論理プログラムによるゲーム開発支援ツール GALOP の開発

川上 広太 †　　鍋島 英知 ‡　　岩沼 宏治 ‡
†山梨大学 大学院 医学工学総合教育部
‡山梨大学 大学院 医学工学総合研究部

1　研究目的

現在，非常に多くのゲームが作られている．そのゲームの多くは 3D ゲームなどグラフィクス技術の向上等によって，非常にリアル感あふれる物へと変化している．このことはゲーム開発者にとって大きな負担となっているが，データの管理などゲームの本質となる部分は大きくは変わっていない．また最近ではゲームが複雑になりすぎ，かなりの数のユーザが CG 系の大作から離れ，任天堂 DS のようなシンプルなゲームなどに移るなどの現象が起きている．面白いゲームを作っていく上でゲームの本質部をより良くしていきたい思う開発者にとって，ゲームの本質部分で如何に良いものを作るかが勝負であり，それを支援するツールの開発が望まれている．

そこでゲームの見た目は単純なものを使用し，ゲームの本質的な部分となるデータの管理等を論理プログラムによって記述する手法を提案する．手続き型の言語に比べて，宣言的な言語である論理プログラムはゲームのロジックを簡潔に記述することができるため，支援ツールとして有用であると思われる．

この手法によってゲームのプロトタイプを簡潔に作るためのツールを提供する．プロトタイプ開発の手間の軽減をすることによって，ゲームの開発者はより多くのプロトタイプを作成することができる．その結果，より面白いゲームの誕生を促進させることが本研究の目的である．

本論文ではまず論理プログラムについて延べ，次にこのツールの開発・実装について説明する．その後ツールを利用したゲームのプロトタイプ開発の実際の流れとその実行画面について説明し，最後に本研究のまとめと今後の課題について述べる．

2　論理プログラム

論理プログラムとは，その世界が持つ**事実**と**ルール**によって作られる宣言的プログラムのことである．「どのように」ではなく「何を」実行するのかを記述するプログラミングである．このプログラムは論理式の集合で与えられ，論理式の導出と呼ばれる推論によってプログラムの計算が行われる．プログラム中には使用される事実とルールのみが記述されており，計算の実行手順は記述されてはいない．つまり，一度宣言してしまえばあとは自動的に動作するのである．このようなプログラムは主に人工知能記述言語として使われている．

論理プログラムの例として，Prolog と呼ばれるプログラミング言語がある．Prolog は，物事の間に成り立つ関係を定義していくことでプログラミングしていく．そして，もっとも基本的な関係を表したものを事実 (fact) と呼び，Prolog のプログラムはこの事実関係を問い合わせることで動作する．すなわち、Prolog は「定義された事実を参照し，質問の答えを導き出す言語」と言える．しかし，ゲームの論理プログラム化においては，質問に対する答えだけではなく，論理プログラムが満たす全ての事実を考慮したい．そこで本研究では Prolog ではなく， smodels という言語を使用した．

2.1　smodels [Niemela 98]

smodels は Prolog によく似た言語ではあるが，答えの返し方が大きく異なる．Prolog は先に述べたとおり，質問に対する個別の答えを返す言語であるのに対して，smodels はハミルトン閉路問題やプランニング問題といった，ある世界の事実と全ての規則から充足される極小のモデル，**安定モデル**を導き出す言語である．

2.2　安定モデル　[Gelfond 88]

ここで論理プログラムから導出される安定モデルの定義を説明する．まず論理プログラム P が以下のようであったとする．

p ← not q,r
q ← not p
r ← not s
s ← not p

このとき，← の右辺をボディ部，左辺をヘッド部と呼ぶ．ボディ部とヘッド部を合わせてルールと呼び，各要素をリテラルと呼ぶ．

ここで原子文集合 S = {r, p} を仮定する．この原子文集合とプログラムに対して以下の削除操作を行う．

(1) ボディ部 に S 中のアトム a の否定リテラル not a をもつ各ルールの削除
(2) 残りのルール中全ての否定リテラルの削除

まず (1) の削除を行うと，body に not p を含む 2 つのルールがプログラム P から削除される．

p ← not q,r
r ← not s

図 1: 積木世界　前

図 2: 積木世界　後

更に (2) の削除を行う．S に含まれない q と s の否定リテラル not q，not s の 2 つのリテラルがプログラム P から削除される．

$$p \leftarrow r$$
$$r$$

原子文集合 S が，S に関するルールの削除を行ったプログラム P の，唯一の極小モデルであった時，そのときに限り S は安定モデルであるという．よって原子文集合 S={r, p} はプログラム P の安定モデルであるといえる．

本研究ではこの安定モデルがそのゲーム世界の**状態**を表していると考えて利用する．

2.3 状態

論理プログラムでゲームを作る場合，**状態**というものを考慮に入れなければならない．この状態とは，ある特定時刻における世界のスナップショットである．状態が異なれば，世界は異なるのである．状態の概念を考慮に入れない場合，世界は変化せず，事実は最初から最後まで真，または最初から最後まで偽として扱われる．しかし，ゲームの世界は常に変化をし続ける世界である．直前の状態をうけて，新しい事実とともに新しい状態が生まれるのである．

2.4 フレーム問題

フレーム問題は状態とその時間変化を考慮したときに考えなければならない問題である．アニメーション作成では一つのシーンを生成する際に，たいていのアニメータはまずいくつかのシーンを通して変化しない背景を描き，その後それを複製し前景の動きを重ねる．フレーム問題とは，行動によって変化しない背景を前景から区別することである．

例えば図 1 のような状態の積木世界があったとする．今，図 2 のように積木 B を積木 C の上に移動させたとする．このとき人間の目から見れば積木 A は動いていないことがすぐわかる．しかしコンピュータは積み木 B が積み木 C の上に動いたときに積み木 A がどうなっているか理解できず，無用の類推をしてしまうのである．この問題を解決するためには，移動しなかった積木は位置を変えないというルールが必要である．つまり状態の概念を考慮する論理プログラムを書く場合，ある事実は次の状態にあっても変化しないということを指示する**フレーム公理**が必要となる．

表 1: 開発環境

項目名	詳細
マシンスペック	CPU: Intel Pentium 4 3.2GHz メモリ容量:2GB
OS	Turbolinux Workstation 8.0
記述言語	JAVA ver1.5.0
必要モジュール	lparse smodels

3 開発支援ツール GALOP

本研究で開発したゲーム開発支援ツールを，論理プログラムに基づくゲーム開発環境 **GALOP**(**GA**me development environment based on **LO**gic **P**rogramming) と命名する．GALOP は論理プログラムによってゲームのプロトタイプを開発するために使用することを目的とする．

本研究では GALOP を，MVC アーキテクチャの流れに沿って，JAVA 言語を用い実装・開発した．論理プログラムから安定モデルを導出する際に smodels を使用するが，JAVA を用いた場合 smodels を簡単に呼び出すことができ，また描画を行っていくうえで JAVA は多くのクラスが用意されているため，実装が容易であると判断したためである．

GALOP の開発環境を表 1 に示す．

4 MVC 「Model-View-Controller」

MVC アーキテクチャとはソフトウェア設計モデルの 1 つである．処理の中核を担う「Model」，表示・出力を司る「View」，入力を受け取りその内容に応じて Model を制御する「Controller」の 3 要素の組み合わせでシステムを実装する方式である．

メインの処理は Model に実装し，Model は画面出力などは行わない．処理結果は View に渡され，画面表示などが行われる．ユーザからの入力は Controller が受け取り，何らかの処理が必要な場合は Model に依頼し，その結果 Model がまた View に処理結果を渡すように作用する（図 3）．

このように，各機能を明確に分離することで開発作業の分業が容易になり，また，互いの仕様変更に影響を受けにくくなる利点がある．

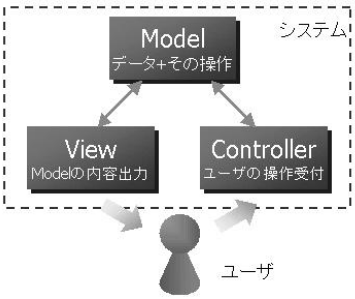


図 3: MVC アーキテクチャ

図 4: 爆破スイッチ

4.1 GALOP における MVC

GALOP における MVC の役割を，例として，爆破スイッチゲーム「ボタンが存在しそのボタンをクリックしたのであれば爆破がおきる」(図 4)，という単純なゲームの事例を踏まえて説明する．

4.2 GALOP の Model

GALOP の Model 部，つまり開発するゲーム本体は論理プログラムによって記述する．ツールユーザが実際にゲームのプロトタイプを作るうえで開発する箇所はこの Model 部の論理プログラムのみである．GALOP の Model モジュールは，ツールユーザによって作られた論理プログラムから安定モデルを導出し，それを View モジュールに送信することが主な役割となる．

爆破スイッチの例を論理プログラムにすると以下のように記述できる．

```
button(0).
blast(T+1) :- button(T), click(T), time(T).
button(T+1):-button(T), time(T).
draw(button.gif):- button(T+1), time(T).
draw(blast.gif):- blast(T+1), time(T).
```

ここで 1 行目は　button(0) は初期時間 0 においてボタンが存在するということを示す．2 行目はある時間 T 中に存在するボタンをその時間にクリックした時，爆破がおきるということを示している．3 行目はある時間にボタンが存在した場合，次の時間においてもボタンは存在するということ，つまりフレーム公理である．

さらに現在の時間をゲームの状態としてこのプログラムに付加する．時間は Model が更新するたびに 1 つずつ進んでいく．初期値は

```
time(0).
```

である．

Model 部ではツールユーザが作成した論理プログラムから安定モデルを導出する．このプログラムの場合，次のような安定モデルが出力される．

```
button(0)
button(1)
time(0)
draw(button.gif)
```

これはつまり，時間 0 でボタンが存在し，時間 1 にボタンが存在する．そのときの時間は時間 0 であり，ボタンを描画するという意味になる．導出された安定モデルは View に送られる．View での動作は次項に詳しく記述するが，ここではボタンが描画されることになる．

ゲームの状態は時間の経過とともに更新されるが，その状態は安定モデルより抜き出される．安定モデルの中で，時間が新しい状態になっている項を取り出し，新しい状態として更新する．例の場合，新しい時間と新しいボタンの項である

```
time(1).
button(1).
```

の 2 つが新しい状態として更新される．

Contorller からユーザがクリックしたという情報が現在の時間に送られてきた場合，ゲームの状態はユーザの動作を含んだ物として更新される．

```
time(1).
button(1).
click(1).
```

更新されたゲームの状態と，元の論理プログラムをからまた新しい安定モデルを導出する．

```
button(1)
button(2)
time(1)
blast(2)
draw(button.gif)
draw(blast.gif)
```

これらの安定モデルがまた View に送られ，View ではボタンと爆破の描画が行われる．

4.3 GALOP の View

GALOP の View モジュールは Model から安定モデルを受け取る．受け取った安定モデルの中に描画用の項が存在した場合，その項の内容にあわせて描画を行う．例の場合

```
draw(button.gif)
```

が存在するため，button.gif というボタンの画像を描画する．

今回の例では，プログラムを単純に説明するために，描画用の項を簡易に表していた．実際には描画用の項には画像，文字，数字の描画の区別と描画位置を表現する必要があるため，次のような項を用意した．

drawImage(Imagefile, 座標 x, 座標 y).
drawString(String, 座標 x, 座標 y).
drawInt(Int, 座標 x, 座標 y).

また，ユニットの大きさを一律に決めておき，そのマップの位置座標をとるようにしたい場合，ユニットのサイズを指定可能である．

setUnitSize(Int).

4.4 GALOP の Controller

GALOP の Controller モジュールはゲームの操作部を担当する．ユーザのクリックやキーボード操作を受け取り，Model にその情報を送っている．

click1(座標 X, 座標 Y, 時間 T). : 左クリック
click3(座標 X, 座標 Y, 時間 T). : 右クリック
key_up(時間 T). ： ↑キー
key_down(時間 T).： ↓キー
key_right(時間 T).： →キー
key_left(時間 T). ： ←キー

5 GALOP によるゲーム開発

本研究で開発した GALOP によってサンプルゲームを作成した．本研究では，ユーザの操作入力があった時のみ時間が進むタイプのゲームとして**倉庫番**を作成した．

5.1 倉庫番

倉庫番は 1982 年 12 月に有限会社シンキングラビットから発売されたコンピュータパズルゲームである．

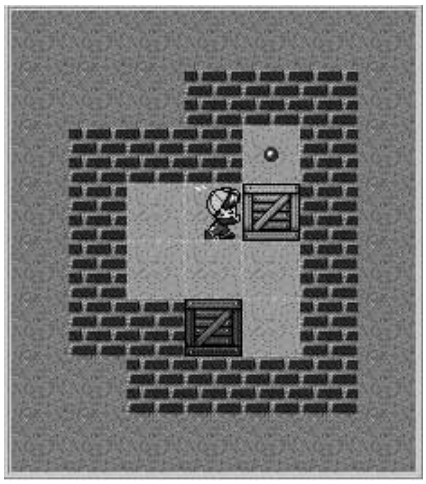
倉庫番のルールは以下の通りである．

・全ての荷物を指定位置に移動させると終了
・荷物を 押すことはできるが引くことはできない
・同時に 1 つの荷物しか押すことはできない

ルールとしてはとても単純である．しかし，この簡潔なルールにゲームの奥深さが隠されている．これらのルールを踏まえて論理プログラムによる倉庫番を作成する．

5.2 倉庫番の論理プログラム

倉庫番の論理プログラムの書き方を順を追って説明する．ツールユーザはゲームの論理プログラムを作成する上で次のことを考えていくとよい．



sokoban®
©1982-2005 HIROYUKI IMABAYASHI
©1989,1990,2001-2005 FALCON CO.,LTD.
ALL RIGHTS RESERVED.

図 5: 倉庫番

5.2.1 初期状態

開発するゲームの論理プログラムではまず最初に，そのゲームの初期状態を記述するべきである．倉庫番では，マップの大きさ，壁の位置，荷物箱の位置，プレイヤーの位置，箱を届ける位置が必要となる．このときプレイヤーと箱の位置は変化する可能性がある．そのため時間変数をプレイヤーと箱の項に組み込み，状態の変化に対応させる．

以下が作成した倉庫番の初期状態である．

```
world(0..7,0..7).
map(1..6,1..6).
wall(X,Y) :- world(X,Y), not map(X,Y).
wall(1,3). wall(3,3).
wall(4,3). wall(6,3).
player(4,6,0).
hako(2,5,0). hako(3,5,0).
hako(4,5,0). hako(5,5,0).
flag(3,1). flag(3,2).
flag(4,1). flag(4,2).
```

5.2.2 終了条件

次にゲームの終了条件を考える．倉庫番のゲームクリア条件はすべての箱が届けるべき位置にある，つまり同じ座標上であるときである．

```
goal(T+1):-
        hako(3,1,T+1), hako(3,2,T+1),
        hako(4,1,T+1), hako(4,2,T+1),
        flag(3,1), flag(3,2), flag(4,1), flag(4,2),
        time(T).
```

5.2.3 ユーザ入力によるルール

次にユーザの入力による各ルールを考える必要がある．倉庫番ではプレイヤーの動き方と箱の動き方を考える必要がある．

まずプレイヤーの動きであるが，ユーザはキーボードの上下左右キーを押したとき，プレイヤーをその方向に動かしたいと考える．よってキーボードの入力を移動するという述語で捉える．

```
move(0,-1,T):-key_up(T), time(T).
move(0,1,T):-key_down(T), time(T).
move(-1,0,T):-key_left(T), time(T).
move(1,0,T):-key_right(T), time(T).
moving(T):-
        1{move(1,0,T),move(-1,0,T),
        move(0,1,T),move(0,-1,T)}1,time(T).
```

次はプレイヤーの動きである．まずプレイヤーは，移動したいポイントに箱も壁も無いマップならば自由に，移動することができる．

```
player(PX+DX,PY+DY,T+1):-
        map(PX,PY), player(PX,PY,T),
        move(DX,DY,T),
        not wall(PX+DX,PY+DY),
        not hako(PX+DX,PY+DY,T),time(T).
```

次は箱があった場合を考える．プレイヤーの移動先に箱があったとき，その箱の更に先が箱でも壁でも無かった場合，プレイヤーはその箱のあった位置に動くことが出来る・

```
player(PX+DX,PY+DY,T+1):-
        map(PX,PY), player(PX,PY,T),
        move(DX,DY,T),
        not wall(PX+DX,PY+DY),
        hako(PX+DX,PY+DY,T),
        not wall(PX+DX+DX,PY+DY+DY),
        not hako(PX+DX+DX,PY+DY+DY,T),
        time(T).
```

また箱の動きも考える必要がある．これは箱があったときのプレイヤーが動く条件とほぼ同じである．押される箱の先が箱でも壁でも無ければ箱は動くと考える．

```
hako(PX+DX+DX,PY+DY+DY,T+1):-
        map(PX,PY), player(PX,PY,T),
        move(DX,DY,T),
        hako(PX+DX,PY+DY,T),
        not wall(PX+DX+DX,PY+DY+DY),
        not hako(PX+DX+DX,PY+DY+DY,T),
        time(T).
```

5.2.4 フレーム公理

前章で述べたフレーム問題についても考える必要がある．「変化しない」というフレーム公理を記述するため，各述語が倉庫番上で変化しない状況を考える．

まず上下左右キーの操作が行われず移動が起きない場合，例えばクリック動作などが行われて状態が更新した場合である．この場合，プレイヤーも箱も位置は変化せず，元の状態の座標の位置にいることになる．

```
player(PX,PY,T+1):-
        map(PX,PY), player(PX,PY,T),
        not moving(T), time(T).
hako(HX,HY,T+1):-
        map(HX,HY), hako(HX,HY,T),
        not moving(T), time(T).
```

次はプレイヤーの移動先が壁だった場合，プレイヤーはその壁には動けずその位置に変わらずいる場合である．また箱の移動先に箱があり，その箱の先に壁，または別の箱があったときもプレイヤーはその場の位置から動けず位置は変化しない．

```
player(PX,PY,T+1):-
        map(PX,PY), player(PX,PY,T),
        move(DX,DY,T), wall(PX+DX,PY+DY).
player(PX,PY,T+1):-
        wall(PX+DX+DX,PY+DY+DY),
        map(PX,PY), player(PX,PY,T),
        hako(PX+DX,PY+DY,T),
        move(DX,DY,T), time(T).
player(PX,PY,T+1):-
        hako(PX+DX+DX,PY+DY+DY,T),
        map(PX,PY), player(PX,PY,T),
        hako(PX+DX,PY+DY,T),
        move(DX,DY,T), time(T).
```

同じように箱についても考える．プレイヤーに押されるであろう箱の押される先が壁，または箱であったときは，その箱はその場の位置から動かない．

```
hako(HX,HY,T+1):-
        map(HX,HY), hako(HX,HY,T),
        move(DX,DY,T),
        wall(HX+DX,HY+DY), time(T).
hako(HX,HY,T+1):-
        map(HX,HY), hako(HX,HY,T),
        move(DX,DY,T),
        hako(HX+DX,HY+DY,T), time(T).
hako(HX,HY,T+1):-
        1{HX!=PX+DX,HY!=PY+DY}2,
        map(HX,HY), hako(HX,HY,T),
        map(PX,PY), player(PX,PY,T),
        move(DX,DY,T),time(T).
```

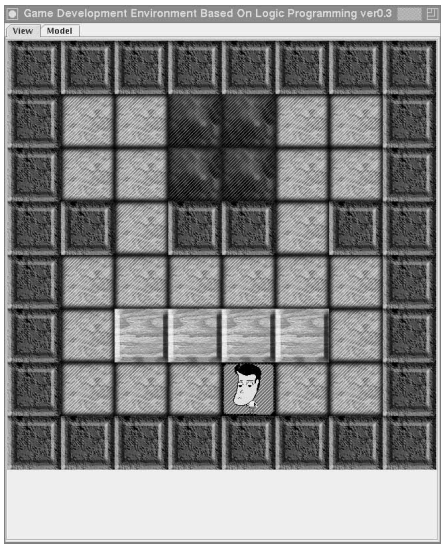


図 6: 倉庫番開始時の画面

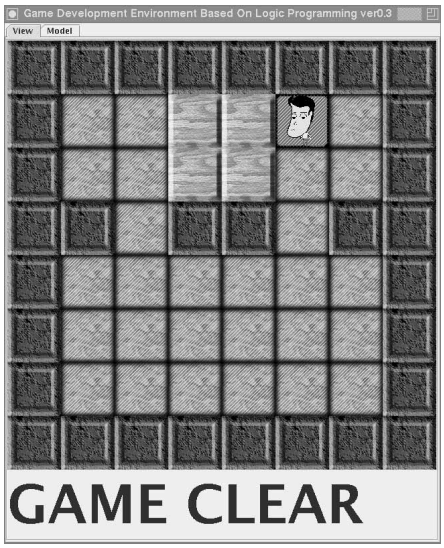


図 7: 倉庫番クリア時の画面

5.2.5 描画用

最後に，GALOP では安定モデルに描画用の項が無ければ描画がされないため，このゲームから導出される事実を描画用の述語に直す必要がある．

```
drawImage(”map2.gif”,X,Y) :-
        map(X,Y),not wall(X,Y),
        not flag(X,Y),
        not player(X,Y,T+1),
        not     hako(X,Y,T+1),time(T).
drawImage(”wall2.gif”,X,Y) :- wall(X,Y).
drawImage(”goal.gif”,X,Y) :- flag(X,Y).
drawImage(”hako.gif”,X,Y) :-
        map(X,Y),hako(X,Y,T+1),time(T).
drawImage(”player.gif”,X,Y) :-
        map(X,Y), player(X,Y,T+1), time(T).
drawString(”GAME CLEAR”,0,8):-
        goal(T+1), time(T).
setUnitSize(80).
```

以上が本研究で作成した倉庫番の論理プログラムである．

5.3 GALOP による倉庫番

前節の倉庫番の論理プログラムを全て合わせ，GALOP により実行した結果は図 6,7 である．

キーボードの上下左右キーの操作に対応し，単純なグラフィックで描画された倉庫番が GALOP を用いて完成できたことをここに示す．

5.4 その他のゲーム

倉庫番以外のゲームとしてマインスイーパーも作成した．マインスイーパーのルールの概略は次の通りである．

・地雷をクリックしたらゲームオーバーする．
・地雷の無い部分をクリックするとその周りの地雷の数が表示される．
・すべての地雷を発見（地雷以外の場所をすべてクリック）できたらゲームクリア．

このルールに沿った論理プログラムのソースは付録に載せる．実行結果は付録の図 8 のようになり，こちらのゲームも完成できたことを示す．

6 まとめ

本研究で，論理プログラムによるゲーム開発支援ツール，GALOP の開発・実装を行い，サンプルゲーム倉庫番を作成することでそのツールの実用性を示した．

今後の課題として，現在の GALOP はユーザの入力があったときのみ Model が更新されていくが，このままでは常に状態が変化していくゲーム，例えばテトリスやシューティングというゲームを開発することが出来ないため，タイマー処理などを追加しこの問題の解決をはかる必要がある．また，使用する言語の検討も考えられる．Smodels 以外にゲーム開発に適した論理言語はないのか，または自分でゲーム開発用論理言語を開発するということも考えていく必要がある．

参考文献

[1] M. Gelfond and V. Lifschitz. The stable model semantic for logic programming. In *Proceeding of the 5th International Conference on Logic Programming*, pages 1070-1080, Seattle, USA, August 1988. The MIT Press.

[2] Ilkka Niemela. Logic Program with Stable Model Semantics as a Constraint Programming Pradigm.Workshop on Computational Aspects of Nonmonotonic Reasoning,May 1998.

A　付録　マインスイーパーのソースと実行図

本システムのプログラムのソースとその実行図 8 を示す.

```
map(1..5,1..5).
world(0..6,0..6).
wall(X,Y):-world(X,Y),not map(X,Y).
num(0..8).
bomb(1,2). bomb(2,3). bomb(3,1). bomb(4,1). bomb(5,5).

game_over(T+1):- bomb(X,Y), not flag(X,Y,T),click1(X,Y,T),time(T).
goal(T+1) :-
    bomb(1,2),bomb(2,3),bomb(3,1),bomb(4,1),bomb(5,5),
    flag(1,2,T+1),flag(2,3,T+1),flag(3,1,T+1),flag(4,1,T+1),flag(5,5,T+1),time(T).

covered(X,Y,0):-map(X,Y),time(0).
covered(X,Y,T+1):-map(X,Y),flag(X,Y,T),click3(X,Y,T),time(T).
opened(X,Y,T+1) :- map(X,Y),covered(X,Y,T), click1(X,Y,T),time(T).
flag(X,Y,T+1) :- map(X,Y),covered(X,Y,T),click3(X,Y,T),time(T).

numBombs(X,Y,Z,T+1):-
  Z { bomb(X-1,Y-1), bomb(X-1,Y), bomb(X-1,Y+1), bomb(X,Y-1),
      bomb(X,Y+1), bomb(X+1,Y-1), bomb(X+1,Y), bomb(X+1,Y+1) } Z,
  opened(X,Y,T+1), num(Z),time(T).

opened(X,Y,T+1) :- map(X,Y),opened(X,Y,T),time(T).
covered(X,Y,T+1):-map(X,Y),covered(X,Y,T),not click1(X,Y,T),not click3(X,Y,T),time(T).
flag(X,Y,T+1) :- map(X,Y),flag(X,Y,T),not click3(X,Y,T),time(T).
game_over(T+1):-game_over(T),time(T).

numBombs(X,Y,Z,T+1):- numBombs(X,Y,Z,T),map(X,Y),num(Z),time(T).

drawImage("block.gif",X,Y) :- covered(X,Y,T+1),time(T).
drawImage("mine.gif",X,Y) :- bomb(X,Y),game_over(T+1),time(T).
drawImage("wall.gif",X,Y) :- wall(X,Y).
drawImage("flag.gif",X,Y) :- flag(X,Y,T+1),time(T).
drawInt(Z,X,Y):- opened(X,Y,T+1), num(Z), not bomb(X,Y), numBombs(X,Y,Z,T+1),time(T).
drawString("Game Over",0,7):- game_over(T+1),time(T).
drawString("Game Clear",0,7):- goal(T+1),time(T).
setUnitSize(50).
```

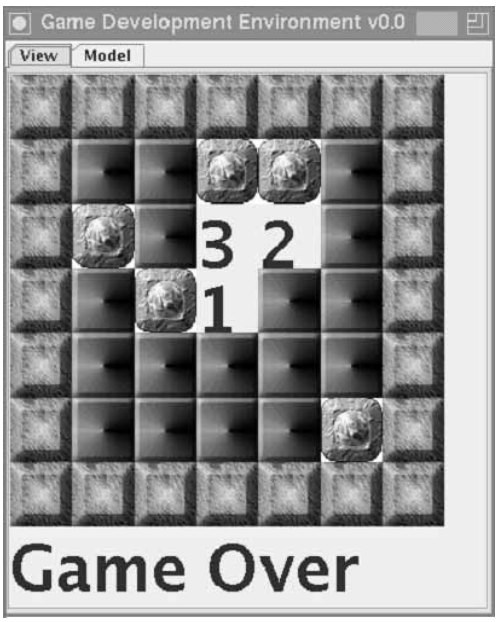


図 8: マインスイーパー