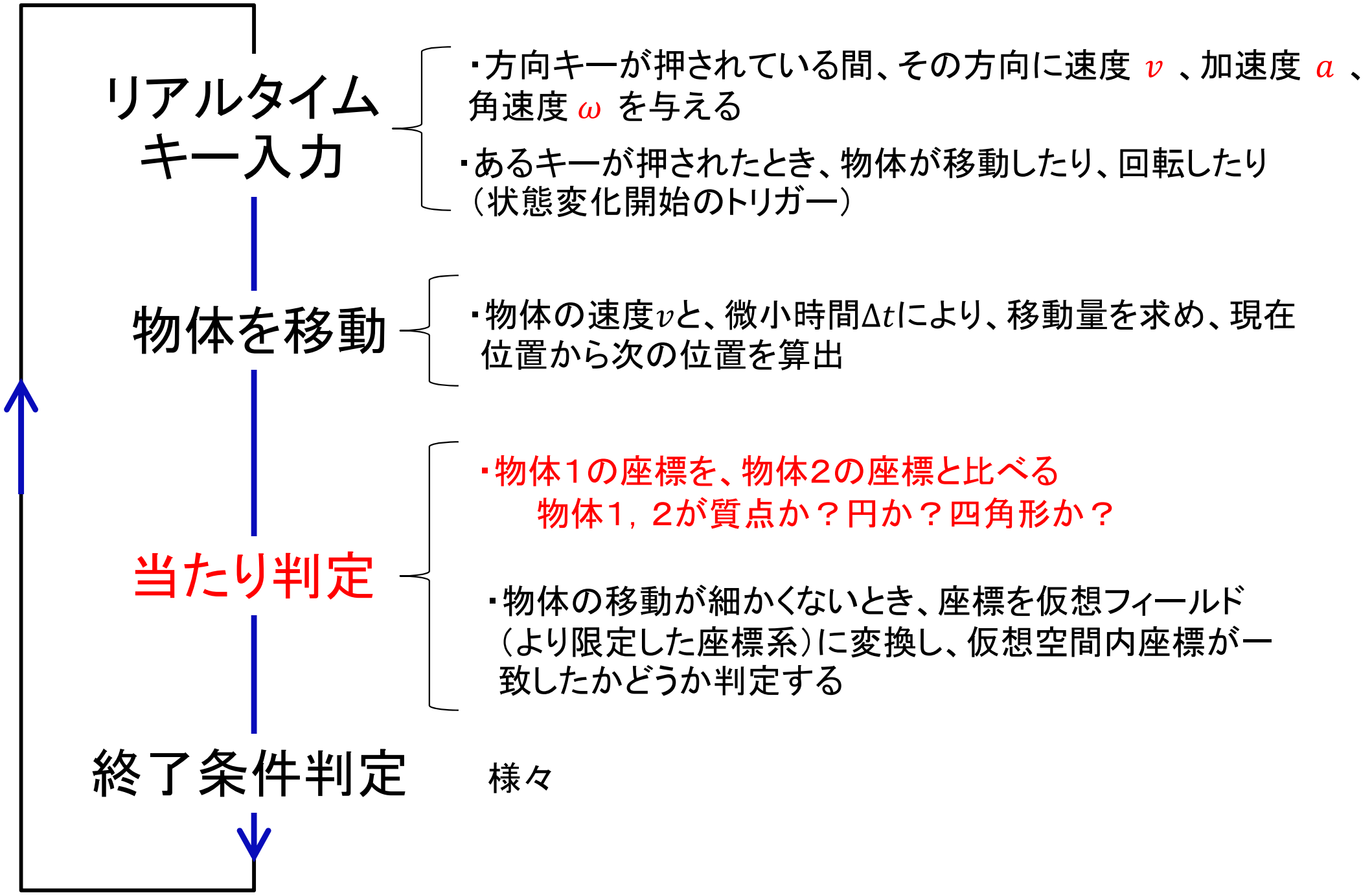
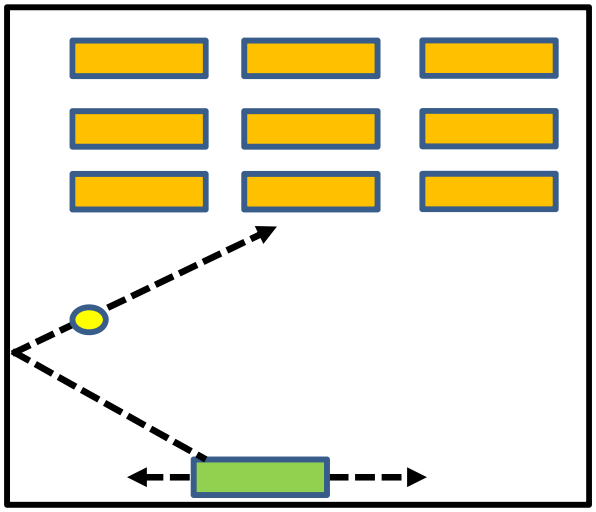
mlibを用いた物理シミュレーション制作（2）

アルゴリズム(1)


リアルタイム
キー入力
・方向キーが押されている間、その方向に速度 v 、加速度 a 、角速度 ω を与える
・あるキーが押されたとき、物体が移動したり、回転したり（状態変化開始のトリガー）
物体を移動
・物体の速度vと、微小時間Δtにより、移動量を求め、現在位置から次の位置を算出
当たり判定
・物体1の座標を、物体2の座標と比べる
物体1，2が質点か？円か？四角形か？
・物体の移動が細かくないとき、座標を仮想フィールド（より限定した座標系）に変換し、仮想空間内座標が一致したかどうか判定する
終了条件判定
様々

シミュレーション例（１）

ブロック崩し

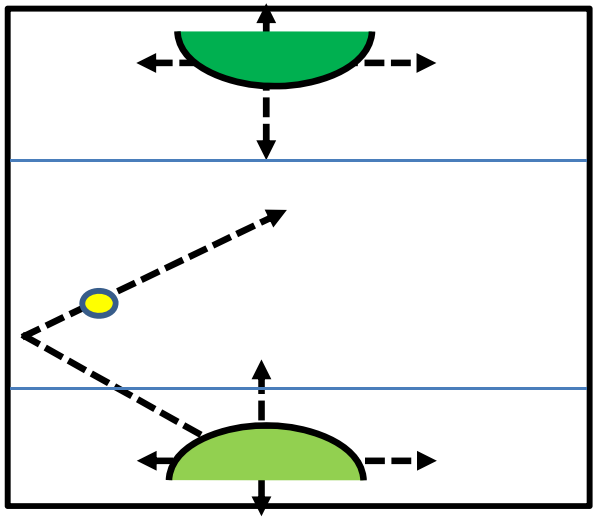


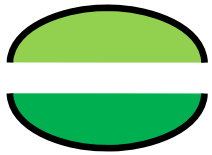
パドル　　方向キーで移動

球　　等速直線運動、壁等で反射

ブロック　　当たるとブロックは消える
球は跳ね返る

テーブルテニス　　対戦





パドル1
パドル2

方向キーで左右移動

球の当たる瞬間に上下キーで
球を加減速

加速度aの物体1と物体2の運動量保存の法則

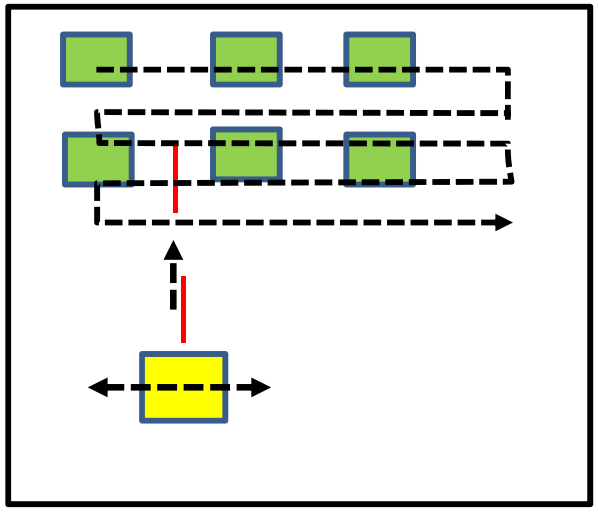
$$m_1(v_1 + a\Delta t) + +m_2v_2 = m_1{v_1}' + m_2{v_2}'$$

円形どうしの衝突は球の当たる
位置によって飛ぶ方向が変わる

球　　等速直線運動、壁等で反射

シミュレーション例（2）

インベーダ系



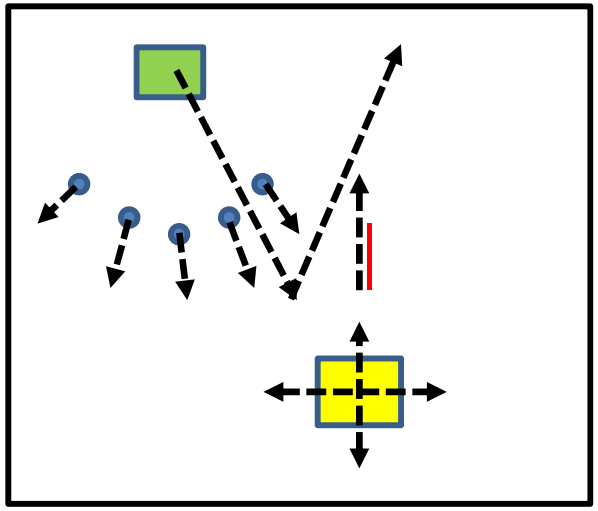
自機　方向キーで移動、発射ボタンでミサイル発射

敵機　複数、同時に、右に動いて、下に移動し、左移動を繰り返す

ミサイル　上に向かって移動

自機の移動速度アップ、ミサイルの連射、敵機もミサイルを出射

ギャラガ系



自機　方向キーで移動、発射ボタンでミサイル発射

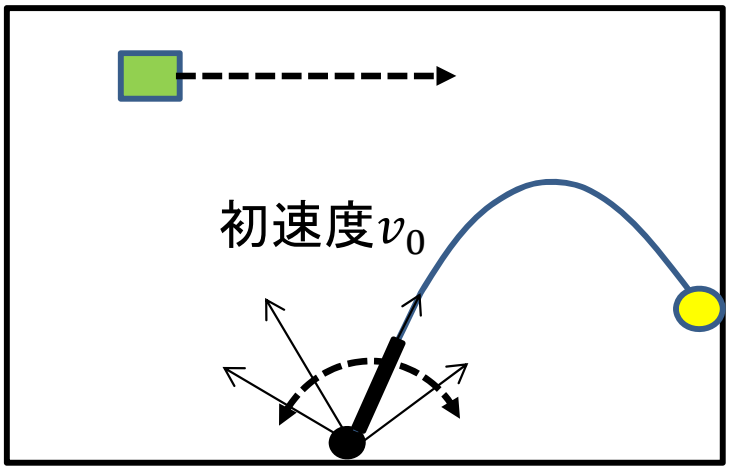
敵機　単体、あるパターンに従って移動
球も発射

ミサイル　直線的に移動

敵機の移動パターン、球の出射パターン、複数の敵機出現、
自機の移動に慣性力

シミュレーション例（3）

自由落下系

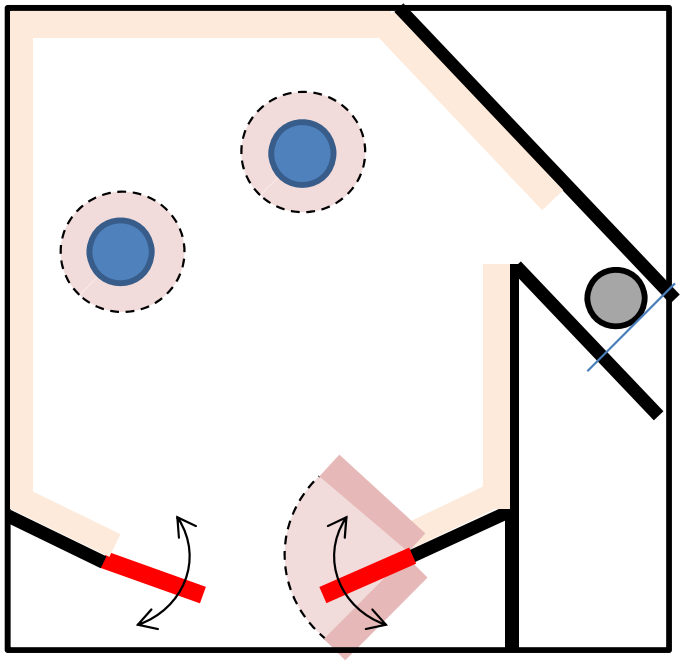


自機　方向キーで向きが回転、発射ボタンを押して、離したときに球発射

敵機　ランダムに等速移動

球　押した長さで初速度、向きで方向を決め放物運動

ピンボール



フリップ　ボタンを押せば上に向かって回転、話せば下に向かって回転、球を跳ね返す

ターゲット　固定された球状の標的。球が当たると、球どうしの衝突に従って球を跳ね返す。ただし、跳ね返り係数が1より大。

球　発射ボタンを押して、離したときに球を発射（押した時間の長さが打つ力）球は自由落下運動。

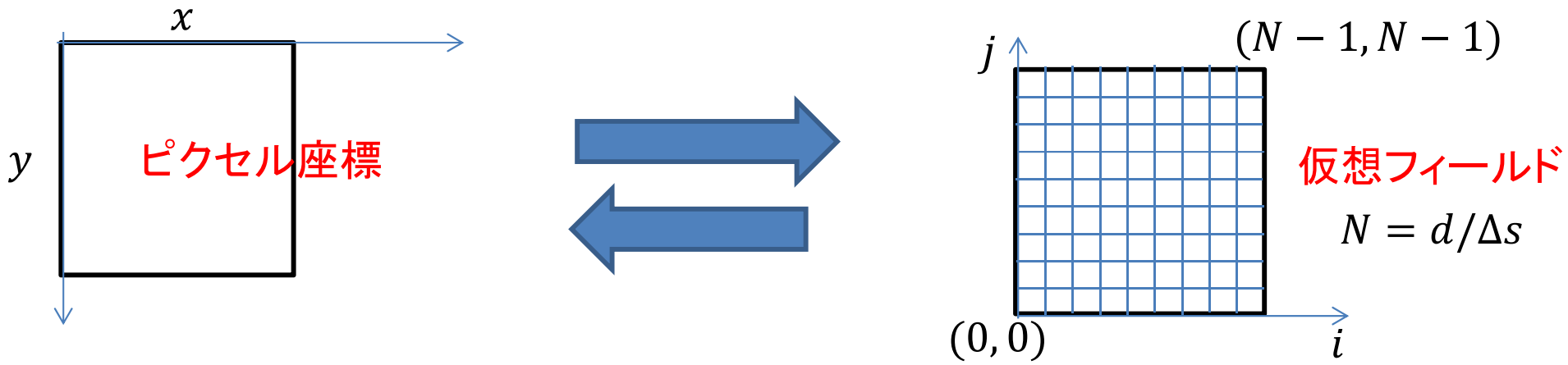
仮想フィールド

・物体の形状が複雑なとき、Collision判定を座標で行うのは非常に面倒。
しかし、全ての物体の動きが細かくない（離散的）ときは仮想フィールドを使う

テトリスであればブロックは複雑な形だが、正方形のブロック単位にしか動かない
ブロック位置は縦20＊横10のフィールドの場所さえわかっていればよい

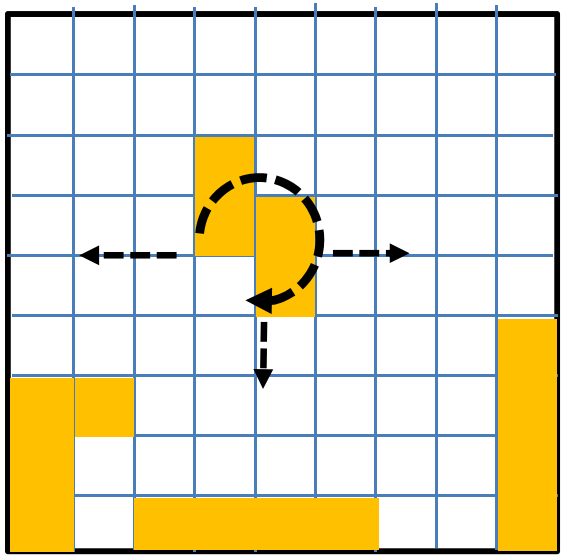
実際に考えるフィールドの範囲が$d*\ d$で、プログラム内で動く物体の1回あたりの移動量の最小値が Δs とすると、$N = d\ /\Delta s$ なる 整数N を使って、$N * N$の2次元仮想フィールドに簡略化できる。

Collision判定は仮想フィールド座標で質点としての判定をすればよい
移動はブロック単位で行い、ピクセル座標に変換



シミュレーション例（4）

テトリス系



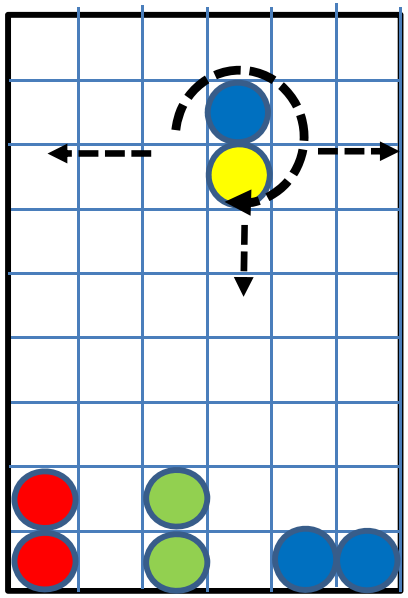
ブロック

4個のブロック7種類の組み合わせ
方向キーで左右移動、等速で落下
ブロックの回転

固定ブロックに当たるか、下まで
落ちればブロックが固定

横1ライン揃えば、そのラインが消える

ぷよぷよ系



ブロック

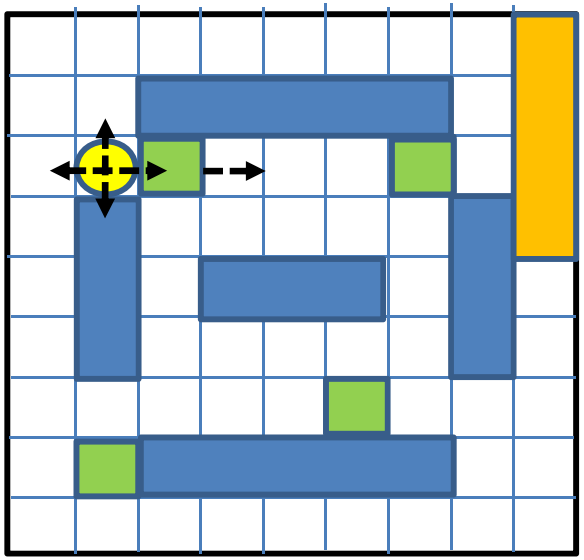
5色以下の色で2個のブロック
方向キーで左右移動、重力により落下
ブロックの回転

移動ブロックが固定ブロックに当たるか、下まで落ちればブロックが固定化

固定化時に4個以上の同色ブロックが縦横につながれば消える
固定ブロックの下に空間ができれば固定化が解除され、落下

シミュレーション例(5)

倉庫番系



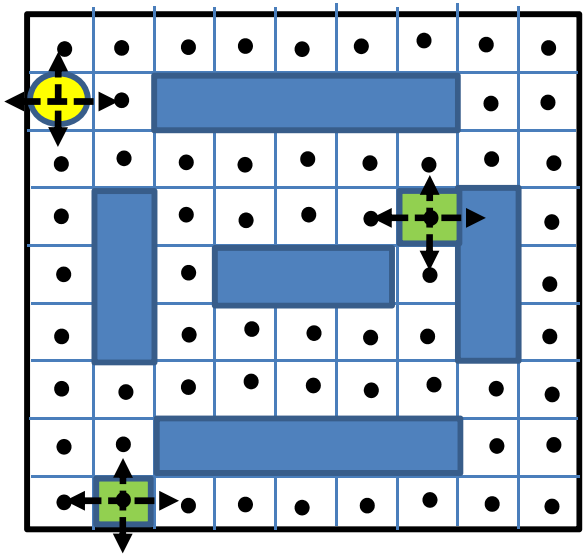
自機　方向キーで移動、壁は通り抜けできない

荷物　荷物に向かって動けば荷物を押して移動できる。引くことはできない。
二つ同時には動かせない

障害物　固定

すべての荷物をオレンジの場所に移動すれば終了

パックマン系



自機　方向キーで移動、壁は通り抜けできない

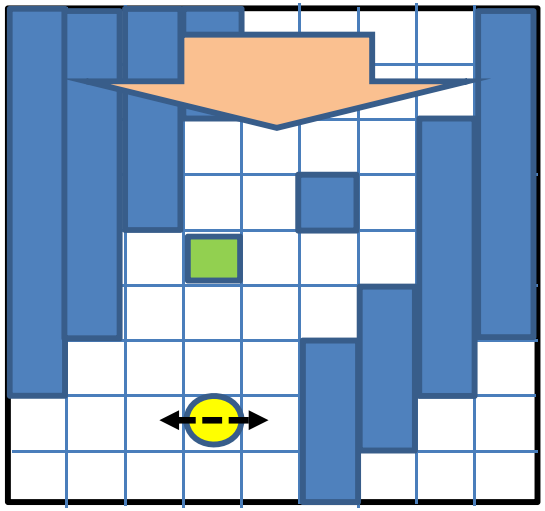
敵機　自機を追いかけるように動く

ドット　自機が通過すると消える。

全てのドットを通過すると終わり

シミュレーション例（6）

障害物避け



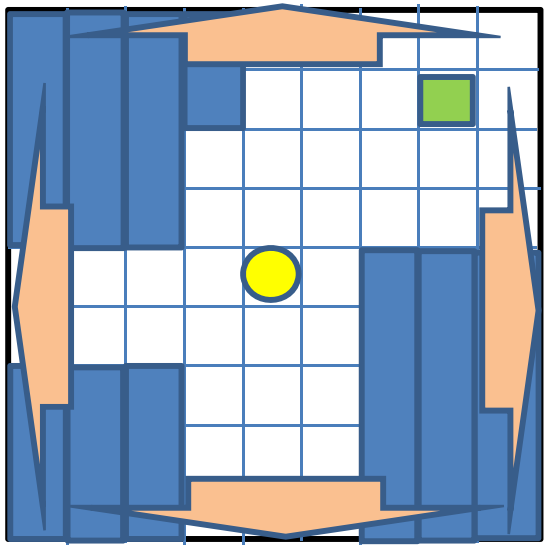
自機　方向キーで左右に移動

障害物　一定時間でスクロール（上から下に）

アイテム　自機の移動速度、スクロール速度変化

障害物にぶつからないようによける

スクロール系（ドラクエ）



自機　移動なし

障害物　キー入力で全方向にスクロール

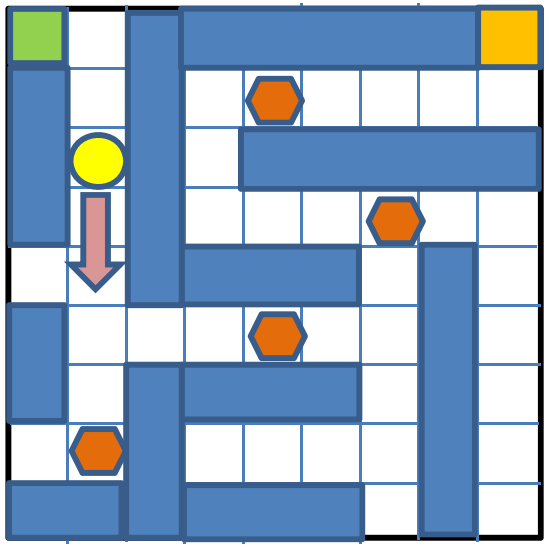
アイテム、町、敵

実際は、もっと大きなフィールドを設定しておき、
表示するのは９×９程度の一部の領域
ストーリーを作製

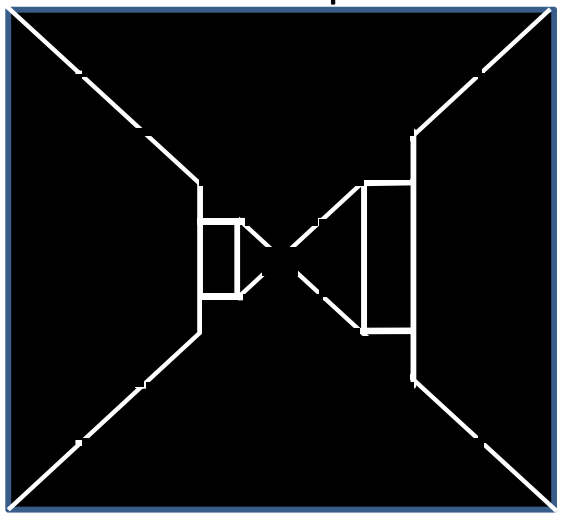
シミュレーション例（7）

ダンジョン系

2Dmap（非表示）



3Dmap



自機　　↑　視点方向への前進

←、→　　視点の左右90度回転

視点方向

壁　　壁のある方向には進めない

スタート、ゴール

敵、or アイテム　　動かない

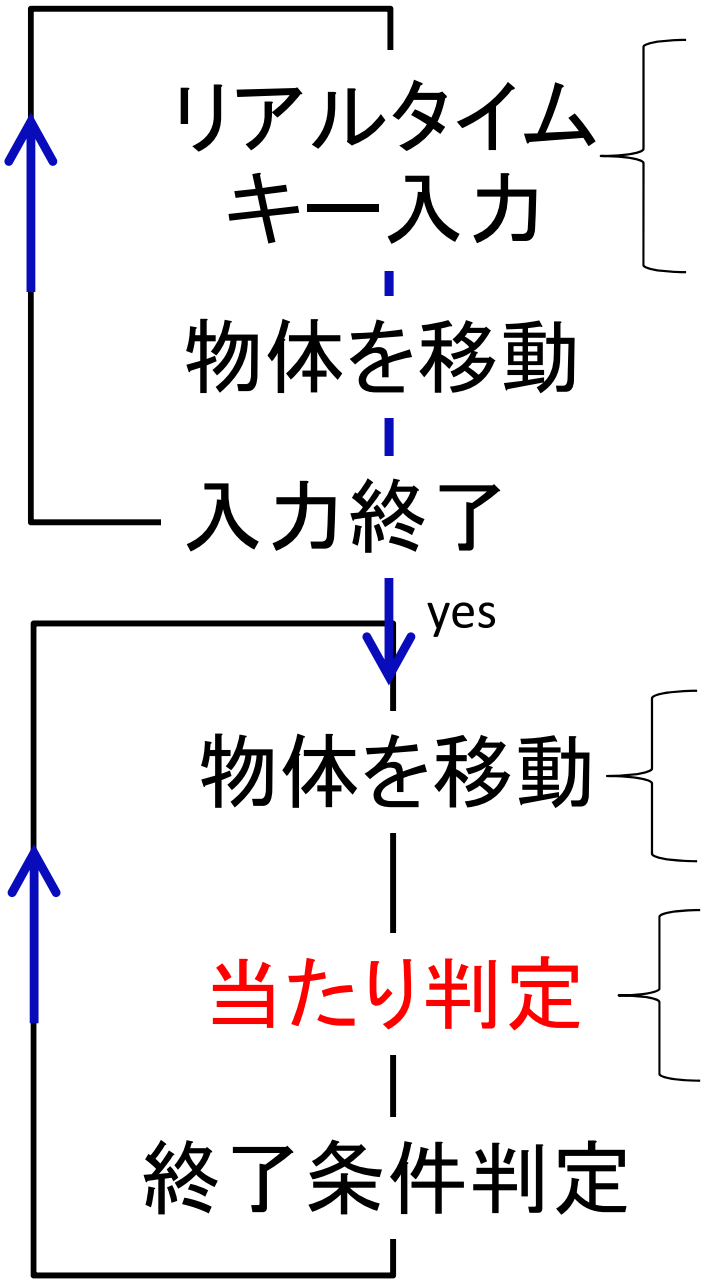
画面には、視点方向の3Dmapが表示

敵に当たると戦闘開始、HPが減る

HPが0になると終了

ゴール地点につくと終了

アルゴリズム(2)



・方向キーが押されている間、その方向に速度 v 、加速度 a 、角速度 ω を与える

・あるキーが押されたとき、物体の移動したり、回転したり

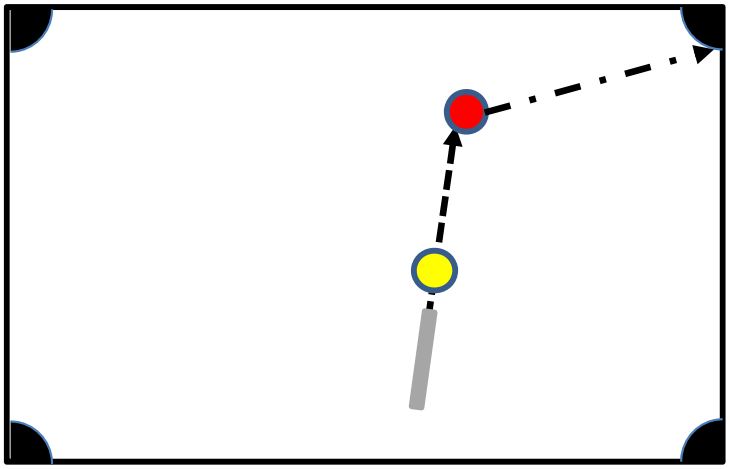
・物体の速度vと、微小時間Δtにより、移動量を求め、現在位置から次の位置を算出

・物体1の座標を、物体2の座標と比べる
　物体1，2が質点か？円か？四角形か？

様々

シミュレーション例（8）

ビリヤード

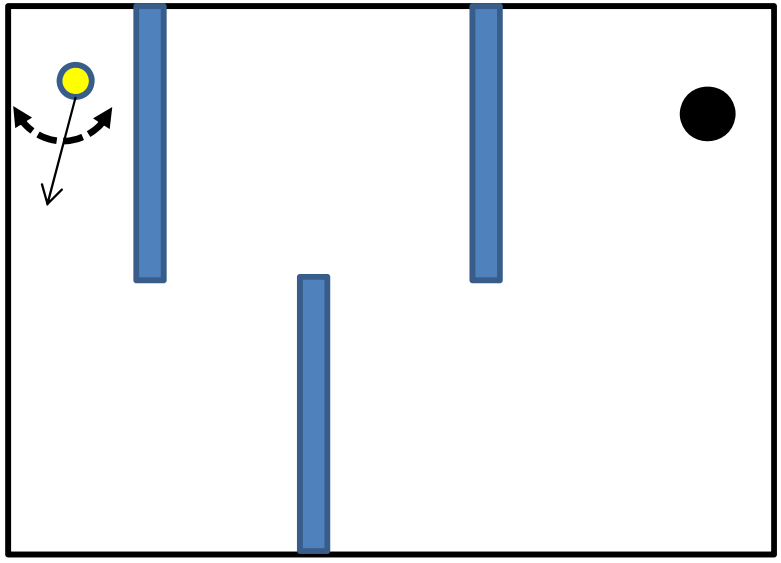


キュー　方向キーでキューの向きが玉を中心に回転、発射ボタンを押して、離したときに球を突く（押す長さが打つ力）

球　打つ強さに応じて初速度が変わる。赤い球にぶつかると2次元の球の衝突の問題にしたがって、跳ね返る

壁は跳ね返り係数が小さい、四隅に入れば終了

パターゴルフ系



球　方向キーで向きが回転、発射ボタンを押して、離したときに球発射

押した長さで初速度、向きで方向を決め、摩擦により減速しながら運動

カップ　入れば終了

壁　反射係数1で跳ね返る

中間課題

以下の2項目を必ず含み

・等速もしくは等加速度運動
・物体と物体間、物体と壁等の衝突（当たり判定）の問題

かつ、以下の2項目のどちらかを含んだプログラムを完成させる。

1. 物理現象をシミュレートする
2. ゲーム性を持たせる

注意点

・過去の課題や他のプログラムをコピーしたと認められる場合は0点
・評価対象はオリジナリティ、プログラムの完成度

締切　12月27日まで。これ以降遅れた場合評価しない。
提出物　Cソースファイル、def.h、課題内容の説明ファイル
提出先　miwa@gunma-u.ac.jp　　メールタイトル：中間課題

特別編　グラフ作成ライブラリ

２次元画像

2次元データの表示

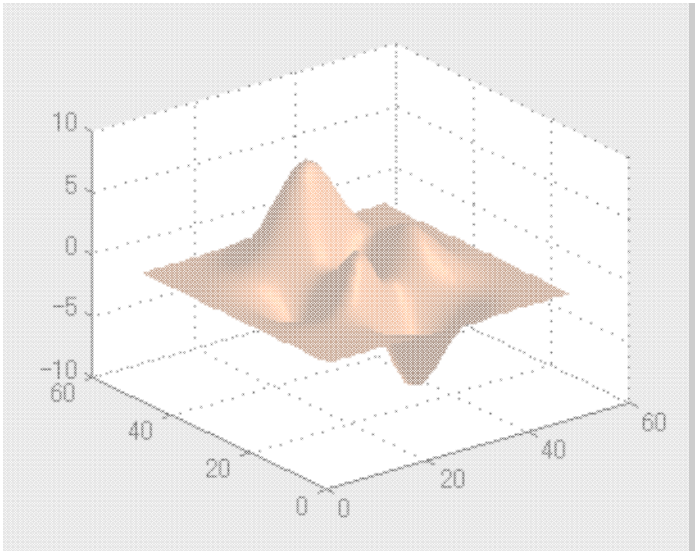
１次元データ

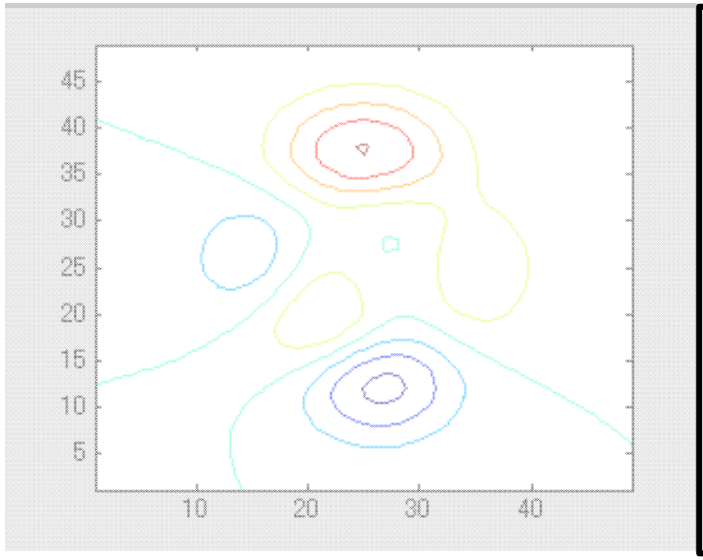
一つのパラメータ変化に着目してある応答（変動）を数値化したもの

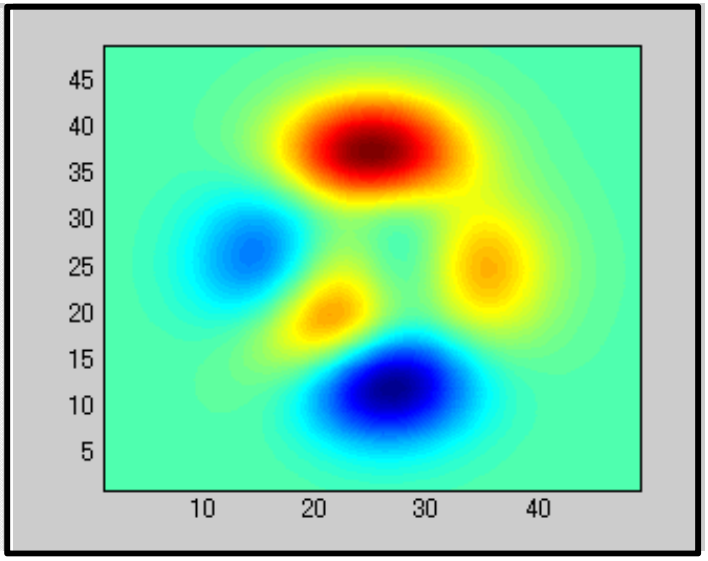
２次元データ

二つの独立なパラメータ変化に着目してある応答（変動）を数値化したもの

画像、１次元データが時間変化する場合　　$z = f(x, t)$
１次元データが場所によって変化する場合　　$z = f(x, y)$







2次元データの表示関数のデータ配置

```
void Plot2d(double *yn, int nx, int ny, double cmin, double cmax
                                    ,int cbflug, int plflug, int szflug)
```

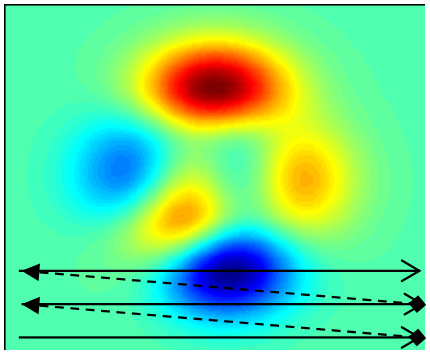
指定されているフィギュアウインドウにdouble型配列 ynの2D画像を描画。

yn ： 1次元配列。二次元データを1次元配列に並び替えたデータとして与える。

nx ： x方向（横軸のデータポイント数）

ny ： y方向（縦軸のデータポイント数）

ny個



nx個

(ny-1)nx	(ny-1)nx+1	(ny-1)nx+2	…	(ny)nx -1
…	…	…	…	…
2nx	2nx+1	2nx+2	…	3nx-1
nx	nx+1	nx+2	…	2nx-1
0	1	2	…	nx-1

2次元画像とPlot2Dに使う1次元配列の順番

上のように見えるべき画像をplot2dで表示させるには、表に書いたような数値の順番で二次元データを1次元に並べ替え、配列変数 yn に格納する。

図の下の行から右方向に向かって行き、右端まで行くと上の行の左端に戻る

2次元データの表示色指定

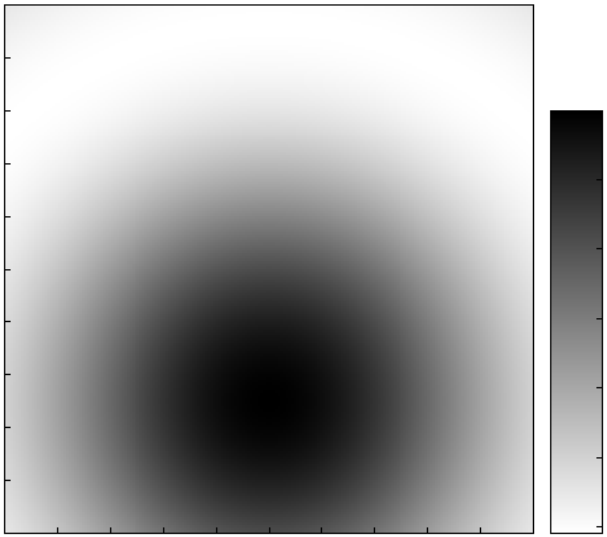
```
void Plot2d(double *yn, int nx, int ny, double cmin, double cmax
                                    ,int cbflug, int plflug, int szflug)
```

cmin ：カラースケールの最小値の指定

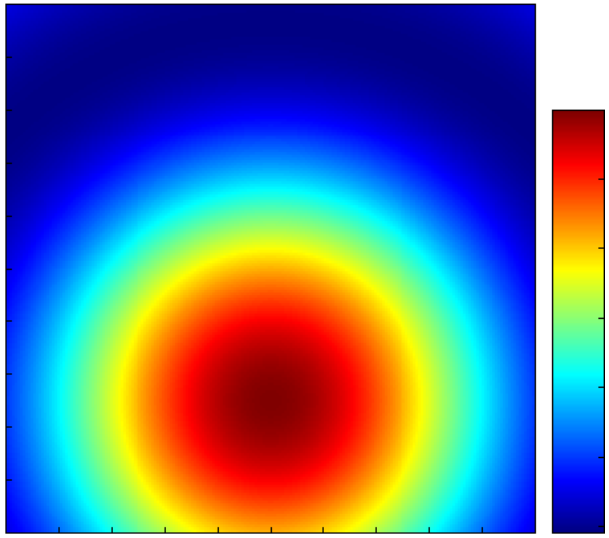
cmax ：カラースケールの最大値の指定、両方とも0のとき最大最小を自動的に探す。

cbflug： 0　グレースケール
　　1　青→水色→黄色→オレンジ→赤
　　2　黒→青→白→赤→黒の循環色（位相の表示等に）

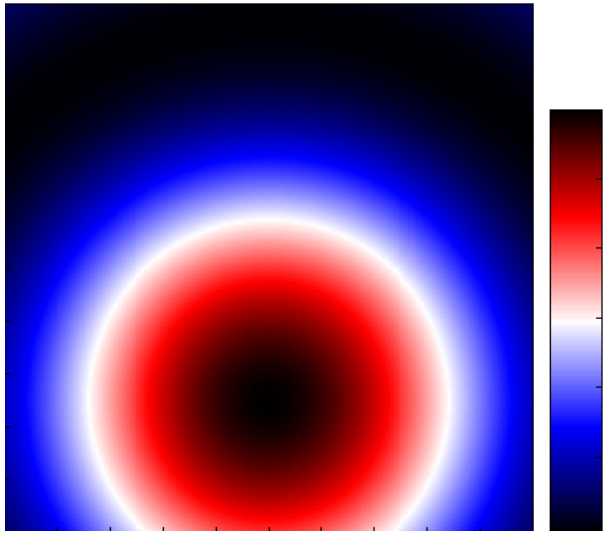
plflug： 0　カラーバーなし
　　1　カラーバー表示



cbflug=0



cbflug=1



cbflug=2

2次元データの表示サイズ指定

```
void Plot2d(double *yn, int nx, int ny, double cmin, double cmax
                                    ,int cbflug, int plflug, int szflug)
```

szflug : 0 画像サイズに関係なくフィギュアウインドウの規定サイズに拡大して描画（データの縦横サイズがウインドウサイズより大きいと画像が乱れるので注意）

1 フィギュアウインドウのサイズを正方形にして描画

2 データのサイズに対応したピクセル数で描画（最もきれい）