☆アルゴリズムの問題(ソース見た方が必要な情報は分かるかも)

〇ソート

　プレイヤーができることは大小比較と指定した二つの要素の入れ替えのみ。

　緑マスは通行可、青マスは落下、岩は通行不可(素材は仮)

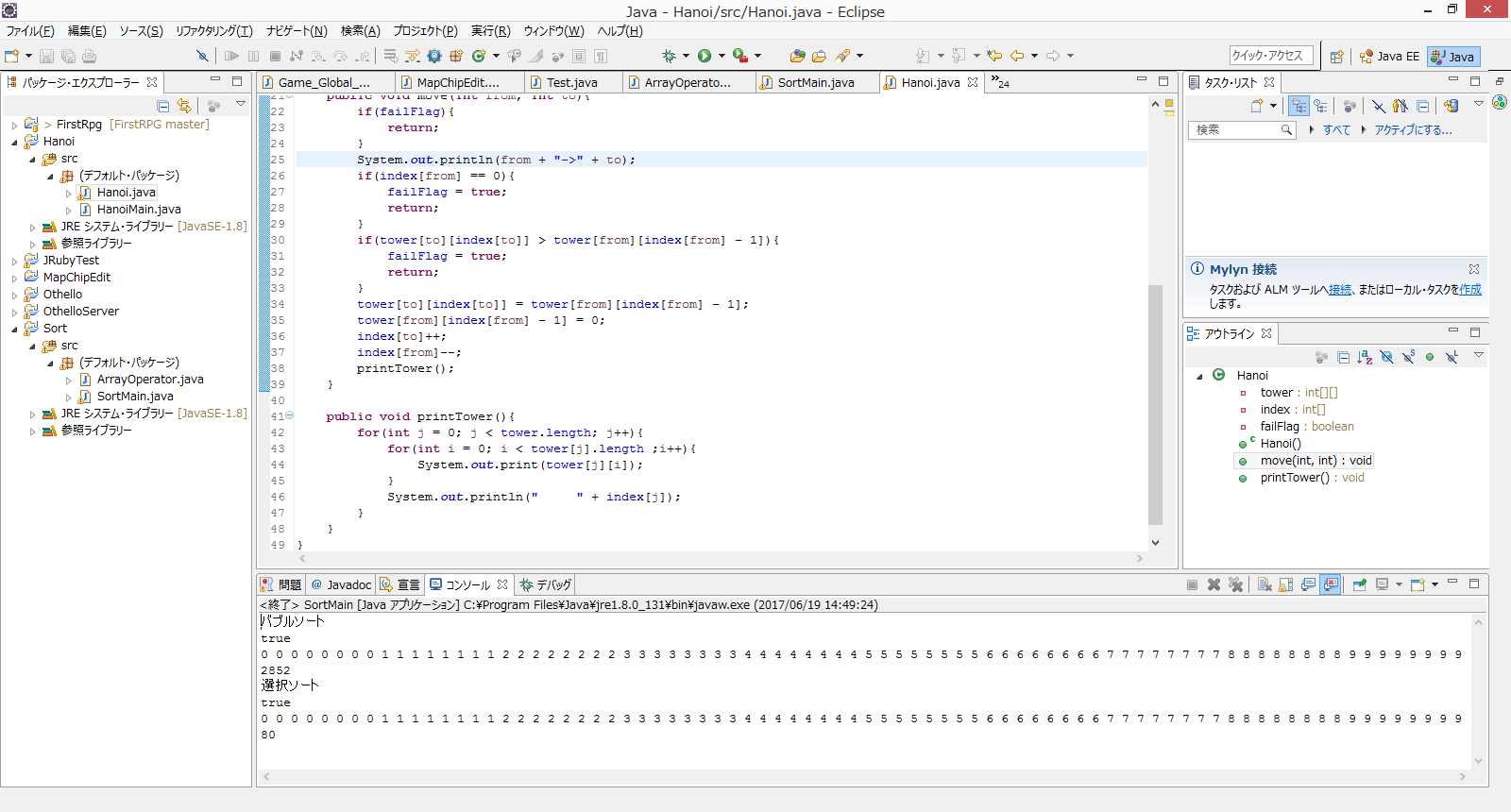
　ソートを実行すると、一つずつ入れ替えが行われる。実行開始から一定時間で元の並びに戻る(ランダムの方がいいかもしれない)。

　ソートの実行時間によって向こう岸までたどり着くのにかけられる時間が変わる。

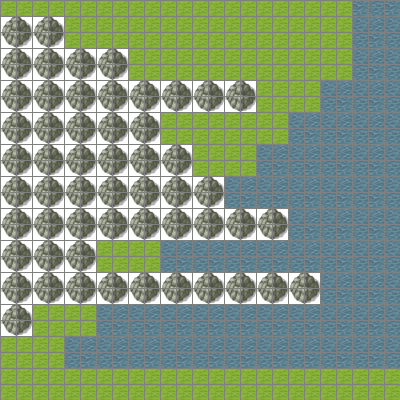
　この例では要素数が少なすぎるため一つのマスを8分割とかするべきかもしれない？

　たぶんソートの実行よりアニメ―ションの方が時間かかるのでそこまで実行時間の差がなくても回数に差があれば十分に差が生まれると思う。制限時間変えて二問分にしたい。

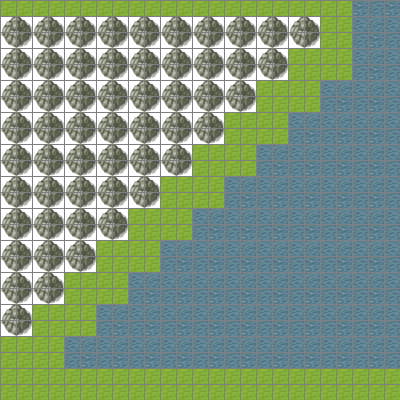
　ソートの交換回数は以下のようになった。要素数は80。

　・ソート

・ソート前



　・ソート後



・知る必要があるメソッド

-int array.length() 配列の長さを返す

-boolean array.compare(int i, int j)　 i, j番目の要素を比較し、i番目がj番目以上なら

true、そうでなければfalseを返す

-void array.exchange(int i, int j)　i, j番目の要素を入れ替える。

○ハノイの塔

　扉の前にハノイの塔を用意しておいて、それをどけなければ扉を通れない。

　積み替えるアニメーションができることが望ましい。

　アクション要素を思いついた人は追記お願いします。

　必要知識　再帰呼び出し

・知る必要があるメソッド

-void move(int from, int to)　from番目の塔からto番目の塔へ円盤の移動を試みる。

〇二分検索

　100個の箱があり、プレイヤーはその箱の中に一つある正解の箱を探す。プレイヤーはプログラム上で、正解の箱であるか調べるメソッドと正解の箱が指定した番号よりも小さい番号か大きい番号かを知るメソッドを使用できる。正解の箱を調べればその箱が光る。ただし、正解の箱か調べるメソッドは時間のかかるメソッドである(要素数が少なくても調べる回数による差を大きくするため)。正解の箱は一定時間で変更され、失敗となる。

　三回連続で正解の箱を選択できれば扉が開く。

　早いコード書ければ箱が光るのが早くなるので正解の箱まで行く時間が稼げる。

・知る必要があるメソッド

-int array.length() 配列の長さを返す

-int array.checkElement(int i,int target)　探している数値が、i番目の要素と等しけれ

ば0、i番目の要素より小さければ-1、i番目の要素より大きければ1を返す。

-void array.setResult(int i)　i番目が答えであると解答する。

○ナイトのツアー

　プレイヤーがナイトに乗ってマス上を移動する。プログラムで求めた通りのナビゲーションが出る。すべてのマスを踏めば扉が開く。

　移動するときにタイミングよく飛ばせるアクション要素？

バックトラックまで記録してプレイヤーにバックトラックを体感させるのもいいかもしれない。

5×5程度のサイズなら単純なバックトラックですぐに求まるが8×8になると長い時間がかかるので、初期の方で5×5の問題を出し、8×8の問題を後半で出題したい。

なお、現在10×10を解くために実装した方法は、欲張り法(ワーンスドロフの規則)であり、ソートと最良優先探索を行う必要がある。

・知る必要があるメソッド・変数

　-int[][] direct = {{1, 2}, {2, 1},{2, -1}, {1, -2}, {-1, 2}, {-2, 1}, {-2, -1}, {-1, -2}}　飛び方配列

-void move(int d) 　dの方向にナイトを移動するのみ。間違ったらその場で移動不可に

-void backtrack(int d)　dの方向から移動してきた来たナイトを移動前に戻す。

-int canMoveCount(int d)　dの方向に飛んだ時にその先からいくつのマスに飛べるか

-boolean canMove(int d) dの方向に飛べるか判定

マスの数はマップからわかる。

最後のメソッドは欲張り法用、難易度を上げたいのであれば移動できるか判定するメソッドだけ用意して数えるのもプレイヤにやらせる方がいいかもしれない。そうなると先読みのための移動のメソッドも必要になるかも？ない方が先読みしてるのがわかりやすくなるかのしれない。(canMove()で指定されたマスを光らせる等)

〇ナップサック問題

　主人公が乗る天秤の反対側に物を載せた箱を載せ主人公を持ち上げる。はこに入る最大の重さまで詰められれば、主人公は持ち上がる。ただし主人公の乗った天秤は時間経過で傾き、箱に詰めるのが間に合わなければ主人公は落ちていく。

　主人公はボタン連打で少しだけ落ちるまでの時間を延ばせる。

　必要な知識　動的計画法

・知る必要があるメソッド・変数

-int getItemNum() 　品物の個数の取得

-int[] getItem(int i)　i番目の品物の情報(重さ,価値)

-int getCapacity()　容量の取得

-pickItem(int i)　 i番目の品物を選択する

〇nクイーン

　アクション要素アイデア求む。

　解法がナイトのツアーと同じだから微妙かも。

　幅を増やしてnクイーン独自の性質を使わないと解けないようにnを大きくする。

　n×nのマスを用意する。クイーンがかかってないマスは黒塗り、かかっているマスはに矢印(上下左右)が見える。マスは実際の部屋に対応していて、入り口のマスから初めてそのマスの矢印の通りに移動すると次の階層に進める。(迷いの森的な)(クイーンが数字にかぶってしまうので後から数字を自動生成？クイーンをどけたり、半透明にするだけでもいいかもしれない)

複数の解を出さなければならないような問題にするためにクイーンのあるマスは見えないようにして複数の解を切り替えるようにすることでゴールまでたどれるのもありかも。

・知る必要があるメソッド・変数

-void boardClear() 　盤の初期化

-void putQueen(int x, int y)　x y にクイーンを置く

〇迷路探索

　構造の把握できない迷路(床が透明で踏み外すと落下)の床のある座標を指定することで光らせることができる(一度に光らせられる数に制限)。プレイヤーはゴールまでの道筋を探索であらかじめ求め徐々に光らせることで通過できる(A\*探索を想定)。

　マップの壁が1, 何もないところが0になるデータとスタート・ゴールの座標を受け取り、ヒープを用いてゴールへの距離が近いマスから探索する。このときヒープにマスを追加するときにそのマスにどの方向から来たかの印をつける。これによってゴールからスタートまで道をたどれる。

・知る必要があるメソッド・変数

-int[][] getMap() マップデータ取得 //map[y][x]

-int[] getStartPos() スタートの座標 //start = {x, y}

-int[] getGoalPos() ゴールの座標 //goal = {x, y}

-int[] setResult(int x, int y) 解答

〇最短経路問題

　制限時間あり迷路の最短経路をもとめる。

　プレイヤーはノード(迷路の分岐点)の間の距離(重み)だけを知ることができプログラムで求めた経路が光って示される。(ちょうどいい迷路作るの難しいかも)

　ダイクストラ法を予定（ヒープ使用）

　迷路内を指示に従って走らせる(AからBにつくと次にCに行くように指示が出る的な)のならワーシャルフロイトのアルゴリズムですべてのノード間の最短距離を出させるのもいいかもしれない。

　ノードにはそれぞれ番号が付いていて、現在はゴールのノードが最大の番号になっている。(ノードの個数を知るメソッドを用意してもよさそうだし、難易度上げたいなら与えない方が難易度が上がる。このままの仕様ならゴールのノードが最大の番号のことも情報として与えるべき)

★暫定順番　アルゴリズム

ソート(1)

ハノイ

二分検索

ナイトのツアー5×5

ソート(2)

nクイーン問題

ナップサック

迷路探索

ナイトのツアー10×10

最短経路問題

/\*15問くらいほしいような気もするけど作れば作るほど大変になるしとりあえずこんなもんかなーと思ったり\*/

☆簡単な問題

**RPGとの関連付けが難しいのでアイデア求む**

変数宣言、代入、四則演算、文字列の出力、ループ、分岐、配列、メソッド、クラス、継承、アクセス制御、モジュール、標準ライブラリ(Numeric,String,Array,DateTime)

変数宣言、代入

型と変数名を宣言させて、値を代入させる。

可視化、壺や樽を型に見立ててしまえばよさそう。

プレイヤーがマップ上のツボに入れたものと対応する数値が引数に渡される(1のカギ、

2のカギ)。それらを加工して用意されてるグローバル変数に代入させる。(1のカギ+2の

カギ=3のカギ)

計算の指定を複雑にすれば四則演算や変数の宣言を必要にできるかも？

四則演算

変数同士の四則演算をさせる。

演算結果があっていればクリア。

文字列出力

文字列を標準出力させる。

プレイヤ名を決めるときについでにやらせる。

ループ

For,while等のループを使って繰り返し処理をさせる。

whileだけしか教えない方が混乱しないかもしれないとか思ったり

 配列と相性がいいかも？

複数の壁をならべて、それを操作するインスタンスをそれぞれ配列で渡し、連続的に開く

ことで駆け抜けられるみたいな。

分岐

if elsif else分岐を使わせる。

ループと組み合わせて敵と戦闘できそう。相手のコマンドに対してどのように動くか、自分のライフにたいしてどのように動くかなどをあらかじめコーディングしてボスと戦う

ループと合わせて、押すメソッドと離すメソッドを繰り返してリフトを上げる(ジャッキアップみたいな)

配列

配列を生成させて演算させる。

カウンタ等を使って数値を代入させる。数値が指示通り格納されて、出力できればクリア。

要素の追加、削除も必要だと思う

メソッド

 　　ほとんど最初に教える必要がある

基本的なデータ構造

問題の中でヒープとか使ってるし…

クラス

 　　オブジェクト指向は必須ではないからない方がわかりやすいかもしれない

継承

アクセス制御

モジュール

標準ライブラリ