミックスダブルスルールにおける動作例 43
5. 改良システムを用いた大会の実施 46
5.1 概要 46
5.2 レギュレーション 46
5.3 大会結果 47
6. 結論 48
7. 今後の展望 51
謝辞 53
参考文献 **エラー! ブックマークが定義されていません。**
付録 54
カーリングのルール 54
改良シミュレータのコード 54
ゲームサーバ（試合進行制御部のコード例 58
設定ファイルの例 60

1. 序論

「デジタルカーリング」とは，カーリングの戦略のみを議論することを目的として当研究室の北清が 2014 年に開発し、発表したカーリングプラットフォームである[1][2]．このシステムは 2D 物理シミュレータを用いて氷上のストーンの動きをできるだけ平易な物理法則で近似して模倣することで，カーリングゲームを擬似的にコンピュータ上でプレイすることを可能にしている．このシステムを介して人間のプレイヤだけでなく，カーリングショットを生成し対戦できるプログラム（以下，カーリング AI と呼ぶ）との対戦も可能となり，年に数度大会が開催されている[3]．このシステムが広く公開され，様々なカーリング AI を同じ条件で対戦できるようになったことで，カーリング AI の発展に大いに寄与してきた[4][5].

このシステムは上述のような成果を収めた一方で，大会参加者から「ショットによるストーンの挙動に不自然な点が存在すること」や「ルールの変更に柔軟に対応出来ないこと」などの問題点も指摘されている．また，本システムとカーリング AI を併用することでカーリングの学習支援の手法が検討されているが[6]，実際のカーリングプレイヤからは，実際の戦略を議論するためには，「ストーンの軌跡が実環境に比べて不自然に感じる点」や「氷の状態やプレイヤのスキルを表現できない点」，「GUI の操作性が悪い点」などの問題点も指摘されている．

現状のデジタルカーリングシステム（以下，旧システムとする）では，現実のカーリングと比べ簡略化している要素（氷の状態変化やプレイヤの技量，スウィープなど）が存在する．カーリングを実環境に近づけるためには，これらの要素を導入することが必要であると考えられる．しかしながら，カーリングにおけるストーンの物理運動に関してはその物理現象自体の原理が解明されていない点が多く，実測データもほとんど存在しないことから，現状ではこれらの要素をただちに取り入れることは難しい．カーリング技術の進歩に伴い近年ルールの変更が行われたことや，ミックスダブルスなどの全く新しいルールが導入されたことなど，レギュレーションの変化も目まぐるしく，デジタルカーリングシステムもこれらの変化に対応する必要がある．

このようにカーリングは様々な要素の状態変化を含むゲームであり，従来のシステムではこれらの変化を十分に表現されていないといえる．表現されている要素に関しても，旧システムの構築時は変更を想定しておらず，システム内でそれぞれのパラメータが散逸していることや，変更不可能な定数として定義されているといった問題がある．よってこれらの問題に対応するためには，システム全体の再構築が必要であると考えられる．

そこで本研究では，これら従来のシステムの問題点に対する改良を行う．具体的には，「ルールや大会レギュレーションの変更に関するパラメータをゲーム開始時に変更可能とする」「氷の状態変化やプレイヤの技量といった要素の追加を可能とする」ことを盛り込んだシステム（以下，改良システムとする）を新たに実装する．この改良システムを用いた大会を実施し，システムの有用性の評価を行う．

2. 関連研究

2.1 ロボカップリーグ

ロボカップリーグは，人工知能やロボット工学といった技術を用いて，サッカーや災害救助といった現実世界の問題解決を行うエージェントの作成を行うプロジェクトである．主なプロジェクトとしてロボカップサッカー[6]，ロボカップレスキュー[7]があり，それぞれに条件の異なるサブプロジェクトが存在する．

ロボカップサッカーは，「2050 年までに人間のワールドカップチャンピオンに勝てるロボットチームを作る」ことを目的として，北野らによって提案された最初のロボカップである．ロボカップサッカーでは実際のロボットがサッカーをプレイする競技の他に，コンピュータ上の仮想的なエージェントが 11 対 11 で対戦を行うシミュレーションリーグが存在する．

この競技では，現実のサッカーにおけるプレイヤに見立てた自立エージェントが，それぞれ独立に思考を行う．それぞれのエージェントが得られる情報（他のエージェントの位置やボールの位置）や蹴ったボールには意図的に誤差が加えられている．このように不完全情報・不確定性を含むゲームであるという点において，人工知能研究の盛んな囲碁や将棋といったチェスライクゲームよりチャレンジングな題材であるといえる．

ロボカップレスキューは地震のような大規模災害の現場で，災害救助の支援を行う自立エージェントを開発すること目的としたプロジェクトである．このプロジェクトもロボカップサッカーと同様に，実際のロボットを用いて災害救助を行う競技と，コンピュータ上で仮想的な災害現場で防災活動を行うエージェントを競うシミュレーション部門が存在する．

これらのプロジェクトはいずれも現実世界の問題を題材として行う競技であるが，現実に存在する要素を全てシミュレーションに加えることは困難である．「どの程度現実の要素を簡略化するか」「どの程度意図的な誤差を加えるか」といった点は競技性に大きくかかわるため，大会を重ねるごとに試行錯誤が行われている．

2.2 Computational Pool

Computational Pool（コンピュータビリヤード）は，ビリヤードのゲームをコンピュータ上で再現したものであり，通常 8 ボールでのプレイが行われる[8]．キューと呼ばれる棒状の道具で手玉を突くショットは，キューの角度と手玉を突く座標，キューの速度のパラメータで表現される．ゲームとしては状態・行動の連続性や, ショットのパラメータにかかる乱数による不確定要素など, カーリングと共通する点が多く存在する．

このシステムを用いて AI プログラム同士の対戦が可能であり，Computer Olympiad にて 2005 年から 2008 年にかけて大会が行われた．各大会での乱数の大きさといったパラメータは，大会開始直前に参加者に通知され，AI プログラムはその場で異なる乱数の大きさに対応することが求められる．

カーリングと異なる点として場の状態変化が存在せず，ボールの挙動も一定である点が挙げられる．物理シミュレーションを伴う手番の処理が当時の計算機資源に対して負荷が大きいこともあり，1 手読みのゲーム木探索や盤面の状態を評価する関数の設計といった手法が用いられたが，2008 年以降大会は行われておらず，研究論文も発表されていない．

2.3 デジタルカーリング

2.3.1 デジタルカーリングシステム

「デジタルカーリング」は，カーリングの純粋な戦略のみを議論することを目的として，当研究室の北清が開発し発表したカーリングプラットフォームである[1][2]．このシステムでは 2D 物理シミュレータである Box2D を用いてカーリングにおけるストーンの動きを模倣することで，カーリングのゲームをシミュレートする．実際のカーリングに存在する様々な不確定要素は一組の乱数として表現することで簡略化が図られている．このシステムを用いることで，GUI を介した人間のプレイヤの対戦が行える．また，通信プロトコルを介することで，カーリング AI の対戦も可能となっており，年に数回このシステムを用いたデジタルカーリング大会が開催されている．

このシステムが公開されたことにより，統一された条件での AI 同士の対戦が可能となった．ゲーム情報学の分野では，カーリングは不確定性や状態・行動の連続性を含むゲームとして分類される．ゲーム AI 研究が盛んな囲碁や将棋と比べ，これらの点においてより複雑なゲームであるといえる．

カーリング AI 研究の例として，加藤らの将棋やチェスの AI で用いられる手法である局面評価関数とゲーム木探索をデジタルカーリングに適用する研究が挙げられる[4]. この研究では, 候補手（すなわちショット）を離散化し，乱数を加えずショットのシミュレーションを行う．ショット後の局面評価値を計算し，デジタルカーリングにおける乱数の分布が既知であることを利用して，近傍のショット後の局面評価値との加重平均を計算することで，あるショットの乱数を考慮した評価値を近似する．この手法を用いることで，多数の候補手の乱数を考慮した評価値を，少ない時間で計算することに成功している．局面評価関数については，ヒューリスティックな関数を用いていたが，のちに機械学習を用いて訓練を行い，ある局面の得点期待値を出力するニューラルネットワークを作成している．

大渡らは囲碁等で用いられるモンテカルロ木探索をデジタルカーリングに適用した[5]．デジタルカーリングにモンテカルロ木探索を適用するにあたり，局面が連続的であり取りうる局面が無数に存在することや，1 手進めるのに物理シミュレーションを伴うため，ランダムにプレイを行うプレイアウトの回数を十分に稼げないという問題がある．大渡らは状態木と呼ばれるデータ構造を導入し，プレイアウトを十分な回数行なえていない局面ついては似た局面をまとめ上げることでこの問題に対処している．

デジタルカーリングシステムが公開されたことで，このように複雑なゲームとしてのカーリング AI 研究が盛んに行われ，その発展に大いに寄与してきたといえる．

2.3.2 デジタルカーリングを用いた学習支援

樋口らは，デジタルカーリングとカーリング AI を用いた，カーリングの戦略学習支援システムの提案を行った[9]．このシステムでは，条件編集ダイアログを用いてストーンの配置やエンド数・投数・得点差を作成し，その時点での着手（ショット）を複数のカーリング AI に思考させる．カーリング AI が出力したショットをストーンの軌跡として盤面上に表示し，自由にアニメーションとして再生できるようにすることで，カーリングの戦略の学習支援を行う．

このシステムは従来のデジタルカーリングシステムを拡張する形で作成されており，条件編集画面と候補手表示画面から構成される．条件編集画面ではマウスによるドラッグドロップとキーボードによる入力を用いて，現在の投数・エンド数，これまでの得点数，ストーンの配置を編集することが出来る．編集した局面は保存・読み込みが可能である．この局面を候補手表示画面で読み込ませ，任意のカーリング AI を指定して思考を行わせる．カーリング AI との通信は，デジタルカーリングシステムで標準的に用いられるプロトコルを用いて行うため，大会上位プログラム等をそのまま使用することが出来る．

しかし，カーリングの氷の条件やプレイヤの個性を表現するデータ構造を有していないため，その表現できる状況は限られている．実際のカーリングの状況を再現するためには，曲がりやすいシートだったり，滑りにくい重たいアイスコンディションだったりというような細やかな氷の条件を自由に変更できる仕様が求められる．

3. 従来のデジタルカーリングシステム

3.1 システムの概要

カーリングの通常ルールに基づいてローカルコンピュータ上にカーリング AI（または人間）同士の試合を行えるゲームサーバが，「Digital Curling - Client」として公開されている[10]．ゲームサーバには物理シミュレータが Dynamic Link Library（DLL）形式で同封されている．ユーザは GUI から試合条件とカーリング AI のパスを指定することで，カーリング AI 同士の対戦を行うことが出来る．

図 3-1 にシステムの概要図を示す．システムはゲームサーバと物理シミュレータから構成される．このほかにもシステムは試合ログの再生機能なども含むが，本論文では試合の実行に関する部分についてのみ扱うため，ここでは割愛する．

ゲームサーバは，グローバル領域に定義された試合進行を行う関数群から構成され，同様にグローバル変数として定義された各種パラメータ群を参照しながら，試合進行の制御を行う．ゲームサーバから実行時に物理シミュレータを読み込むが，物理シミュレーションに関わるパラメータの多くは定数として定義されているため変更することが出来ない．

現状のデジタルカーリングシステム（このシステムを便宜上，「従来システム」と呼称する）では氷のコンディションやプレイヤの技量といった不確定要素は，初速度ベクトルの各成分に正規乱数を加えることで表現される．氷のコンディションの変化や，個々のプレイヤの技量の差は考慮されておらず，またスウィーピングの要素も実装されていない．

なお，本章以降で取り扱うカーリングの基本的なルールと用語を巻末に付録として添付した．

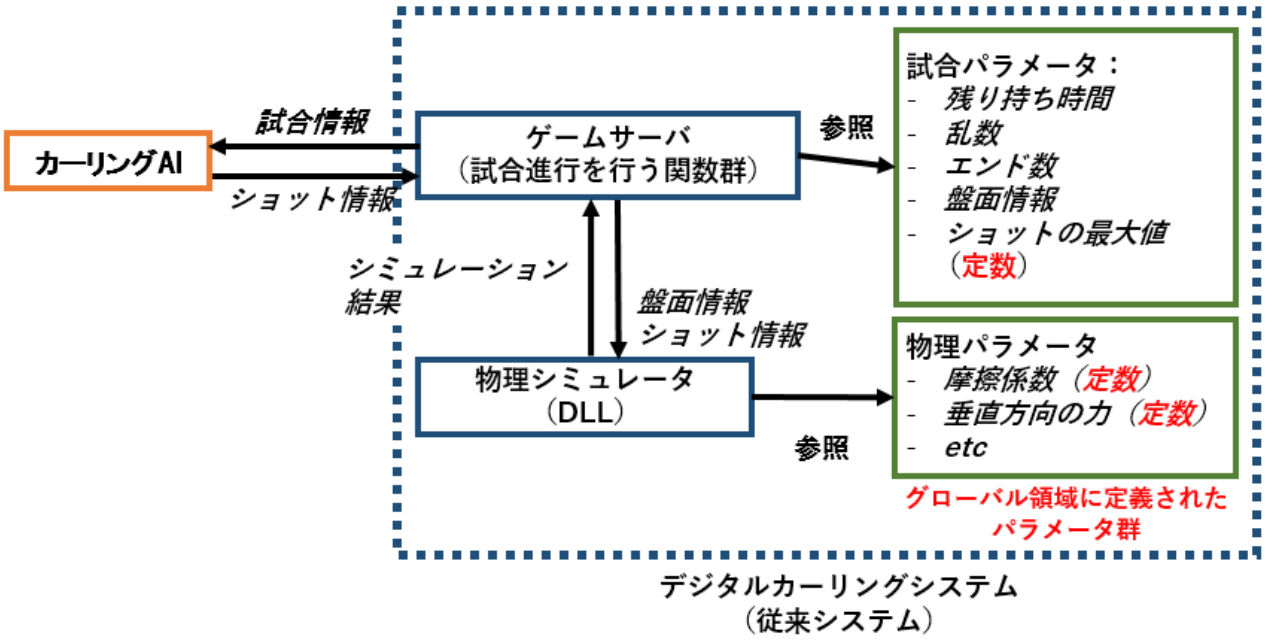


図 3-1 従来システムの構成

3.2 物理シミュレータ

3.2.1 概要

ゲームサーバで用いられる物理シミュレータのライブラリは「Curling Simulator」としてゲームサーバとともに公開されている．このライブラリは DLL 形式で配布されており，ゲームサーバやカーリング AI が実行時に読み込むことで利用することが出来る．現在の最新バージョンは ver.2.3 および大会用に(x, y)成分にかかる乱数の比率を変更した ver2.4GAT となっている．

シミュレータ内におけるストーンの大きさや，プレイエリア・ハウスといったリンク内の各エリアの寸法は実際のカーリングに準拠した値となっている．図 3-2 にシミュレータ内の座標系を示す．リンク内の各部の座標は，ストーンの進行方向を上としてリンクの左上を原点とし，x軸は右方向，y軸は下方向を正とする直交座標系で表され，z軸方向（リンクに対して垂直方向）については考慮しない．

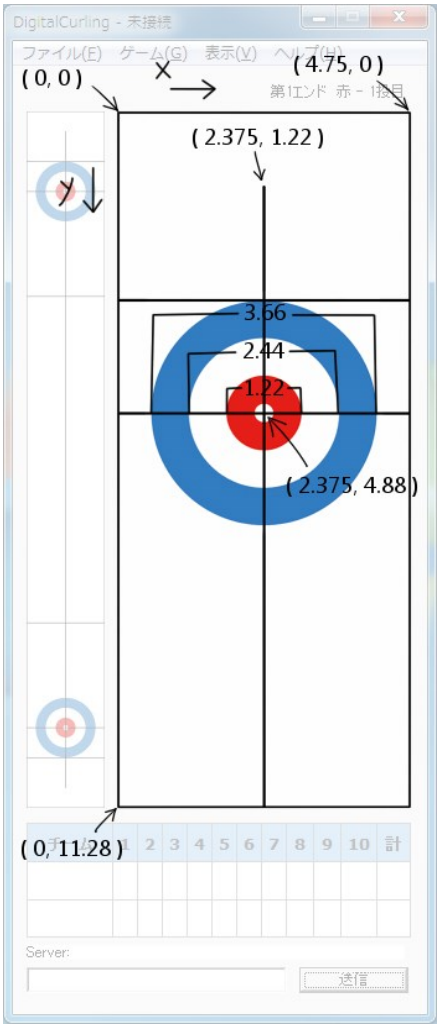


図 3-2 デジタルカーリングにおける座標系[10]

3.2.2 ショットのシミュレーション

カーリングのストーンが曲がる（カールする）物理的な原理は解明されておらず，厳密な物理シミュレーションによってストーンの動きを再現することは困難である．そこで物理シミュレータでは，ストーンのあるステップ数での速度ベクトルに対して，垂直方向の定数ベクトルが加わるものとしてストーンの動きをモデリングしている．定数ベクトルが加わることにより速度ベクトルが増大するため，摩擦による減速を考慮して速度ベクトルの調整を行う．以下に計算の手順を示す．

あるステップ数でのストーンの速度ベクトルをv，ストーンと氷の摩擦力をa，垂直の力をbとする．まず減速のために速度ベクトルvからaを引く（図 3-3）．このベクトルに対して垂直方向のベクトルbを加える（図 3-4）．

このままでは速度ベクトルの大きさが$|\sqrt{(v-a)^2+b^2}|$となるため，もとの$|v-a|$へ大きさを調整する（図 3-5）．このベクトルをシミュレーションの次のステップにおける速度ベクトルとし，これを$|v| \leq 0$となるまで繰り返すことでシミュレーションを行う．

なお，ストーン同士の衝突処理は，オープンソースの 2D 物理シミュレータである Box2D v.2.1.2[11]を用いる．衝突後のストーンの軌道については直線運動と仮定し，垂直方向の力は考慮しない．

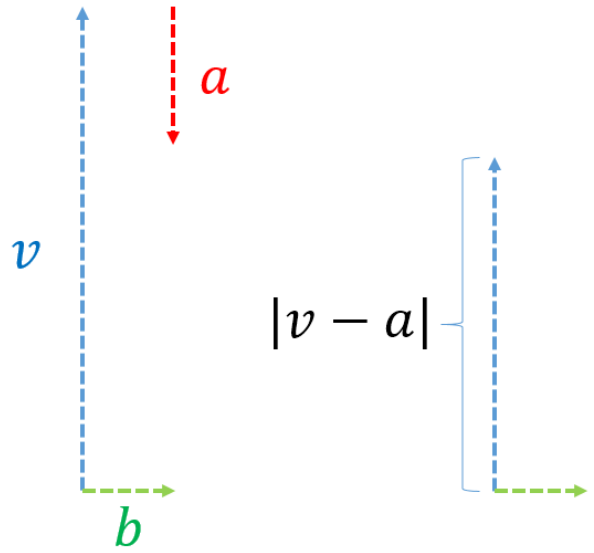


図 3-3 ショットベクトルの計算 1

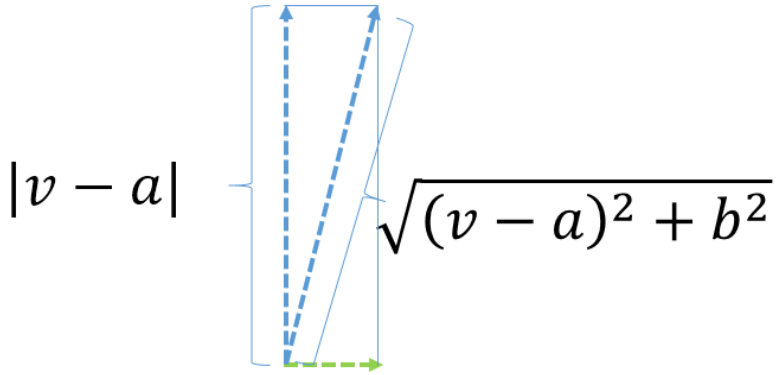


図 3-4 ショットベクトルの計算 2

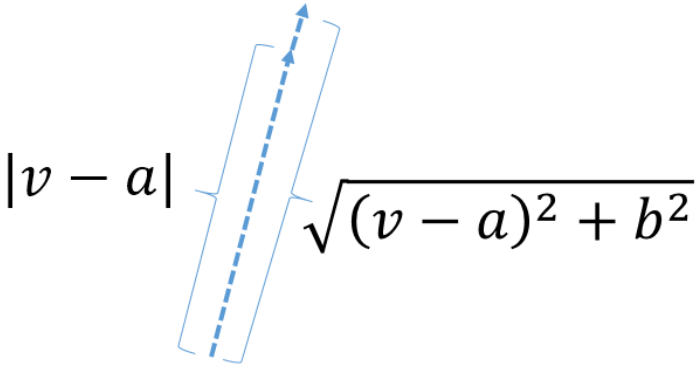


図 3-5 ショットベクトルの計算 3

3.2.3 乱数

実際のカーリングでは，氷のコンディションやプレイヤのコンディション・技量に由来するショットの精度，ストーンの個体差による挙動の変化といった様々な不確定要素が存在する．これらの不確定要素を全て個別に考慮するとゲームが複雑になりすぎることや，これらの要素がどの程度ショットに影響するか数値化することが困難であることから，デジタルカーリングでは不確定要素を一組の正規乱数で表し，これをショットの初速度ベクトルに加えることで不確定要素の簡略化を行っている．

また実際のカーリングでは，氷やプレイヤのコンディションは時間とともに変化するが，デジタルカーリングでは時間変化を考慮せず，乱数の値はゲーム中常に一定であると仮定している．

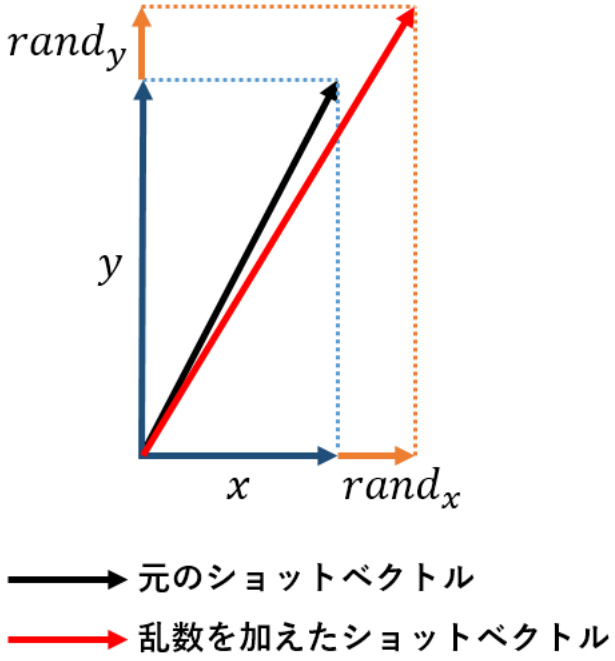


図 3-6 従来システムにおける乱数モデル

3.2.4 ライブラリのインターフェース

ライブラリのヘッダファイルでは，局面の状態を表す GAMESTATE 構造体や，ショットベクトルを表す SHOTVEC 構造体・ストーンの座標を表す SHOTPOS 構造体が定義されている．

このライブラリで提供される関数として，Simulation 関数，CreateShot 関数，CreateHitShot 関数がある．これらの関数について以下で解説する．

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

pGameState：現在の局面

SHOTVEC：実行するショットの初速度ベクトル

float Rand：初速度ベクトルにかかる乱数の大きさ

lpResShot：乱数を加えた初速度ベクトルを格納する変数

LoopCount：シミュレーションを行うステップ数の上限

現在の局面に対して，指定されたショットの物理シミュレーションを行う関数である．局面情報は現在局面が格納された変数を上書きすることで更新される．

CreateShot(SHOTPOS, SHOTVEC *ShotVec)

SHOTPOS：ショットの目標座標

SHOTVEC *ShotVec：生成したショットの初速度ベクトルを格納する変数へのポインタ

目標座標で停止するショットのベクトルを生成する関数である．

CreateHitShot(SHOTPOS, float Power, SHOTVEC *Shotvec)

SHOTPOS：ショットの目標座標

float Power：ショットの強さを表す数値（0.0-32.0）

SHOTVEC *ShotVec：生成したショットの初速度ベクトルを格納する変数へのポインタ

指定された強さの，目標座標を通過するショットを生成する関数である．

3.3 ゲームサーバ

3.3.1 概要

ゲームサーバでは通信プロトコルを介して AI との情報のやり取りを行い，物理シミュレータを用いてショットのシミュレーションを行うことで，試合進行の制御を行う．従来システムにおいて試合進行を行う関数は GUI やログの再生機能といった機能を実現する関数とともに，グローバル関数として定義されている．すなわちゲームサーバというものが他の機能と明確に区別されているわけではないが，本論文では便宜上試合進行を行う関数群を従来システムにおけるゲームサーバとして扱う．ただし試合進行関数のなかには，試合条件の設定（3.3.2 節）のように GUI を用いて実現している機能も存在するため，GUI についても本節で簡単に解説する．

GUI は，現在のストーンの配置や得点といった試合中の情報を表示するダイアログ（図 3-6）や，試合条件やカーリング AI の実行ファイルのパスを指定するダイアログ（図 3-7）を提供する．

ゲームサーバでは，ある局面の情報（ストーン座標，ショット数，エンド数，これまでの得点）に対して，ショットの初速度ベクトル（x, yの直交座標系で表現される）および曲がり（カール）の方向を与えると，物理シミュレーションが実行され次の局面が生成される．これを各エンド 16 投分行うと，エンド終了時の得点が計算される．これを規定エンド数繰り返し，合計得点を計算することでカーリングのゲームをシミュレートする．

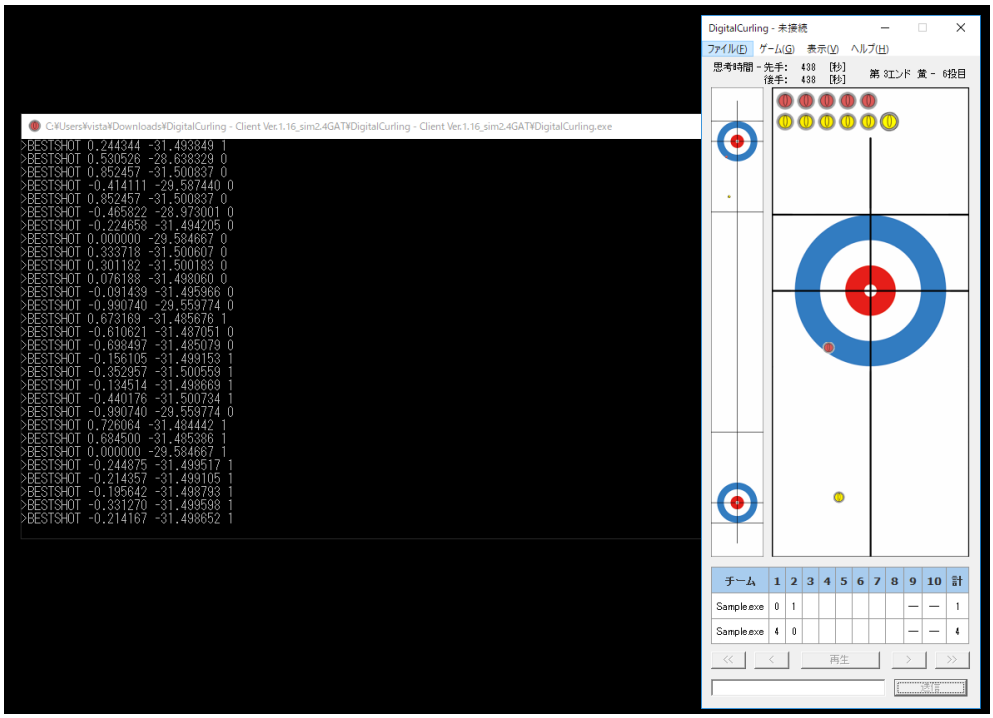


図 3-7 従来システムの GUI

3.3.2 試合条件の設定

ゲームサーバでは試合開始前に, 条件設定ダイアログにて試合条件の指定が行える. ダイアログを図 3.3.2.1 に示す. 指定可能な条件を以下に示す.

1. 先手・後手のプレイヤ情報（カーリング AI であれば実行ファイルのパス・人間プレイヤであればプレイヤ名）
2. 先手・後手それぞれの思考時間
3. 乱数の大きさ（x, y共通）
4. 試合のエンド数
5. 連続対戦の有無

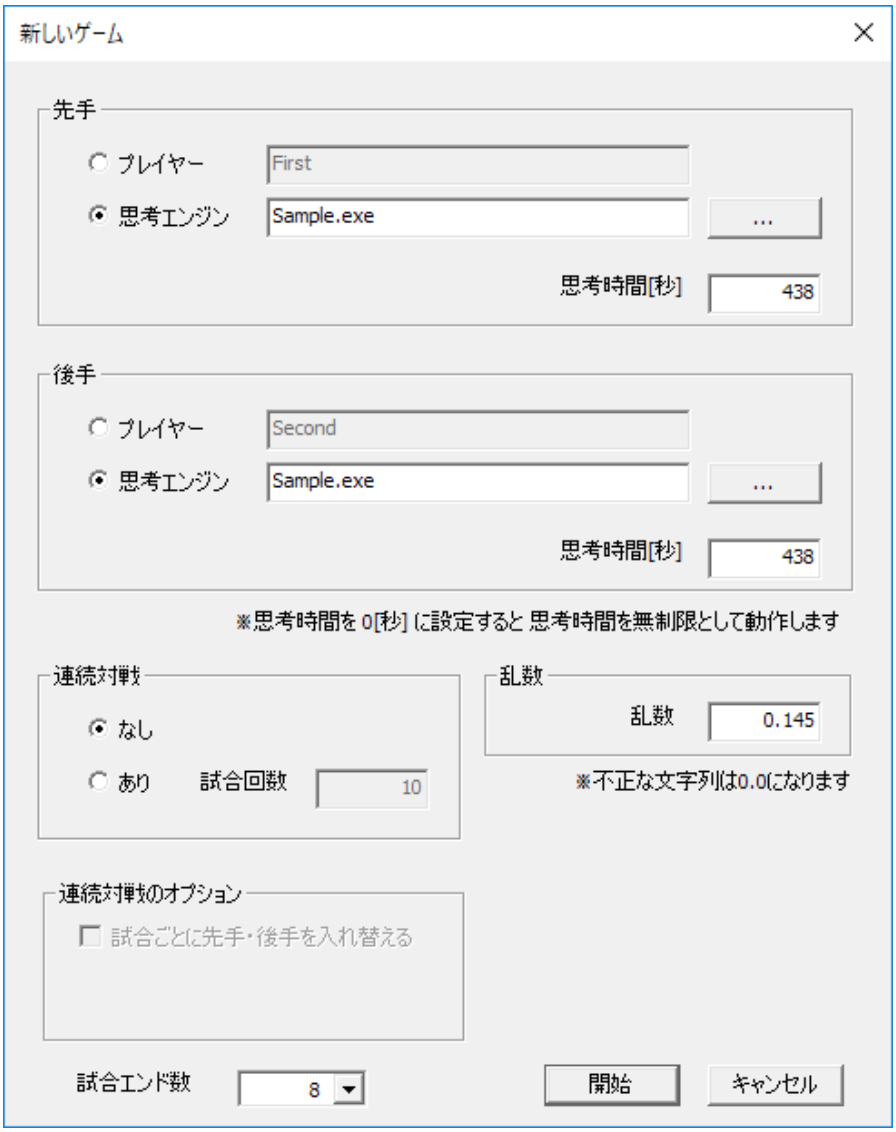


図 3-8 試合条件設定ダイアログ

3.4 通信プロトコル

3.4.1 概要

ゲームサーバとカーリング AI の通信を行うプロトコルが用意されており，公式サイト上で公開されている[10]. この通信プロトコルでは様々なコマンドが定義されており，試合情報のやり取りに加え，ゲームの開始や終了・カーリング AI の試行開始といった指令の通知に用いられる.

3.4.2 コマンドの詳細

以下に具体的なコマンドと引数を示す.

- ISREADY（ゲームサーバ→カーリング AI）

  カーリング AI の準備完了を確認するコマンドである.

- READYOK（カーリング AI→ゲームサーバ）

  カーリング AI が準備完了を通知するコマンドである．カーリング AI は ISREADY コマンドを受信したら READYOK コマンドを返さなければならない. ISREADY コマンドを受信してから READYOK コマンドを返すまでの間に，カーリング AI は必要な初期化処理を行うことができる.

- NEWGAME *name1 name2*（ゲームサーバ→カーリング AI）

  試合開始を通知するコマンドである．NEWGAME コマンドの通知から GAMEOVER コマンドの通知までが 1 試合となる． それぞれのプレイヤの名前が name1・name2 の引数として与えられる.

- SETSTATE *shot_num current_end last_end move*（ゲームサーバ→カーリング AI）

  現在の試合情報を通知するコマンドである．ショット数を表す shot_num（0 から 15 の整数），現在のエンド数を表す current_end（0 から 9 の整数），最終エンド数を表す last_end（1 から 10 の整数），手番情報を表す move（0 または 1）の 4 つの引数が与えられる．手番情報を表す move が 0 の場合，第 1 エンドで先手のプレイヤの手番となる.

- POSITION *x0 y0 ... x15 y15*（ゲームサーバ→カーリング AI）

  各ストーンの座標を通知するコマンドである．まだ投げられていないストーンを含む 16 個のストーンのx座標とy座標が通知される．プレイエリア外およびまだ投げられていないストーンの座標は(0,0)で表される．SETSTATE コマンドおよび POSITION コマンドは GO コマンドの直前に両プレイヤへ通知さ

れる．

・ GO *timelimit1 timelimit2*（ゲームサーバ→カーリング AI）

カーリング AI に思考開始を通知するコマンドである．第 1 エンドで先手・後手のプレイヤの残り制限時間を表す timelimit1・timelimit2 が引数として与えられる（単位はミリ秒）．

・ BESTSHOT *x y angle*（カーリング AI→ゲームサーバ）

カーリング AI が投げたいショットの初速度ベクトルを通知するコマンドである．カーリング AI は GO コマンド受信後，制限時間内に BESTSHOT コマンドを返さなければならない．制限時間を超えた場合は時間切れとみなされ，直ちに試合が終了する．

・ SCORE *n*（ゲームサーバ→カーリング AI）

得点を通知するコマンドである．引数として，第一エンドの先手からみた得点が与えられる．$n > 0$ならば先手の得点，$n < 0$ならば後手の得点，$n = 0$ならばブランクエンド（どちらも得点しない）となる．SCORE コマンドは各エンドの終了後に通知される．

・ GAMEOVER *win/lose/draw*（ゲームサーバ→カーリング AI）

試合終了を通知するコマンドである．それぞれのプレイヤの勝敗もしくは引き分けに応じて，win・lose もしくは draw が引数として与えられる．

3.5 問題点

3.5.1 システム全体の問題点

システム全体の問題点として，「それぞれのパラメータが散逸しており参照・変更が困難であること」「カーリングのゲームを表現するのに必要なパラメータが十分に存在しない，もしく変更不可であること」があげられる．

システムの概要図（図 3-1）にあるように，試合に関するパラメータや物理シミュレーションに関するパラメータはゲームサーバと物理シミュレータがそれぞれ独立して保持している．このほかにも，大会毎に変更が行われているショットの最大値（カーリング AI が指定可能な初速度ベクトルの最大値）やカーリング AI の実行ファイルのパスといった情報がグローバル変数として個別に定義・保持されている．このようにそれぞれのパラメータが散逸していることから，ゲームサーバ（を操作するユーザ）が参照もしくは変更することが困難である．

カーリングのゲームには「様々なルールが存在すること」「氷のコンディションが時間ごとに変化すること」「各チーム内でそれぞれのプレイヤが異なる技量を持つこと」など試合毎に変化する要素を持っている．これがカーリングの戦術を複雑にしているとともに，カーリングのゲームとしての醍醐味でもある．しかし，従来システムではこれらの要素は表現されていないことから，カーリングのゲームを十分に表現したシステムであるとは言えない．

デジタルカーリングは「カーリングのゲームをコンピュータ上で再現するシステム」である．よってカーリング AI 研究の発展やカーリングの学習支援の観点からも，これらの要素を取り入れて，ゲームサーバから容易に参照・変更が行えるべきである．

3.5.2 物理シミュレータ

ショットのシミュレーション

従来システムにおけるショットの曲がり（以下，カールとする）は，ベクトルに対して垂直方向に一定の力がかかるものとして表現されているが，現実のカーリングプレイヤからはカールが弱く感じると指摘されている．ベクトルにかかる垂直方向の力の大きさ（プログラム内では定数値で表演されている）を変更することでカールの強さを変更することは可能である．

しかしながら 3.2.4 節で述べたように，ショット生成関数が現状の設定値に依存したヒューリスティックな計算式を用いているため，この定数値を変更することが困難である．よって，カールの強さの変更へ対応するためには，定数値に依存しないショット生成関数を設計する必要がある．

また，従来システムのコードはマジックナンバーを用いたハードコーディングが多用されているという実装上の問題もあり，パラメータの変更をより困難にする要因となっている．今後変更を行うためにはこれら

の数値も整理したうえで，より理解しやすいコーディングを行う必要があると考えられる．

現実のカーリングと比較すると，特にストーンが静止する直前のカールが弱いという指摘もされており，ベクトルに対して一定の力がかかるというモデルの妥当性にも疑問が残る．

乱数モデル

従来システムの乱数モデルでは，直交座標系で表したショットの初速度ベクトルの各成分(x, y)に対して，独立した正規乱数を加えることでショットのぶれを表現している．しかしながら現実のカーリングプレイヤは，「投げる方向」と「投げる強さ」を決めてショットを行っていると考えられるため，この乱数モデルは不自然であるという指摘がされている．

よって，ショットの始点を原点とする極座標系で表した初速度ベクトルの各成分に対して乱数を加えることで，より現実に近い乱数モデルを実現できると考えられる．

ライブラリのインターフェース

現在公開されているシミュレータのライブラリでは，ゲーム情報を保持するのに必要な基本的な構造体と，ショットを生成する関数，およびシミュレーションを行う関数が用意されている．シミュレーション関数に渡すことのできるパラメータは乱数の大きさのみである．物理シミュレーションに関わるパラメータ（たとえば，3.2.2 で述べたストーンと氷の摩擦係数や垂直方向の力の大きさ）の多くはコンパイル時定数として定義されており，容易には変更することが出来ない．これに加え 3.5.1 節で述べたように，これらの定数に依存したヒューリスティックな処理も存在しており，大会等でレギュレーションの変更があった場合に対応することが困難であるという問題がある．

3.5.3 ゲームサーバ

条件変更への対応

従来システムで試合条件として「先手・後手の持ち時間」「乱数の大きさ」「エンド数」を指定することが出来る．過去のデジタルカーリング大会では主にこれらの条件を変更することでレギュレーション変更への対応を行っていた．これ以外の変更点については，3.5.1 節で述べたように各種パラメータがコンパイル時定数として定義されているため，これらの数値を変更したうえでビルドしなおしたシステムを公開することで対応を行った．

例として，2016 年 11 月に開催された第 2 回 GPW 杯以降，乱数の大きさはダイアログで指定した数値に対してx方向が$1/2$倍，y方向が 2 倍となるように変更されている．この変更に対しては，シミュレータをビルドしなおすことで対応したため，通常の乱数を用いるバイナリと合わせて異なる仕様のシステムが 2 つ公開されている．このように同じバージョンで仕様の異なる複数のバイナリが存在することは大会等で混乱の原因となっていた．

プレイヤ毎の技量差の再現

カーリングは通常 1 チーム 4 人のプレイヤで構成され，それぞれのプレイヤが異なる技量を持つ．そこで現実のカーリングの試合ではそれを考慮した上でショットを投げる順番を決定するなど，プレイヤの技量がチームの戦略に影響を与えていることが知られている．

現状のシステムではプレイヤ毎の技量差は考慮されていないことから，これを導入することでより現実のカーリングに近い条件にて試合を行えるシステムが望ましいと思われる．

3.5.4 通信プロトコル

従来システムで用いられているプロトコルはシステムの開発初期に策定されたものであり，ルールやレギュレーションの変更を想定していない．そのため，乱数の大きさのような試合条件を通知するコマンドの不足が指摘されている．

また，現実のカーリングではある程度得点差が付いた際に投了（コンシードと呼ばれる）が行われる．デジタルカーリングではコンシードを行うコマンドが用意されていないため，逆転が不可能な得点差であっても試合が続行してしまうことがしばしば起こる．大会における時間短縮や，機械学習のために大量の試合を実行する際の効率を考慮すると，コンシードを行うコマンドを追加するべきであると考えられる．

4. システムの改良

4.1 システムの概要

3.5 節で述べた旧システムの問題点を踏まえ，物理シミュレータおよびゲームサーバの再構築を行う．再構築した物理シミュレータとゲームサーバを合わせて新たに「Digital Curling」システムと呼称するが，本論文では便宜上，これを改良システムとする．通信プロトコルの見直しも行い，新たに Digital Curling Protocol として定義した．

改良システムの概要図を以下に示す．改良システムにおけるゲームサーバは各試合の進行を制御するクラス（試合進行クラス）のインスタンスと，設定ファイルのパスを保持する．試合進行クラスは，いわばカーリングのゲームそのものを表現したクラスであり，試合情報（ルールの種類やエンド数）やシミュレータ・各カーリング AI（本章以降では 4.3.3 節で述べるチーム内のプレイヤと区別するため，カーリング AI をクライアントと呼称する）の情報といった，カーリングのゲームの表現に必要なパラメータを全て一括で保持する．これらのパラメータは試合の実行時に設定ファイルから読み込まれ，試合毎に変更を行うことでルール・レギュレーションの変更に対応する．また，このクラス内で試合の進行を行う処理をまとめた関数群（「試合進行関数群」とする）が定義されており，「試合進行制御部」が試合進行関数を順次呼び出すことで，試合の進行を行う．

従来システムにおいて散逸していたパラメータを一括で管理できるように変更を加えることで，今後のルール・レギュレーション変更に対して柔軟に対応できるシステムの構築を行った．これに加え，改良システムでは，ルール変更の一例としてミックスダブルスカーリングへの対応を行い，プレイヤ毎の技量差を表すパラメータの導入を行った．

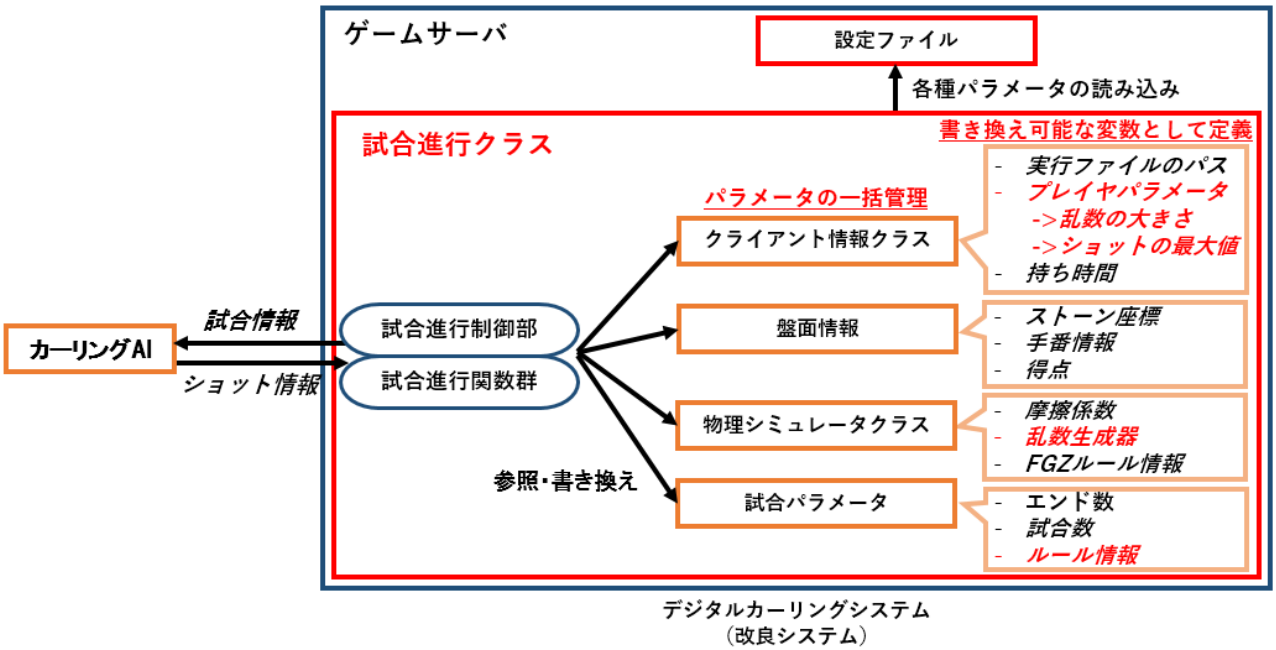


図 4-1 改良システムの構成

4.2 物理シミュレータ

4.2.1 概要

改良システムにおける物理シミュレータライブラリ（以下, 改良シミュレータとする）を新たに構築した．ストーン物理運動のシミュレーションに関しては，従来システムの手法を踏襲しつつ，今後のルール・レギュレーション変更を想定し，パラメータの変更に対して柔軟に対応できるように関数の整備を行った．

4.1 で述べたように，改良シミュレータはゲームサーバのビルド時に静的にリンクされる．従来システムを用いた大会においては，公式の物理シミュレータを用いるクライアントは多く存在しているため，改良システムにおいても物理シミュレータを単体で利用することが考えられる．改良シミュレータを含めシステムのソースコードは公開されているため，改良シミュレータを単体でビルドすることで，静的あるいは動的ライブラリとして利用することが可能である．

4.2.2 ショットのシミュレーション

3.5.1 節で述べたように，ショットの運動モデルの妥当性については疑問が残る．現状ではストーンがカールする物理的な原理が解明されていないことや，実際のストーンが描く軌跡の実測データも十分に存在しないことから，従来システムよりも妥当な運動モデルを構築するのは困難である．

そこで改良シミュレータにおいても同様に速度ベクトルに対して垂直方向の力を加える点や，衝突の処理に Box2D を用いる点など，基本的なシミュレーション手法は従来システムのものを踏襲する．なお，使用する Box2D のバージョンは，2018 年時点で最新版である v.2.3.0[11]とした．

4.2.3 乱数生成器

乱数モデルに関しては，従来システムで用いられていた直交座標系の乱数生成器に加え，極座標系で表した初速度ベクトルの各成分へ乱数を加える乱数生成器を実装した．

シミュレータ内部ではショットベクトルはすべて直交座標系(v_x, v_y)で表されるため，この乱数生成器ではこれを極座標系(v, θ)（v：ベクトルの大きさ，θ：角度）へ変換する．C++の標準ライブラリを用いて独立した正規乱数rand_v，rand_θを生成し，これらを加えたベクトル$(v + \mathrm{rand}_v, \theta + \mathrm{rand}_\theta)$を直交座標系に変換し実際に投げるベクトルとする．

なお，直交座標系と極座標系どちらの乱数生成器を用いるかは，ゲームサーバ側から設定可能としている（4.2.3 節参照）．

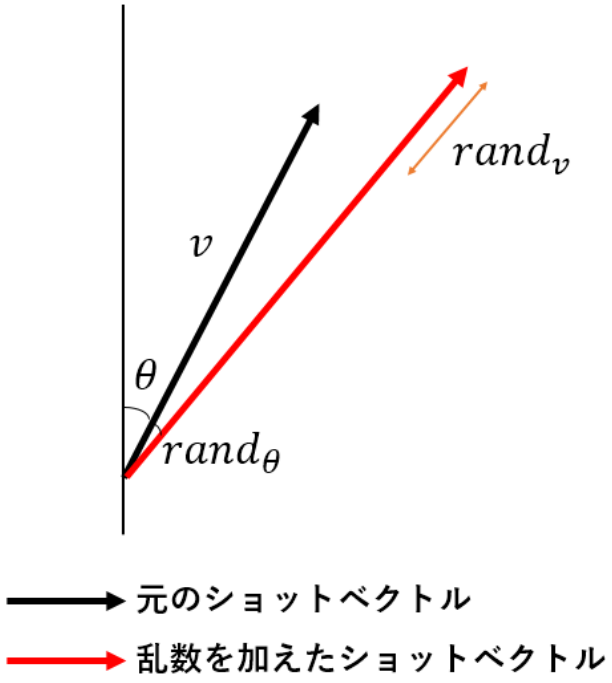


図 4-2 改良システムの乱数モデル

4.2.4 ライブラリのインターフェース

従来シミュレータの問題点として，パラメータがコンパイル時に定数として定義されており，ルールやレギュレーションに応じた変更が困難である点が挙げられる．また，クライアントプログラムはこのシミュレータを利用するものが多く存在するが，シミュレータで利用される定数（ストーンの大きさやリンクの各部の座標）が定義されておらずマジックナンバーが多用されている点や，シミュレーション関数やシミュレーションに必要な変数がグローバル関数・グローバル変数として定義されている点などライブラリとしての可用性の低さも指摘されている．

そこで，改良シミュレータでは，これらライブラリのインターフェースの整備を行う．シミュレーションを行うクラス b2simulator::Simulator を名前空間 DigitalCurling に定義した（以降のクラス・変数はすべて名前空間 DigitalCurling 下に定義されているものとする）．以下にシミュレータのヘッダファイル dcurling_simulator.h にて宣言・定義されるクラスについて解説する．なお，ヘッダファイルのコードについては巻末に付録として添付した．

シミュレータクラス

シミュレータのクラス図を図 4-3 に示す．

- int num_freeguard_
- StoneArea area_freeguard_

  それぞれ，フリーガードゾーンルールを適用する投数とエリアを表す．
- int random_type

  関数 AddRandom2Vec にて用いる乱数生成器の種類を表す．
- float friction_
- float stone_friction

  それぞれ氷とストーンの摩擦係数，およびストーン同士の摩擦係数を表し，プライベートなメンバ変数として定義される．これらの摩擦係数はストーンの描く軌跡に影響を与えることから，後述するショット生成関数で用いるストーンの軌跡テーブルの初期化処理以降は変更できない．ストーンの軌跡を格納するテーブルはメンバ変数 shot_table_として定義される．

- 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);

  ショットのシミュレーションを行う関数である.

  - GameState* const gamestate
    現時点でのゲーム情報を格納する変数. ショット後の情報がこの変数に上書きされる.
  - ShotVec vec
    実行するショットの初速度ベクトル.
  - Float random_x, float random_y
    初速度ベクトルの各成分にかかる乱数の大きさ
  - ShotVec* const run_shot
    実際に実行された（乱数が加えられた）ショット初速度ベクトル
  - Float *trajectory
  - Size_t traj_size
    各ストーンの軌跡を格納する配列とそのサイズ. 必要であれば 16*2*(ステップ数)*sizeof(float)分の領域を確保して渡すことで, 実際に実行されたシミュレーションの, 16 個のストーンの軌跡が格納される. 必要なければ, trajectory にヌルポインタを渡す.

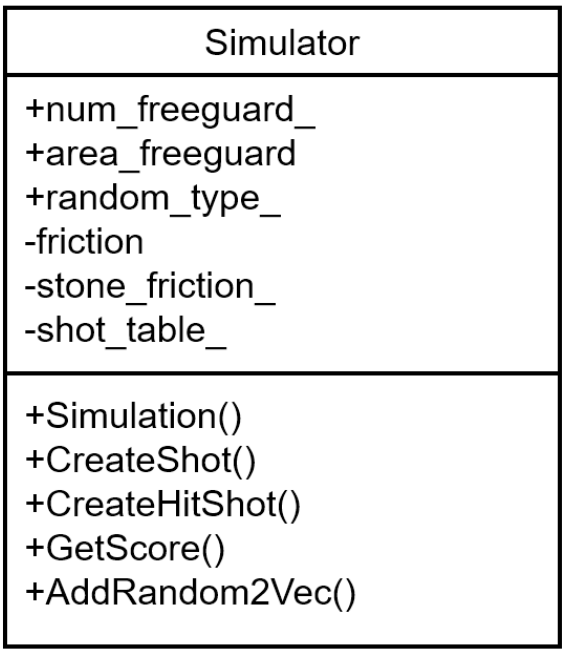


図 4-3 シミュレータクラス

盤面情報・ショット情報

盤面情報を表すクラス GameState を図 4-4 に示す．これらのクラスは，従来システムにおいて用いられていた構造体と同等のメンバ変数を持つ．GameState クラスのメンバ変数として，投数を表す ShotNum，現在のエンド数を表す CurEnd，最終エンドを表す LastEnd，各エンドでの得点を表す配列 Score，現在の手番を表す WhiteToMove，16 個のストーンの座標を表す二次元 body が定義されている．ストーン座標 body が持つ座標をすべて消去する関数 Clear()，エンド数や得点も含めて初期化する関数 ClearAll()，任意の座標にストーンを追加する関数 Set()がメンバ関数として定義されている．

また，ストーンの座標を表すクラス ShotPos，ショットの初速度ベクトルを表すクラス ShotVec を図 4-5 に示す．それぞれ直交座標系で表した座標ないしベクトルの(x, y)成分と，カールの方向を示す angle がメンバ変数として定義されている．

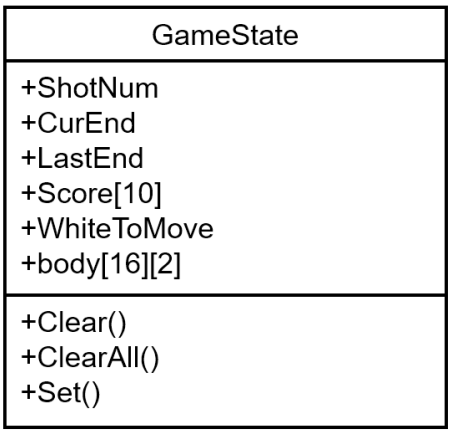


図 4-4 ゲーム情報クラス

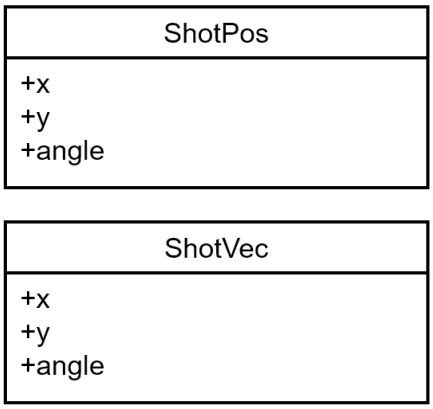


図 4-5 ショットベクトルおよび座標クラス

4.3 ゲームサーバ

4.3.1 概要

改良システムにおけるゲームサーバ（「改良サーバ」とする）は，各試合の進行を制御するクラス（便宜上「試合進行クラス」と呼ぶ）を持つ．試合進行クラスは，カーリングのゲームの表現に必要なパラメータを全て一括で保持する．また，試合の進行に必要な処理をまとめた関数群が定義されており，クライアントとの通信やシミュレーションの実行はこれらの関数内で実行される．試合進行制御部と呼ばれる関数がこれらの試合進行関数を順次呼び出していくことで，試合の進行を行う．

従来システムではゲーム進行を行うプログラムと GUI が統合されていたのに対し，改良サーバはメンテナンスの容易さを考慮し CUI として実装する．試合の実行や実行する試合の指定はコマンドライン上から行う．

試合の実行時に試合進行制御部は，JSON 形式で表現された設定ファイルから，試合設定やクライアント情報，物理シミュレータの設定など各種パラメータを読み込む．読み込んだパラメータを用いて試合進行クラスの初期化を行う．規定試合数が終了するまで，試合進行関数を順次呼び出すことで，試合が実行される．実際の呼び出し順序については，付録にゲームサーバ（試合進行制御部）のソースコード例という形で掲載した．

4.3.2 試合進行クラス

試合進行クラス（クラス GameProcess）は，各試合におけるカーリングのゲームを表現したクラスである．試合情報（ルールの種類やエンド数）やシミュレータ（4.2 節のクラス Simulator）・各クライアント（クライアント情報クラス Client）の情報といった，カーリングのゲームの表現に必要なパラメータを全て一括で保持する．

試合進行クラスは試合開始前に，試合毎のパラメータを設定ファイルから読み込む．読み込んだパラメータに基づいてクライアントやシミュレータの初期化を行い，試合の開始をクライアントに通知する．クライアントとの情報のやり取りは，クライアントクラス内で定義された送受信関数を用いる．

試合進行クラスのクラス図を図 4-6 に示す．試合進行クラスは試合情報やシミュレータ，クライアントの情報をメンバ変数として持つほか，試合の各ステップでの進行処理を行うメンバ関数を持つ．以下で各メンバ変数およびメンバ関数の解説を行う．

メンバ変数

- int rule_type_

  ルールの種類を表す変数である．現状では 0：通常ルール，1：ミックスダブルスルールを指定することが出来る．

- GameState_

現在の盤面状態を表す変数である．リンク上のストーンの座標や現在の投数・エンド数，最終エンド数，各エンドでの得点といった情報を含む．シミュレーションを実行する際にはこの変数を用いる．

- ShotVec best_shot_

  最後に BESTSHOT コマンドで受け取ったショットの初速度ベクトル．

- ShotVec run_shot_

  最後にシミュレーション実行したショットの初速度ベクトル．best_shot に乱数を加えたベクトルとなる．

- Simulator_

  シミュレータクラス．

- Client client1_

- Client client2_

  各クライアントの情報を表すクラス．4.3.3 節にて解説する．

- LogFile log_file_

  ログファイル情報を表すクラス．ログファイルのパスや読み書きを行う関数を持つ．

- Int prepetition

  試合の繰り返し回数を表す変数．1 以上の整数を指定する．

メンバ関数（試合進行関数）

- IsReady()

  クライアントの初期化処理を行う関数である．この関数では，指定されたファイルのパスからプログラムの起動を行い，ISREADY コマンドを用いてクライアントが準備完了かどうかの確認を行う．

- NewGame()

  各試合の開始処理を行う関数である．この関数で NEWGAME コマンドおよびゲーム情報を通知する GAMEINFO・PLAYERINFO コマンドの送信を行う．通常ルールでは，クライアントがショットを投げる順番を指定する SETORDER コマンドもこの関数内で処理を行う．

- Prepareend()

  各エンドの開始処理を行う関数である．この関数ではミックスダブルスルールにおけるストーンの初期配置を行う PUTSTONE コマンドの処理を行う．まだ，ミックスダブルスではショットを投げる順番をエンド毎に変更可能であるため，SETORDER コマンドの処理をこの関数内で行う．

- SendScore()

  各エンドでの得点計算と得点の送信を行う関数である．SCORE コマンドの送信を行う．

- Go()

各手番でのクライアントの思考開始と思考時間の制御を行う関数である．現在の手番情報を通知するSETSTATE コマンドおよび試行開始の通知を行う GO コマンドの処理を行う．制限時間内に BESTSHOT もしくは CONCEDE コマンドが受信できない場合は時間切れとして終了処理を行う．

- RunSimulation()

  シミュレーションの実行を行う関数である．メンバ変数 best_shot_に格納されたショットの初速度ベクトルを，シミュレータクラスに定義された関数を用いてシミュレーションを実行する．実際に実行されたショットの初速度ベクトルをメンバ変数 run_shot_に格納する．

- SendScore()

  SCORE コマンドを用いて得点の送信を行う関数である．

- Exit()

  試合の終了処理を行うコマンドである．この関数内では GAMEOVER コマンドの送信を行い，クライアントのプロセス終了処理を行う．

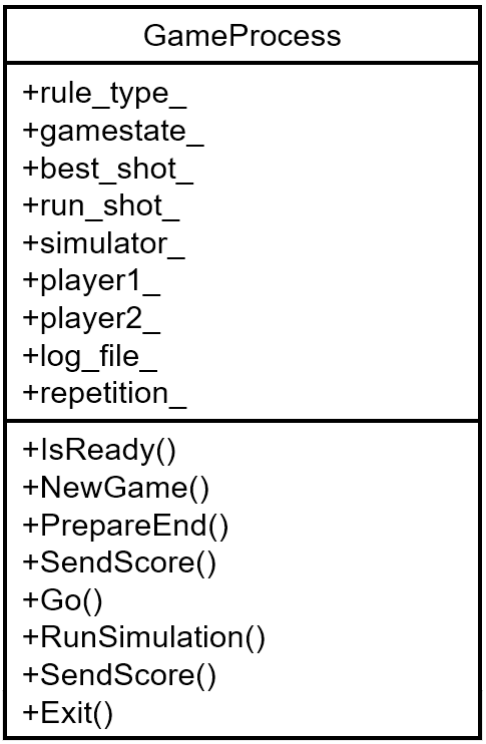


図 4-6 ゲーム進行クラス

4.3.3 プレイヤ毎の技量差の表現

3.5.2 節で述べたように，より現実に近いカーリングの試合を再現するために，改良システムでは，従来システムでは存在しなかったプレイヤ毎の技量差を導入する．プレイヤ毎の技量差としては様々な要素が挙げられるが，改良システムではショットを投げる精度と投げられるショット強さをプレイヤの技量差として用いる．

具体的には，初速度ベクトルの各成分にかかる乱数の大きさと，実行可能なショットの初速度ベクトルの最大値を，1 チーム最大で 4 人のプレイヤがそれぞれ異なる値を持つものとして設定ファイルに指定することが出来る（4.3.2 節参照）．このパラメータは設定ファイルにおいて配列で記述されており，従来どおりすべてのプレイヤが同一のパラメータを持つようにするには，1 人分のパラメータのみを記述すればよい．

試合進行システム内では，プレイヤ毎の技量差はクライアント情報クラス”Player”（図 4-7）内で表現される．クライアント情報クラスはクライアントの名前や持ち時間・残り持ち時間に加え，PlayerInfo クラス型のメンバ変数として，最大 4 人分の乱数の大きさとベクトルの最大値を保持することができる．

チーム内でストーンを投げる順番は，通常ルールで試合毎に，ミックスダブルスルールではエンド毎に変更することが可能である．プレイヤが投げる順番を変更できるように，改良システムでは専用のコマンドの追加を行った．

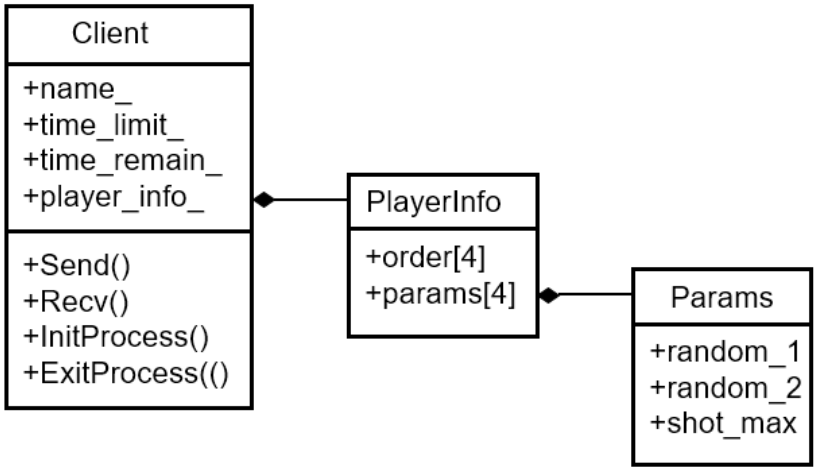


図 4-7 クライアント情報クラス

4.3.4 設定ファイル

シミュレータとサーバ，および各試合における設定値は，すべてひとつの設定ファイルに記述される．設定ファイルは JSON 形式で表現される．実際の設定ファイルの内容を巻末に付録として添付した．現在のバージョンで指定可能なパラメータを以下に示す.

サーバ

- “timeout_isready”

  サーバが ISREADY コマンドを送信後，カーリング AI の READYOK コマンドを待つ時間
- “timeout_preend”

  サーバが PUTSTONE コマンドを送信後，カーリング AI の PUTSTONE コマンドを待つ時間
- “output_dcl”

  試合のログを出力するかどうか

シミュレータ

- “friction”

  氷とストーンの摩擦係数
- “stone_friction”

  ストーン同士の摩擦係数
- “rand_type”

  乱数生成器の種類（"RECTANGULAR”：直交座標系，”POLAR”：極座標系）
- “freeguard_num”

  フリーガードゾーンルールを適用する投数

試合条件

- “rule_type”

  ルールの種類（"normal”：通常ルール，”mix_doubles”：ミックスダブルス）
- “ends”

  エンド数（1-10 の整数）
- “Client_1” “Client_2”

  各クライアントに関する情報
  - “path”: 実行ファイルのパス

- “time_limit”

  クライアントの持ち時間（ミリ秒）

- “params”

  各プレイヤのパラメータ（4.3.3 節参照）．配列となっており最大で 4 人分指定可能.

  - “random_1”

    乱数の大きさ 1（直交座標系：x，極座標系：v）

  - “random_2”

    乱数の大きさ 2（直交座標系：y，極座標系：θ）

  - “weight_max”

    ショットの初速度ベクトル最大値

- “repetition”

  試合の繰り返し回数（1 以上の整数）

4.3.5 ミックスダブルスルールへの対応

ミックスダブルダブルスはカーリングにおける変則ルールのひとつで，平昌オリンピックより正式種目として採用された．通常のカーリングと異なり，1 チームは男性 1 人，女性 1 人の 2 人から構成される．1 エンドあたりの投数は各チーム 5 投・合計 10 投であり，フリーガードゾーンルールを適用する投数が 3 投となる．

また，各エンドの最初にあらかじめストーンの配置を決めることができ，この配置がエンドの戦略に大きく影響を与える．ストーンの配置パターンは，ハウスの中央に 1 つ・センターガード 1 つ（図 4.3.4 左：センターガード）およびハウスのサイドに 1 つ，サイドガード 1 つ（図 4.3.4 右：パワープレイ）の 2 つ，またどちらのチームのストーンを手前にするかを加えて合計 4 種類が存在する．ストーンの配置は前のエンドを失った（得点できなかったチーム）が決定し，各チームパワープレイは 1 試合に 1 度しか指定できない．

改良システムでミックスダブルスルールの試合を実行するために，各エンドのはじめにストーンの配置を行うコマンドを用意した．投数（GameState.ShotNum）は 6 から始まり，15 が最終投とすることで，1 エンド 10 投のゲームを再現している．

現実のカーリングにおいてストーンが配置される具体的な位置は，各試合で氷のコンディションなどを考慮して決定される．位置の決定はアイスメーカーと呼ばれるスタッフの裁量によって変わることから，改良システムにおける配置座標は，カーリングのルールブック[12]に記載されている典型的な値を用いることとする（図 4-8）．

このようにミックスダブルスは通常ルールとは大きく異なるため，どちらのルールにて試合が行われるかは，ゲーム情報の通知を行う「GAMEINFO」コマンドによって各カーリング AI へ通知される．

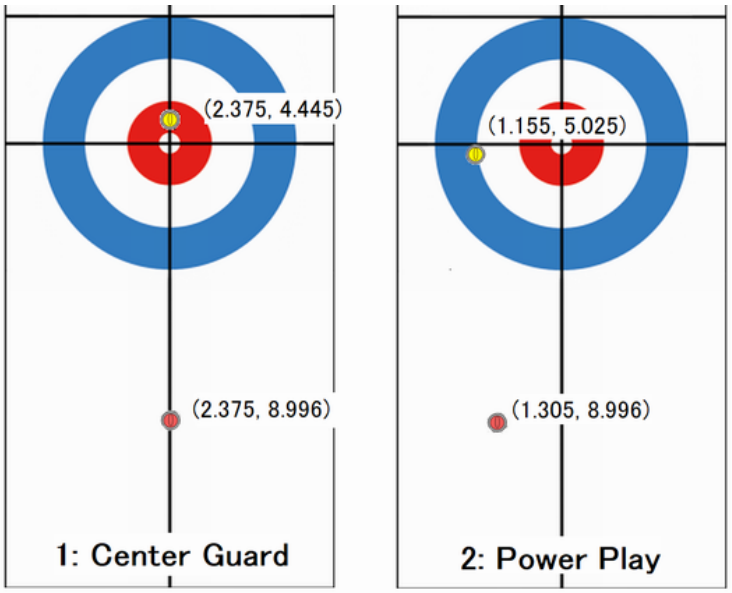


図 4-8 ミックスダブルスにおけるストーンの初期配置

4.4 通信プロトコル

従来システムの問題点を踏まえ, 改良システムでは従来のプロトコルを拡張した Digital Curling Protocol（以下，DCP）の定義を行なった．従来のプロトコルと異なる点として，各種情報（試合設定・プレイヤのパラメータ）を通知するコマンド・プレイヤの投げる順番を指定するコマンド・ストーンの配置を行うコマンド・コンシードを行うコマンドが追加されている点が挙げられる．

以下に DCP にて追加・変更されたコマンドと引数を記載する．

- GAMEINFO *rule random num_players*（ゲームサーバ→クライアント）

  試合情報を通知するコマンド

  - “rule”：ルールの種類（0：通常，1：ミックスダブルス）
  - “random”：乱数生成器の種類（0：直交座標系，1：極座標系）
  - “num_players”：1 チームあたりのプレイヤの数

- PLAYERINFO *rand1_0 rand2_0 shotmax_0 … rando1_3 rand2_3 shotmax_3*（ゲームサーバ→クライアント）

  プレイヤのパラメータを通知するコマンド

  - “rand1_n”：n 番目のプレイヤの乱数の大きさ 1
  - “rand2_n”：n 番目のプレイヤの乱数の大きさ 2
  - “shotmax_n”：n 番目のプレイヤのショットの上限値

- SETORDER（ゲームサーバ→クライアント）
- SETORDER *p1 … p3*（クライアント→ゲームサーバ）

  プレイヤの投げる順番を指定するコマンド

  - “pm”：m 番目に投げるプレイヤの番号（PLAYERINFO で通知された順番）

- PUTSTONE（ゲームサーバ→クライアント）
- PUTSTONE *type*（クライアント→ゲームサーバ）

  ストーンの配置を指定するコマンド

  - “type”：ストーンの配置

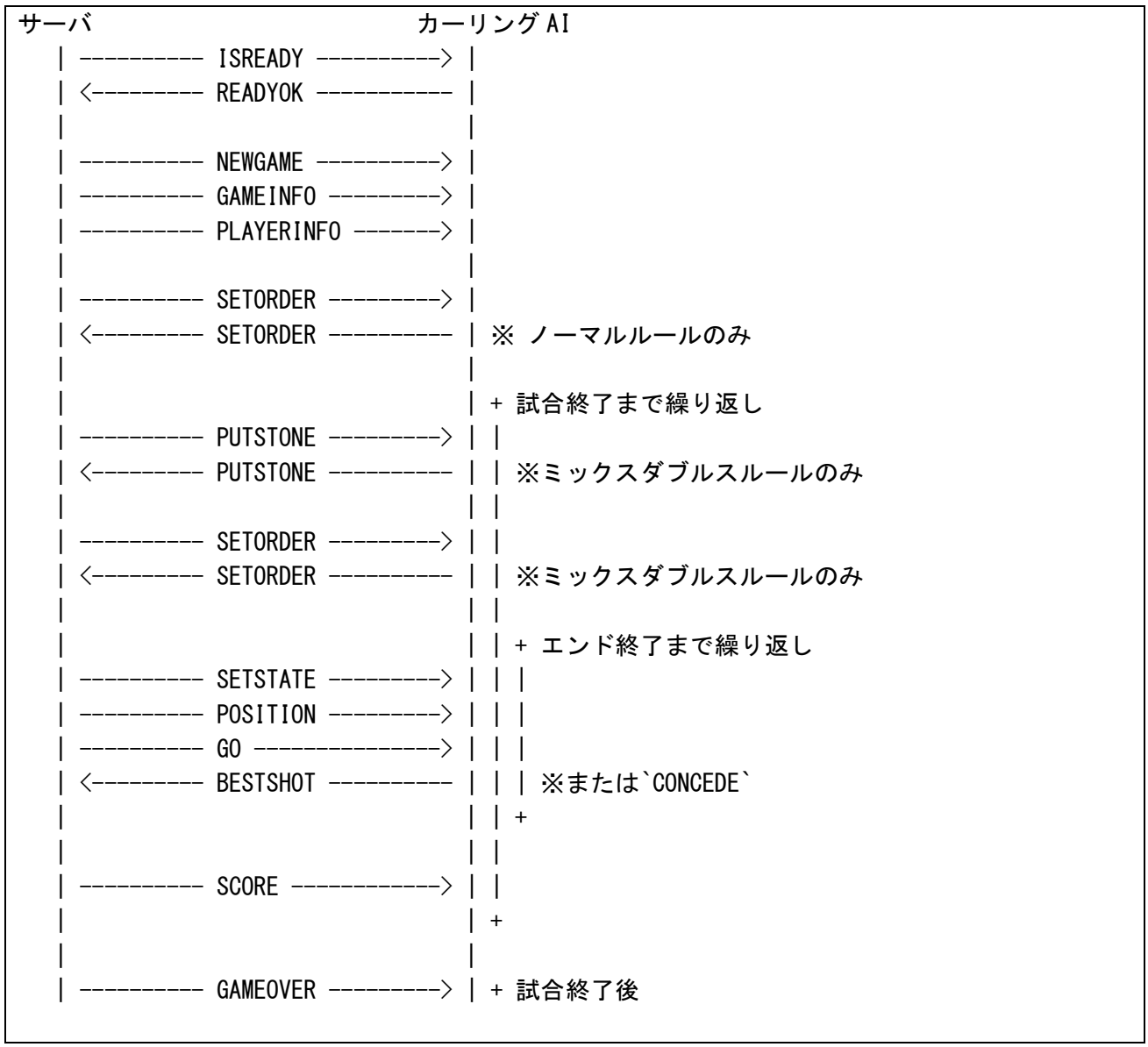
    （0：センターガード，配置プレイヤが後手）

    （1：センターガード，配置プレイヤが先手）

    （2：パワープレイ，配置プレイヤが後手）

（3：パワープレイ，配置プレイヤが先手）

以下にコマンドのタイムラインを記載する．



4.5 改良システムの動作例

4.5.1 通常ルールにおける動作例

本章では改良システムの動作例を示す．改良システムは CUI のみでの動作となるが，プレイエリア内のストーン配置を示す簡易的な図を表示させている. 先手番のストーンは〇･後手番のストーンは×で表示され，最後に投げられたストーンはそれぞれ◎・※で表示される

本節ではルールの種類は通常ルール・エンド数 8 とし，公式でサンプルプログラムとして公開されている SampleAI 同士の対戦を行う. 乱数の設定は付録 : 設定ファイル例における”matches_default”の設定を用いた.

図 4-9 に試合開始時点での画面を示す．サーバプログラムを起動し，「run」と入力することで，設定ファイルから試合設定を読み込み，初期化処理を行う．各手番で現在の投数，エンド数，および手番情報と持ち時間が表示（図 4-8 の 8 行目）され，その下にストーンの配置図が表示されている．また，その下には得点表が表示されているが，1 エンドも終了していないため空欄となっている．また，現在は 1 投目を投げる前であるため，プレイエリア内にストーンは存在しない.

図 4-10 に 1 投目の BESTSHOT 受信後にシミュレーションを行った画面を示す．SampleAI はハウス内にストーンが存在しない場合，ハウスの中央めがけてショットを投げるが，乱数により手前にずれたため，ハウスの中央手前にストーンが位置している.

図 4-11 に 1 エンド終了後の画面を示す．1 エンド終了時点で盤面はリセットされ，後手番の得点 1（および先手番の得点 0）が得点表に表示されている.

図 4-12 に試合終了後の画面を示す．得点表に 8 エンドそれぞれの得点が表示され，その右側には合計得点が表示されている．合計得点は 3-4 となり，後手番の勝利として終了処理が行われている.

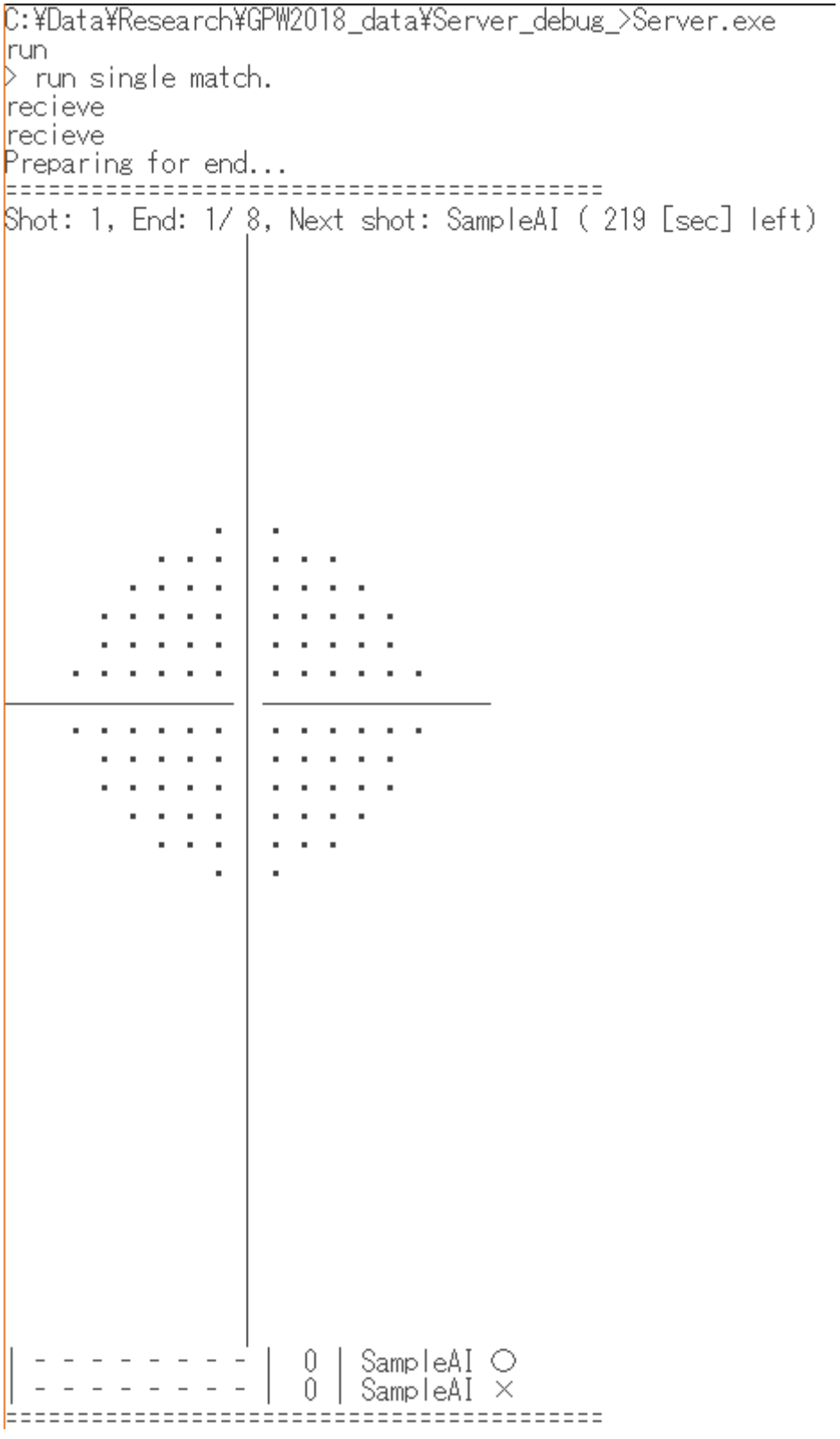


図 4-9 第 1 エンド・1 投目前の画面

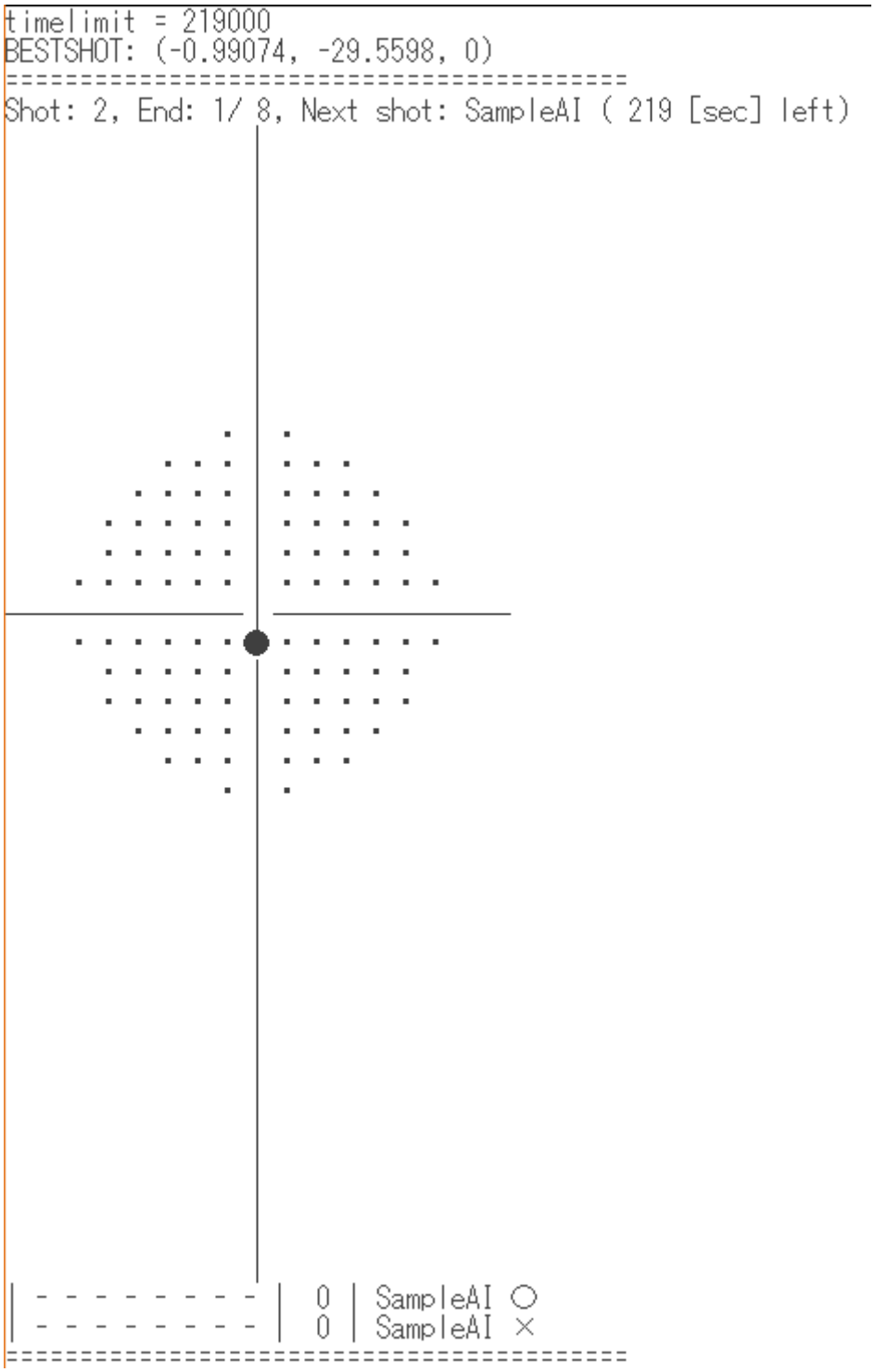


図 4-10 第 1 エンド 1 投目投球後の画面

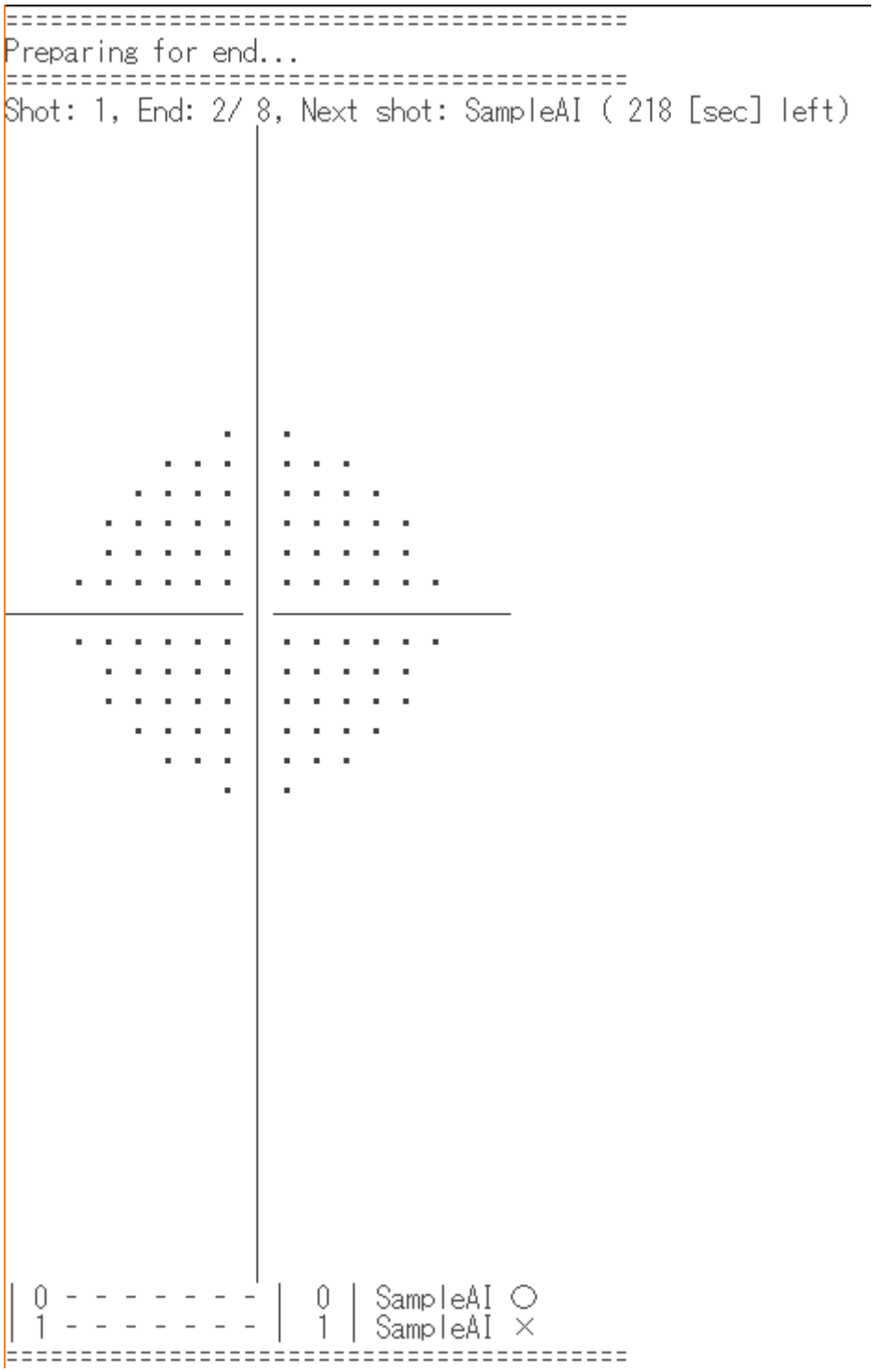


図 4-11 第 1 エンド終了後の画面

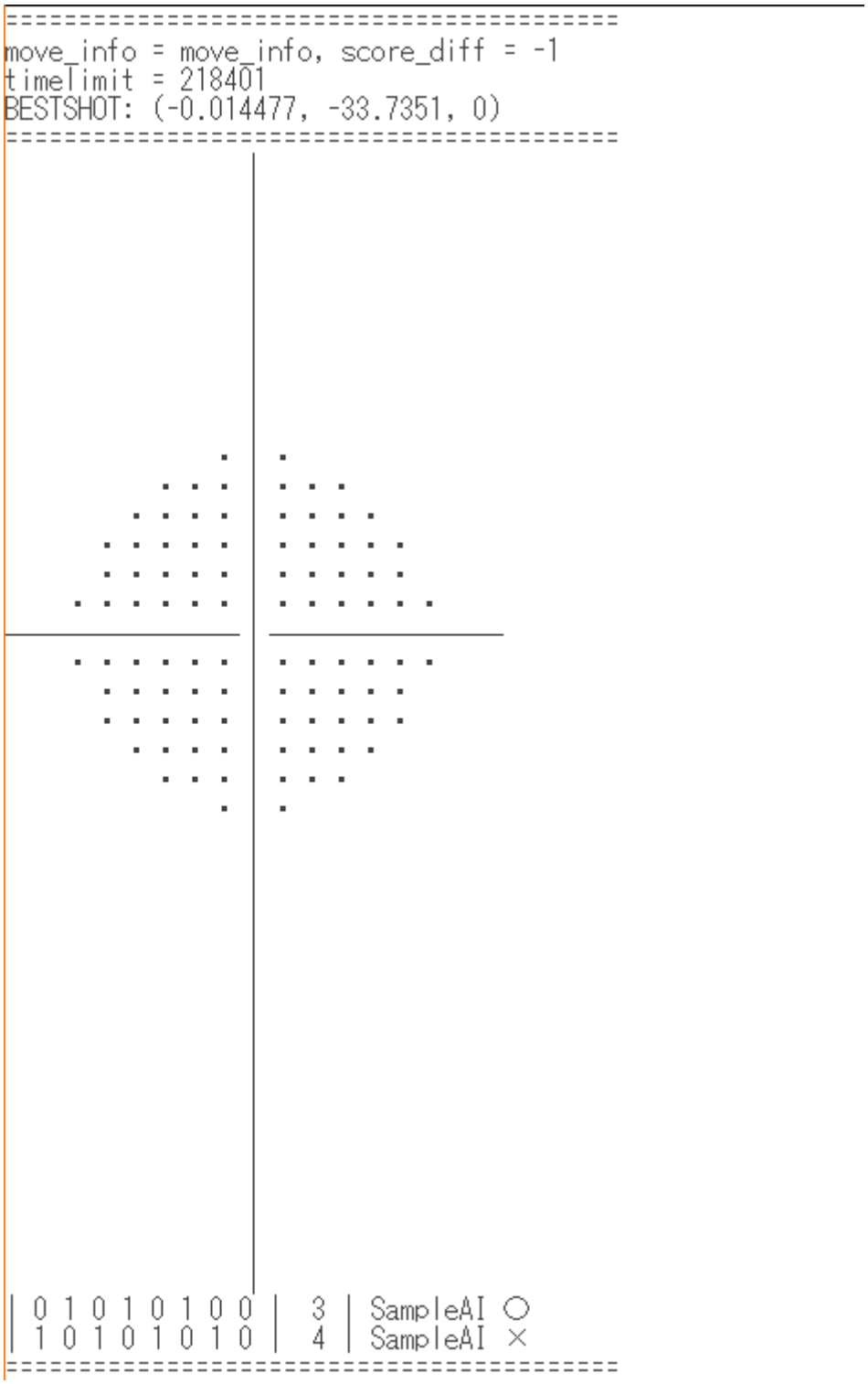


図 4-12 第 8（最終）エンド終了後の画面

4.5.2 ミックスダブルスルールにおける動作例

本節ではミックダブルスルールにおける動作例を示す．エンド数 4 とし，GPW 杯 2018 デジタルカーリング大会ミックスダブルス部門優勝プログラムである TAI_2sigma 同士の対戦を行った．

図 4-13 に第 1 エンド・1 投目開始前の画面を示す．通常ルールとは異なり，ストーンの初期配置が行われるため，この時点でプレイエリア内にストーンが存在している．また，1 エンド 10 投のゲームであるため，この時点でショット数は 7 となっている．

図 4-14 に第 1 エンド・10 投目開始前の画面を示す．実際の投数は 10 であるが，システム内部では 16 投目となるため，投数が 16 と表示されている．

エンド終了後の得点表示等は通常ルールと変わらないため，表示画面は割愛する．このように，通常ルールに加え，ミックスダブルスルールでのプレイが可能となっていることがわかる．

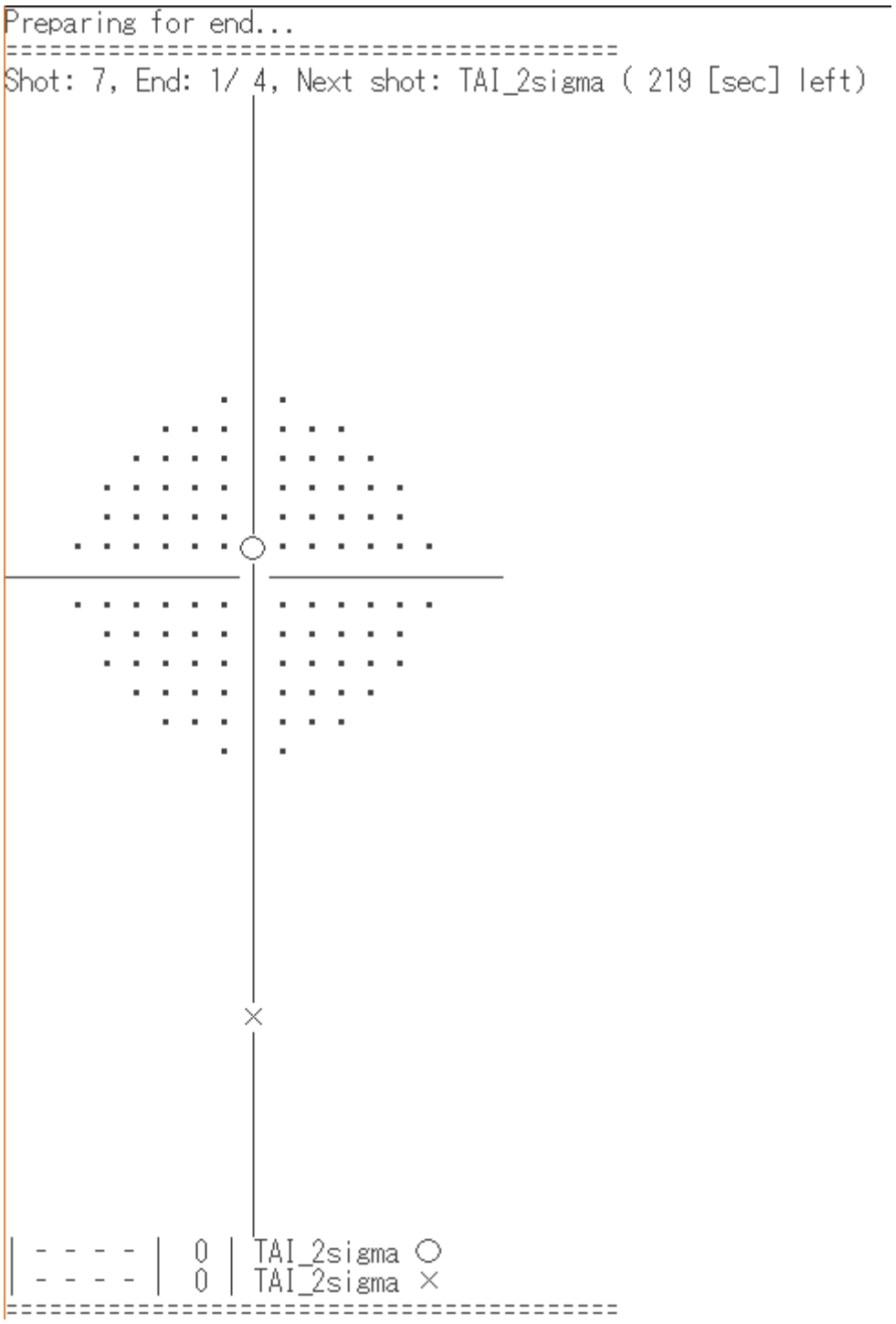


図 4-13 第 1 エンド 1 投目前の画面（ミックスダブルス）

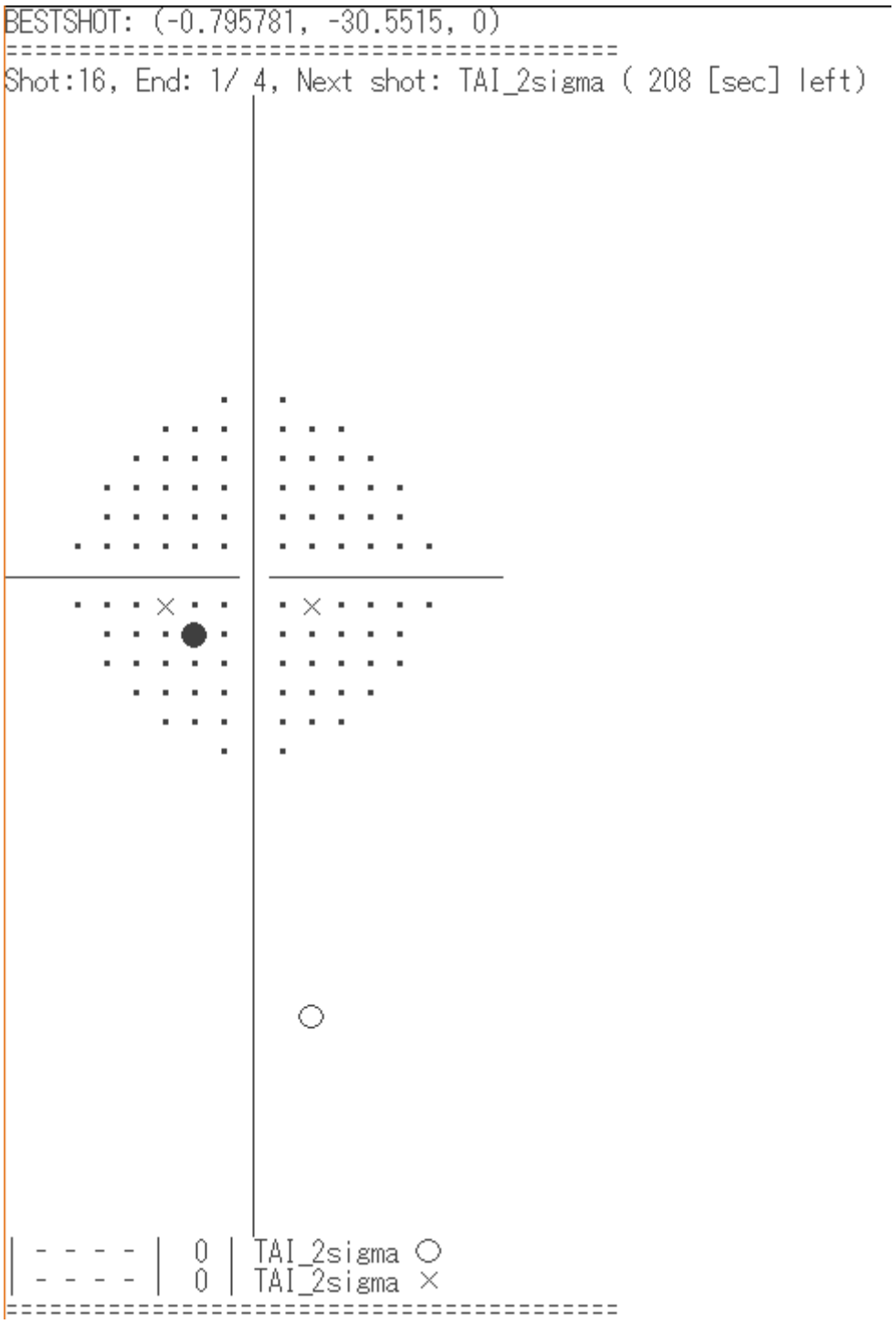


図 4-14 1 エンド目 10 投目投球前の画面（ミックスダブルス）

5. 改良システムを用いた大会の実施

5.1 概要

改良システムを用いたデジタルカーリング大会を，GPW 杯 2018 デジタルカーリング大会として，第 23 回ゲームプログラミングワークショップにて開催した．大会は従来システムで行われたものと同形式の部門を通常部門とし，改良システムによる条件変更の一例として新にミックスダブルス部門を設け，2 部門それぞれについて優勝を決定した．

参加チームは 4 チーム，うち 3 チームは本学からのエントリーであった．

5.2 レギュレーション

- トーナメント形式：総当たり戦

通常部門

- エンド数：8
- 投数：16
- 制限時間：219 秒
- 乱数の大きさ：(0.0725,0.29000)

ミックスダブルス部門

- エンド数：8
- 投数：10
- 制限時間：219 秒
- 乱数の大きさ：(0.0725,0.29000)

5.3 大会結果

大会は大きなトラブルなく行われ，改良システムを用いてデジタルカーリングの競技を実施できることが実証された．大会の結果を以下に示す．

通常部門

-	MogMog	MontCurlingAI	SampleAI	TAI_2sigma	勝	敗
MogMog	-	○○	○○	×○	2	0
MontCurlingAI	××	-	××	××	0	3
SampleAI	××	○○	-	○×	1	1
TAI_2sigma	○×	○○	×○	-	1	0

ミックスダブルス部門

-	MogMog	MontCurlingAI	SampleAI	TAI_2sigma	勝	敗
MogMog	-	○○	○○	××	2	1
MontCurlingAI	××	-	××	××	0	3
SampleAI	××	○○	-	××	1	2
TAI_2sigma	○○	○○	○○	-	3	0

先手	後手	得点（通常）		得点（ミックスダブルス）	
MogMog	SampleAI	11	4	16	0
MontCurlingAI	SampleAI	5	11	5	8
TAI_2sigma	SampleAI	3	4	6	4
SampleAI	MogMog	2	11	1	17
MontCurlingAI	MogMog	4	11	5	14
TAI_2sigma	MogMog	7	8	11	9
MogMog	MontCurlingAI	11	1	13	2
SampleAI	MontCurlingAI	6	4	8	4
TAI_2sigma	MontCurlingAI	11	3	15	1
MogMog	TAI_2sigma	5	8	4	14
SampleAI	TAI_2sigma	3	4	3	12
MontCurlingAI	TAI_2sigma	4	12	3	16

6. 結論

本研究では，従来のデジタルカーリングシステムの問題点を踏まえ，ルールや物理シミュレーションのパラメータの変更に対して柔軟に対応できる改良システムを新たに構築した．従来システムでは散逸していたうえ変更できなかったパラメータをひとまとめにし，ゲーム進行に必要な関数群も合わせたゲーム進行システムを定義した．これにより，ルールだけでなく物理シミュレーションに必要なパラメータ等の変更もゲームサーバから行えるようになった．また従来システムで表現されていなかったプレイヤ毎の技量差を，乱数の大きさとショットベクトルの最大値として表現した．

本システムを用いた GPW 杯 2018 大会を，通常ルールに加えカーリングの変則ルールであるミックスダブルスにて実施し，ルールの変更に対して柔軟な対応が可能であることを実証した．プレイヤ毎の技量差についてはレギュレーションの告知を行う時間が十分に取れなかったため導入を見送ったが，2019 年 3 月に GAT2018（Game AI Tournament 2018[13]）にて実施予定の第 5 回 UEC 杯デジタルカーリング大会では，ミックスダブルス部門において異なるパラメータを持つ 2 人のプレイヤを想定し導入を行う．

変則ルールや技量差の導入を行いより現実に近い複雑なゲームを実現することが可能となったことで，改良システムがカーリング AI 研究のより一層の発展に寄与することが期待できる．

また，改良システムを 2.3.2 節で述べたカーリングの戦略学習支援システムへ応用することも考えられる．この学習支援システムではストーン配置や得点の編集を行いカーリング AI に思考させることができるが，従来システムを踏襲しているため物理シミュレータのパラメータ変更やプレイヤの技量差といった要素には対応していない．ストーンの曲がりの強さやショットの精度（乱数の大きさ）・強さといった要素は，カーリング AI や実際のカーリングプレイヤの戦略の影響を与えうる要素である．

例として，図 6-1 に樋口らの学習支援システムにおいて，異なるショットの強さの上限値を持つ 2 つ AI が思考した場合のショット示す．この局面は第 3 エンドの 14 投目であり，図 6-1 左側に表示されているのが現在のストーン配置である．この局面において，異なるショットの強さを持つ 2 つのカーリング AI に思考させた結果が，図左の 2 つの線で示されている．従来大会と同等の標準的なショットの上限値をもつ AI の結果が赤，より強いショットの上限値を持つ AI の強さが緑で示される．赤の標準的なショットは相手のストーンに軽く当てるショットであり，相手のストーンをハウスからはじき出すことは出来ていない（図 6-2 左）．それに対して，より強く投げることのできる緑のショットでは，相手のストーンをハウス内からはじき出す強力なショットを選択している（図 6-2 右）．このように，ショットの強さのようなパラメータはカーリング AI の戦略に大きく影響を与えるといえる．

この例ではカーリング AI のソースコードを直接書き換えることで，ショットの強さの変更に対応している．本研究で構築した改良システムを用いることで，ソースコードの改変を伴わずに，このようなパラメータ変更に対応することが出来る．実際に学習支援に用いる際は，ユーザ（プレイヤ）の技量に応じて様々な

値を設定することになると考えられるため，今後の学習支援システムの発展に際して，改良システムの柔軟性は大変有用であると考えられる．

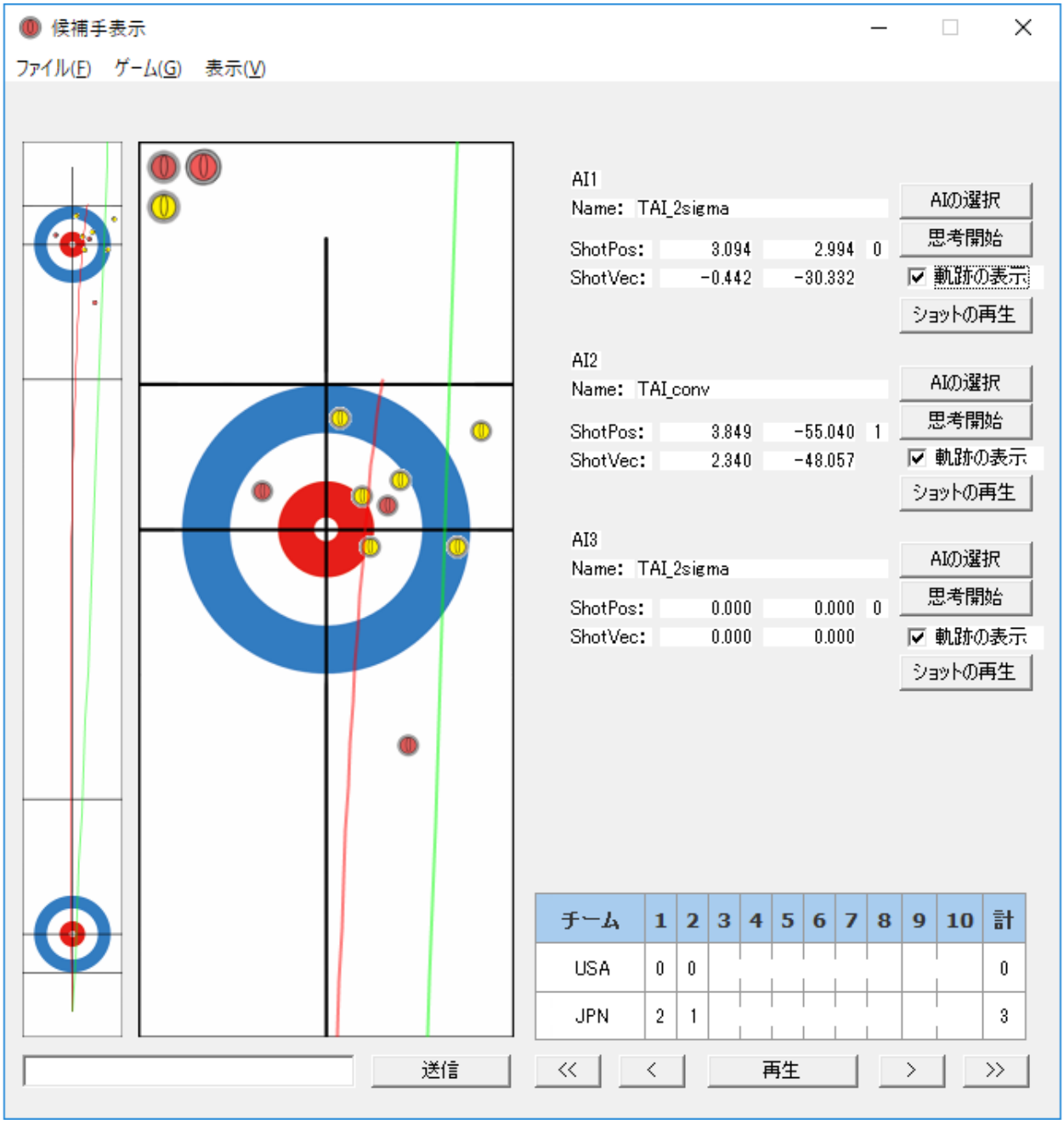


図 6-1　ショットの強さを変更した場合の着手の変化

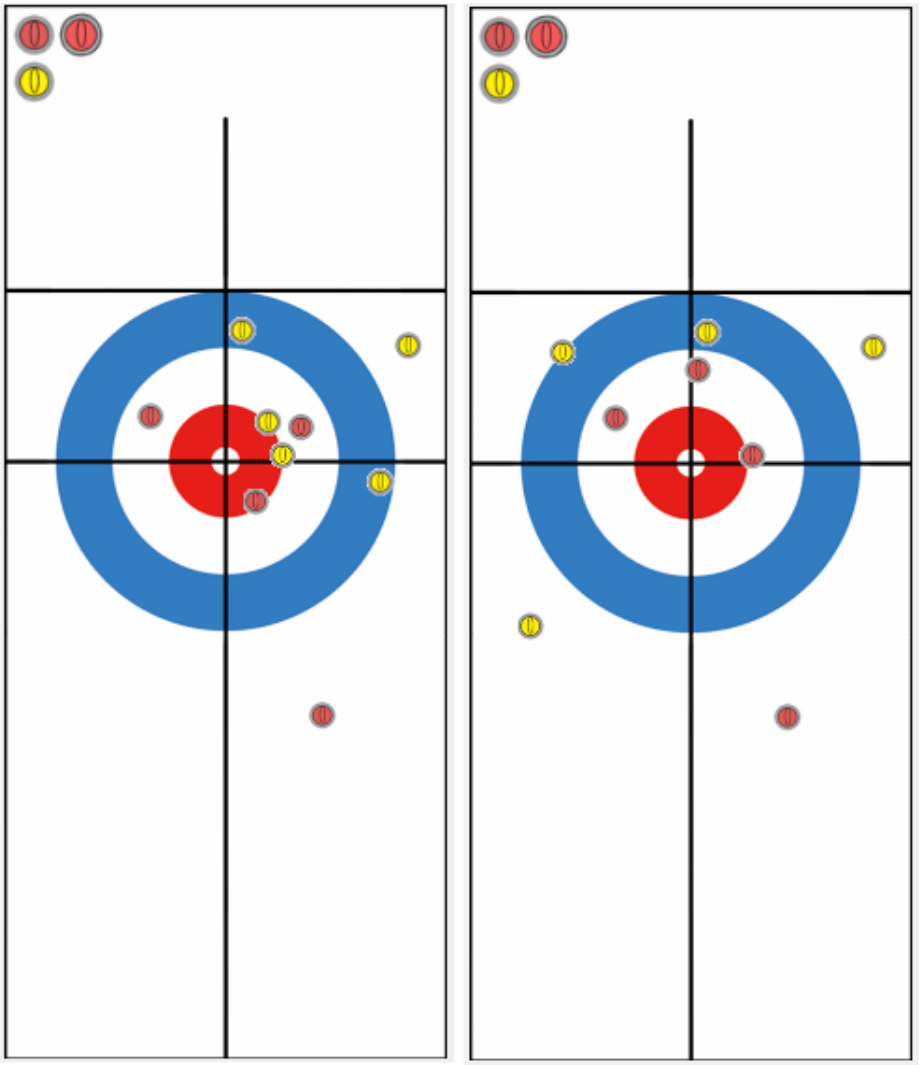


図 6-1 標準ショット（左）と強いショット（右）の実行結果

7. 今後の展望

本研究で構築した改良システムによって，従来システムの問題点であった「現実のカーリングを十分に表現できていない点」について改善を行った．しかしながら，「氷のコンディションの時間変化」や「スウィーピング」といった改良システムでも表現されていない要素は多く存在している．

これらの要素の導入を見送った理由として，カーリングにおけるストーンの物理運動は原理が解明されていない点が多いことが挙げられる．一般的に氷は時間とともに解けていくため摩擦係数が増大しストーンが滑りにくくなるほか，ストーンが通過した部分も氷が解けるため，氷の状態は局所的にも変化するといわれている．しかし具体的にストーンの運動にどのように影響を与えているかは分かっておらず，厳密に測定された軌跡のデータも存在しない．スウィーピングについても同様に，ストーンの運動へどの程度影響するか数値化するのは困難である．

実測データを用いてストーンの運動を数値化する試みが行われている．本論文の執筆時点では具体的な数値の計測には至っていないが，このデータを用いてデジタルカーリングにおけるストーンの運動モデルを新たに構築することで，より現実に近いゲームの再現を行えると考えられる．これにあわせて氷の時間変化やスウィーピングの影響についてもデータが得られれば，デジタルカーリング上でのモデル化・実装も可能であると考えられる．

具体的な実装方法については検討が必要である．もっとも単純な実装を考える場合，氷の状態変化を 1 投ごとの非連続な変化ととらえ，8 エンドであれば 8*16 投分の氷の摩擦係数を用意することが考えられる．すなわち，ゲーム進行システム（4.3.2 節）が保持するシミュレータクラスのインスタンスを 1 試合の合計投数分確保し，それぞれ異なる摩擦係数を用意することで，氷の状態変化を疑似的に再現することができると考えられる．

参考文献

[1]Takeshi Ito and, Yuuma Kitasei, Proposal and Implementation of “Digital Curling”, 2015 IEEE Conference on Computational Intelligence and Games (CIG2015), pp.469-473 (2015).

[2] 北清勇磨, 岡田雷太, 伊藤毅志, デジタルカーリングサーバーの提案と紹介, 情報処理学会ゲーム情報学研究会報告, GI-31(2) (2014).

[3] 伊藤毅志, 森健太郎, デジタルカーリング大会報告 2015 年度, エンタテインメントコンピューティング, 2016-EC-41, pp.1-6 (2016).

[4] 加藤修, 飯塚博行, 山本雅人, 不確定性を含むデジタルカーリングにおけるゲーム木探索, 情報処理学会論文誌, 57(11), pp.2354-2364 (2016)

[5] 大渡勝己, 田中哲郎, カーリング AI に対するモンテカルロ木探索の適用, ゲームプログラミングワークショップ 2016 論文集, 2016, pp.180-187 (2016)

[6] 秋山英久, ロボカップサッカーシミュレーションリーグ, 人工知能学会誌 24(3), pp.349-354 (2009).

[7] 田所諭, ロボカップレスキューロボットリーグ, 日本ロボット学会誌 vol.29 No.9, pp.983-986 (2009).

[8] Christopher Archibald, Alon Altman, Michael Greenspan, Yoav Shoham, Computational Pool: A New Challenge for Game Theory Pragatics, AI Magagine Vol 31 No 4: Winter 2010, pp.33-41 (2010).

[9] Toshiyuki Higuchi, Kentaro Mori, Takeshi Ito, An AI-Assisted Strategy Learning Support System Using Digital Curling, WCI2018 論文集, pp.21-25 (2018)

[10] デジタルカーリング, 入手先(http://minerva.cs.uec.ac.jp/curling/wiki.cgi) (参照 2019-1-21)

[11] Box2D, 入手先(https://box2d.org/) (参照 2019-1-21)

[12] World Curling Federation - Rules and Regulations, 入手先(http://www.worldcurling.org/rules-and-regulations-downloads) (参照 2019-1-25)

[13] GAT@UEC 入手先(http://minerva.cs.uec.ac.jp/gat_uec/wiki.cgi) (参照 2019-1-25)

謝辞

本論文の執筆にあたり，様々なご指導を頂いた主任指導教員の伊藤毅志先生に深謝の意を表する．本研究室の樋口利幸君には，学習支援システムとカーリング AI のデータ提供を頂いた．ここに感謝の意を表明する．GPW 杯 2018 デジタルカーリング大会に参加いただいた参加者の方々に感謝の意を表する．

本研究は JSPS 科研費 15H02797 の助成を受けたものである．

付録

カーリングのルール

カーリングは「リンク」と呼ばれる氷上の，約 40 メートル先にある「ハウス」めがけて，2 つのチームが交互に「ストーン」を投げる（滑らせる）競技である．通常各チーム 8 投ずつ・合計 16 投を投げ，16 投終了時点でのハウス内のストーン配置によって得点が計算される（これを 1「エンド」と呼ぶ）．これを 8 エンドないし 10 エンド繰り返し，合計得点の高いチームが勝利となる．

リンク内には「プレイエリア」と呼ばれるエリアが存在し，各投の終了後にこのエリア外に出たストーンはプレイから除外される．各エンドで最後に投げるチームが得点しやすいため，各エンド最初の 1 投は直前のエンドで得点したチームが投げる．

カーリングには「フリーガードゾーンルール」と呼ばれるルールが存在する．各エンドで規定投数が投げ終わるまで，相手チームのストーンを特定のエリアからプレイエリア外はじき出すことが禁止される．このルールに違反した場合，その投球は無効とされ，ストーンの配置は投げる前の状態に戻される．このルールが適用される投数とエリアはルールによって異なる．またこれまでに数回ルール改定が行われており，通常ルールでは適用投数が 3 投であったが，2018 年の改定により 4 投に変更されている．

改良シミュレータのコード

dcurling_simulator.h

```
// Constant values
  constexpr unsigned int kLastEndMax = 10;  // Maximum number of LastEnd

  constexpr float kCenterX     =  2.375f;  // X coord of Center Line
  constexpr float kTeeY        =  4.880f;  // Y coord of Tee Line
  constexpr float kSideX       =  4.750f;  // X coord of Side Line
  constexpr float kHogY        = 11.280f;  // Y coord of Hog Line
  constexpr float kHackY       = 41.280f;     // Y coord of Hack?
  constexpr float kRinkHeight  = 42.500f;  // Height of Rink
  constexpr float kStoneR      =  0.145f;  // Radius of Stone
  constexpr float kHouseR      =  1.830f;  // Radius of House (12 foot circle)
  constexpr float kHouse8FootR =  1.220f;  // Radius of House ( 8 foot circle)
  constexpr float kHouse4FootR =  0.610f;  // Radius of House ( 4 foot circle)

  // State of DigitalCurling game
  class DLLAPI GameState {
  public:
   GameState();
   GameState(unsigned int last_end);
   ~GameState();

   // Clear body and Set ShotNum = 0
```

```
 void Clear();

 // Set stone
 void Set(unsigned int num, float x, float y);

 //  Note: Same member variables as GAMESTATE in CurlingSimulator older ver2.x
 unsigned int ShotNum;    // Number of current Shot

 unsigned int CurEnd;     // Number of current End (0 to LastEnd - 1)
 unsigned int LastEnd;    // Number of last End    (up to kLastEndMax)

 int Score[kLastEndMax];  // Score of each End

 bool WhiteToMove;        // which have next shot

 float body[16][2];       // position of stones
};

// Position of Stone
class DLLAPI ShotPos {
public:
 ShotPos();
 ShotPos(float x, float y, bool angle);
 ~ShotPos();

 float x;
 float y;
 bool angle;
};

// Vector of Stone
class DLLAPI ShotVec {
public:
 ShotVec();
 ShotVec(float x, float y, bool angle);
 ~ShotVec();

 float x;
 float y;
 bool angle;
};

// Vector of Stone (Polar Coordinate System)
class DLLAPI ShotVecP {
public:
 ShotVecP();
 ShotVecP(float v, float theta, bool angle);
```

```
   ~ShotVecP();

   float v;
   float theta;
   bool angle;
  };

  // Simulator with Box2D 2.3.0 (http://box2d.org/)
  namespace b2simulator {

   // Area of stone
   typedef enum {
    OUT_OF_RINK = 0b0000,
    IN_RINK = 0b0001,
    IN_PLAYAREA = 0b0010,
    IN_FREEGUARD = 0b0100,
    IN_HOUSE = 0b1000
   } StoneArea;

   enum {
    RECTANGULAR,
    POLAR
   };

   class DLLAPI Simulator {
   public:
    Simulator();
    Simulator(float friction);

    // Simulation with Box2D, returns number of steps taken
    // - GameState* game_state    : Current state and updated state after simulation
    // - ShotVec* shot_vec        : Shot Vector
    // - float random_x, random_y : Size of random number (v, theta when random_type_ =
POLAR)
    // - ShotVec* run_shot        : Shot Vector which with random numbers (pass nullptr
if you don't need)
    // - float trajectory         : trajectory log (pass float[traj_size][2], or nullptr
if you don't need)
    // - float traj_size          : size of trajectory log (number of steps)
    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);

    // Create ShotVec from ShotPos which stone will stop at
    void CreateShot(ShotPos pos, ShotVec* const vec);
```

```
    // Create ShotVec from ShotPos which stone will pass through
    void CreateHitShot(ShotPos pos, float weight, ShotVec* const vec);

    // Add random number to ShotVec
    //  random_1 : x (rectangular), v (polar)
    //  random_2 : y (rectangular), theta (polar)
    void AddRandom2Vec(float random_1, float random_2, ShotVec* const vec);

    unsigned int num_freeguard_;   // Number of shots which freeguard rule is applied
    StoneArea area_freeguard_;     // Area of freeguard
    unsigned int random_type_;      // Type of random number generator (0: )

   private:
    int init_shot_table();

    float friction_;

    static const unsigned int kTableSize = 10000;//8192;
    struct ShotTable {
     unsigned int index_end;
     ShotPos pos[kTableSize];
     ShotVec vec[kTableSize];
    } shot_table_;
   };

   // Return score of second (which has last shot in this end)
   int GetScore(const GameState* const game_state);

   ShotVecP ConvertVec(ShotVec vec_rect);
   ShotVec ConvertVec(ShotVecP vec_polar);
  }

  // Operators
  DLLAPI ShotPos operator+(ShotPos pos_l, ShotPos pos_r);
  DLLAPI ShotPos operator-(ShotPos pos_l, ShotPos pos_rs);
  DLLAPI ShotPos operator+=(ShotPos &pos_l, ShotPos pos_r);
  DLLAPI ShotPos operator-=(ShotPos &pos_l, ShotPos pos_r);
  DLLAPI ShotVec operator+(ShotVec pos_l, ShotVec pos_r);
  DLLAPI ShotVec operator-(ShotVec pos_l, ShotVec pos_r);
  DLLAPI ShotVec operator+=(ShotVec &pos_l, ShotVec pos_r);
  DLLAPI ShotVec operator-=(ShotVec &pos_l, ShotVec pos_r);
  DLLAPI ShotVec operator*(ShotVec &vec_l, float ratio);
}
```

ゲームサーバ（試合進行制御部のコード例

```
// Open config file w/ json
std::string config_path = "config.json";
Options opt;

GameProcess *gp = Init(config_path, match_name, opt);  // 設定ファイルの読み込みを行う
GameProcess game_process = *gp;

// Send "ISREADY" to both players
if (!game_process.IsReady(game_process.client1_, opt.timeout_isready)) {
          cerr << "failed to recieve ISREADY from client 1" << endl;
          return 0;
}
if (!game_process.IsReady(game_process.client2_, opt.timeout_isready)) {
          cerr << "failed to recieve ISREADY from client 2" << endl;
          return 0;
}

// Run match(es)
for (unsigned int i = 0; i < game_process.repetition_; i++) {
          if (i > 0) {
                    // Create new log file
                    game_process.log_file_.Create(game_process.client1_, game_process.client2_);
          }

          // Send "NEWGAME" to clients
          game_process.NewGame();

          while (game_process.gs_.CurEnd < game_process.gs_.LastEnd) {

                    // Prepare for End
                    game_process.PrepareEnd(opt.timeout_preend);
                    while (game_process.gs_.ShotNum < 16) {
                            // Send "SETSTATE" and "POSITION" to clients
                              game_process.SendState();

                              // Send "GO" to client
                              int status = game_process.Go();
                              if (status != GameProcess::BESTSHOT) {
                                        break;
                              }

                    // Simulation
                    game_process.RunSimulation();
          }
```

```
                    // Send 'SCORE' to clients
                    game_process.SendScore();
          }

          // Exit Game
          game_process.Exit();
          cerr << "game_end" << endl;
}
```

設定ファイルの例

```
{
  "server": {
    "timeout_isready": 15000,
    "timeout_preend": 5000,
    "output_dcl": true
  },

  "simulator": {
    "friction": 12.009216,
    "friction_default": 12.009216,
    "stone_friction": 0.500,
    "rand_type": "RECTANGULAR",
    "rand_type_all": ["RECTANGULAR", "POLAR"],
    "freeguard_num": 5
  },

  "match_mix" : {
    "rule_type": "mix_doubles",
    "ends": 8,
    "player_1": {
      "type": "local",
      "path": "SampleAI.exe",
      "timelimit": 219000,
      "params": [
        {
          "random_1": 0.0725,
          "random_2": 0.2900,
          "weight_max": 50.000
        },
        {
          "random_1": 0.10875,
          "random_2": 0.4350,
          "weight_max": 75.000
        },
        null,
        null
      ]
    },
    "player_2": {
      "type": "local",
      "path": "SampleAI.exe",
      "timelimit": 219000,
      "params": [
        {
          "random_1": 0.0725,
```

```
        "random_2": 0.2900,
        "weight_max": 50.000
      },
      {
        "random_1": 0.10875,
        "random_2": 0.4350,
        "weight_max": 75.000
      },
      null,
      null
    ]
  },
  "extended_end": true,
  "repetition": 1
},

"match_default" : {
  "rule_type": "normal",
  "ends": 8,
  "player_1": {
    "type": "local",
    "path": "SampleAI.exe",
    "timelimit": 219000,
    "params": [
      {
        "random_1": 0.0725,
        "random_2": 0.2900,
        "weight_max": 50.000
      },
      null,
      null,
      null
    ]
  },
  "player_2": {
    "type": "local",
    "path": "SampleAI.exe",
    "timelimit": 219000,
    "params": [
      {
        "random_1": 0.0725,
        "random_2": 0.2900,
        "weight_max": 50.000
      },
      null,
      null,
      null
```

```
      ]
    },
    "extended_end": true,
    "repetition": 1
  }
}
```