Project Mid-term report

FOrML

Crowdsourced Theorem Proving

October 2, 2019

Abstract

The idea of using an entertaining game to extract valuable outputs can go back to [AD04] which uses human power to help label images. The work of [Gov13] suggests to use a similar approach in theorem proving by translating logic formulae into game's objects and proof calculus into game's rules. Hence, playing a game instance turns out to be the same as looking for a proof of the given logic formula. The purpose of this project is to implement a game based on [Gov13] from scratch to increase gameplay experience and extend it to first order logic. Furthermore, this project has the ambition to spread worldwide the resulted crowdsourcing game and maybe even to surpass current automated theorem provers.

Team leader is Firmin Martin, other members are Jérôme Boillot, Quentin Corradi, Avril de Goër de Herve, Orégane Desrentes, Dina El Zein, Paul Géneau de Lamarlière, Thibault Marette, Guillaume Rousseau, Justine Sauvage, and Victoria Sedig.

Website :`https://forml.rudelune.fr/`

Keywords: Human-based computation game, Theorem proving.

Contents

1 Human-based computation game

Many game-like puzzles can be found among undecidable problems, for example the well-known Post Correspondence Problem (PCP) or the Tiling Problem (given a set of tiles S, determining whether one can create a tiling of the plane using only tiles from S). The Constraint Logic model was introduced as a way to represent these types of games, and to associate every game with a decision problem (undecidable, naturally). The aforementioned problems are not only undecidable, but also able to **capture** First Order Logic, meaning that First Order Logic can be expressed in them somehow. However, they are not well suited for practical use as they lack properties needed for what we aim to do.

A *suitable* game/problem should:

- Be fun to play.
- Have a tractable representation. A good example of this is that it is possible to translate the SAT problem into a bejeweled (or candy crush) instance, which is a fun game to play, but the translation makes it too big to be a fun game. For instance, the formula $x_1 \wedge (x_2 \vee x_1)$ is translated in a bejeweld instance composed in no less than 6000 lines...(https://www.isnphard.com/g/bejeweled/).

- Be able to produce a readable proof. For instance in the tiling problem, we can't prove in a game that it is not possible to tile the plan, or that an aperiodic tiling works, even if the player knows that it does.
- Allow players to play this game even if they know nothing about logic, theorems, etc.

A good example of such is the foldit project. Biologists often get to work on large proteins and to study how these proteins naturally fold. As of now no formula is known that gives an easy answer to that question, and the state-of-the-art algorithms are mediocre at best. However, using crowdsourcing, the foldit team (`https://fold.it/portal/`) was able to solve many instances of protein folding problems much more efficiently than any algorithm. The game being very graphical, players were able to move the protein around and use strategies that a machine couldn't, thus finding better configurations. The foldit game, however, presents itself as a biology problem-solving game: the user knows that he is working on a protein. Our goal would be to create a game where the user has no idea that he is solving first order problems, and where the logic aspect is never shown in the game as such due to its lack of user-friendliness.

2 State of the Art

A part of the game described in the thesis that we studied [Gov13] has already been implemented, it is available on this website: `http://compete.catabotrescue.com/` and it covers all the propositional logic.

Players assemble some "ants" (catabots) in order to create a legless one which could escape from the jail represented by the screen. By assembling these ants, we use in fact some rules of logic on clauses (ants). By creating a legless ant, we prove that it exists a valuation function such that the formula is a truth assignment.

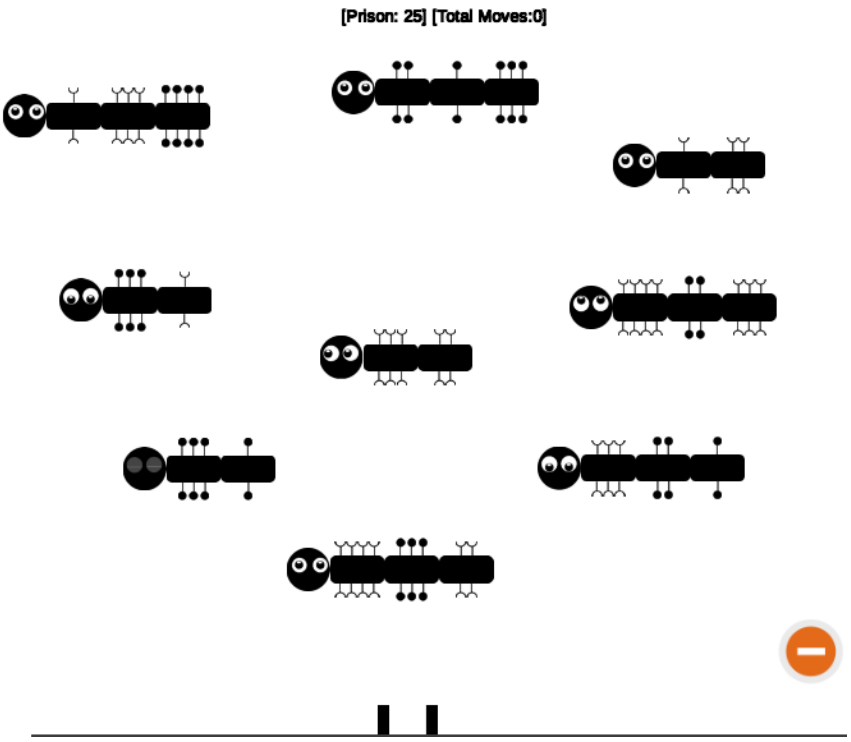
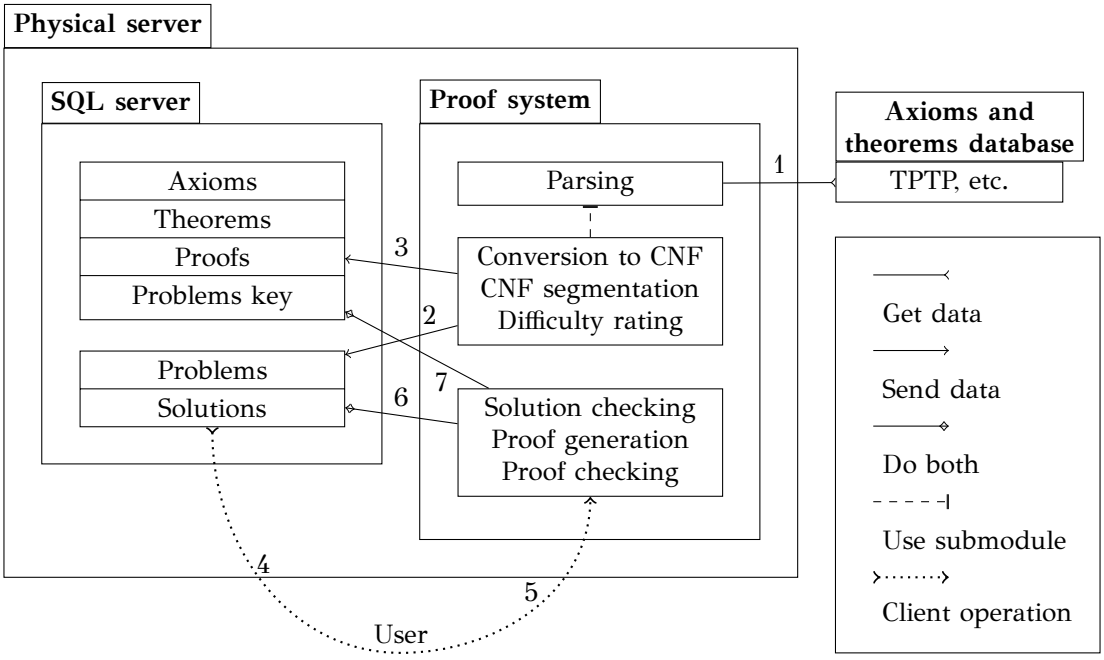


Figure 1: Catabot Rescue – The game created on the principles described in [Gov13].

As described in [NB16], minimum proofs lengths produced by minimally trained humans are smaller than proofs given by automated theorem provers such as Prover9 for the 50 levels implemented. The paper claims that "These results, while humble, indicate that theorem provers might be able to benefit from untrained humans" but adds that "Future experiments would involve a more extensive set of problems [...]". This is exactly what we will try to figure out.

3 Architecture of the project



1 Retrieve a theorem and axioms with HTTP request to an online database

2 Add a problem (CNF + difficulty grade) through SQL request

3 Store a theorem entry (SQL request)

4 Get a problem (SQL request)

5 Submit a solution, protocol not defined

6 Add a new valid solution to a problem (SQL request)

7 Add a new theorem proof (SQL request)

Figure 2: Server side view of theorem proving % terms

Standard theorem proving procedure:

1. Get the theorem and corresponding axioms from a theorems and axioms database.
2. Convert the theorem into Conjonctive Normal Form (CNF, a.k.a. Clausal Normal Form).
3. Segment it into small enough pieces (*e.g.* a problem $A \wedge B$ can be split into problems A and B which are individually simpler to solve).

4. Rate the difficulty of subproblem and add a problem (a CNF and a difficulty grade) to the database.
5. Add segmentation info (contains the problems keys, the axioms, and the original theorem to prove) to the database.
6. A user tries to solve a problem: they gets it from the server, make a solution (a sequence of actions to solve the problem) and submit it.
7. The solution is verified and stored in the database if it is valid.
8. Go to step 5 until all the segment are solved.
9. Assemble the proof of the global problem, check that it is sound. If it is, make the proof of the original theorem and store it in the database.

4 Work packages

The current work packages are described below, but keep in mind that they will change during the project because of the nature of it: when the structure of the game will be strong enough, the game engine work package will be less important and more people will be involved in the backend given that it mostly relies on the game engine.

1. **Communication**

   Role: Creation of a website to communicate about our project and communication with other people like the author of the main article.

   Members: Firmin, Justine (leader), Avril, and Victoria.

2. **Game Design**

   Role: Creation of interface between logical expressions and game.

   Members: Orégane (leader), Justine, and Avril.

3. **Game Engine**

   Role: Creation of the game graphics and mechanisms involved.

   - **Interface**:

     Role: handling how to display the game properly (usage of `Godot` engine).

     Members: Orégane, Jérôme (leader), and Avril.

   - **Storage management**:

     Role: manage how the game state will be stored in order to be integrated with the network later.

     Members: Quentin (leader), Jérôme, Thibault, and Victoria.

4. **Backend**

Role: At this point, converting game's proofs to logical proofs that are checkable.

Members: Dina, Firmin, Guillaume (leader), Justine, and Paul.

5. **Research**

   Role: see below.

   Members: Dina, Guillaume.

The schedule is for now the following. The midterm break is not precised but it will be later.

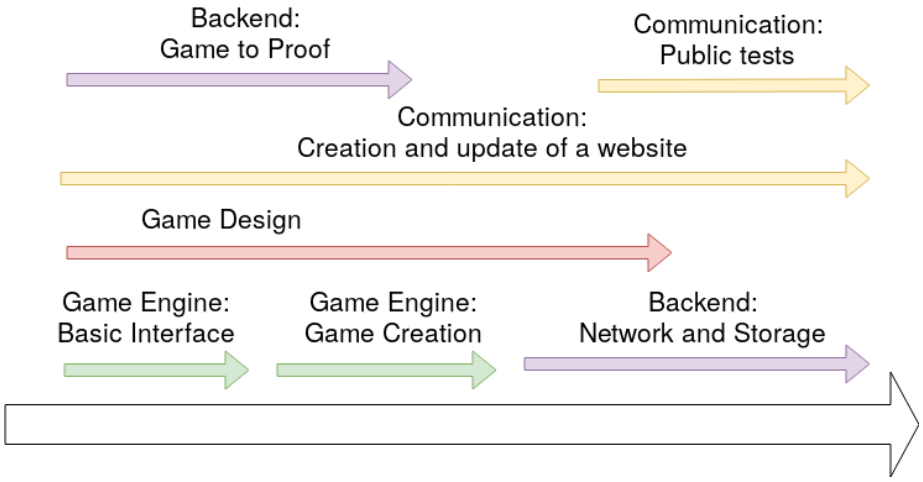


Figure 3: Initial schedule of the work packages

5 Resources Requirements

In order to develop this project we have some requirements:

- A server which is accessible online to host the website, the database of the game and the back-end application.
- Transport fees for invited researcher(s) who worked on the core article.
- A graphic artist who could improve the homemade images of the game.
- Apple Store fee (99$/year) and Google play one-time fee (25$).

6 Goals and Evaluations

This section describe the goals we would want to reach and how we evaluate the resulted software.

6.1 Goals

The main goal of this project is to implement a crowdsourcing game for theorem proving based on [Gov13]. In addition of what the author has already done, we would implement the game in a more general case which is first order logic.

Furthermore, we would implement possible improvements mentioned in [Gov13], including:

- Increase gameplay experience (*cognitive validation*): the crowdsourcing application should have industrial-level stability, excellent user interface, and, improved gameplay features in order to be attractive [Gov13, p.8].
- Several problems are raised by the author inside the future scope section. They include but are not limited to:
  - Dividing up large/hard problems with **preprocessing** ;
  - Convert the game into a **multi-player game** by adding, for instance, a competitive mode ;
  - Incorporating more rules of inference.

6.2 Evaluation

- Benchmarking on TPTP problem library [Sut17] by comparing proof length with Automated Theorem Provers (ATPs).
- Evaluate gameplay experience through methods described in [NDG10], such as questionnaire or qualitative interviews.

7 Mid-term progress

7.1 Communication

Progress made so far: Create a website based on the CMS WordPress. This one is hosted on the server of Jérôme: `https://forml.rudelune.fr/` because the web server of the ENS has outdated versions of PHP, etc.

We also sent some mails to Dr. Govindarajulu and asked him some details about his project and what he could advise us for our project.

Further goals: Fill the website with some articles and details about the project.

7.2 Game Design

Progress made so far: Design several graphical models for the game, and choose one. Detailed information are in the report of the work package, in appendix A. Figure 4 sums up the three models developed compared this the model described in [Gov13]. The model chosen was the third.

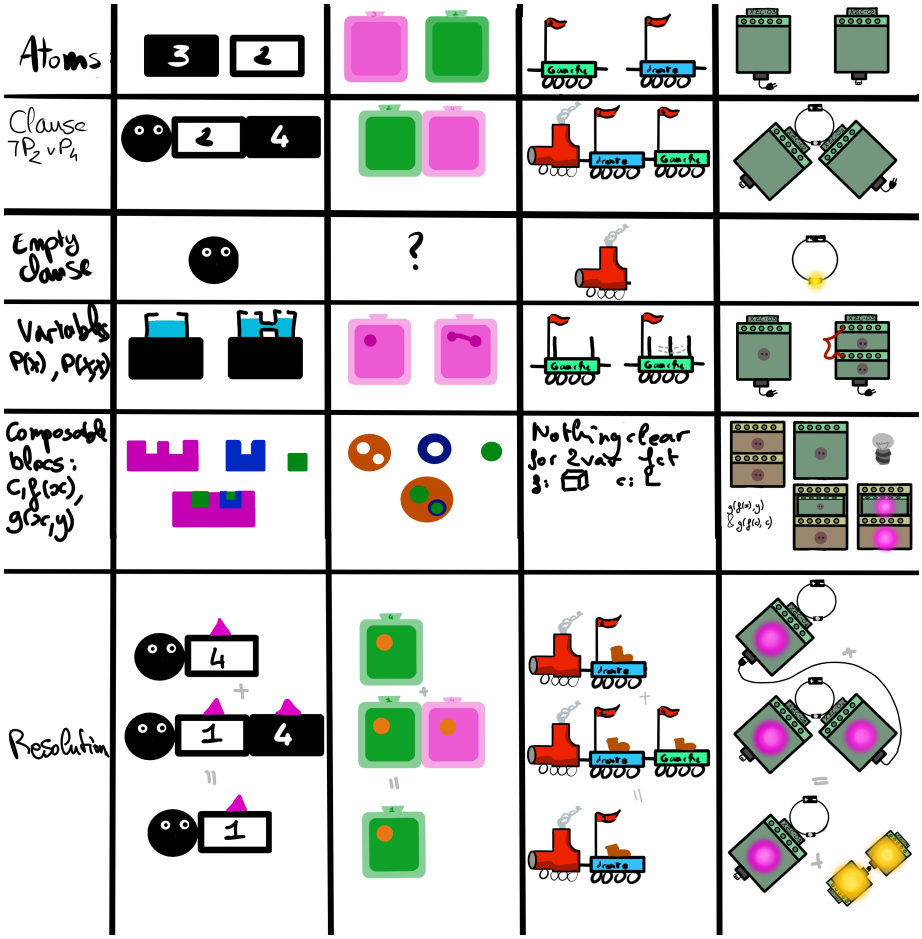


Figure 4: Table summarizing the inductive representation of FOL in the different models. Catabots model designed in [Gov13] are presented as well as an element of comparison

Further goals: Contact graphic artists, end of the work package.

7.3 Game Engine

- **Interface**:

  Progress made so far: We have chosen to use Godot, a game engine which allows us to be focused only on the game part. Then we learned how to use it with the official guide. We want to implement in a first part only the propositional logic, like the original game CatabotRescue. We already have the beginning of a such game with some basic objects so the project is modular. We have configured Godot such that it is possible to export the game on phones and on browsers. This work package does

not have a lot of things to explain because it is mostly understanding a library and using it.

Further goals: Continuing the integration of the game on Godot. We will also start to explore the backend part with the server side of the game.

- **Storage management**: As the game engine was the priority during this first half of the class, we have not yet started to focus on storage management

7.4 Backend

Progress made so far

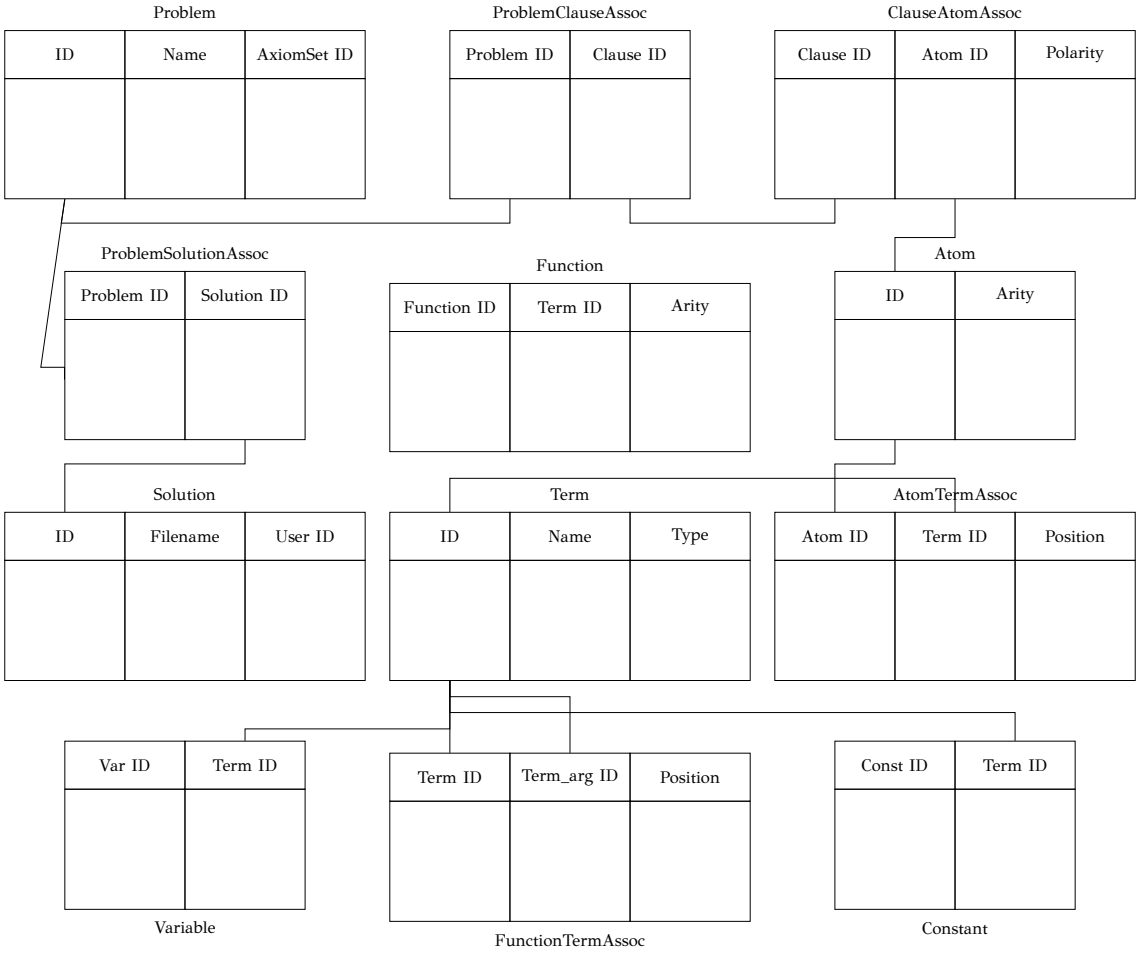
In the initial stage, the WP has investigated the possibility to check the correctness of a proof generated by the game. In order to do so, we first planned to use ANTLR4 as a parser, since there is already a TPTP ANTLR4's grammar implemented. Then, we decided to pick the automatic theorem prover Z3 in view of its built-in functions we can use to proof-check and preprocess formulae. After playing a while with Z3, we observed that Z3 can indeed produce a model for satsfiable formulae, or a proof for unsatisfiable ones as one can expect for an ATP. However, it cannot read back its proof (as stated in `https://stackoverflow.com/a/49882310/8274517`) and it's therefore impossible to use it directly to check proof correctness without third-party tool, such as interactive theorem prover Isabelle. The most relevant article [Böh] we can refer to remains technical for total beginners in Isabelle. To acquire more insight, we have inquired the mailing-list of Isabelle to know if a built-in resolution-based proof-checker is available and whether there is a specific proof format. The main purpose of this inquiry was to estimate whether we should go with a Z3 & Isabelle solution or directly implement our proof-checker inside Isabelle. The reply of Dr. Nipkow (co-author of Isabelle) confirmed that such format does exist (proof term format specified in [BN00]) and the possibility to write custom proof-checker as [Böh] does. Consequently, it seems that the work mentioned in [Böh] is not integrated in Isabelle. We will contact Dr. Böhme when needed to know where we can find the implementation.

Even though ensuring the correctness of proof generated by the game is important, since we don't want to allow the reuse of false proofs, we can assume that there is no way to cheat the proof generation process for a first glimpse and move on to more prioritised tasks (see below).

Further goals

Preprocess TPTP problems and import them in SQL database. We plan to use TPTP utilities `tptp2x`, `tptp4x` instead of Z3 to preprocess formulae before storing them in the SQL database.

The issue of how to store problems in our server was hard to handle: we needed a format that was easily readable, and did not need an entire parser. Eventually we decided to store our problems in an SQL database, which would contain the data for the different clauses that make up our axioms and problems. The following figure summarises the structure of the SQL tables in which we would store our problems.



Provide server-side APIs. As the game engine team got nearer and nearer to having an operational game, it seemed a priority to know how we would handle communications between the game and the back-end. We decided to provide an API for the game engine team that would allow them to interact with the back-end: load some problem and its axioms, load some user data, store a proof, store a game save, etc...

Proof-checking. We should have an agreement with the game engine work package on the proof format generated by the game. The best would be Isabelle's proof term format for forward compatibility. But Lisp-like S-expression would be enough to later conversion.

7.5 Research

Progress made so far

We created this work-package during the semester, for several purposes. First, compiling a short class on first-order logic, explaining the basics as well as the processes of clausification and resolution. Secondly, we want to look into extensions of resolution that are present in automated provers, that we may want to implement in the game, to add more gameplay elements as well as more proving power.

Further Goals

As mentioned, we want to look into extensions of resolution. A possible concept to look into is Deduction Modulo [Bur11], which would yield some very interesting gameplay elements. However, for each extension we add, we also need to adjust our eventual proof-checker, as all checkers are not familiar with all existing variations of resolution. Another future goal of this work-package is that of writing some articles about the project : explaining how the game works with regard to first-order resolution, what crowd-sourcing is, and perhaps more. Finally, as we want the game to be playable by humans, we should look into what is called ”preprocessing”: breaking down formulas in order to get simpler clauses, and therefore simpler game instances. This preprocessing should then be implemented in the back-end.

References

[AD04] Luis von Ahn and Laura Dabbish. “Labeling Images with a Computer Game”. 2004.

[BN00] Stefan Berghofer and Tobias Nipkow. “Proof terms for simply typed higher order logic”. 2000.

[Böh] Sascha Böhme. *Proof Reconstruction for Z3 in Isabelle/HOL*.

[Bur11] Guillaume Burel. “Experimenting with deduction modulo”. July 2011.

[Gov13] Naveen Sundar Govindarajulu. “Uncomputable games: games for crowdsourcing formal reasoning”. 2013.

[NB16] G. NaveenSundar and Selmer Bringsjord. “Crowdsourcing Theorem Proving via Natural Games”. 2016.

[NDG10] Lennart Nacke et al. “Methods for Evaluating Gameplay Experience in a Serious Gaming Context”. 2010.

[Sut17] G. Sutcliffe. “The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0”. 2017.

Appendix A: Game design models

In this section we present the three main graphical concepts/models which have been explored while working on the project's graphical user interface (GUI) design, and we expose the pros and cons which lead us to our final choice. For each model we develop all technical aspects of representing first-order logic in a graphical way and the story-related aspects we imagined in relation to the graphical concept and the in-game objective for the player.

Introduction

Just a little reminder of the dynamics the game needs before presenting the graphical interfaces. The game hides logical features, among them:

- Variables (x, y, *etc.*).
- Constants (that variables instantiate into).
- Relations (like $R(x, y)$) which take variable as arguments.
- Functions (take variables in arguments and can be arguments of another function or relations).
- The "not" operator ($\neg$)
- Atoms: One relation with its variables/functions/constants that could be behind a "not" (*e.g.* not($R(f(x), n, x)$))
- Clauses (that are conjunction of relation(s) and their variable(s) instantiated or not)
- Empty clauses: clauses with no relations in it. The goal of the game is to create an empty clauses.

Their are three logical dynamics that the game must translate:

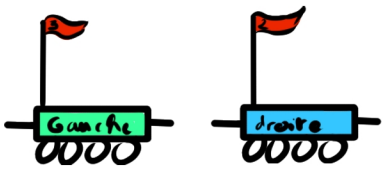
- Instantiating and Linked variables: a variable x could appear several times in a clauses, and instantiating x must affect all of its appearance.
- Resolution rules: associating two specific atoms that are in two different clauses (let say c_1 and c_2). This creates a new clauses that contains all the atoms of c_1 and c_2 except the associated atoms.

Trains

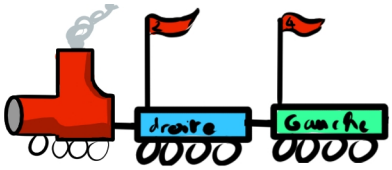
The overall idea of this model is to depict clauses as freight trains. The goal of the player would be unloading trains by combining compatible trains. An explanation to the necessity of unloading trains two by two is that trains be loaded with goods from different locations, which a company combines to make new products out of them, *e.g.* combining left-foot shoes with right-foot shoes into a matching pair of shoes.

Representing FOL

Atoms Every atom is depicted as a train wagon. Positive atoms are represented by a left shoe wagon, and negative atoms are represented by a right shoe wagon. Each relation has a number, which is indicated on the flag.

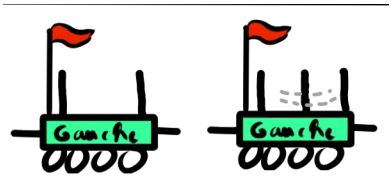


Clauses Clauses are depicted by a train locomotive followed by some wagons. The wagons are ordered by the number on the flags.



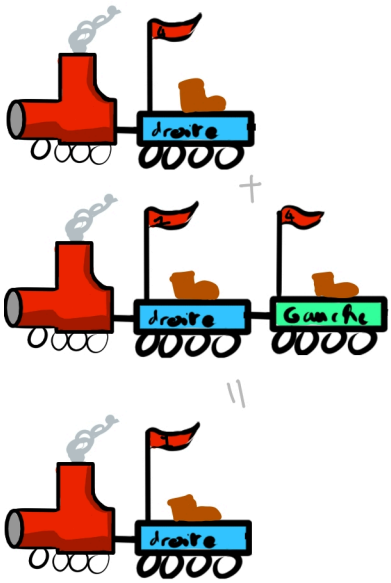
Empty clause The empty clause is represented by a train locomotive on its own.

Variables Variables are containers and we did not figure out how to do clean connections for same variables in the clause. This is why we then build our representation starting with the connections (board representation).



Composable items: functions and constants This is a weakness of the model, as we could use shoes as the constants, but it was very hard to come up with clear functions. We thought of having numbered crates but it was hard to show what was inside them.

Depicting the resolution rule



Doing logical elimination of atoms (resolution rule) was done by putting the two wagons next to a crane, which would unload them. The two remaining trains would then be fused into one to have a unique remaining clause. The parent clauses would be kept in a menu if needed again.

Narrative

The idea is that a merchandise usually found in pairs is separated, and should be reunited. The player needs to unload a train completely in order to win, creating the empty clause needed in resolution.

Pros and cons: summary

Pros :

- The narrative is nice.
- The trains would have been put on tracks, which makes it easier to keep a clean screen.

Cons :

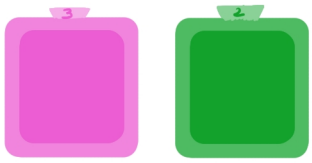
- Bad representation of linked variables
- Problem with functions
- The name of the wagon is in small writing, so is the left/right (to know if the atom is positive or negative)

Slime creatures

In this model the player controls a tribe of slime creatures (more simply called "slimes" in the rest of this section) which have items stucked in their bodies. The concept would be digesting or extracting those annoying foreign bodies.

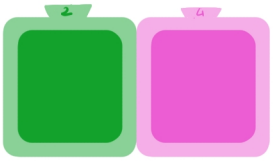
Representing FOL

Atoms



Every atom is depicted as a slime. Negative atoms can be depicted as differently-colored slimes.

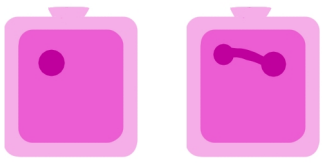
Clauses



Clauses are depicted as slimes agglomerates or pluricellular slimes.

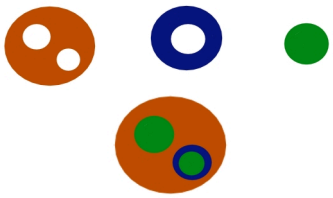
Empty clause (Model weakness.) As we make a correspondence between atoms and slimes, and clauses are depicted as slimes agglomerates, it is unclear how to depict an empty clause in a consistent way.

Variables



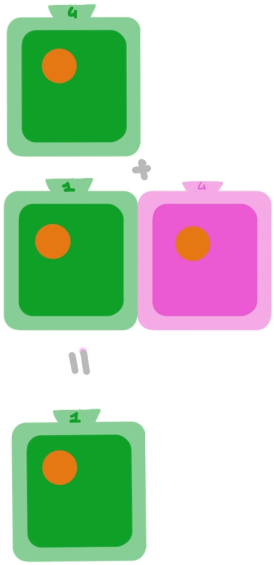
We depict variables as stomach-like organs inside the slime's body, which can be filled with items. Linked variables are depicted as organs connected by a duct, as shown in the right example of the above picture.

Composable items: functions and constants



Constants are depicted as coloured disks that can be dragged into a variable slot to instantiate it. Functions are depicted as holed spheres, with one hole per argument.

Depicting the resolution rule



We associate two clauses which include matching instantiated atoms (one positive and one negative). This step causes the "cleansed" slimes to disappear from the GUI.

Narrative

An interesting point about the "slimes" graphical concept was that we could build a narrative about them. We thought about the story of a tribe of slimes which accidentally gather detritus inside their (semiliquid) bodies as they travel through a polluted area. As this becomes a

nuisance, the tribe of slimes tries to get rid of these foreign bodies, which are depicted as variables. The player's role would then be to help them figure out how to get rid of it.

We have thus considered depicting variables, functions and constants as different kinds of garbage pieces. The matching idea of the resolution rule might then be depicted as matching items such as a can and a can-opener. However, this would lead to a more complicated and possibly obscure user interface, and it is unclear how to depict composition cases implicating functions: if we depict composition as boxes, canisters, etc., how can the player tell a variable instantiated as a function of some constants from ?

On the other end, simplifying the graphical design to clarify the GUI means distancing from the graphical characteristics which support the universe and narrative built around the concept, which is a pity, and makes the depiction of the resolution rule more artificial.

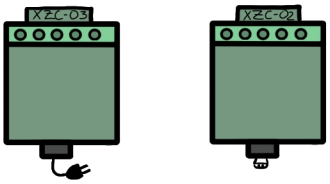
Pros and cons: summary

- This model is primarily lead by the idea of creating a narrative and a game universe to create more fun for the players.
- However, it is difficult to keep a good balance between GUI clarity and a graphical design which consistently supports the narrative built around the concept.
- It is furthermore unclear how to depict an empty clause.

Circuit boards

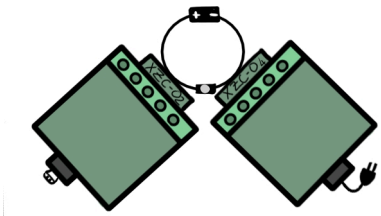
Representing FOL

Atoms



A small electrical block represents a relation. The name of the relation is written on top of it. An atom is equipped with an electrical wire if it's negative (*i.e.* behind a "not" operation), or with a plug if it is positive (in the example, the negative block in on the left, and the positive on the right).

Clauses

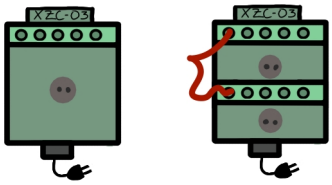


To create a clause we connect all its atoms on a common circuit, that forms a circle. Here, the clause has two atoms.

Empty clause

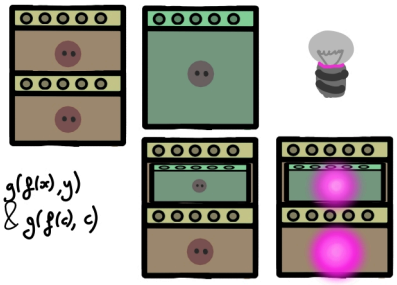
A circuit that made a clause also posses a battery and a small lamps. As long as there is atoms in the circuit, the battery cannot power the lamps. Once there is no more blocks, only the battery and the lamp remain and the lamp light up.

Variables



On block that represent relations, it is possible to connect composables items (lamps, other blocks...). The numbers of plugs to connect those on the block is the number of arguments the relation takes. On the left, the relation takes one argument, and ,on the right, the relation takes two arguments. On top on each variable, there is a pluging bench. When two variables are linked (the same variable appears two times in the logical formula),the associated bench are connected with a wire. On the example, the right-side block posses two variables linked with a wire. It thus translates a logical formula of the form $R(x, x)$.

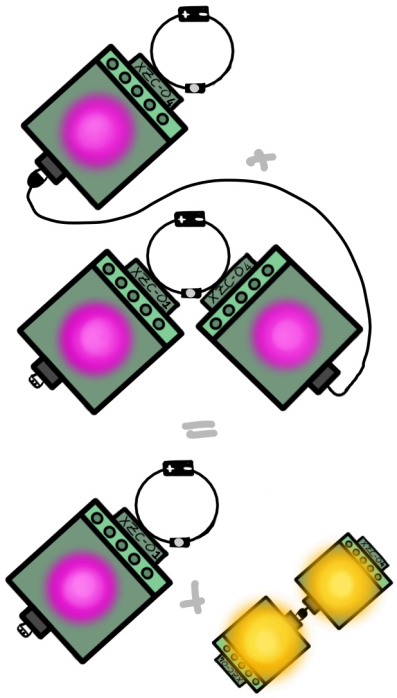
Composable items: functions and constants



In this construction, function are smaller block, that can be plugged in another block (that could be a relation or another function)? Note that functions differ from relations block, since relation block has a device to connect them with each other. Furthermore, variables are lamps.

To instance a variable is then equivalent to connect the lamps to a relation's or a function's plug. The lamp then can access power and lights up, which gives enlighten block, such as the example block on the bottom right.

Depicting the resolution rule



Doing logical elimination of atoms (resolution rule) is equivalent to connect one relation block to another by connecting the electrical wire of one atom to the bottom plug of the other. When connecting, they become autonomous and both enlighten and can be remove of the clause (which gives the yellow device in the example) and of the game. The wires of the remaining clause merges and keep all the remaining atom.

In the example, the transformation made is of the form $c_1 + c_2 \rightarrow c_3$ where:

- $c_1 = R(n)$
- $c_2 = Q(n)) \wedge \neg R(n))$
- $c_3 = Q(n)$

Narrative

As it has been said, a circuit that made a clause posses a battery and a small lamps. As long as there are electrical blocks in the clause, the power from the battery is "stolen" by these blocks. The objective is to light one small lamp, and thus, to remove, in at least one clause, all the blocks,

aka to reach the empty clause. Blocks can be connected to one another : some have power outlet and other have electrical wire to connect to outlet. But so as to match, two blocks as to be identical. Once connected, the two blocks fuel one another and the pair becomes stand-alone. The pair can thus be removed from clauses' circuits.

Pros and cons: summary

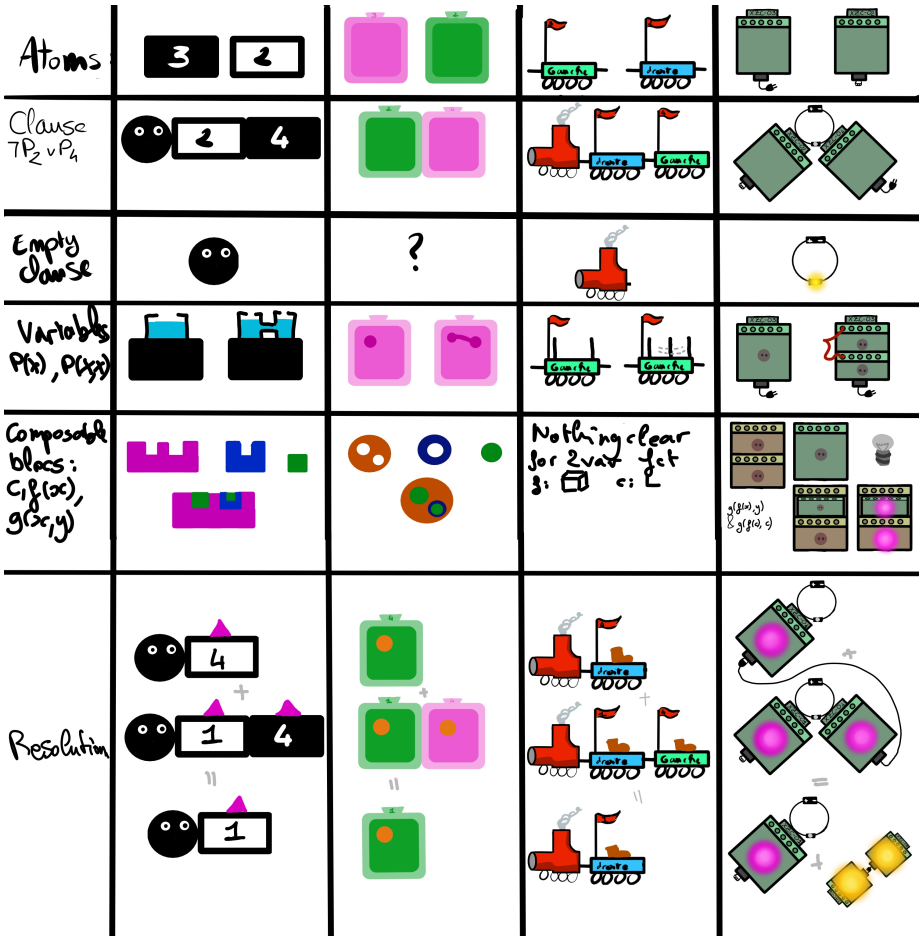
Pros :

- A clearer representation of linked variables
- Every logical features can be represented clearly.
- We don't have to move the clauses to connect, but just to insert electrical wires.

Cons :

- Name of relation written on blocks won't be easily read
- Narrative less user friendly that the first two models
- At every reduction, one lamp and one battery disappear

Summary, synthesis and final choice



Choice We eventually choose the third model

Additional features of the game

In this section, we are going to list briefly all feature the interface will need to make the game user-friendly.

- An inventory to store variables.
- An inventory to store clauses, like created clauses during the game, by a given order and by last seen.
- Feature to change size of clauses
- Feature to make the bunch rotate on itself