ADVANCED JASON

BE4M36MAS - Multiagent systems

LAST TUTORIAL ISSUES

Malfunctioning Jason

Hopefully resolved (if not, tell me about that!)

Ordering of plans

- Source file is scanned top down
- **First** applicable plan is executed

```
+!step <- !random_move ; !step.
+!step :  cell(X,Y,gold) <- +gold(X,Y) ; !step.   ← unreachable

+!step :  cell(X,Y,gold) <- +gold(X,Y) ; !step.   ← "infinite loop"
+!step <- !random_move ; !step.
```

Ordering of plans

- Source file is scanned top down
- **First** applicable plan is executed

```
+!step <- !random_move ; !step.
+!step :  cell(X,Y,gold) <- +gold(X,Y) ; !step.   ← unreachable

+!step :  cell(X,Y,gold) <- +gold(X,Y) ; !step.   ← "infinite loop"
+!step <- !random_move ; !step.
```

Ordering of plans

- Source file is scanned top down
- **First** applicable plan is executed

```
+!step <- !random_move ; !step.
+!step :  cell(X,Y,gold) <- +gold(X,Y) ; !step.   ← unreachable

+!step :  cell(X,Y,gold) <- +gold(X,Y) ; !step.   ← "infinite loop"
+!step <- !random_move ; !step.
```

ASSIGNMENT

Mining world

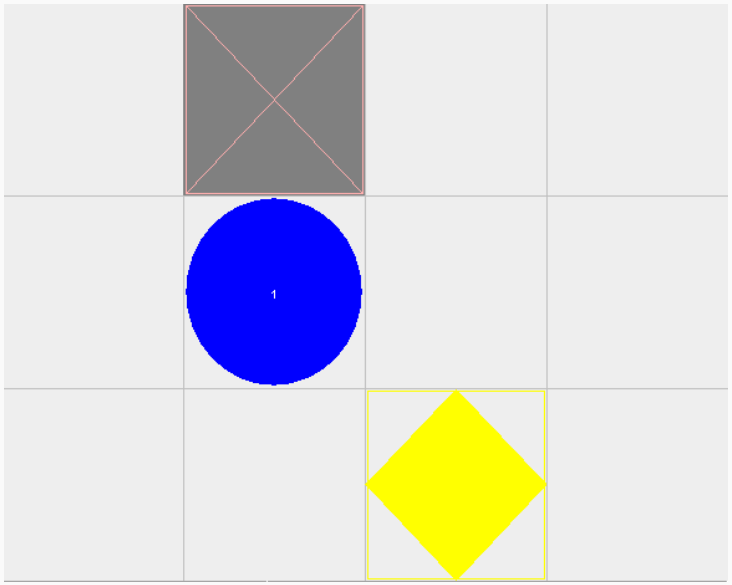
Find, collect and carry all gold stones from their location to a depot!

- Miners do not know positions of gold stones and depots — they must find them
- They may carry at most one gold stone at a time
- They have limited range of sight (8-neighbourhood)

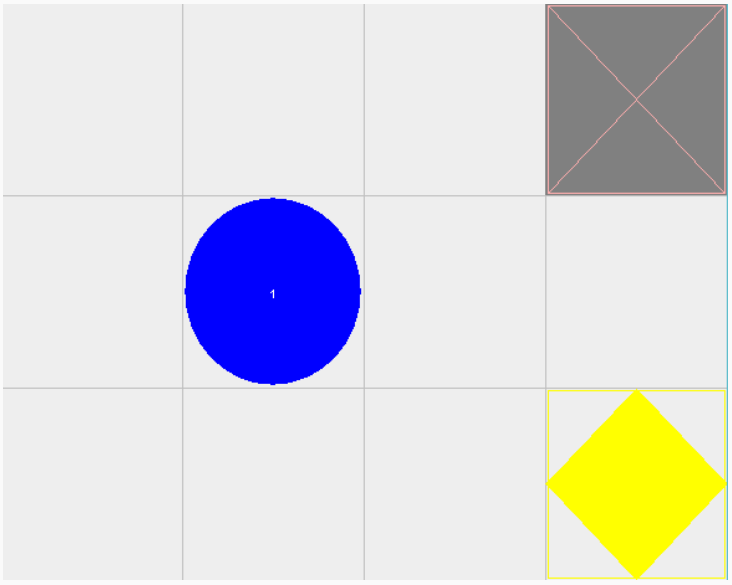
Mining world — percepts

- `pos(X,Y)` — (X, Y) position of the miner
- `name(N)` — name of the current miner
- `gsize(_,W,H)` — width and height of current map
- `cell(X,Y,gold)`, `cell(X,Y,depot)`, `cell(X,Y,ally)`, `cell(X,Y,obstacle)`
- `carrying_gold`

Mining world — percepts



```
cell(2,2,gold).
cell(1,0,depot).
```



No `cell` percepts!

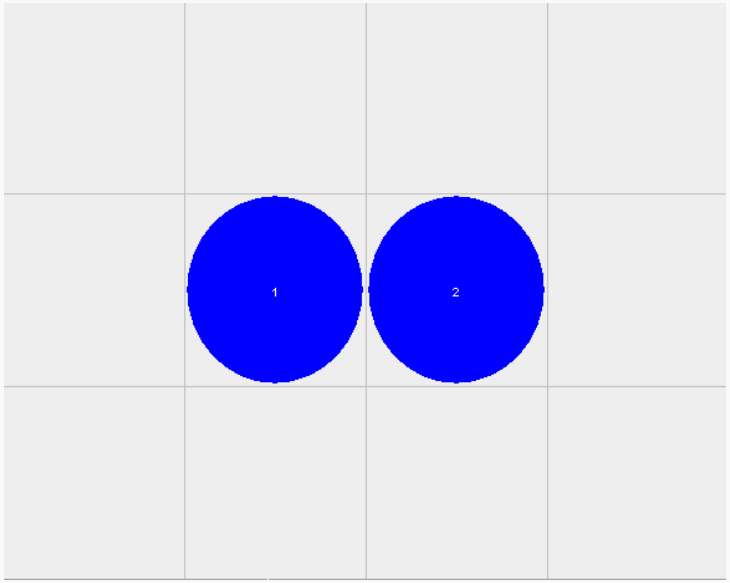
Mining world — actions

- `do(left)`, `do(right)`, `do(up)`, `do(down)` — movement in the grid
- `do(pick)`, `do(drop)` — manipulating gold stones
- `do(skip)` — use it to update your percepts (nearly no delay)

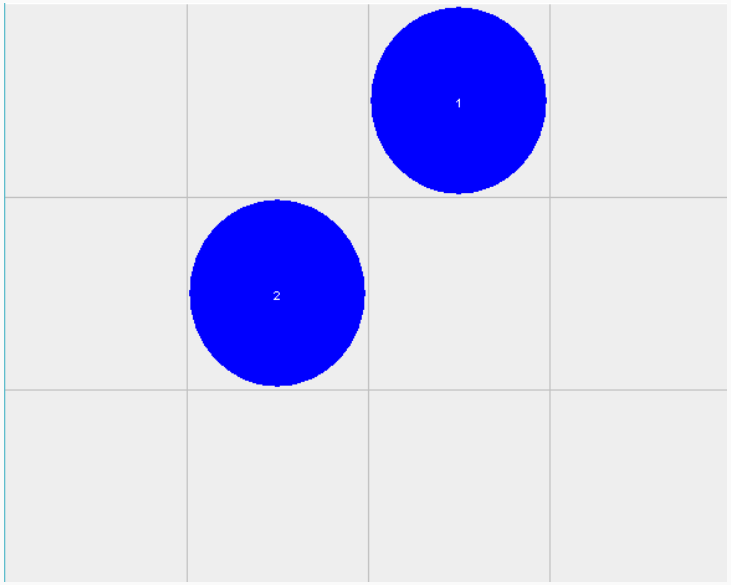
Mining world — Problem 1

Gold stones are **heavy**.

$\rightarrow$ there must be another miner in 4-neighbourhood for `do(pick)`



`do(pick)` succeeds



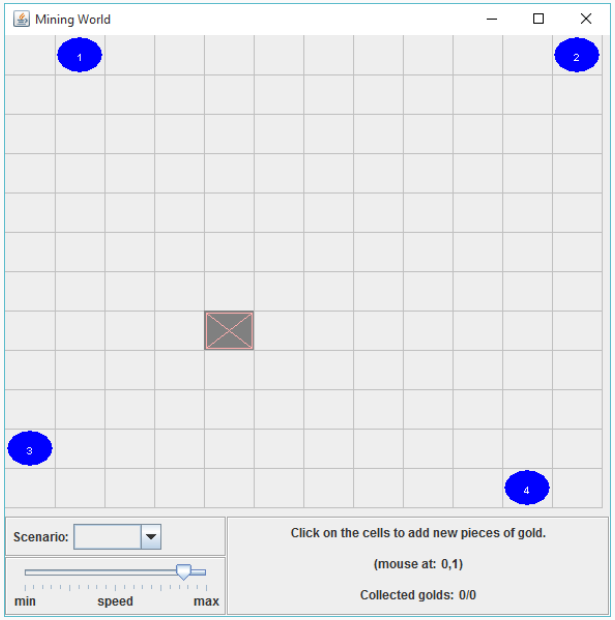
`do(pick)` fails

Mining world — Scenario 1

Gold stones are **added in runtime**

$\rightarrow$ Your miners must be able to find them at any time

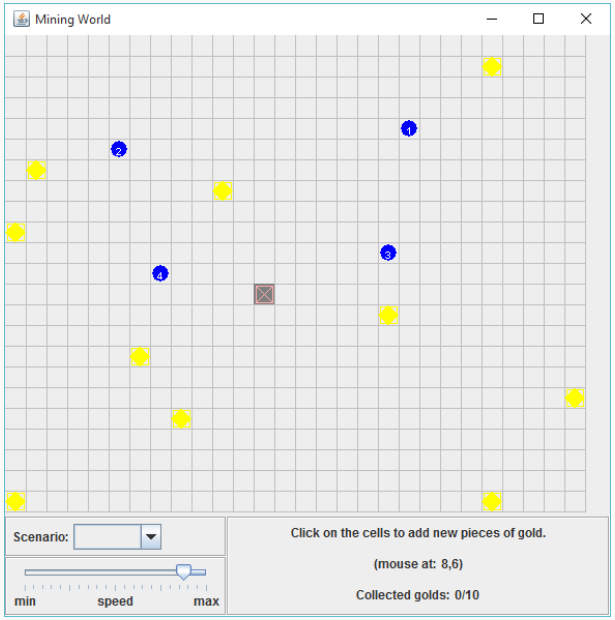
2 points



Mining world — Scenario 2

- You are racing the **time** now
- Your miners should not be much slower than (inefficient) reference solution

2 points

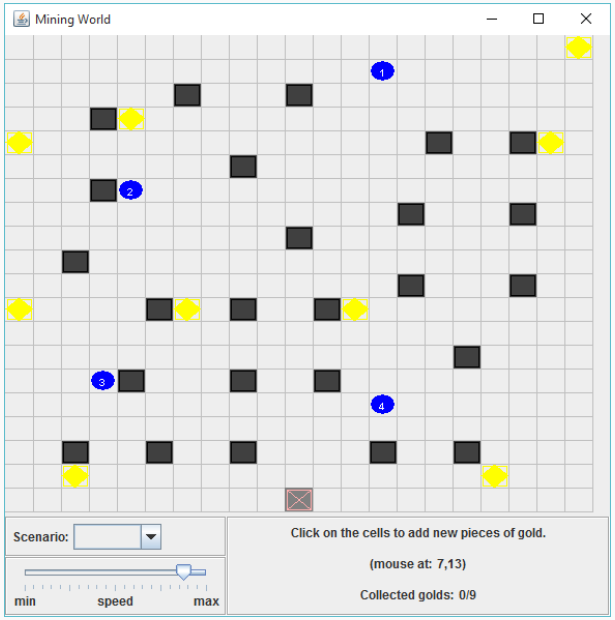


Mining world — Scenario 3

Beware of **obstacles**

→ Your team should make the way through the mine in **time** again

2 points

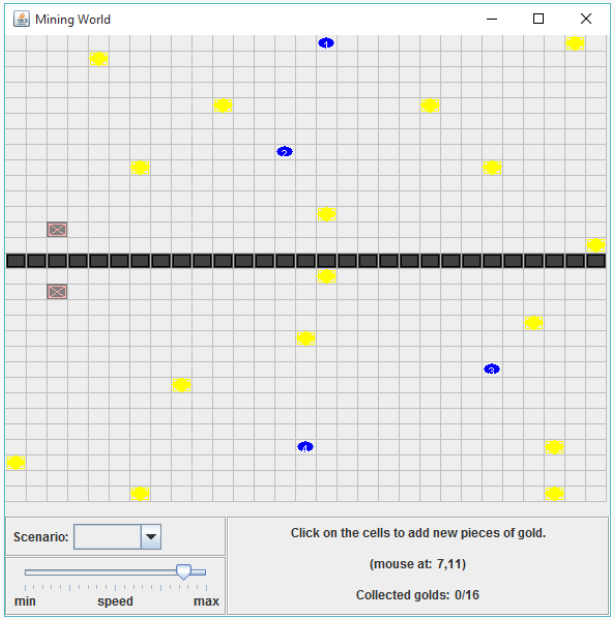


Mining world — Scenario 4

Pairs of your miners got **separated**

→ Hardcoded pairs helper–carrier will get into troubles

2 points

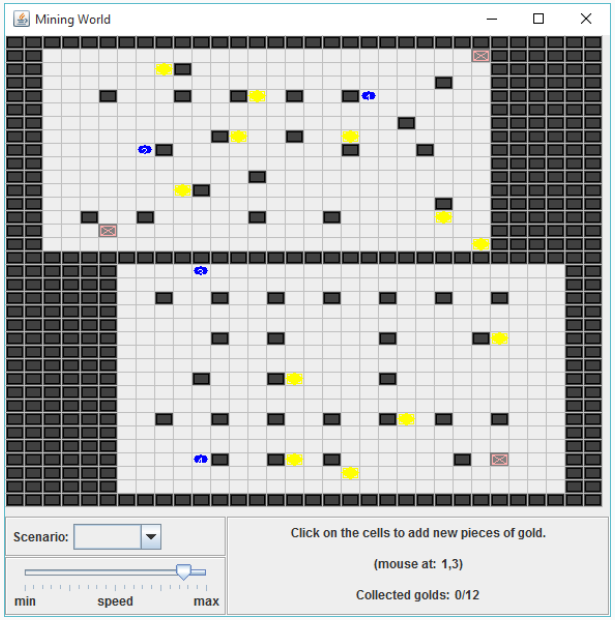


Mining world — Scenario 5

The final blow, is it?

(there might be multiple depots)

1 point



Mining world — Competition

You can get **1 more point** for implementing a **fast** mining team.

A competition between your submissions will be held

$\rightarrow$ Results from multiple runs on **Scenarios 3–5** will be averaged

(Average of values containing ∞ is infinite)

Mining world — Evaluation

Mines used for evaluation **will** ~~not~~ be identical to the public instances!

$\rightarrow$ see the package ,,Testing scenarios”

Report

You are asked to submit a short report:

- What approach have you used for discovering gold stones and depots?
- How have you solved synchronization problems?
- What issues have you encountered and how have you overcome them?

Reward: **1 point**

Too easy?

Advanced solvers are encouraged to try to deal with more difficult setups...

- Narrow passages
- Deadends
- Complex shapes of obstacles
- ...

Possible reward: **extra points**

(number of your points from tutorials can be at most 40 unfortunately)

ADVANCED JASON

Test goals

```
+!say_hello(N) <- ?greeting(G) ; .print(G," ",N).
```

1. `greeting(G)` matches the belief base $\rightarrow$ `G` gets unified
2. A plan for `+?greeting(G)` is executed
   $\rightarrow$ `G` gets unified with applicable value
3. A failure plan for `-?greeting(G)` is applied

Test goals

Example:

```
+?random_move(left) :  math.random < 0.25.
+?random_move(right) :  math.random < 0.33.
+?random_move(up) :  math.random < 0.5.
+?random_move(down).

+!step <- ?random_move(D) ; do(D).
```

Communication

Talking with **one** colleague:

$$\texttt{.send(Rcpt, } ilf\texttt{, Message)}$$

$ilf \in \{\texttt{tell}, \texttt{untell}, \texttt{achieve}, \texttt{askOne}, \ldots\}$

- `tell` — adds belief `Message` to `Rcpt`'s belief base
- `untell` — removes a belief previously told
- `achieve` — adds intention `!Message` for `Rcpt`

Communication

Example:

```
+!run <- ?name(N) ; .send(miner1, achieve,
                    say_hello(N)).
+!say_hello(N) <- .print("Hello from ",N).
```

Communication

The `askOne` variant of `.send`:

```
.send(Rcpt, askOne, Goal, Result)
```

Similar to `achieve` — `?Goal` test goal is added.

Execution of the intention is `paused` until the `?Goal` is (un)satisfied.

- `?Goal` unsatisfiable — `Result=false`
- otherwise — `Result` contains `Goal` with all free variables unified

Communication

Example

Lists

```
X = [ 1, 2, 3 ]
```

- Prepending element into list:

```
[ 0 | X ] = [ 0, 1, 2, 3 ]
```

Lists

```
X = [ 1, 2, 3 ]
```

- Prepending element into list:

```
[ 0 | X ] = [ 0, 1, 2, 3 ]
```

Advanced unification

Variables can get unified for **more complex** terms, e.g.:

```
!greet(greeting("Hello ", "!  How are you?"), "Bob").
+!greet(greeting(Before,After), Who) <- .print(Before,Who,After).
```

```
!first([1, 2, 3]).
+!first([X | Xs]) <- .print(X).
```

Question: What happens if `!first([])` *is requested?*

Task: Write plans for `!print_all([1,2,3])` *intention that lists all elements of the list.*

Prolog–style beliefs

Example:

```
valid(X,Y) :- gsize(_,W,H) & X>=0 & X<W & Y>=0 & Y<H.
```

Atomic plans

An atomic plan is executed **intact**.

$\rightarrow$ No other plan can interfere with actions from the atomic plan

Example:

```
@pickGoldPlan[atomic]
+?pick_gold(X,Y) <- !go_to(X,Y) ; do(pick) ; ...
```

Disclaimer: Beware of deadlocks!

TIPS

Possible caveats

- Helping miners leaving their square before the `do(pick)` action is fully executed
- Miners blocking the way of other miners
- ...

Try to anticipate possible caveats **before** you encounter them.
$\rightarrow$ It will be easier to deal with them

Tips

- Think before implementation
- Decompose the problem into simple problems ($\sim$ intentions) first $\rightarrow$ It will just remain to implement and debug them
- Be prepared for possible issues!

Tips

- Think before implementation
- Decompose the problem into simple problems ($\sim$ intentions) first $\rightarrow$ It will just remain to implement and debug them
- Be prepared for possible issues!

Tips

- Think before implementation
- Decompose the problem into simple problems ($\sim$ intentions) first $\rightarrow$ It will just remain to implement and debug them
- Be prepared for possible issues!