第3章

倉庫番

上野 篤, 中山 康, 疋田輝雄

1. 倉庫番とは

広い意味でのゲームプログラミングには，ゲームをユーザに提供するソフトウェアを作成することと，ゲームを解くプログラムを作成することとの，2つがある．もちろん両者は密接に関係し，2つをはっきり分けられない場合もある．ここは後者で，倉庫番の問題を解くプログラムをC言語を用いて作成した．

倉庫番はコンピュータ上で楽しめる1人用のパズルゲームである．国産品であり，1982年に発表されて以来，その時々のほとんどすべてのパーソナルコンピュータや家庭用ゲーム機上で発売されてきた[3]．外国においても，Sokobanの名称で広く通用し，X Window版のソースコードは文献2)にある．

図1はゲームの進行の様子を示している．画面に表示される倉庫は，壁と，その間の通路からなる．通路上には，1人の「**番人**」がいて，またいくつかの「**荷物**」が散在している．さらに，荷物と同数の「**ゴール**」地点が指定されている．ゲームの目的は，番人を画面上で操作して，荷物をゴール地点まで運ぶことである．荷物を動かす際，「荷物は押すことしかできない」，「1度に1つしか押せない」という制約があり，この制約がこのゲームの複雑さを生み出す元となっている．

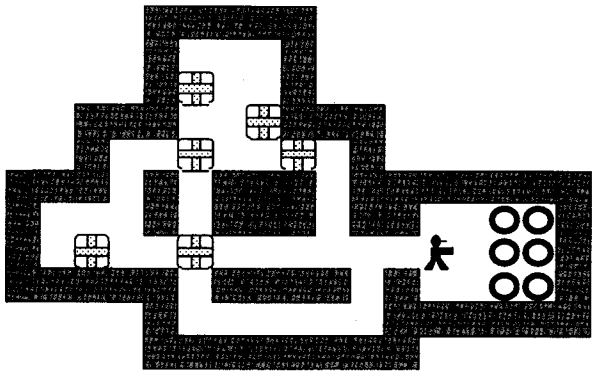
倉庫番をコンピュータ上で解くための技法，すなわち解の探索法，データの内部表現や，この問題特有の困難な点について述べる．倉庫番のようなパズルゲームの解法は，詰将棋や各種のパズルなどと大枠は同じで，人工知能の教科書にあるとおりである[5]．しかし細部は問題に特有の工夫が必要であり，この倉庫番の場合はやさしくない．とくに局面の評価関数の作り方と，荷物の作ってしまう「袋小路」への対策が重点である．

ここで紹介するプログラムは，上野（主として評価関数部分）と中山（主として袋小路部分）が作成した[4]．本稿は主として文献4)の要点をまとめたものだが，参考文献は新しい．なお，村瀬・松原・平賀[1]は，倉庫番の問題例を自動的に作成することを扱っている．そこでは解の幅優先探索を用いている．

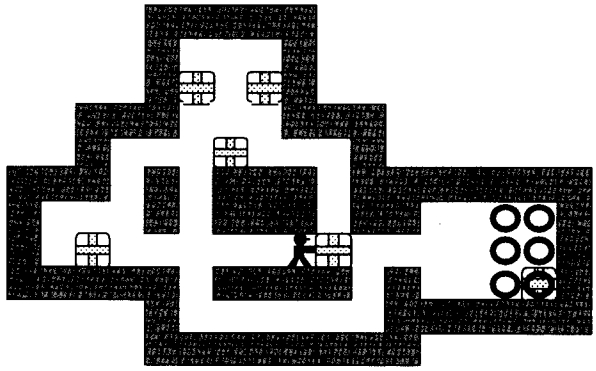
2. 準　備

2.1 用語

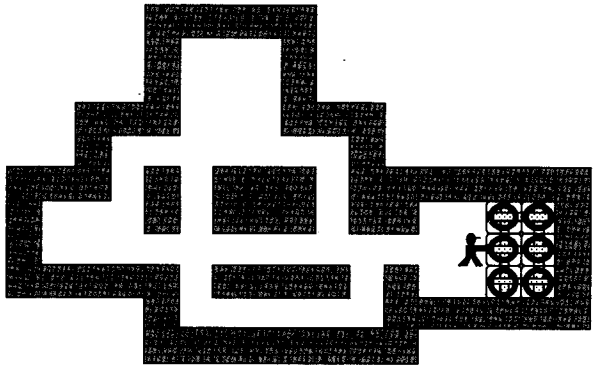
このゲームを論じるための用語を用意する．倉庫番には数多くの倉庫つまり初期局面が用意されている．ソフトによって呼び名は，n面，n番倉庫などというが，ここではそれぞれを，**マップ**と呼ぶ．マップを構成する各マス目のことを，**マス**と呼ぶ．また，倉庫のレンガ模様の壁の部分を1マス単位で**ブロック**と呼び，通路上のマスで，荷物の最終位置として指定されたものを**ゴール**，ゴールでなく，かつ番人も荷物も存



初期局面



途中



最終局面

図1　倉庫番のゲームの進行

在しないマスを**スペース**と呼ぶ．

また，このゲームの操作は，番人の移動と，荷物の移動の2つのみであり，これらの動作を以下の用語で表わす．

● **1歩**：プレイヤは，番人を縦か横方向に1マス移動させることができる．このような番人の単なる移動を，1歩と呼ぶ．

● **1手**：番人が荷物を押して1マス移動させる操作を1手という．これは1歩でもあるとみなす．荷物の行き先のマスは現在，スペースまたはゴールでなければならない．この操作で番人も，その荷物のあった位置へ移動する．

2.2　同一の局面

荷物と番人の配置された局面において，どのような2つの局面を「同じ」局面とみなすかを考える．1歩の操作を基本として考えると，同一の局面とは，荷物と番人の両方の配置がまったく同一のものである．1手を基本とする場合は，荷物の配置はまったく同じであるが，番人の位置については，移動可能な範囲のどこにいても同一の局面とみなし，番人の移動可能な範囲が異なるときは違う局面とみなすことになる．この定義は本稿におけるキーポイントの1つである．

図2において，網のかかっているマスが，現在番人の移動可能な範囲である．1手という意味で考えるとき，(a)と(b)は同じ局面である．ともにあと2手で最終局面となる．(c)は荷物の配置は(a)や(b)と同じだが，異なる局面である．番人が荷物を押すことのできる方向が違う．また最終局面まであと4手必要である．

私たちは，倉庫番の解を考えたり探したりするとき，1手のほうを用いる．その理由は，番人の位置を厳密には区別しないことにすると，たとえば番人の歩数を最小にするような解は求めることができないが，解の探索において生成して調べるべき局面の数はずっと少なくてすむからである．大きなマップを解くときの効率を重視して，こちらの定義を採用した．

2.3　主な死に手

ゲームの進行途中で，マップ上に1つでも，もはや動かすことができない荷物や，限られた範囲にしか動かせない荷物ができてしまうと，最後まで解けなくなってしまう．いわば「**死に**」が局所的に生じる．ここで主な死に手をあげる．次の2種類に大別できる．

●荷物をまったく動かせなくなる死に手（図3）．

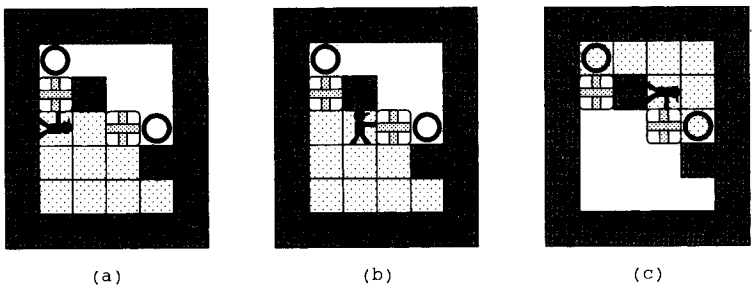


図2　同じ局面の識別

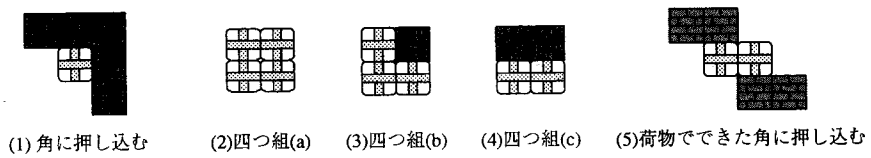
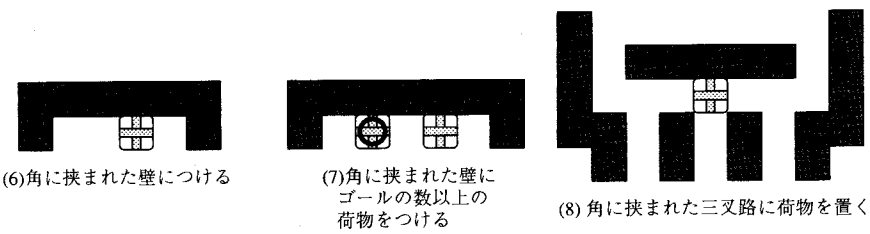


図3　死に手-1



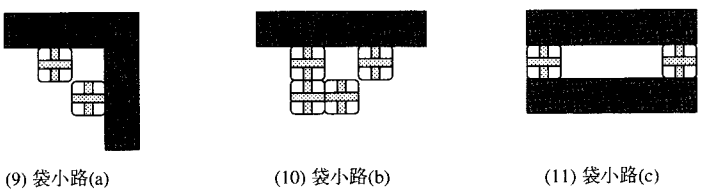


図4　死に手-2

●荷物を限られた範囲にしか動かせなくなる死に手（図4）.

四つ組と名づけた死に手は，もっとも基本となるものである．図で**袋小路**と名付けた死に手は，図よりもっと大規模なものもあり，扱いの最もやっかいな代物である．

なお死に手というとき，本来はそのような**指し手**（move）つまり操作のことをいうのだろうが，ここでは操作の結果としての回復不可能な状態ないし形のことも死に手といい，本稿ではもっぱらこの意味である．

3. プログラムの概略

3.1 データの内部表現と主な変数

マップの各構成要素ないし状態は，**表1**のように表現する．

マップ上での要素の配置を表わす配列は，maze[][]とbags[][]である（添字はその座標）．配列mazeには，マップの地形に関する不変のデータ（荷物や番人がどこにいようと変わらないデータ）を格納する．具体的には，SPACE，GOAL，AVOID（後

表1 要素の内部表現（*は後出）

要　素	マクロ
スペース	SPACE
番人	MAN
荷物	BAGGAGE
閉領域*	SHUT
番人の到達可能なマス*	CAN_MOVE
ゴール	GOAL
荷物の入ったゴール	GOAL_IN
荷物を置けないマス*	AVOID
ブロック	BLOCK

出)，BLOCKである．この配列を参照することにより，

(番人が移動できるマス)

＝SPACEまたはGOALまたはAVOID

(荷物を置くことができるマス)

＝SPACEまたはGOAL

が成り立つ．

配列bagsには可変のデータを格納する．具体的には，SPACE，BAGGAGE，CAN_MOVEのいずれかである．BAGGAGEは荷物を表わし，CAN_MOVEは番人がその時点で到達可能なマスを意味する．この配列を参照することにより，

(荷物を置けないマス)＝BAGGAGE

が成り立つ．(ある荷物がすでにそのマスに存在しているため．)

配列maze，bagsの両方を参照することにより，荷物の移動および番人の移動を制限している．

3.2 前処理

マップの形状を読み込んだ時点で判別できる事柄を，解の探索前に行なう．具体的には，特定のマス目に値AVOIDを割り当てる．先の主な死に手のうちで，

(1)角に押し込む

(6)角に挟まれた壁に荷物をつける

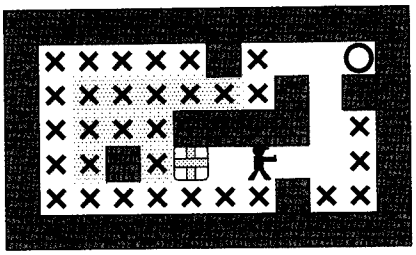
(8)角に挟まれた三叉路に荷物を置く

を避けるために，

- 角を探す
- 角に挟まれた壁ないし三叉路を探す

ことを行なう．発見した角，壁，三叉路の該当するマスにAVOIDを割り当てる．AVOIDとは荷物を置くことを避けるという意味である．ただし，これらにゴールが含まれる場合はAVOIDは割り当てない．このAVOIDのマスの発見は解の探索の効率に大きな役割を果す．

角や壁以外にも，一般に荷物を置いてしまうと，そこからはどのゴールにも荷物を運ぶことができなくなるようなマスにAVOIDを割り当てる．その方法は，荷物1つをゴールに置いておいて，その荷物をあらゆるマスに引っ張っていくという（押すと逆の）操作で移動させる．それをすべてのゴールに対して行なうと，どのゴールから引っ張ってきても荷物を置くことができないマスが見つかる．その場所にAVOIDを割り当てる．実際はこの方法だけですべてのAVOIDを割り当てることができる（図5）．



× ＝角、壁、三叉路によるAVOID

× ＝その他のAVOID

図5 AVOIDの割当て

なお，ゴール周辺の壁に対しては，角に挟まれた壁でもその中にゴールがある場合には，荷物を置くことは当然許される．前処理の段階では，壁につけてよい荷物の個数を表にしておき，実際に探索する時点でその個数を数える．

3.3 プログラムの流れ

プログラムの流れの概要は，図6のようになっている．

4. 解の探索

解の探索は定石どおり，**最良優先探索**（best-first search）を採用する．局面の評価，言い換えると解までの予想コストを立てるのが難しく，早く解を見つけるためのポイントである．

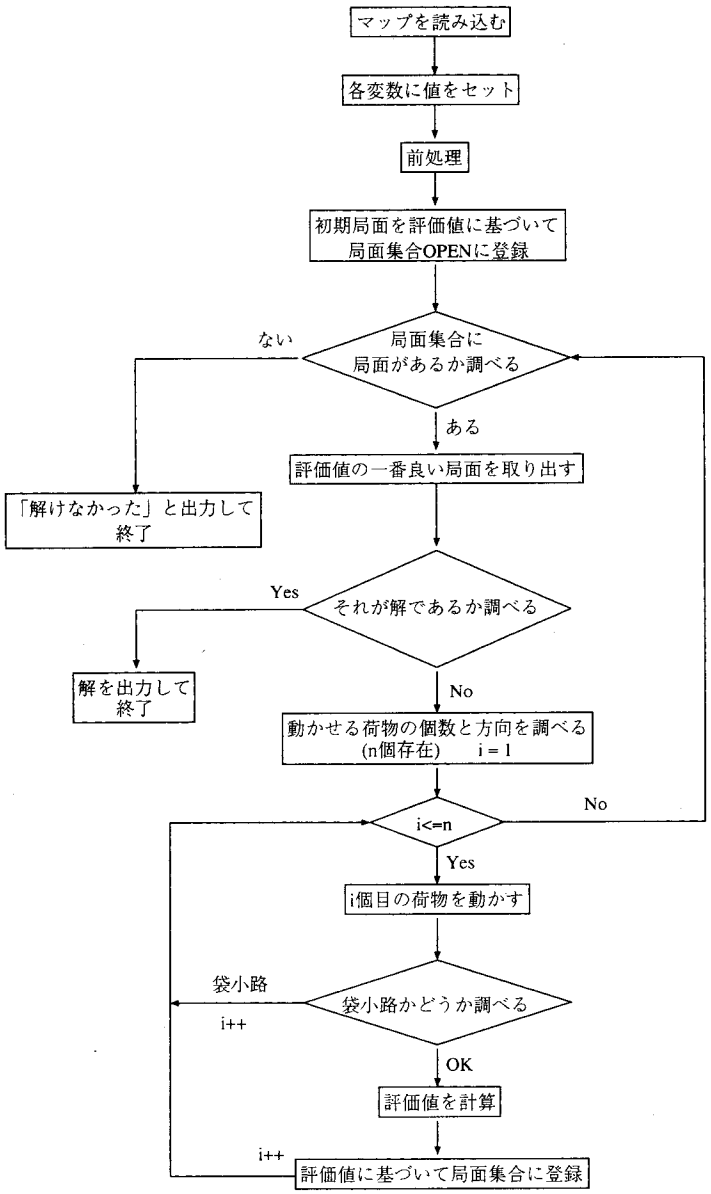


図6　アルゴリズムの構造

4.1　最良優先探索

解の探索の様子は探索木で表現される（図7）.

すでに展開された局面からなる集合をCLOSEとする．これらの局面は探索木の内点である．またこれから展開されるべき局面の集合をOPENとする．これらの局面は探索木の葉である．今から展開する局面として，集合OPENの中から評価値の最良であるものを取り出す.

4.2　評価関数の準備——ゴールへ入れる順番

局面の評価関数を具体的に作らなければならない．ある局面の評価値というものを，ゴールを目指しての荷物の位置の善し悪しにより与えたい．ゴールに荷物が近いほど評価値が高くなり，ゴールにすべての荷物が入ったときに評価値は最大となるようにする.

各ゴールに対する各マスでの評価値，という概念を導入する．各ゴールの評価値は，荷物を入れる順番が先のものほど高くなるようにする．評価値の実際の算出は，次のような段階を経て行なう.

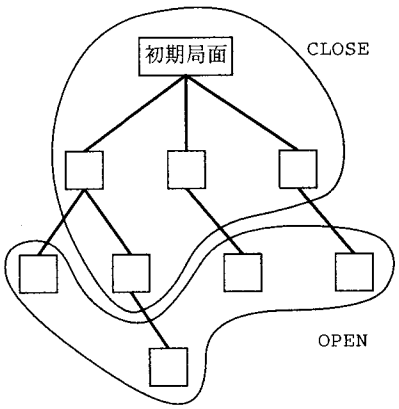


図7 探索木

1.探索の前処理として，各ゴールに順番付けをする．先に荷物を入れるべきゴールにより高い評価値を与えるためである．先に入れたいゴールから順に，1，2，…，の順番をつける．
2.やはり前処理の段階で，各ゴールに対しての，マップの各地点における評価値というものを，そこに荷物があるとして，計算しておく．
3.実際に荷物を動かして解を探索する際に，この値を用いることによって，局面の評価値を計算する．

まずゴールに荷物を入れる順番に関してである．解を実際に求める前にこの順番を完全に正確に決めることは容易ではない．問題によっては，ゴールに1度入れた荷物を出したりすることが必要なこともある．ここではある簡単な方法でゴールにある程度の順番付けをし，それを探索時の評価法に反映させるということにしている．結果としては，マップにもよるが，それなりの値を返している．

ゴールの順番付けには2つのアルゴリズムを用意した．第1の方法は図8のようにする．要するに，荷物をすべてゴールに入れた局面から，逆に荷物を取り出していくわけである．取り出せる荷物を調べるのには，方法1では，各荷物を引っ張って初期局面の荷物のうちのいずれかの位置に運ぶことができたら，その荷物は取り除けるということにする．そしてその順にゴールに番号を付ける．

第1の方法で行き詰まったら，その時点で第2の方法に切り替える（図9）．この方法では，荷物をゴールから取り除ける条件として，最小限の条件「その荷物から（荷物と番人のための）荷物のないマスが2つ続いていること」だけを採用する．

4.3 評価関数の準備——ゴールまでの道のり

次に，各マスからゴールまでの道のりを調べる．これは，単にそのマスからゴールまで番人が移動するときの道のり（歩数）とはせずに，実際に荷物を押してゴールまで運ぶのにどれだけ手数がかかるかにする．両者は異なることがある．図10(a)のように単純に歩数で道のりを定めると，斜線のマスではゴールまでの道のりが1となっているが，実際にはそこにある荷物を1回押しただけではゴールに入れることはできない．

さらに，各マスのゴールまでの道のりはただ1つとは限らず，その荷物に対する番人の位置によって，各マスのゴールまでの道のりは複数あり得る．

4.4 評価関数

局面の評価値は次のように決めることにする．まず，その局面において，荷物を入れたい目標のゴールを，まだ空いているもののうちで順番の最も高いものとして決める．荷物の現在の位置からそのゴールまでの道のりが近いほど高い評価値とし，すでにゴールに入っている荷物にはかなり高い値を与える．荷物全体での評価値の和をその局面の評価値とする．

ゴールに近い荷物を優先的に動かし，できる限り入れていくことが，解法への近道であると経験によりわかっている．図11の場合，局面(1)から展開される，局面(2)と(3)の評価値としては，(2)のほうを(3)より高くすることが望まれる．そこで各地点の評価値を，そのゴールに近づくにつれて道のりの2乗に反比例して上がるようにする．こうすることで，ゴールにより近い荷物を先に動かすように局面を展開していくようになる．

4.5 局面集合OPENとCLOSEのデータ構造

解の探索において，局面一つに対して記憶すべきデータは，

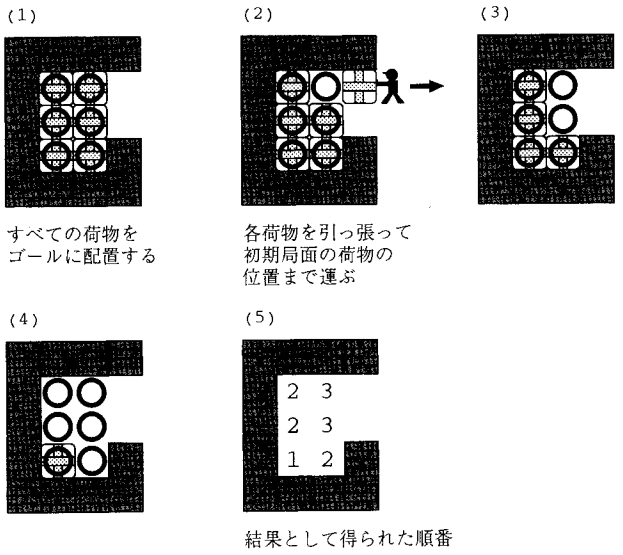


図8　ゴールに入れる順番を求める

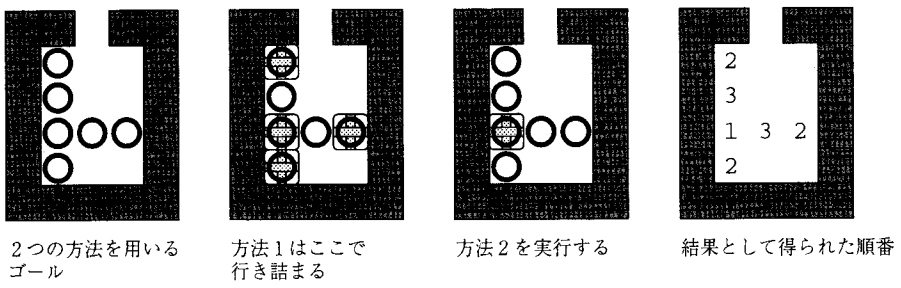


図9　方法1と方法2を用いる順番決定

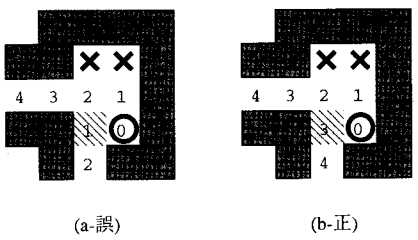


図10　ゴールまでの距離

- ●番人の座標
- ●番人のこれまで歩いた歩数
- ●全荷物の座標
- ●親の局面

の4つである．

集合OPENのデータ構造は図12のようである．局面の評価値自体をハッシュ値とするハッシュ表である．同じハッシュ値（評価値）をもつ局面は線形リストでつなぎ，線形リストの中では，これまでの歩数の少ない順に局面を並べる．線形リスト中で，歩数の少ない局面を先に置いているのは，解としてできるだけ歩数の少ないものを得たいためである．

集合CLOSEは展開ずみの局面を記録しておくためのものである．そのデータ構造はOPENとほぼ同じであるが，ハッシュ関数は高速化のために，局面評価関数よりも簡単なものを用いる．

局面のデータベースにハッシュ法を用いるのは，このような解の探索において定石の1つである．

5. 荷物を1手動かす

ここでは，番人が荷物を動かすために必要な処理について説明する．

5.1　動かせる荷物を探す

押して動かせる荷物と押す方向を判別するには，次の2つの条件を満たしていることを調べればよい．

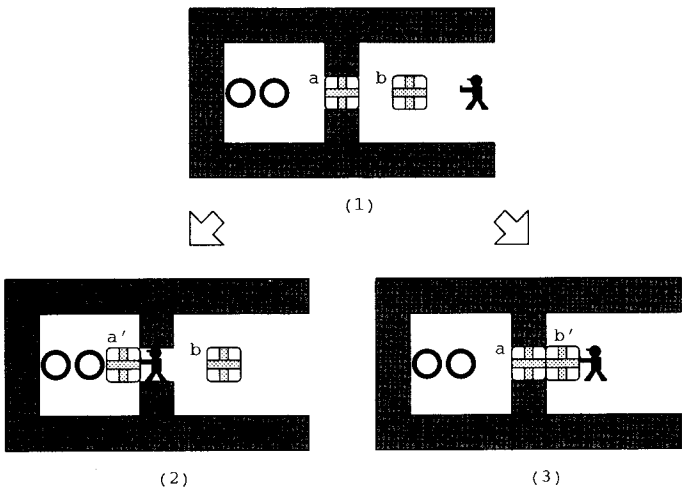


図11　荷物をゴールに近づける

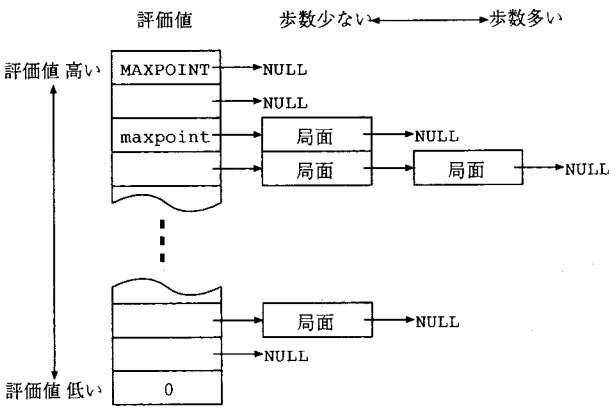


図12　集合 OPEN のデータ構造

1.その荷物の隣のマスに番人がたどり着くことができること．

2.移動後の荷物の位置となるマスが，BLOCK，AVOID のどちらでもなく，かつそこに荷物がないこと．

条件1を調べる準備として，マップ上で，番人が現在到達できるすべてのマスに印をつける．印としては，可変のデータを格納する配列 bags[][] に，CAN_MOVE という値をいれる．そのうえで条件2を調べるため，各荷物の各辺に対して以下の処理を行ない，動かせる荷物を特定する．

1.着目している荷物のある1辺が，CAN_MOVE をもつマスに接しているかを調べる．

2.接していたら，その接している辺の向こう側のマス（移動後の荷物の位置となるマス）が，BLOCK，AVOID のどちらでもなく，かつそこに荷物がないことを調べる．

3.上の2つを満たしていたら，仮にそのマスに荷物を動かしてみて，（袋小路以外の）死に手になっていないかを調べる．

以上の1～3をすべて満たしていたら，その荷物は動かせる荷物である．

動かせる荷物については，その荷物に付けられている通し番号，その荷物の動かせる方向，番人の現在位置から着目している荷物にたどり着くまでの最小の歩数の3項目を記録しておく．

5.2　実際に動かす

動かせる荷物とその方向が判明したので，その1つずつについて，実際に1手動かしてみる．荷物を動かした後の局面が，

- 以前に存在した局面でなく（集合 CLOSE の中になく）
- これから展開される局面の集合 OPEN の中になく
- 袋小路になっていない

ならば，その局面を集合 OPEN の中に入れる．

6. 死に手かどうか調べる

荷物を移動できるかどうかを調べるためには，その手が死に手かどうかを，死に手の種類ごとにチェック

する必要がある．

6.1　四つ組

四つ組とは，荷物とブロックが田の字形に4つ固まってしまう死に手である．これの判別の際には，動かしてみる荷物を含む前方2方向の4マスについて，荷物とブロックが4つ固まっていないかを調べればよい（図13）．

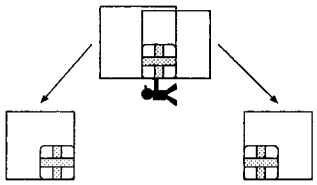


図13　四つ組を調べる4マス

調べる4つのマスの中にゴールが含まれている場合は注意が必要である．壁を背にして2つ並んでいるゴールが含まれている場合は，四つ組として却下してはいけない．逆にゴールが含まれている場合は無条件に許すことにしてしまうと，壁を背にしたゴールが1つあり，そのゴールに荷物が入っている場合は，もう1つの荷物をくっつけてもよいことになってしまう．結局，注目している4つのマスにおける，荷物の数，ブロックの数，ゴールの数をそれぞれ数えておき，

（荷物の数＋ブロックの数＝4）

∧（荷物の数≠ゴールの数）

を満たすとき，四つ組として却下すればよい．

6.2　荷物によって角ができ，動かせなくなる場合

四つ組のルールは，「壁の角に荷物を置いてはいけない」というルールと本質的に同じである．四つ組の場合と同様の処理として，図14のようなものがある．このような荷物によってできる角は，ブロックによってできている角とは異なり，荷物の移動により変化するので，前処理によって調べるわけにはいかない．

これは荷物がいくつかつながっており，そのつながった荷物の両端の荷物が壁などによって動けなくなっている場合に起こる死に手†である．この場合は，単にそこが角になっているというだけでは判別できない．押した荷物につながっている荷物を調べて，それらが互いに動けなくなっているということを調べなくてはならない．これを調べるには，動かせる荷物の条件を「その荷物の両側が空いていること」のみとし，動かせる荷物を取り除いていく．最終的に荷物が残ったならば，それは死に手であると判断する．

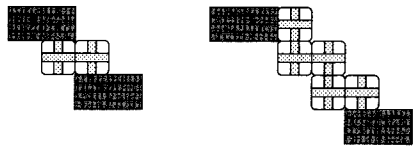


図14　荷物によって角ができる

図示すると図15のようである．（a）の局面から押した荷物につながっていない荷物を取り除くと，（b）のようになる．動かせるであろう荷物すべてを取り除くと，後に残るのは（c）のように互いに動けなくなっている荷物だけとなる．最後に，残された荷物がゴールに入っているかを調べる．

6.3　ゴール周辺の壁

角に挟まれた壁には荷物を置いてはいけないが，そこにゴールが含まれている場合にはゴールの数までは置くことが許される（図16）．

実際のプログラムでは，3つの表（縦の壁，横の壁，L字およびコの字型の壁の表）を用意し，ゴールを含む壁には通し番号を付け，それらの表を基にして，その壁についていてもよい荷物の個数を監視する．

7．袋小路

倉庫番の数種類の死に手のうちで，袋小路は最も発見しづらい，厄介な死に手である．他の死に手は，それができた時点で比較的容易に判別できるのだが，袋小路は，一意に判別できず，その局面からかなりの手数を動かしてみないと判別できない場合があるからである．

† 村瀬はこの死に手を**シチョウ**と名付けている．

図 15　荷物による角の判別

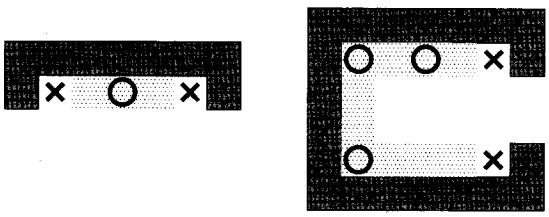


図 16　ゴール周辺の壁

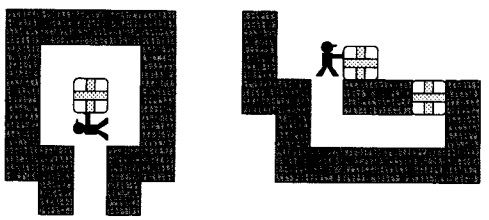


図 18　ほら穴の例

7.1　袋小路に関する用語

ここでは袋小路という用語は，本来の行き止まりとはやや違う意味で使っている．以下では，袋小路やそれに関連する用語について説明する．

- **閉領域**：ブロックと荷物によってできてしまった，番人がその中に到達できない閉ざされた領域である．図 17 の(a)，(b)，(c)が例である．
- **ほら穴**：閉領域の局面から回復できるものをほら穴と呼ぶ．図 17 の(b)と(c)がその例であり，回復後の局面が図 18 である．
- **袋小路**：閉領域の局面において，どのように荷物を動かしてみても閉領域のままであるものを袋小路と呼ぶ．たとえば図 17(a)において，さらに動かせる荷物は図 19(a)の A か B である（その他の 2 つは四つ組になってしまう）．A を動かしてみる（図 19(a)-2）と，今度はどの荷物を動かしてもすべて四つ組になってしまうので，事実上，もうこの局面から荷物を動かせないわけである．よって図 17(a)は袋小路だったわけである．

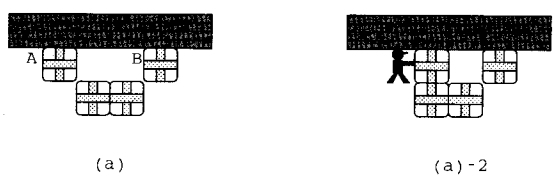


図 19　袋小路の例

以上をまとめると，閉領域は 2 種類に分かれ，回復可能なものがほら穴で，回復不可能なものが袋小路である．

- **栓**：閉領域とは，ある領域をいくつかの荷物がふさいでしまっている局面であるが，栓とはそのふさいでいる荷物のことをいう．
- **接する，隣接する**：ある荷物に接しているマスという表現の場合は，その荷物の上下左右の 4 つのマスのことを指す．荷物に隣接しているマスと表現した場合は周囲 8 マスのことをいう．つまり隣接の場合は斜め隣も加える．

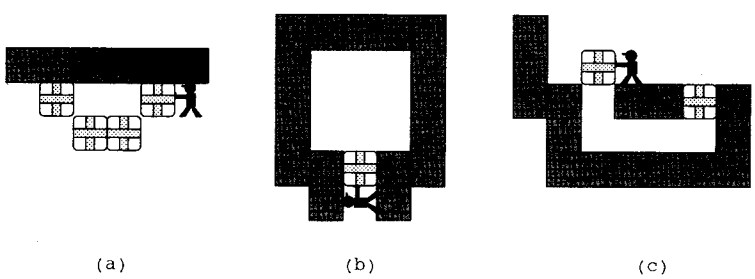


図 17　閉領域の例

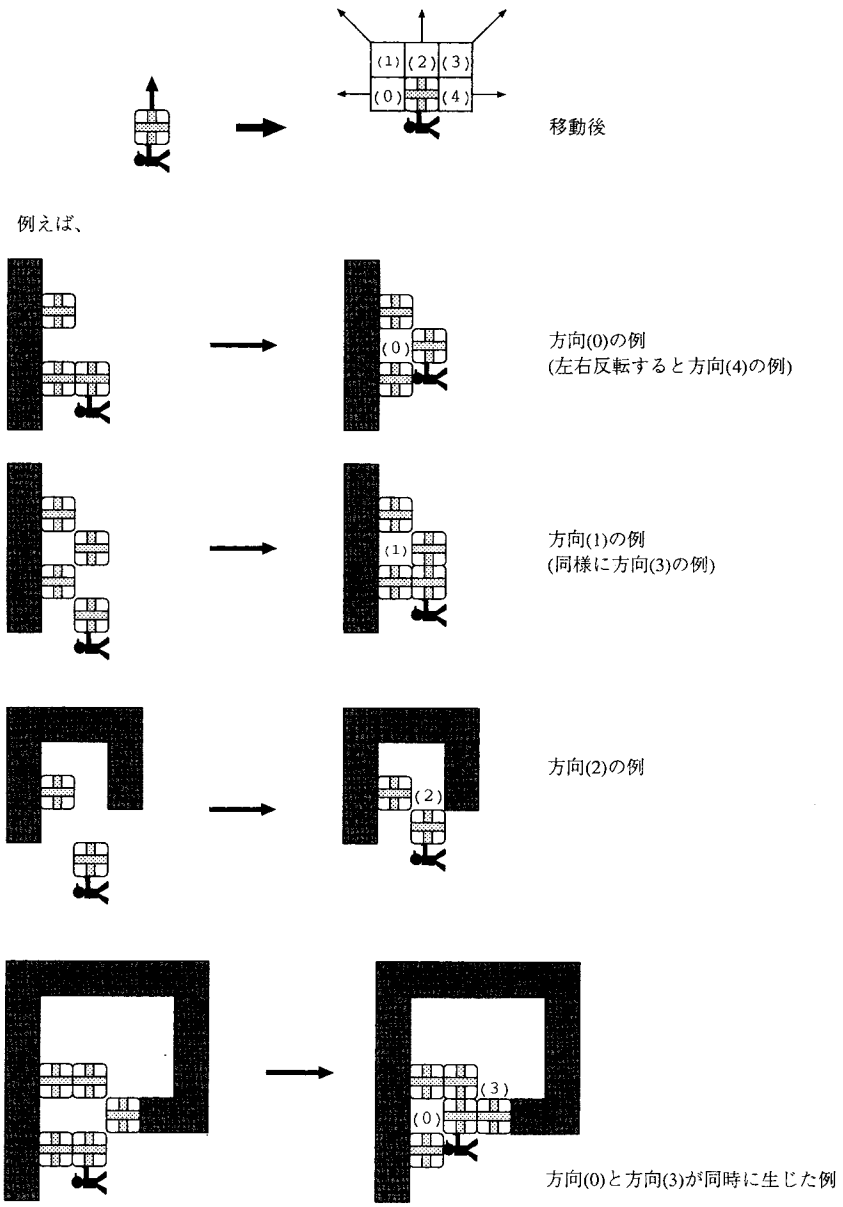


図20　閉領域のできる可能性のある5方向

7.2　袋小路の却下

荷物を動かして閉領域を作ってしまった時点で，その閉領域が袋小路かどうかを判別し，袋小路ならば死に手として却下する．

ある1つの荷物を動かすことによって，その荷物の前方5方向に閉領域ができる可能性がある（図20では，左，左上，上，右上，右の5方向）．なお，斜め方向（図20において方向(1)と(3)）は，荷物を動かす前から閉領域となっているが，これはその閉領域が形成されたときにほら穴であることが確認済みのはずである．

処理は最初に，上記5方向それぞれについて，閉領域ができているかどうかを調べる．まず，番人が到達できる範囲のマスに印をつける．動かした荷物に隣接している，指定された方向のマスを始点として，接するマスに対して次々に以下のような作業を行ない，閉領域を見つけ出す．

- そのマスに何も要素がなかったら（厳密にいうとSPACEかAVOID），閉領域専用の配列に，閉領域であるという意味で値SHUTを割り当て

る．そして次のマスのチェックに移る．

- そのマスに要素があれば，次のマスに移る．
- そのマスがCAN_MOVEであったら，番人が到達できるので，閉領域ではないとしてチェックを終了する．

これを再帰的に行なうことで，閉領域を検出することができる．閉領域となっているマスは専用の配列上に記録しておく．

次に，栓となっている荷物を探す．すでに閉領域のマスは調査済みなので，すべての荷物に対して，その荷物が閉領域のマスに隣接しているかどうかを調べる．もし隣接していたらその荷物を栓として記録しておく．

これらの処理の後，見つけた各閉領域について「閉領域回避探索」を行ない，ほら穴であるかそうでないかを判別する．この閉領域回避探索とは，マップ上の荷物のうちで栓になっているもの以外を取り除き，栓を通常の探索の場合と同様に何手か動かしてみて，袋小路になっているかどうかを調べる作業である．着目している閉領域が閉領域の状態から回避されたかどうか調べ，回避されていたら，ほら穴であったとして探索は終了する．栓として残された荷物を可能な限り動かしてみたが，どう動かしてみても閉領域の状態から回避できなかったら，着目している閉領域は袋小路だったということである．解の探索において，最も処理が重いのがここである．

この探索の手順は，通常の解の探索の場合とほとんど変わらない．しかし，通常の探索と閉領域回避探索と決定的に違うのは探索の目的である．通常の探索では荷物をゴールに運ぶことが目的であるが，閉領域回避探索では閉領域の局面からの回避が目的である．探索の終了条件は，閉領域である範囲に番人が到達できるようになることである．目的が異なるので，通常の探索のときのように評価値を割り当てることはしない．図21は閉領域回避探索を始めるときの様子を示している．斜線部が閉領域で，栓が2つある．この閉領域は，ほら穴となり回避できる．

閉領域回避探索で荷物を動かしたことによって，着目している閉領域とは別の閉領域ができてしまう場合がある．そのため，これまでに述べた処理を再帰的に行なう．これは興味深いことで，このゲームの奥深さを示していると言える．

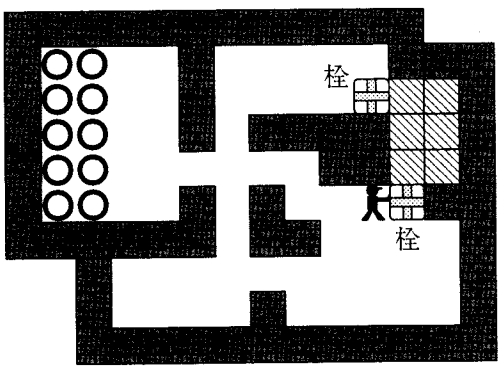


図21　閉領域回避探索

7.3　袋小路についての補足

ゴールの近くに閉領域ができたときは処理は異なる．閉領域の中にゴールがある，図22のような場合である．これはゴールに荷物を入れるときに当り前に生じる状況である．閉領域の中に1つでもゴールが含まれている場合は，閉領域に関する処理を行なわないことにしている．しかしこの場合，袋小路ができてしまっている可能性はある．

また，栓がすべてゴールの上に乗っている場合がある．図23は袋小路の形であるが，もちろんこの局面は却下してはいけない．この場合も同様に，閉領域に対する処理を行なわない．

閉領域回避探索においては，栓になっている荷物のみを残して回避探索を始めるのだが，実は栓以外の特定の荷物も残すべき場合がある．図24の局面は袋小路である．この局面から閉領域回避探索を行なうが，栓になっている荷物のみを残すと，荷物Aと荷物B

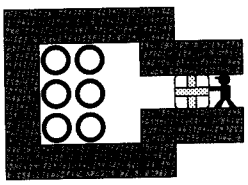


図22　閉領域の中にゴールがある場合

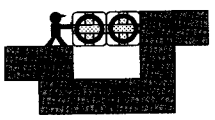


図23　栓がすべてゴールの上に乗っている場合

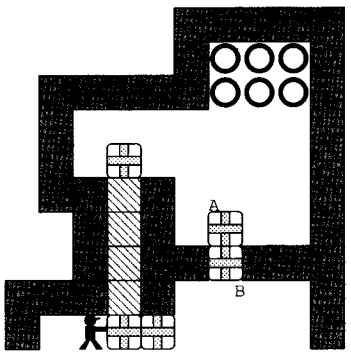


図24　栓以外の荷物を残すほうがよい場合

が取り除かれ，ほら穴とみなされてしまう．このように，栓になっている荷物のみを残すと，本来回避できない閉領域が閉領域回避探索においては回避できてしまい，袋小路として却下できなくなってしまうことがある．

以上のように，袋小路のチェックは，わずかだとは思われるが，見逃しを許してしまっている．つまり袋小路であるものを，そうでないとみなして，探索を続けてしまう．これは，現時点では袋小路を完全に判別できる方法が見つかっていないためである．この点の改善は非常に難しく，もしできれば，「倉庫番を解くプログラム」の性能はさらに向上すると思われる．

8. 実行結果

作成したプログラムに実際に問題を解かせてみた．実行はUNIXワークステーションSPARCstation 10（TOSHIBA　AS 4080/30 GX，86.5 MIPS）で行なった．

問題は，「倉庫番パーフェクト」（シンキングラビット社製：全306ステージ）を使用した．倉庫番パーフェクトのうち，荷物の少ないマップ（荷物数3〜13個）を順に解かせてみた（問題例は付録参照）．局面を記憶するバッファの容量は2Mバイトとした．合計133題を解かせたところ，

解けたもの　　77題
解けなかったもの　56題

という結果が得られた．荷物数ごとの結果は**表2**のとおりである．

付録に，解かせてみたマップの代表的ないくつかについて，展開された局面の数，生成された局面の数，総手数，番人の歩いた総歩数，実行に要した時間（秒）を示す．マップの番号は，倉庫番パーフェクトでのステージ番号である．どのマップも，解法にくせのあるものを取り上げてみた．なお，マップ28は荷物数が20個で，解けたマップ中でもっとも大きいものである．

表2　解けた問題の割合

荷物数	マップ数	解けた数	割合(%)
3	3	3	100
4	7	7	100
5	11	11	100
6	21	21	100
7	7	6	85.7
8	13	11	84.6
9	14	6	42.9
10	11	5	45.6
11	18	5	27.8
12	13	2	15.4
13	15	0	0

解けなかった問題の内訳は以下のようである．

1.局面領域オーバフロー　　44題
2.閉領域回避探索のための作業領域オーバフロー　　5題
3.解けないと判断して人間による中断（目安：1時間半以上）　　7題

局面領域オーバフローの場合は，30分前後経つと出始める．傾向としては，10分以上かかるものは解けそうにないと判断してよいようである．

解けなかった理由としては，袋小路の見逃しと，評価値の不完全さによるものと考えられる．本来，死に（袋小路）の状態となっているものに偶然高い評価値が与えられると，その局面を優先的に展開していくので，もっとも解に近いはずの局面の展開が後回しにされて，長時間にわたる探索の継続のために結果として記憶域のオーバフロー，あるいは人間によって中断されたものがほとんどだと思われる．

なお，本プログラムにおいては原則として，展開可能な局面集合が空になることはない．その状態が発生するのは，もともと解の存在しないマップを解かせようとした場合に限られている．

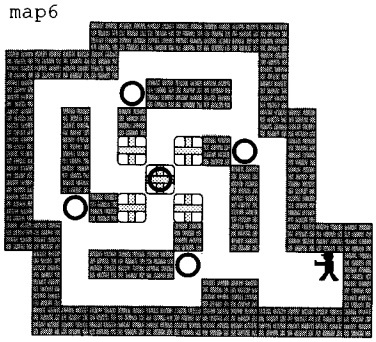
参考文献

1) Y. Murase, H. Matsubara, Y. Hiraga : Automatic making of sokoban problems, PRICAI'96 Topics in Artificial Intelligence, Lect. Notes in Art. Intelligence, No. 1114, pp. 592-600, 1996.
2) A. Myers : XSokoban, http://clef.lcs.mit.edu/~andru/xsokoban.html, 1995.
3) シンキングラビット社：倉庫番パーフェクト，1982, 1989.
4) 上野篤，中山康，疋田輝雄：「倉庫番」を解くプログラム（上）準備，解の探索，局面の評価，**bit**, Vol. 26, No. 12, pp. 40-51, 1994.
中山康，上野篤，疋田輝雄：「倉庫番」を解くプログラム（下）死に手，袋小路，解答例，**bit**, Vol. 27, No. 1, pp. 92-100, 1995.
5) P. H. Winston : *Artificial Intelligence (2nd ed.)*, Addison-Wesley, 1984.

付 録

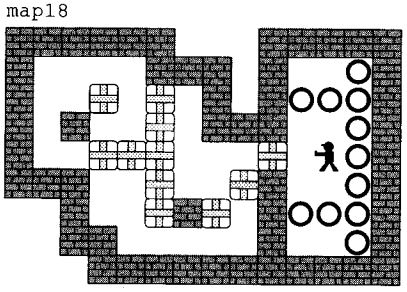
[問題例]

map6



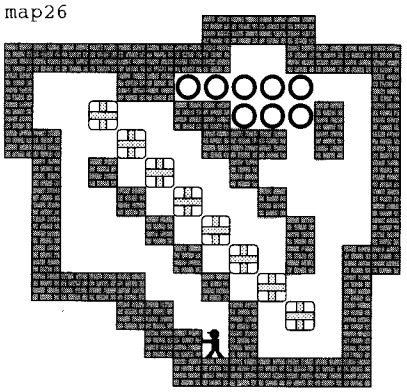
CLOSEの局面数 = 24
生成された局面数 = 154
総手数 = 22
総歩数 = 140
実行時間(秒) = 0.4

map18



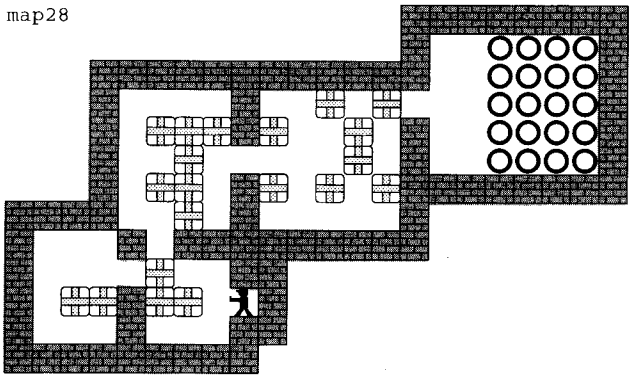
CLOSEの局面数 = 131
生成された局面数 = 534
総手数 = 122
総歩数 = 347
実行時間(秒) = 15.9

map26



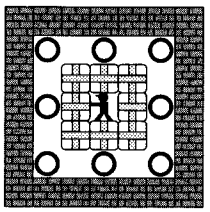
CLOSEの局面数 = 137
生成された局面数 = 648
総手数 = 136
総歩数 = 468
実行時間(秒) = 10.6

map28



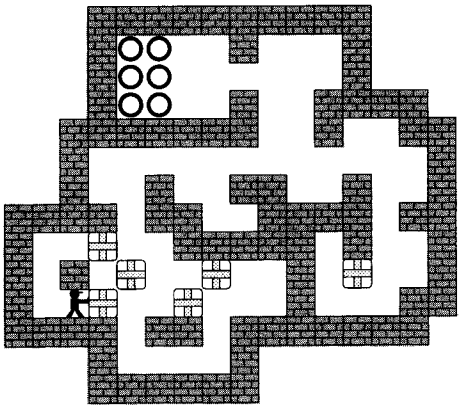
CLOSEの局面数 = 363
生成された局面数 = 2378
総手数 = 346
総歩数 = 901
実行時間(秒) = 9003.2

map126



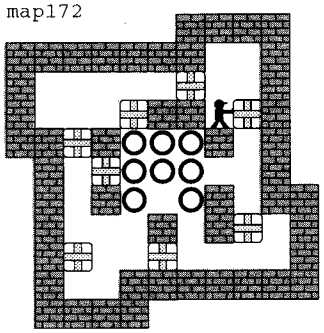
CLOSEの局面数 = 1470
生成された局面数 = 1604
総手数 = 24
総歩数 = 71
実行時間(秒) = 2.6

map165



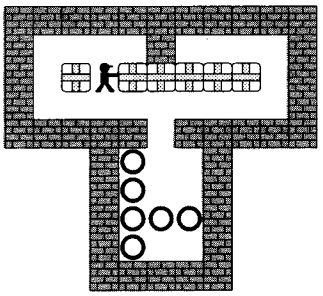
CLOSEの局面数　=　14760
生成された局面数　=　15559
総手数　=　217
総歩数　=　575
実行時間(秒)　=　528.8

map172



CLOSEの局面数　=　12342
生成された局面数　=　16250
総手数　=　133
総歩数　=　643
実行時間(秒)　=　2505.4

map294



CLOSEの局面数　=　79934
生成された局面数　=　96254
総手数　=　67
総歩数　=　238
実行時間(秒)　=　399.1