Game Description Language for Real-time Games

Jakub Kowalski, Andrzej Kisielewicz

GIGA

27 July 2015

Evolution of Game Description Language

GDL Evolution

GDL (Genesereth, Love, Pell, 2005)

n-player, turn-based, finite, deterministic, full information games

GDL Evolution

GDL (Genesereth, Love, Pell, 2005)

n-player, turn-based, finite, deterministic, full information games

GDL-II (Thielscher, 2010)

n-player, turn-based, finite, ~~deterministic, full information~~ games

GDL Evolution

GDL (Genesereth, Love, Pell, 2005)

n-player, turn-based, finite, deterministic, full information games

GDL-II (Thielscher, 2010)

n-player, turn-based, finite, ~~deterministic,~~ ~~full information~~ games

rtGDL (2015)

n-player, ~~turn-based,~~ finite, deterministic, full information games

GDL Evolution

GDL (Genesereth, Love, Pell, 2005)

n-player, turn-based, finite, deterministic, full information games

GDL-II (Thielscher, 2010)

n-player, turn-based, finite, ~~deterministic,~~ ~~full information~~ games

rtGDL (2015)

n-player, ~~turn-based,~~ finite, deterministic, full information games

rtGDL-II (2015)

n-player, ~~turn-based,~~ finite, ~~deterministic,~~ ~~full information~~ games

What is real-time in games?

Real-time games

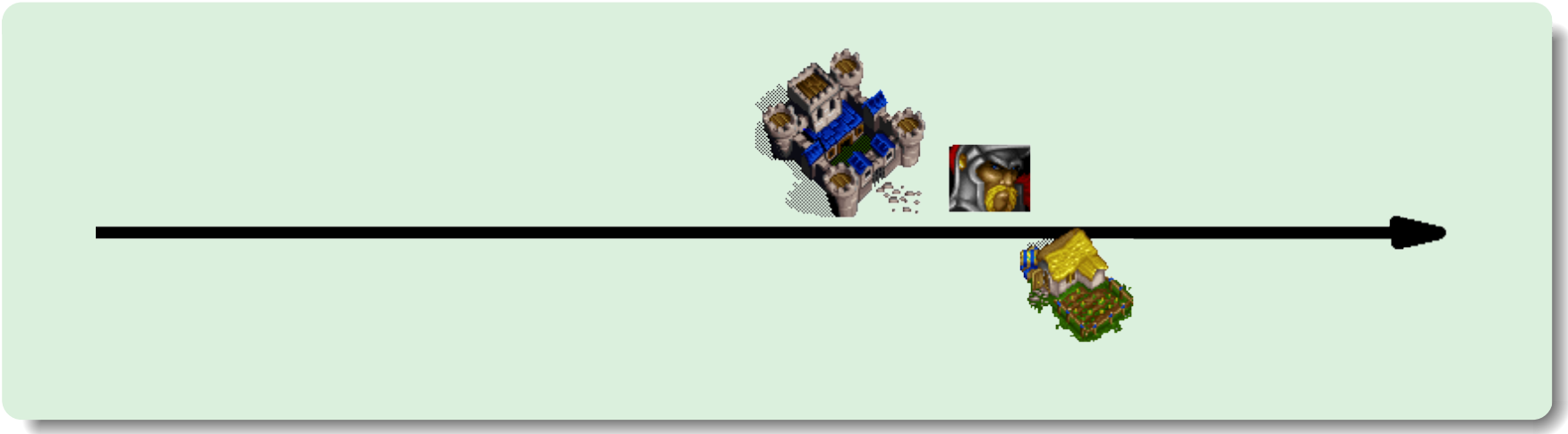
Real-time games

Real-time games

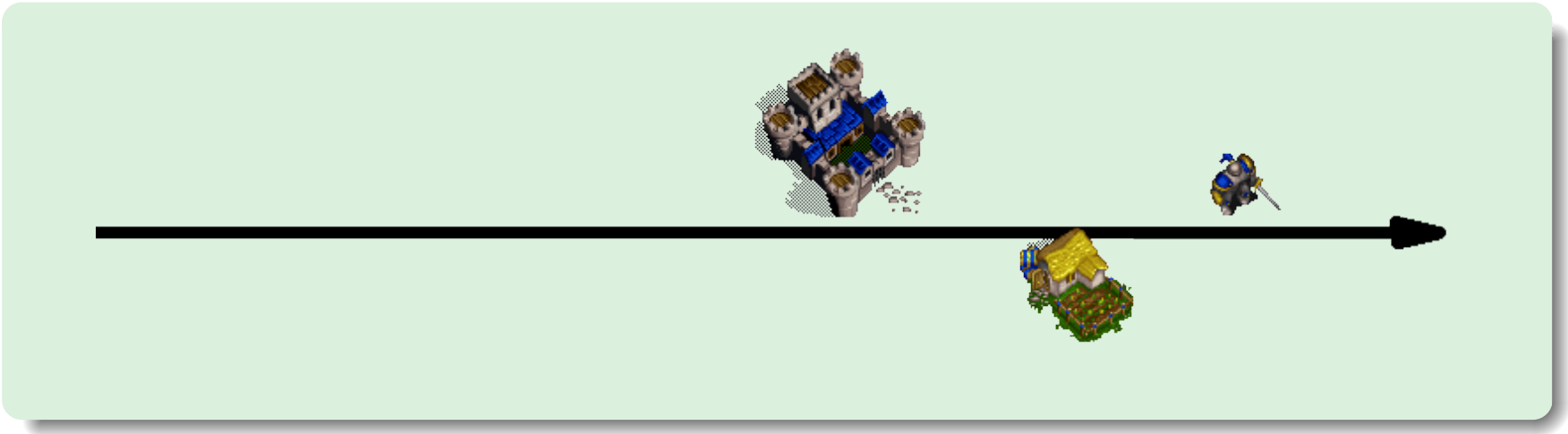
Real-time games

Real-time games

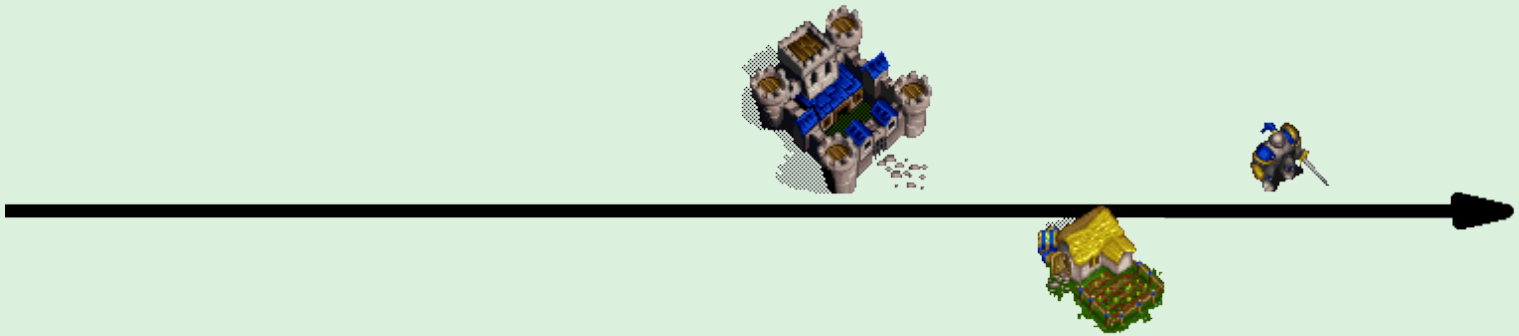
Real-time games



Real-time games



Real-time games



- Players can take actions at any moment.
- Events can have duration time.

Real-time in GDL

Goal

- Preserve purely declarative style
- No arithmetic in rule engine

Real-time in GDL

Goal

- Preserve purely declarative style
- No arithmetic in rule engine
- Reasoners should remain unchanged

Real-time in GDL

Goal

- Preserve purely declarative style
- No arithmetic in rule engine
- Reasoners should remain unchanged

Problem

There have to be real numbers somewhere inside the GDL code.

Real-time in GDL

Goal

- Preserve purely declarative style
- No arithmetic in rule engine
- Reasoners should remain unchanged

Problem

There have to be real numbers somewhere inside the GDL code.

But we already have natural numbers (in goal relation)!

Real-time in GDL

Goal

- Preserve purely declarative style
- No arithmetic in rule engine
- Reasoners should remain unchanged

Problem

There have to be real numbers somewhere inside the GDL code.

But we already have natural numbers (in goal relation)!

Solution

- Treat time as immutable values, only copying is allowed.
- Arithmetic operations are allowed only by the search engine (between state updates).

Real-time Games Description Language (rtGDL)

Keywords

GDL	rtGDL
`role(R)` `init(F)`	

`true(F)`
`legal(R,M)`
`does(R,M)`
`next(F)`	

`terminal` `goal(R,N)`	

Keywords

GDL	rtGDL
`role(R)`	`role(R)`
`init(F)`	`init(T,F)`
`true(F)`	`true(T,F)`
`legal(R,M)`	`legal(R,M)`
`does(R,M)`	`does(R,M)`
`next(F)`	`next(T,F)`
`terminal`	`terminal`
`goal(R,N)`	`goal(R,N)`
	`infinity`
	`expired(F)`

$T \in \mathbb{R}^+ \cup \{\infty\}$
(lifetime)

Sketch of semantics

Event

An *event* occurs when at least one fact becomes obsolete or at least one player performs a move.

Time update

Update of state S after time Δt:

$$\text{Holding facts} : \{\texttt{true}(t_i - \Delta t, f_i) : \texttt{true}(t_i, f_i) \in S \wedge \ t_i > \Delta t\}$$

$$\text{Expired facts} : \{\texttt{expired}(f_i) : \texttt{true}(t_i, f_i) \in S \wedge \ t_i \leqslant \Delta t\}$$

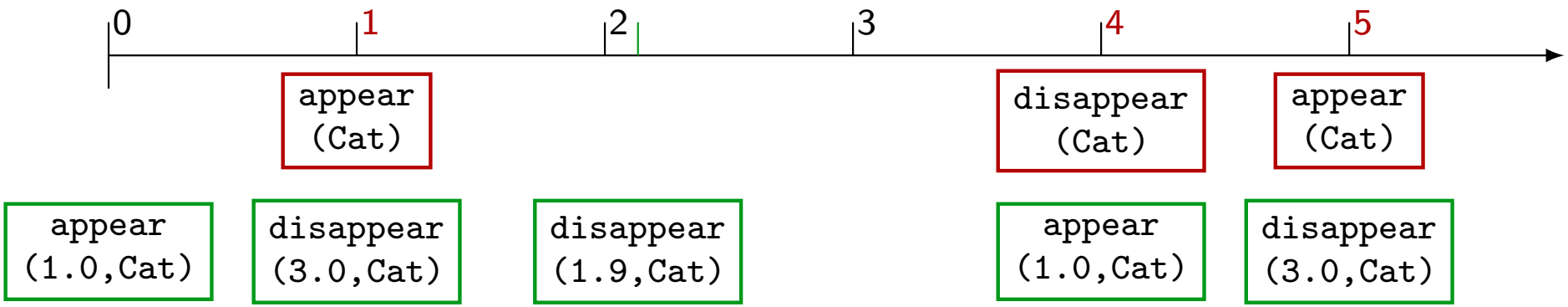
State update

State update is performed with the first occurrence of an event.

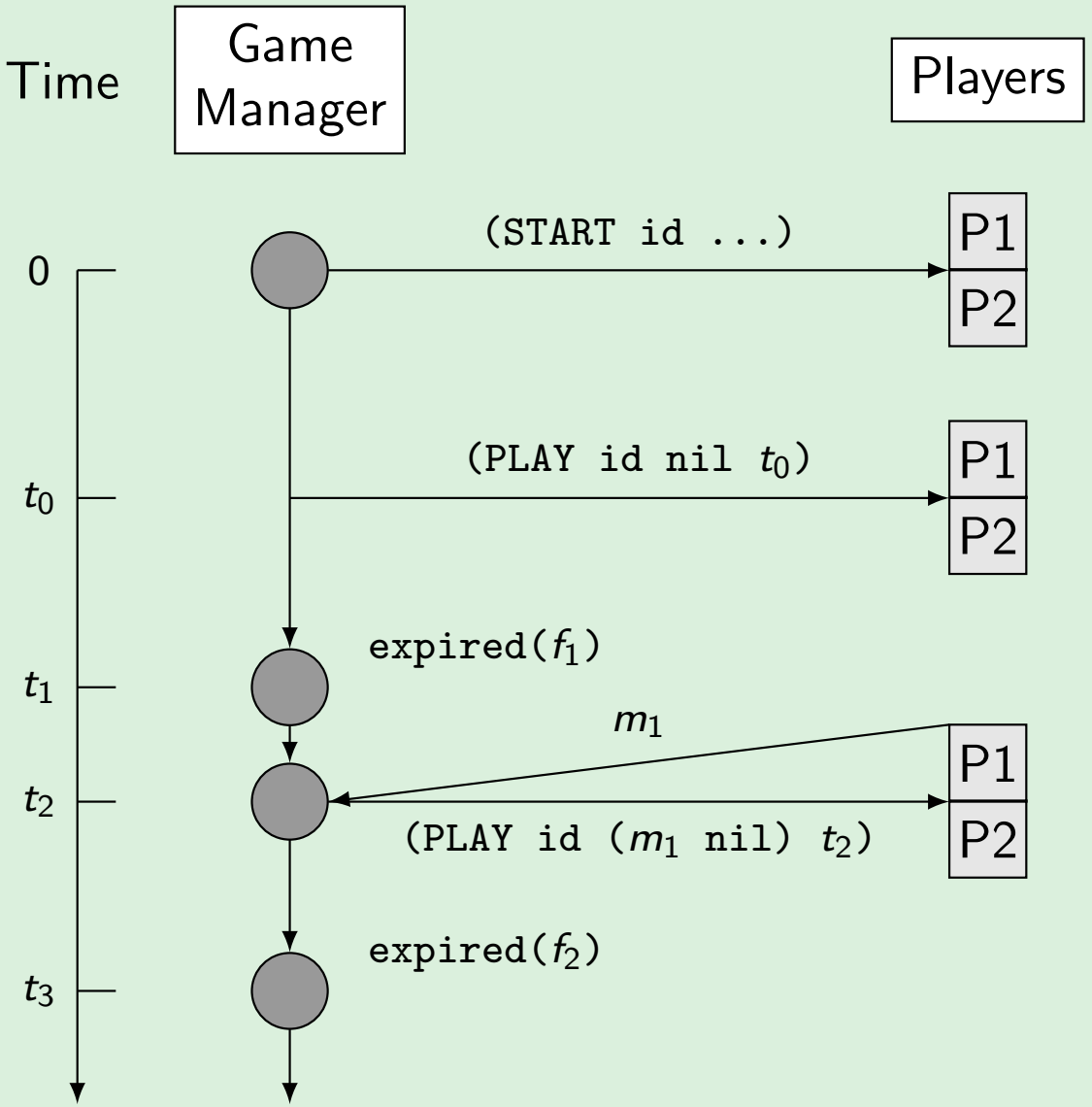
Simple example

Self appearing and disappearing object

```
1  init(1.0,appear(cheshireCat))
2
3  next(3.0,disappear(C)) ⇐ expired(appear(C)).
4  next(T,disappear(C))   ⇐ true(T,disappear(C)).
5
6  next(1.0,appear(C)) ⇐ expired(disappear(C)).
7  next(T,appear(C))   ⇐ true(T,appear(C)).
```



Communication Protocol



Execution model

Game Manager

1. Send `START` messages.
   Set $S := S_0$. Turn the timer on.
2. Send initial `PLAY` messages.
3. Restart the timer.
   Calculate t_u = the minimal time until something from S expires.
4. Wait until move received or the timer equal to t_u.
   Set $M :=$ 'players moves', $\Delta t :=$ 'timer indications'.
5. If $M \neq \emptyset$ send `PLAY(`M`,`Δt`)`.
6. Update state $S := u(\Delta t, M, S)$.
7. If S is not terminal go to 3.
   Otherwise, send `STOP`.

Clocks

`STARTCLOCK` – unchanged
`PLAYCLOCK` – is a multiplier for every `T`

Examples

Turn-based games

```
role(xplayer).  role(oplayer).
init(infinity,cell(1,1,blank)).
...
init(infinity,cell(3,3,blank)).
init(1.0,control(xplayer))

next(1.0,control(S))
 ⇐ does(R,M)
 ∧ role(S)
 ∧ distinct(R,S).

next(infinity,cell(M,N,R))
 ⇐ does(R,mark(M,N))
 ∧ true(infinity,cell(M,N,blank)).
next(infinity,cell(M,N,C))
 ⇐ true(infinity,cell(M,N,C)).
 ∧ distinct(C,blank).
next(infinity,cell(M,N,blank))
 ⇐ true(infinity,cell(M,N,blank))
 ∧ does(R,mark(J,K))
 ∧ (distinct(J,M) ∨ distinct(K,N)).
```

```
legal(R,mark(M,N))
 ⇐ true(infinity
        ,cell(M,N,blank))
 ∧ true(T,control(R)).

terminal
 ⇐ expired(control(R)).

goal(R,0)
 ⇐ expired(control(R)).
goal(S,100)
 ⇐ expired(control(R))
 ∧ role(S)
 ∧ distinct(R,S).

terminal ⇐ ¬boardopen.
terminal ⇐ line(R).
goal(R,100) ⇐ line(R)).

...
```

Chess clock

```
 1  role(white).  role(black).
 2  init(infinity,timer(R,120.0)) ⇐ role(R).
 3  init(infinity,control(white)).
 4  init(120.0,clock).
 5
 6  next(infinity,timer(R,T))
 7     ⇐ does(R,M)
 8     ∧ true(T,clock).
 9  next(infinity,timer(S,T))
10     ⇐ true(infinity,timer(S,T))
11     ∧ does(R,M)
12     ∧ distinct(R,S).
13
14  next(T,clock)
15     ⇐ true(infinity,timer(S,T))
16     ∧ does(R,M)
17     ∧ distinct(R,S).
18
19  terminal ⇐ expired(clock).
20  goal(R,0)
21     ⇐ expired(clock)
22     ∧ true(infinity,control(R)).
```

Respawning objects

```
1  init(infinity,onMap(2,3,yellowArmor,5.5))
2  init(infinity,onMap(11,11,redArmor,10.0))
3
4  next(T,toRespawn(X,Y,A,T))
5      ⇐ true(infinity,onMap(X,Y,A,T))
6      ∧ playerAtPosition(X,Y).
7  next(T,toRespawn(X,Y,A,S))
8      ⇐ true(T,toRespawn(X,Y,A,S)).
9
10 next(infinity,onMap(X,Y,A,T))
11     ⇐ true(infinity,onMap(X,Y,A,T))
12     ∧ ¬playerAtPosition(X,Y).
13 next(infinity,onMap(X,Y,A,T))
14     ⇐ expired(toRespawn(X,Y,A,T))
```

Self-moving objects

```
 1  speed(xwing,2,2,1)
 2  init(infinity,space(35,86,40,xwing))
 3  init(0.5,refresh)
 4
 5  next(0.5,refresh) ⇐ expired(refresh)
 6  next(T,refresh) ⇐ true(T,refresh)
 7
 8  next(infinity,space(F,G,H,S))
 9     ⇐ true(infinity,space(X,Y,Z,S))
10     ∧ expired(refresh)
11     ∧ speed(S,A,B,C)
12     ∧ plus(X,A,F)
13     ∧ plus(Y,B,G)
14     ∧ plus(Z,C,H).
15  next(infinity,space(X,Y,Z,S))
16     ⇐ true(infinity,space(X,Y,Z,S))
17     ∧ ¬expired(refresh).
```

Player ordering time-taking vents

```
 1  cost(barracks,160,6.0).
 2  cost(blacksmith,140,7.0).
 3  init(infinity,gold(500)).
 4
 5  legal(player,build(B))
 6     ⇐ true(infinity,gold(G))
 7     ∧ cost(B,F,T)
 8     ∧ greaterEqual(G,F).

 9  next(T,underConstruction(B))
10     ⇐ does(player,build(B))
11     ∧ cost(B,G,T).
12  next(T,underConstruction(B))
13     ⇐ true(T,underConstruction(B)).
14
15  next(infinity,constructed(B))
16     ⇐ expired(underConstruction(B)).
17  next(infinity,constructed(B))
18     ⇐ true(infinity,constructed(B)).
```

Real-time GDL with Incomplete Information (rtGDL-II)

Nondeterminism

What happens? (following GDL-II)

Using a non-empty set of `random`'s actions:
perform an action drawn with uniform probability and update the state.

Nondeterminism

What happens? (following GDL-II)

Using a non-empty set of `random`'s actions:
perform an action drawn with uniform probability and update the state.

When it happens?

If `random`'s action arity > 1 and the first argument is a real number:
perform state update after time t' drawn using the $\mathcal{U}(0, t)$ distribution.

Example

- `wait`,
- `move(right)`,
- `shot(0,pistol)`,
- `shot(2,blaster)`.

Incomplete Information

Basics (following GDL-II)

- `sees` as a predicate for players' percepts.
- Game Manager sends percepts instead of the joint move.

Incomplete Information

Basics (following GDL-II)

- `sees` as a predicate for players' percepts.
- Game Manager sends percepts instead of the joint move.
- We should not give a clue that someone made a move.

Incomplete Information

Basics (following GDL-II)

- `sees` as a predicate for players' percepts.
- Game Manager sends percepts instead of the joint move.
- We should not give a clue that someone made a move.

Solution

Compute two sets of percepts:

- after state update,
- as if update never happened.

Send message only if the sets differ.

Leads to silent updates problem
(which can be solved by making them verbose)

Summary and future work

Conclusion

We have presented a real-time extension of GDL, which:

- remains in the spirit of GGP,
- preserve pure logical reasoning,
- is the largest GDL family language so far,
- opens a new area of GGP research,
- can be further extended by merging with GDL-II.

Future work

- Investigate limitations of the language and provide more examples.
- Investigate relations with other formalisms (eg. Extensive form games).
- Implement the Game Manager.
- Implement a reasonable player.
- Formally describe rtGDL-II.

Future work

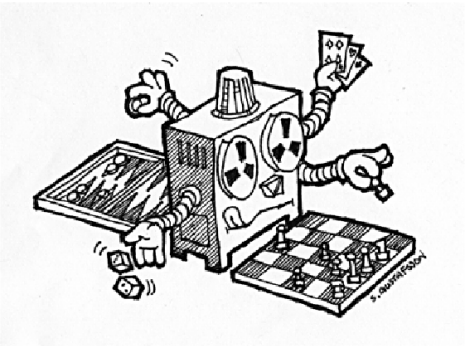
- Investigate limitations of the language and provide more examples.
- Investigate relations with other formalisms (eg. Extensive form games).
- Implement the Game Manager.
- Implement a reasonable player.
- Formally describe rtGDL-II.

If anyone is interested, I am very willing to cooperate ☺.

Future work

- Investigate limitations of the language and provide more examples.
- Investigate relations with other formalisms (eg. Extensive form games).
- Implement the Game Manager.
- Implement a reasonable player.
- Formally describe rtGDL-II.

If anyone is interested, I am very willing to cooperate ☺.



THANK YOU