MineFOL: a Game for Learning First Order Logic

Adrian Groza and Maria Monalisa Baltatescu
Department of Computer Science
Technical University of Cluj-Napoca, Romania
Cluj-Napoca, Romania
Adrian.Groza@cs.utcluj.ro,Baltatescu.Ne.Maria@utcluj.didatec.ro

Mihai Pomarlan
Faculty of Linguistics
University of Bremen
28359 Bremen, Germany
pomarlan@uni-bremen.de

***Abstract*—The role of games in computing science has been given some consideration in the literature. We aim to design games that can be used for teaching logic. Specifically, we build on the Minefield game used in the Stanford Intro to Logic course, and present a version, which we call MineFOL, that incorporates time constraints. We also move beyond the Stanford examples in that we provide a formalisation of the game, with a view to eventually generating game instances automatically as well as from user input, and show how MineFOL can be used as a learning tool. We also developed an online platform supporting learners in at least three ways. First, the learners can practice reasoning in First Order Logic (FOL) and proving strategies such as resolution through various MineFOL games. Second, the learners can practice formalisation in FOL, by allowing learners to build their own games. Third, the learners become aware of the need of interleaving various technologies to solve a logic-based task. In this line, MineFOL is formalised in such way to allow augmenting reasoning in FOL with search strategies.**

***Index Terms*—First Order Logic, theorem proving, Minefield game, teaching logic**

I. Introduction

Teaching logic with games has already proved successful no matter the learner level or age. Recent worldwide examples include the "Intro to Logic" class at Stanford[1], the "Logic with Fun" [9] at Australian National University, "Dynamic Epistemic Logic" at Amsterdam University [10], or the "Artificial Intelligence" class at Technical University of Cluj-Napoca[2].

The role of games in computer science has been given some consideration in literature [11]. From the teaching perspective, we need to teach logic by building upon the learners' existing logical thinking and improve it through practice. Simple logic exercises help with this as well, but often suffer from being too small and self-contained, whereas the solution of a game or puzzle requires putting together several clues into a coherent chain of reasoning. Puzzles breach the gap between logic small exercises and larger logical-based applications. From the learner perspective, benefits of solving logical games include mental [2], motivational [8], or anthropological aspects [3].

Our aim is to design games that can be used for teaching logic. We took the idea of combining Minefield with FOL from the Stanford "Intro to Logic" course. Henceforth, we refer to this game as MineFOL. We will next present a formalization of it, show examples, and investigate how it can be integrated into a learning platform for logical reasoning skills.

[1]http://logic.stanford.edu/intrologic/extras/minefield.html
[2]https://minefielder.azurewebsites.net/

978-1-7281-9080-8/20/ $31.00 © 2020 IEEE

II. Introducing the MineFOL game

You start in the upper left corner $cell(1,1)$ and you navigate with $\downarrow$, $\uparrow$, $\rightarrow$, $\leftarrow$. Some cells contain mines; some contain messages, others are empty. The goal is to locate all mines. You should move in a square only if you prove that there is no mine in that cell. Otherwise, if you move into a cell containing a mine, the game ends. The messages are written in First Order Logic and they help you locate mines. If you move into a cell containing a message, the message can be used to prove that some cells are (not) safe. The message $XBox_{14} : mine(6,4)$ means that there is a mine in row 6 and column 4, and $\boxtimes_{ij}$ means that the message is in line i column j.

A. Formalising the MineFOL game

Definition 1 (MineFOL game): A MineFOL game is defined by the tuple $\mathcal{G} = \langle \mathcal{M}, m \times n, g, \tau \rangle$, where $\mathcal{M}$ is a set containing FOL messages, $m \times n$ is the size of the grid, the goal g is the number of mines to find, while τ represents a budget of moves to find all the mines.

Example 1 (A MineFOL game): Let

$$G_1 = \langle \{ \lfloor m_{11} : \neg\exists x\ mine(1,x) \rfloor, \\ \lfloor m_{13} :\ mine(1,6) \vee mine(5,1) \rfloor \}, 6 \times 5, 1, 4 \rangle$$

The game G_1 has two messages. The message located in $cell(1,1)$ states that the first row is safe. The message in $cell(1,3)$ states there is a mine either on line 6 column 1, or in line 1 column 5. The grid has six lines and five columns, the goal is to find a single mine, while the agent should find the mine within 4 steps.

Figure 1 shows a solution for the game G_1 in two steps. After the first message, one logical agent can prove that all cells in the first column are safe. After two down moves, the agent collects the second message. At this moment, the agent can prove that there is $mine(5,1)$. Since the game specifies a single mine, the game is solved within the available time budget. Note that if the agent explored all the safe cells (1,2), (1,3), (1,4), (1,5), (1,6) to collect messages before attempting to use any new messages it finds, the time would elapse without finding a mine.

Let $\mathcal{K}^t$ be the set of messages known after move t; $\mathcal{K}^0 = \{m_{11}\}$. The messages form a logical theory about the board,

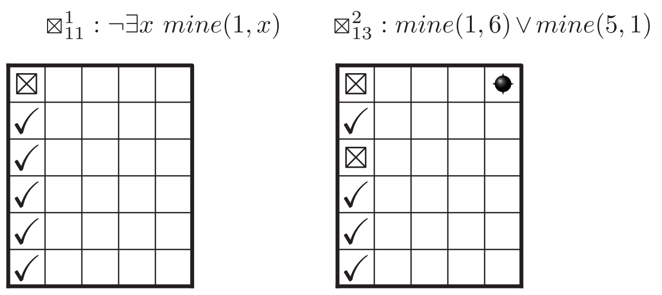

$\boxtimes^1_{11} : \neg\exists x\ mine(1,x)$
$\boxtimes^2_{13} : mine(1,6) \vee mine(5,1)$

Fig. 1: Solving the G_1 game in 2 steps

from which it would be possible to infer whether a particular cell (x, y) is safe or has a mine. We denote this by $\mathcal{K}^t \models safe(x, y)$ or $\mathcal{K}^t \models mine(x, y)$ respectively.

Let $\mathcal{B}^t$ be the set of mines known after move t: $\mathcal{B}^t = \{(x, y) | \mathcal{K}^t \models mine(x, y)\}$, and likewise $\mathcal{S}^t$ is the set of known safe cells after move t: $\mathcal{S}^t = \{(x, y) | \mathcal{K}^t \models safe(x, y)\}$. Let $\mathcal{V}^t$ be the set of cells visited after move t, and $\mathcal{V}^0 = \{(1, 1)\}$. Obviously, we require $m_{(x,y)} \in \mathcal{K}^t \leftrightarrow (x, y) \in \mathcal{V}^t$.

A safe cell is defined as a cell proved not to contain a mine:

$$\forall x, y\ safe(x, y) \leftrightarrow \neg mine(x, y) \tag{1}$$

Definition 2 (Reachable cell): A cell c_r is reachable from the current position c_0 if there is path $(c_0, c_1, ...c_n, c_r)$ where all c_i cells are proved safe.

Definition 3 (State graph): A state graph for the MineFOL game is a graph in which each vertex is labeled by a tuple $(p, \mathcal{K})$, i.e. current position of the player on the board and the set of messages they have seen. An edge exists from vertex $(p, \mathcal{K})$ to $(p', \mathcal{K}')$ iff:

1) p and p' are neighbors on a column or row and both provable safe from $\mathcal{K}$;
2) the size of $\mathcal{B} = \{(x, y) | \mathcal{K} \models mine(x, y)\}$ is less than g
3) $\mathcal{K}'$ equals $\mathcal{K}$ together with the message, if any, at p'

A vertex $(p, \mathcal{K})$ is winning if the size of the set of mines that can be proven to exist from $\mathcal{K}$ is at least g.

Definition 4 (Logical validity): A game is logically valid iff

1) the set of its messages is consistent, i.e. $\forall \mathcal{K} \subseteq \mathcal{M}, \forall (x, y) : \mathcal{K} \not\models safe(x, y) \wedge mine(x, y)$
2) the game is feasible, i.e. there is some path from the vertex $((1, 1), \mathcal{K}^0)$ to a winning vertex of the state graph of G, and the shortest such path is at most τ in length

Further, a game is fair if for any vertex $v = (p, \mathcal{K})$ of the state graph, v reachable from $((1, 1), \mathcal{K}^0)$ implies every shortest path from v to a winning vertex of the state graph of G is at most $\tau - l$ in length, where l is the length of the shortest path from $((1, 1), \mathcal{K}^0)$ to v.

A logical valid game always contains a message in the starting position (1,1). Based on this initial message the agent should prove that at least one neighbour cell (i.e. (1,2) or (2,1)) is safe. Some examples of invalid games follow:

Example 2 (Invalid game because of no winning path): Let

$$G_3 = \langle \{\lfloor m_{11} : \neg\exists x\ mine(x, x)\rfloor, \lfloor m_{22} : mine(1, 4)\rfloor\}, 4 \times 4, 1, 4\rangle$$

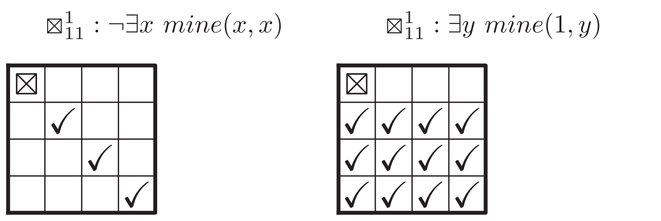

$\boxtimes^1_{11} : \neg\exists x\ mine(x,x)$
$\boxtimes^1_{11} : \exists y\ mine(1,y)$

Fig. 2: Sample of invalid games

Here, the agent can prove that cells (2,2), (3,3) and (4,4) are safe (see Figure 2, left). But there is no way to reach one of these cells, since the agent has no legal moves in that position (both neighbors are not provable safe). Thus, an agent is not able to collect the second message m_{22} from $cell(2, 2)$ that would suffice to solve the game. The reason for the G_3 game's invalidity is the absence of a path to some winning state.

Example 3 (Invalid game from inconsistent messages): Let

$$G_4 = \langle \{\lfloor m_{11} : \exists y\ mine(y, 1)\rfloor, \lfloor m_{22} : mine(3, 3) \vee mine(4, 4)\rfloor\}, 4 \times 4, 1, 4\rangle \tag{2}$$

Since the first message m_{11} says there is a mine on the first line and the game specifies there is only one mine, an agent can prove that lines 2, 3, and 4 are safe (Figure 2, right). By exploring these safe cells, the agent collects the message m_{22} stating there is a mine in (3,3) or (4,4). Knowing there is only one mine, the agent can prove that line 1 is safe, thus rising a contradiction. Henceforth, a resolution-based prover would prove every given goal. The reason of game G_4's invalidity is because its messages are inconsistent.

Two clarifying observations follow: First, observe that each agent receives only the first message. For instance, an agent a knows initially only the following about game G_1:

$$G_1^a = \langle \{\lfloor m_{11} : \neg\exists x\ mine(1, x)\rfloor\}, 6 \times 5, 1, 4\rangle \tag{3}$$

Second, note that MineFOL has two components: the logical component and the search component. The *logical component* requires proving, based on the collected messages, which cells are safe or not. This part requires a theorem prover. The *search component* requires finding the optimal path between the current agent position and the known safe cells. The agent aims to explore the safe cells in search for more messages. This part requires planning or search algorithms such as A^*. By removing the time-step component with $\tau = \infty$, we are interested only in the logical part; that is the agent should find g mines by entering only safe cells, no matter the number of steps required.

The interleaving of the logical (i.e. proving) component with the searching (exploration) component leads to various agent strategies. One such strategy is exemplified in the following.

B. Unfolding a MineFOL game

Let the game

$$G_2 = \langle \{\boxtimes^1_{11}, \boxtimes^2_{16}, \boxtimes^3_{22}, \boxtimes^4_{66}, \boxtimes^5_{16}, \boxtimes^6_{44}, \boxtimes^7_{53}\}, 6 \times 6, 1, 35\rangle$$

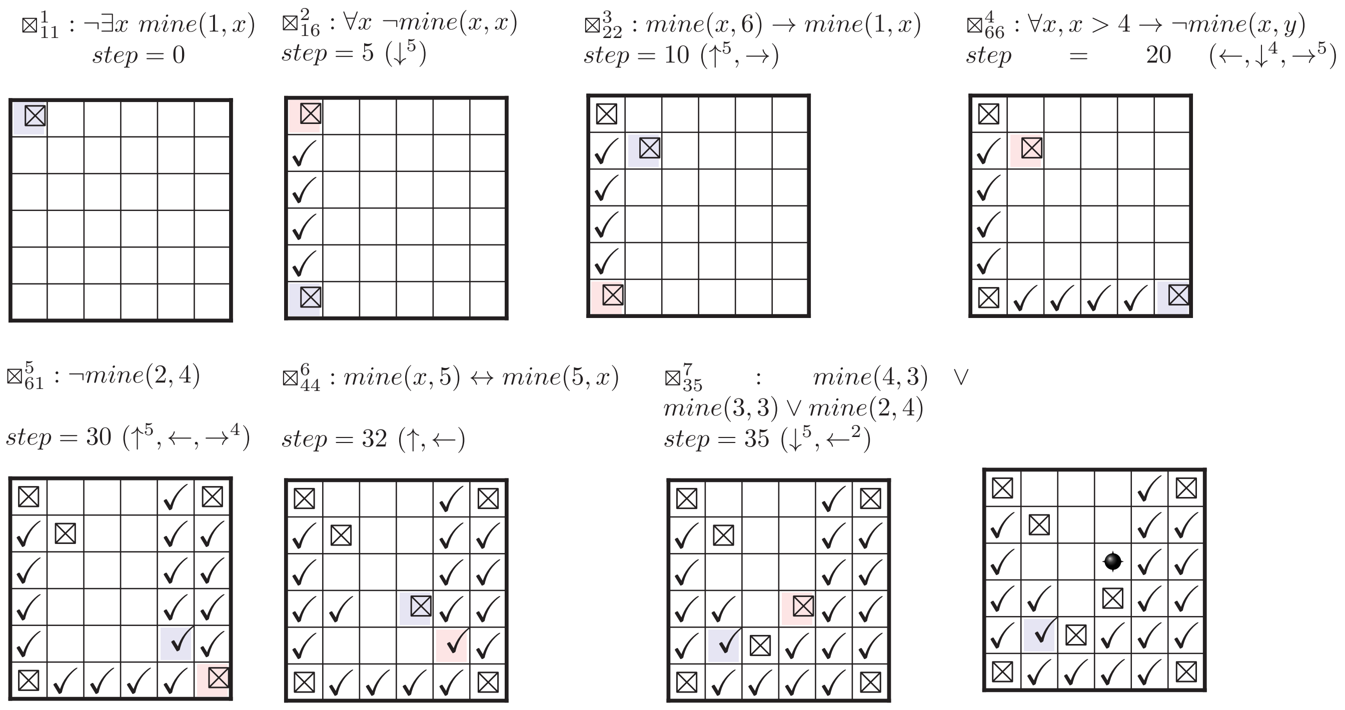

$\boxtimes^1_{11}: \neg\exists x\ mine(1,x)$
$step = 0$
$\boxtimes^2_{16}: \forall x\ \neg mine(x,x)$
$step = 5\ (\downarrow^5)$
$\boxtimes^3_{22}: mine(x,6) \rightarrow mine(1,x)$
$step = 10\ (\uparrow^5, \rightarrow)$
$\boxtimes^4_{66}: \forall x, x > 4 \rightarrow \neg mine(x,y)$
$step = 20\ (\leftarrow, \downarrow^4, \rightarrow^5)$
$\boxtimes^5_{61}: \neg mine(2,4)$
$step = 30\ (\uparrow^5, \leftarrow, \rightarrow^4)$
$\boxtimes^6_{44}: mine(x,5) \leftrightarrow mine(5,x)$
$step = 32\ (\uparrow, \leftarrow)$
$\boxtimes^7_{35}: mine(4,3) \vee mine(3,3) \vee mine(2,4)$
$step = 35\ (\downarrow^5, \leftarrow^2)$

Fig. 3: Unfolding a MineFOL game with a single mine. The blue colored cell is the current position. The pink cell is the previous position. Here, the agent strategy is to collect all messages from the safe, not yet visited cells and then to ask Prover9 [7] for more safe cells. This exploration strategy leads to a solution in 35 steps.

The messages are explained in Table II. The agent a gets only the first message $G^a_2 = \langle\{\boxtimes^1_{11}\}, 6 \times 6, 1, 35\rangle$ (see Figure 3). Here, the logical agent will start exploring the first column, since the initial message $\boxtimes^1_{11}$ ensures that it is safe (Figure 3 up first and second picture). In the last cell of the first column, a new message $\boxtimes^1_{16}$ gets discovered confirming that the diagonal doesn't contain any mines so the logical agent starts the exploration on the diagonal where it finds another message $\boxtimes^1_{22}$ (Figure 3 up third picture). Together with the $\boxtimes^1_{22}$, this last discovered message indicates that the last line is free and consequently, the logical agent continues its exploration there (Figure 3 up fourth picture). After the $\boxtimes^1_{66}$ message asserts the fifth and sixth columns are free, the agent arrives at the (6,6) coordinate where the unique unexplored and safe cell is (5,5) on the diagonal (Figure 3 down first and second picture). Here a new message gets discovered which together with $\boxtimes^1_{66}$ shows the fifth line is safe as well. While exploring this line, the agent finds new information through the message $\boxtimes^1_{35}$ (Figure 3 down third picture). The agent already knows that the diagonal doesn't contain mines so the statement $mine(3,3)$ is obsolete and thanks to the $\boxtimes^1_{61}$ the agent can eliminate the possibility of the mine's existence in the (2,4) cell. In order for the message to be true, the predicate $mine(4,3)$, must be true so the agent wins after locating the mine at (4,3), having performed 35 steps (Figure 3 down fourth picture). Note that the steps are incremented even if the agent backtracks to any previously visited coordinate.

III. Interleaving of Proving and Searching

Definition 5 (Winning a game): The game is won on the agent has a proof of g cells containing mines, and these proofs were obtained within the given time budget.

Definition 6 (Losing a game): The game is lost if: i) the allocated steps are consumed; ii) the agent steps in a mine-cell; or iii) the agent steps in a cell that does not contain a mine, but there is no proof of the safety of the cell. Here, stepping is related to taking the allowed moves: up, down, left right. Since we are interested in learning logic, there is no benefit to allow agents that risk stepping in an unknown cell. Stepping in such a cell means that the agent wrongly proved that the cell is safe. Hence, we are quick to penalise this logical mistake by losing the game.

The game is controlled by Algorithm 1. Here $step$ represents the number of steps used during mine search. The set $Mines$ includes the cells that are already proved to contain mines (initially the set is empty). The set $Unknown$ contains the cells for which Prover9 is not able to prove either safety or mine presence. The current location is represented by the variable $current$. $Messages$ is the set collecting all the messages, $Visited$ is the set of visited cells, and $Safe$ is the set of cells for which Prover9 found a proof that they are safe. Initially, the set $Messages$ contains the message from $cell(1,1)$, which is considered visited and safe. Since there could be more than one safe neighbour at each step, a choice of which safe neighbor to explore next is given by a search strategy like Depth First Search, Breadth First Search, and A^*

TABLE I: A valid 6×6 MineFOL game with 7 messages. At each step, the agent has at least one safe place to go

Message	FOL	Meaning
$\boxtimes^1_{11}$	$\neg\exists x\ mine(1,x)$	There is no mine on the first column
$\boxtimes^2_{16}$	$\forall x\ \neg(mine(x,x)$	There is no mine on the main diagonal
$\boxtimes^3_{22}$	$mine(x,6) \rightarrow mine(1,x)$	Mine on the sixth line implies mine on the first column
$\boxtimes^4_{66}$	$\forall x, x > 4 \rightarrow \neg mine(x,y)$	There are no mines in 5th and 6th columns
$\boxtimes^5_{61}$	$\neg mine(2,4)$	No mine on line 4 and column 2
$\boxtimes^6_{44}$	$mine(x,5) \leftrightarrow mine(5,x)$	Mine on the 5th column if and only if mine on the 5th line
$\boxtimes^7_{35}$	$mine(4,3) \vee mine(3,3) \vee mine(2,4)$	There is a mine in at least one of these cells

Algorithm 1: Winning or losing the game

```
Data: $G(\{m_{11}\}, m \times n, g, \tau)$
Result: result
$step = 0$, $Mines = \emptyset$, $Unknown = \emptyset$;
$current = cell(1,1)$, $Messages = \{m_{11}\}$;
$Visited = \{cell(1,1)\}$, $Safe = \{cell(1,1)\}$;
$searchAlg = A^*$;
while $step < \tau$ and $|Mines| < g$ do
    foreach $n_i \in neighbours(current)$ do
        $status \leftarrow prover9(n_i, Messages)$;
        case status=safe do
            $Safe \leftarrow Safe \cup \{current\}$
        end
        case status=unsafe do
            $Mines \leftarrow Mines \cup \{current\}$;
            if $|Mines| = g$ then
                "Win-Congrats"
            end
        end
        case status=failure-to-prove do
            $Unknown \leftarrow Unknown \cup \{current\}$
        end
    end
    if $Safe \setminus Visited = \emptyset$ then
        return "InvalidGame";
    else
        $current \leftarrow$
          $getNext(searchAlg, current, Safe, Visited)$ if
          $current \in Unknown$ then
            "Unproven cell"
        end
        if $contains(current, m_i$ then
            $Messages \leftarrow Messages \cup \{m_i\}$
        end
    end
    $step \leftarrow step + 1$;
    if $step = \tau$ then
        "Times Up"
    end
end
```

in the current case). While time available and not all mines are found, Prover9 checks the status of all neighbours. If a single neighbor is proved to be safe then the agent will choose that coordinate to explore next. If more than one cells are safe, the search strategy will decide what action to take. Otherwise, if none of the unexplored cells are safe, the agent must backtrack by choosing the visited cell with the minimum visited count. Again, if all the visited neighbors have the same visited count value, the search strategy will indicate what is the agent's next position. All the cells that couldn't be proved safe represent the possible candidates to contain a mine. Therefore, we try to find if any of the unproved cells has a mine and the process continues until the agent still has available steps to take or until it finds all the mines.

To win the game, several strategies can be explored. The top level of the "Explore-All" strategy is illustrated in Algorithm 2. Note that this strategy was used in Fig. 3 game.

Algorithm 2: "Explore-All" strategy

```
Data: $G(\{m_{11}\}, m \times n, g, \tau)$
Result: $path$
repeat
    $RSafe \leftarrow Reachable \cap Safe$;
    $Message \leftarrow Messages \cup explore\_all(RSafe)$;
    $Mines \leftarrow Mines \cup prove(FOLprover, Messages)$;
    if $|Mines| = g$ then
        "You won"
    else
        $Reachable \leftarrow computeReachable()$;
        $\tau \leftarrow \tau - 1$;
    $path \leftarrow path \cup Reachable$
until $\tau > 0$;
```

A second strategy is to call a FOL theorem prover (e.g. Prover9 [7]) once a new message is found (see Algorithm 4).

Algorithm 3: "Exploit-Prover9" strategy

```
Data: $G(\{m_{11}\}, m \times n, g, \tau)$
Result: $path$
repeat
    $RSafe \leftarrow Reachable \cap Safe$;
    $m_i \leftarrow getMessage(RSafe)$;
    if $Messsage = \emptyset$ then
        $Reachable \leftarrow computeReachable()$;
        $\tau \leftarrow \tau - 1$;
        $result \leftarrow result \cup Reachable$
    else
        $Mines \leftarrow Mines \cup prove(FOLprover, m_i)$
    if $|Mines| = g$ then
        "You won"
    else
        $Reachable \leftarrow computeReachable()$;
        $\tau \leftarrow \tau - 1$;
        $path \leftarrow path \cup Reachable$
until $\tau > 0$;
```

Another strategy is calling Prover9 after each step. Firstly, we try to prove if any of the neighbors are safe. Then, if Prover9 failed to prove the safety of one neighbor, we try to find if that cell contains a mine.

Algorithm 4: ”Exploit-Neighbors” strategy

```
Data: $G(\{m_{11}\}, m \times n, g, \tau)$
Result: $path$
$Current = cell(1,1)$;
repeat
    $Neighbors \leftarrow getNeighbors(Current)$;
    $Safe = \{Current\}$;
    $Unsafe = \emptyset$;
    foreach $n_i \in Neighbors$ do
        if $safe(n_i)$ then
            $Safe \leftarrow Safe \cup \{n_i\}$
        else
            $Unsafe \leftarrow Unsafe \cup \{n_1\}$
    if $Unsafe \neq \emptyset$ then
        $Mines \leftarrow Mines \cup prove(Unsafe, Messages)$
    if $|Mines| = g$ then
        ”You won”
    else
        $Current \leftarrow$
          $getNext(Safe, searchAlg, Messages)$
        $\tau \leftarrow \tau - 1$
        $path \leftarrow path \cup \{Current\}$
until $\tau > 0$;
```

```
 1 formulas(minefol).
 2   all x all y (safe(x,y) <-> -(mine(x,y))).
 3   safe(1,1).
 4 end_of_list.
 5
 6 formulas(game)
 7   mine(1,2).
 8   all x (-mine(x,x)).
 9   all x safe(x,0) -> safe(3,x).
10   mine(1,3).
11   -(exists x (mine(x,0))).
12 end_of_list.
13
14 formulas(goals).
15    safe(3,1).
16 end_of_list.
```

Listing 1: Formalising MineFOL game in Prover9 syntax.

Example 4 (A game which can be successfully solved by a human user, but not by a logical agent): Let

$$G_6 = \langle \{ \lfloor m_{11} : \exists y\ mine(1,y) \rfloor, \lfloor m_{13} : mine(3,3) \rfloor \}, 4 \times 4, 1, 4 \rangle \quad (4)$$

In G_6, when the human agent reaches the (1,3) cell, he finds out that the mines are contained in the (3,3) cell so he wins the game before consuming his available steps. Although the logical agent has the information about the mine location in its knowledge base, it will try to prove that the (3,3) coordinate has a mine only when its current position is nearby. In fact, for this game instance, he couldn't resolve the game with the allocated steps.

The agent should have the capability to maintain an internal state of knowledge, to reason over that knowledge, to update its state after observations, and take actions. This agent should be able to represent the world with some formal representation using FOL and act intelligently.

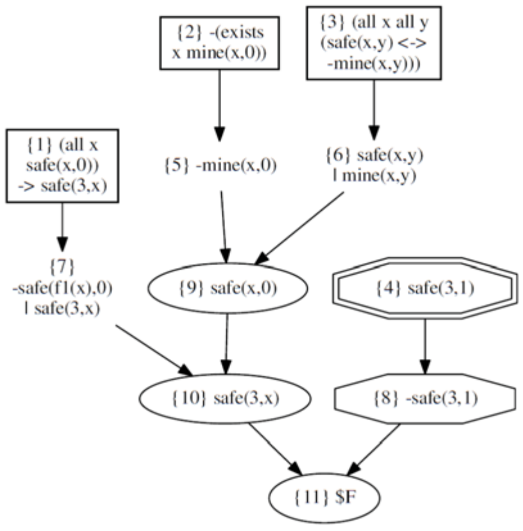

{2} -(exists x mine(x,0))
{3} (all x all y (safe(x,y) <-> -mine(x,y)))
{1} (all x safe(x,0)) -> safe(3,x)
{5} -mine(x,0)
{6} safe(x,y) | mine(x,y)
{7} -safe(f1(x),0) | safe(3,x)
{9} safe(x,0)
{4} safe(3,1)
{10} safe(3,x)
{8} -safe(3,1)
{11} $F

Fig. 4: Proving by resolution the goal $safe(3,1)$

Resoning in FOL is performed here with Prover9, [6], which is a theorem prover for equational first-order logic. An example input file for Prover9 after the agent has explored a few cells is given in Table II. The formalisation already includes seven messages found by the agent. The messages are kept within the `formulas(sos)`. The active goal is to demonstrate that cell (5,7) is safe (line 16 in Listing1). Note that in Prover9 the interval [1,8] is translated to [0,7].

Prover9 relies on resolution as its proving mechanism. The given FOL statements are converted into CNF, the theorem is negated, and the prover starts searching for a contradition. The steps for converting to CNF are listed in Table III.

A sample resolution proof is depicted in Figure 4 where the square framed formulas represent the formulas as they are in the input file, the formulas without any frame represent the conjunctive normal form of the formulas, the circle framed formulas represent the result of the inference and the hexagonal framed formulas represent the theorem. In the first step of resolution graph, $\neg mime(x,0)$ and *mine(x,y)* get resolved by substitution of {0/y}, and we are left with *safe(x,0)*. In the second step, *safe(x,0)* and $\neg safe(f(x),0)$ get canceled by substitution {0/f(x)}, and we are left with *safe(3,x)*. In the last step, *X* being a Skolem constant, it can be replaced with *1* so *safe(3,X)* and $\neg safe(3,1)$ get resolved, so we can consider the theorem proved.

Example 5: $\forall x, \neg mine(x,x)$. $mine(7,7)$. In this case the knowledge base is inconsistent. Since Prover9 uses resolution as proving mechanism, it could prove anything based on proof by contradiction.

To signal inconsistency of the set $Messages$, we ask Prover9 to prove both a sentence and its negation. Since, we assumed that $cell(1,1)$ is safe (recall line 3 in Listing 1), we just need to ask for a proof of $\neg safe(1,1)$. The only case when Prover9 finds proofs for both $safe(1,1)$ and also $\neg safe(1,1)$ is when the collected messages are inconsistent.

TABLE II: Possible messages in MineFOL game

Message	FOL	Natural language
$\boxtimes^{8}$	$\neg\exists y\ mine(1,y)$	The first column is safe.
$\boxtimes^{9}$	$mine(6,4)$	There is a mine in the cell having (6, 4) coordinate.
$\boxtimes^{10}$	$\forall x\ (safe(x,1) \leftrightarrow safe(1,x))$	If the first row has no mines, then the first column has no mines too.
$\boxtimes^{11}$	$\exists x\ mine(3,x) \rightarrow \exists x\ mine(8,x)$	If there is a mine in the third column, then there is a mine in the eighth column as well.
$\boxtimes^{12}$	$\exists x\ mine(x,6) \rightarrow \neg\exists y\ mine(y,8)$	If there is a mine in the sixth row, there is a mine on the eighth row too.
$\boxtimes^{13}$	$mine(2,4) \wedge mine(7,5)$	There are mines in the cells having (2, 4) and (7, 5) coordinates.
$\boxtimes^{14}$	$mine(5,3) \vee mine(3,5)$	There is a mine in the cell having (5, 3) or (3, 5) coordinate.
$\boxtimes^{15}$	$\forall x\ (mine(x,8) \rightarrow mine(x,1))$	If there is a mine on the eighth row, than there is a mine on the first row too.
$\boxtimes^{16}$	$\neg mine(2,3) \wedge \neg mine(3,2)$	The cells having the coordinates (2, 3) and (3, 2) are safe.
$\boxtimes^{17}$	$\forall x\ \neg mine(x,x)$	There are no mines on the main diagonal.
$\boxtimes^{18}$	$\forall x\ \forall y\ \forall z\ (mine(x,y) \wedge y \neq z \rightarrow \neg mine(x,z))$	There is only one mine per column.

TABLE III: Automatic conversion by Prover9 into Conjunctive Normal Form

Step	1	2	3	4	5	6	7
Eliminate implications	$\forall x\forall y\neg safe(x,y) \vee \neg mine(x,y)\wedge mine(x,y) \vee safe(x,y)$	$\neg\exists x mine(x,0)$	mine(3,5)	$\exists y\neg mine(2,y)\vee \exists y mine(7,y)$	$\forall x\neg mine(x,7) \vee mine(x,0)$	$\forall x\neg mine(x,x)$	$\forall x\neg mine(x,0) \vee mine(0,x)\wedge \neg mine(0,x) \vee mine(x,0)$
$Move\neg inwards$	$\forall x\forall y\neg safe(x,y) \vee \neg mine(x,y)\wedge mine(x,y) \vee safe(x,y)$	$\forall x\neg mine(x,0)$	mine(3,5)	$\exists y\neg mine(2,y)\vee \exists y mine(7,y)$	$\forall x\neg mine(x,7) \vee mine(x,0)$	$\forall x\neg mine(x,x)$	$\forall x\neg mine(x,0) \vee mine(0,x)\wedge \neg mine(0,x) \vee mine(x,0)$
Standardize variables	$\forall x\forall y\neg safe(x,y) \vee \neg mine(x,y)\wedge mine(x,y) \vee safe(x,y)$	$\forall x\neg mine(x,0)$	mine(3, 5)	$\exists x\exists y\neg mine(2,x)\vee mine(7,y)$	$\forall x\neg mine(x,7) \vee mine(x,0)$	$\forall x\neg mine(x,x)$	$\forall x\neg mine(x,0) \vee mine(0,x)\wedge \neg mine(0,x) \vee mine(x,0)$
Skolemize	$\forall x\forall y\neg safe(x,y) \vee \neg mine(x,y)\wedge mine(x,y) \vee safe(x,y)$	$\neg mine(X,0)$	mine(3,5)	$\neg mine(2,X)\vee mine(7,Y)$	$\forall x\neg mine(x,7) \vee mine(x,0)$	$\forall x\neg mine(x,x)$	$\forall x\neg mine(x,0) \vee mine(0,x)\wedge \neg mine(0,x) \vee mine(x,0)$
Drop universal quantifiers	$\neg safe(x,y) \vee \neg mine(x,y)\wedge mine(x,y) \vee safe(x,y)$	$\neg mine(X,0)$	mine(3,5)	$\neg mine(2,X)\vee mine(7,Y)$	$\neg mine(x,7) \vee mine(x,0)$	$\neg mine(x,x)$	$\neg mine(x,0) \vee mine(0,x)\wedge \neg mine(0,x) \vee mine(x,0)$
Distribute $\vee$ over $\vee$	$\neg safe(x,y) \vee \neg mine(x,y)\wedge mine(x,y) \vee safe(x,y)$	$\neg mine(X,0)$	mine(3,5)	$\neg mine(2,X)\vee mine(7,Y)$	$\neg mine(x,7) \vee mine(x,0)$	$\neg mine(x,x)$	$\neg mine(x,0) \vee mine(0,x)\wedge \neg mine(0,x) \vee mine(x,0)$

IV. Learning FOL with MineFOL

From a user perspective, Minefield has two roles: a) a logic game, b) a tool for learning FOL.

First, Minefield as a logic game is activated when the user chooses *Play it yourself!* option. The system creates page with an 8×8 board (the default dimension) having various configuration options. The user can explore the board using the $\downarrow$, $\uparrow$, $\rightarrow$, $\leftarrow$ keys to navigate and the mouse to expose the cell containing a mine. When the user reaches a cell containing a hint, the system signals back, displaying a message icon in the corresponding cell and a tooltip with the message in natural language. The system responds with notifications when the user finds all the mines and wins the game or when he reaches a cell with a mine and loses the game.

Second, Minefield is a a learning tool for first-order logic. If the user chooses *Challenge a software agent!* option then a software agent based on Prover9 handles the game. The user only needs to press *Start* and from that moment the software agent takes over all the responsibilities: reading the messages, choosing the safe cells, finding all the mines. The agent only knows at the beginning how many mines he needs to expose and the initial hint located in the cell with *(1,1)* coordinates. The other messages are accessible only when the agent reaches a cell containing a message.

The agent asks Prover9 to demonstrate which of the agent's current position's neighbors are safe or if any of them has a mine. Having this in mind, one more question needs to be addressed: *Which neighbor should the agent choose when Prover9 decides that at least two of them are safe?*.

The solution for this specific problem is to implement a search algorithm which can help the agent decide. A search algorithm usually takes the initial state of the problem and the goal and builds a path the agent should follow to reach its goal. However, the implementation of the search algorithm for MineFOL cannot be done traditionally since such an algorithm can't prove if a cell is safe or not. The search algorithm only takes over the responsibility to choose between two possible safe next moves. In the process of finding mines, the Prover9-based agent is empowered with four search algorithms: Depth-First Search, Breadth-First Search, Random Search, and A*.

The A* algorithm will decide which move to choose according to a value $f = d + h$, with d the cost of the path from the start node to the current one, while heuristic h is the remaining mines to find. At each step, the agent software picks the cell having the lowest *f* value.

Assume the trace of the logical agent on a 6×6 board using both the breadth-first search and depth-first search strategies. The influence of the search strategy is highlighted in the fifth picture of both 5 and 6 figures, when the agent selects different next moves. In both of the cases, the agent asks, in exactly this order, the right, down, left, and up neighbor to find out which is safe. The agent implementing the breadth-first search strategy extracts a cell from the safe neighbors queue using the FIFO strategy (the selected cell in this case is $cell(1,2)$, while the agent implementing the depth-first search strategy extracts the cell using the LIFO strategy (the selected cell in this case is $cell(2,1)$.

These different methodologies lead the logical agent on different paths and with different numbers of steps (the sum of the numbers in each cell): the agent implementing the breadth-first search strategy solves the game in 37 steps while the agent implementing the depth-first search strategy solves the game in 44 steps. We exemplify here the fifth step since the first four steps are identical with the ones presented in Fig 5.

A third feature of the system allows the user to choose the *Create your own game!* option, in which case the system redirects him to a page with an 8×8 board, which is the initial setup of the game, where the user can add new messages

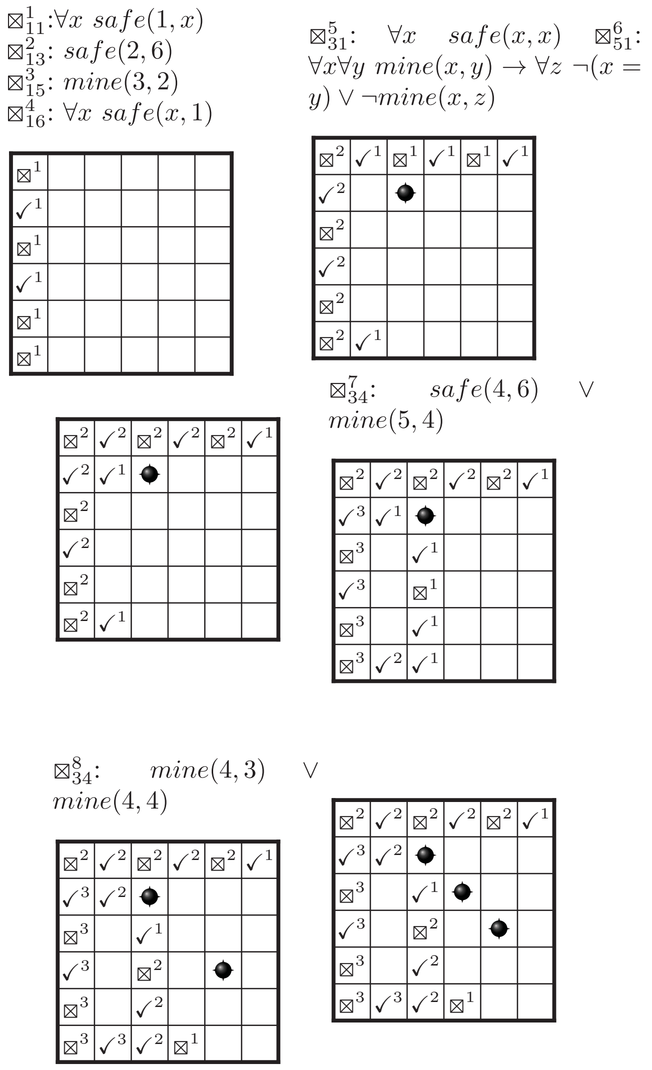

$\boxtimes^1_{11}$: $\forall x\ safe(1,x)$
$\boxtimes^2_{13}$: $safe(2,6)$
$\boxtimes^3_{15}$: $mine(3,2)$
$\boxtimes^4_{16}$: $\forall x\ safe(x,1)$
$\boxtimes^5_{31}$: $\forall x\ safe(x,x)$
$\boxtimes^6_{51}$: $\forall x \forall y\ mine(x,y) \rightarrow \forall z\ \neg(x=y) \vee \neg mine(x,z)$
$\boxtimes^7_{34}$: $safe(4,6) \vee mine(5,4)$
$\boxtimes^8_{34}$: $mine(4,3) \vee mine(4,4)$

Fig. 5: Comparing search strategies - Breadth-First Search (steps 1-7)

in the cells he considers useful. When clicking on a cell, a panel opens on the right side of the page containing the cell coordinates and the board dimension as shown in Figure 7. The user can choose from an existing hint in the system to be associated with the selected cell, or he can create another hint. The creation of a new hint requires both FOL and natural language versions to be specified. To help users, the panel contains three controls showing the quantifiers, the connectors, and the predicates employed in the system that are already translated to Prover9 syntax. The user doesn't have any restrictions when it comes to writing his own hints. He can validate his work by challenging the software agent to solve the newly created game.

V. Discussion and Related Work

Minefield is used as a tool to teach "Introduction to logic" at Stanford University. The grid contains 8×8 cells, and the user needs knowledge about first-order logic to interpret the messages. This initiative was the inspiration for developing the MineFOL platform, as an extension of the Stanford's approach to teach FOL.

One line of extension was to use a theorem prover. With most resolution based provers being appropriate for the task, we used Prover9. The interleaving of Prover9 and the model generator Mace4 has proven a powerful tool to study algebraic structures [1]. Here, Arthan and Oliva have investigated how to develop human-readable proofs in research on algebraic structures using the proofs found by Prover9 and examples found by Mace4.

GamePad has been used to investigate the application of machine learning to theorem proving [4]. Huang has proposed a system to synthesize proofs for a simple algebraic rewrite problem and train baseline models for a formalization of the Feit-Thompson theorem. GamePad provides a structured Python representation of Coq proofs, where formal theorem proving is seen as a game. The assumption is that a game is a useful mental model of the proving process.

Slaney has argued [9] that the only effective solution for learning logic is to understand both natural and formal languages and to match the two. That was the reason he developed a Web site, *Logic for fun*, on which the users are invited to express a set of logical problems so a solver on the site can find solutions. This tool is intended to help understand logic and critical reasoning so that the learners don't give up due to the difficulties of writing a well-formed formula to express claims about a domain of discourse.

Using FOL reasoners is an obvious way of solving logic puzzles, as mentioned, e.g, in research on solving puzzles using logic [5]. Since first-order logic representations are more general and widely applicable to many domains, logic is a tool for logic representation for puzzle solving attempts. Having as an example the sculptures puzzle, the research paper concludes with presenting a prototype system that analyzes and correctly solves all the game requirements.

VI. Conclusions

The MineFOL aims to support students to learn First Order Logic. The system provides a platform to train understanding of FOL formulas, but also to write their own MineFOL games. One advantage is that students have prompt feedback on the FOL formulas they do not completely understand. For instance, if, based on a FOL message, the students steps in a mine cell, the system explains why a mine is there by displaying a resolution-based proof.

We took the idea of combining Minefield with FOL from the Stanford Introduction to Logic. We build on this idea with the following scenarios:

First, students decipher secret messages written in FOL and translate them in natural language. The translation is needed in order to understand the message and to solve the game.

Second, students develop logical agents capable of solving the game. Each logical agent relies on a theorem prover to

$\boxtimes^7_{34}$: $safe(4,6) \vee mine(5,4)$ $\boxtimes^8_{34}$: $mine(4,3) \vee mine(4,4)$

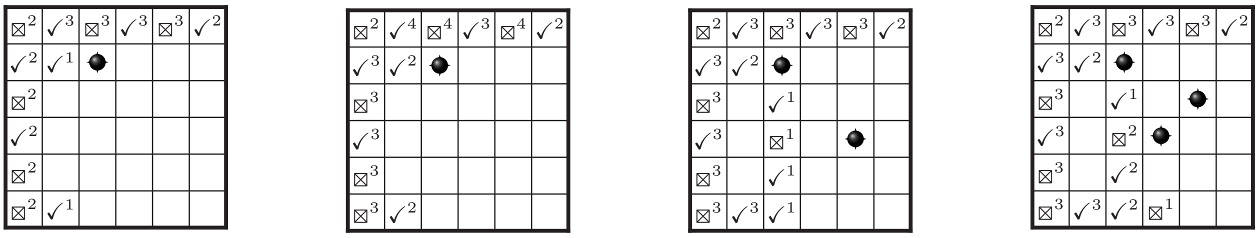
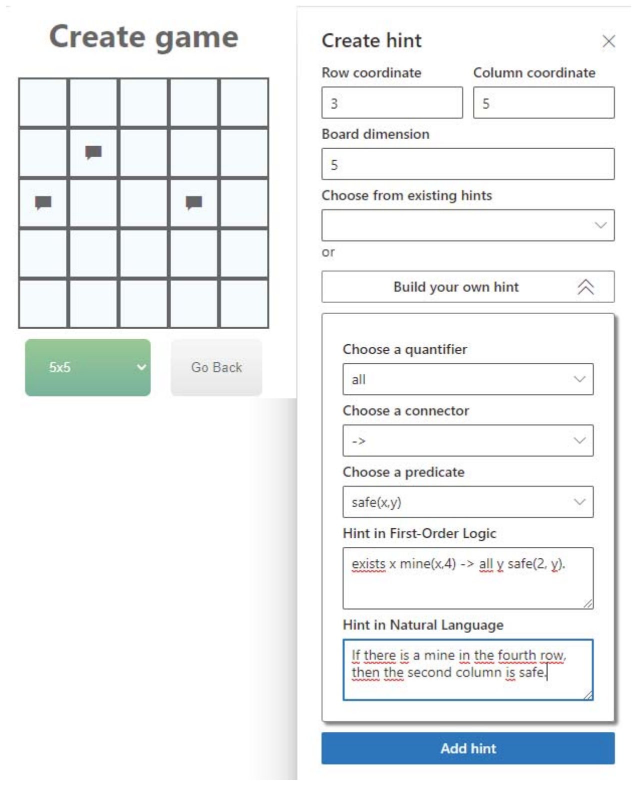


Fig. 6: Comparing search strategies - Depth-First Search (steps 5-9)



Create game

5x5

Go Back

Create hint

Row coordinate: 3

Column coordinate: 5

Board dimension: 5

Choose from existing hints

or

Build your own hint

Choose a quantifier: all

Choose a connector: ->

Choose a predicate: safe(x,y)

Hint in First-Order Logic: exists x mine(x,4) -> all y safe(2, y).

Hint in Natural Language: If there is a mine in the fourth row, then the second column is safe.

Add hint

Fig. 7: Students can create their own MineFOL games

make inferences based on the received FOL messages. A competition can be organised for such logical agents.

Third, students manually design MineFOL games. The aim is to write their own messages in FOL. The challenge is to design messages and to order them in a proper sequence to result in a valid game. The students can challenge each other to solve their own designed MineFOL games.

Forth, students design algorithms that automatically generate MineFOL games. Based on a library of FOL messages previously formalised, a constraint solver or a genetic algorithm can pick from these messages those that would generate a valid MineFOL game.

Fifth, students can assess the complexity of a MineFOL game, by defining various complexity metrics. The metrics may include criteria such as: the length of the messages, the gain of information enclosed in each message, the generality or the specificity of a message, the board dimension, or the number of the mines. All these properties together would provide a method to organize the games based on levels of difficulty. From the student and teacher perspective, this feature can help to keep track of the progress.

Sixth, teachers and students can link the MineFOL domain (learning FOL) with other topics from AI. We have already shown the interleaving of theorem proving with search algorithms. The mix between proving and searching can be formalised under the umbrella of planning. With planning, given a game the agents can dynamically build their plans, where actions alternate between moving and asking for proofs. A complementary line would be to add a translator from natural language to FOL for the MineFOL game. This translation is feasible since the domain is closed and relatively small in terms of types of messages.

ACKNOWLEDGMENT

We thank the reviewers for their valuable comments which were included in the current version.

REFERENCES

[1] Rob Arthan and Paulo Oliva. Studying algebraic structures using Prover9 and Mace4. In *Proof Technology in Mathematics Research and Teaching*, pages 91–111. Springer, 2019.
[2] Marcel Danesi. *Ahmes' Legacy Puzzles and the Mathematical Mind*. Springer, 2018.
[3] Marcel Danesi. *An Anthropology of Puzzles: The Role of Puzzles in the Origins and Evolution of Mind and Culture*. Bloomsbury Publishing, 2018.
[4] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *arXiv preprint arXiv:1806.00608*, 2018.
[5] Iddo Lev, Bill MacCartney, Christopher D Manning, and Roger Levy. Solving logic puzzles: From robust processing to precise semantics. In *Proceedings of the 2nd Workshop on Text Meaning and Interpretation*, pages 9–16. Association for Computational Linguistics, 2004.
[6] William McCune. Mace4 reference manual and guide. *arXiv preprint cs/0310055*, 2003.
[7] William McCune. Release of Prover9. In *Mile High Conference on Quasigroups, Loops and Nonassociative Systems, Denver, Colorado*, 2005.
[8] Edwin F Meyer, Nickolas Falkner, Raja Sooriamurthi, and Zbigniew Michalewicz. *Guide to teaching puzzle-based learning*. Springer, 2014.
[9] John Slaney. Logic for fun: an online tool for logical modelling. *Journal of applied logics*, 4(1):171–192, 2017.
[10] Jan Van Eijck and Rineke Verbrugge. *Discourses on social software*. Amsterdam University Press, 2009.
[11] Luis Von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.