



Canada School
of Public Service

École de la fonction
publique du Canada



Rules as Code in Canada

Summary of Experiments and Lessons Learned

Public Sector Experimentation Team (PSX)

February 2024

What is Rules as Code (RaC)?

Rules as Code (RaC) is an innovative approach to rulemaking that encourages governments to create a trustworthy interpretation of rules in a machine-consumable form. When rules are encoded in a clear, accessible, consistent manner, and made publicly available, they can be used to power legal automation, simulation, and verification tools. This could significantly enhance public service delivery while simultaneously providing more transparency on government decision-making.

First Canadian Experiments (2019-2022)

The Canada School of Public Service (CSPS) has been experimenting with RaC tools and approaches since 2019. Like many other experimenters in this space, our first projects focused on converting existing regulations into code using the microsimulation tool OpenFisca. We selected rules that were descriptive, interconnected, expansive, and subject to relatively frequent change (as opposed to rules that are subjective, self-contained, circumscribed, and static) and assembled a multidisciplinary team of experts to help transcribe their meaning.

Discoveries

These initial projects supported our hypothesis that writing laws in computer languages could significantly enhance public service delivery. For example, writing rules into code helped reveal gaps, loopholes, and ambiguities that may go unnoticed when reading and writing the natural language on its own. If these rules were shared publicly via API (Application Programming Interface), it could help others implement them in a more streamlined, comprehensive, and consistent manner. In short, encoded rules make legal automation, simulation, and verification possible.

Lessons Learned

However, we also identified some major obstacles that made the process long, arduous, and unsustainable. For example, deciphering the meaning of existing rules to convert them into code was difficult. Furthermore, the interdisciplinary nature of the workgroup (e.g. drafters, subject matter experts, coders) presented communication challenges. Finally, it was very difficult to capture the whole meaning of a law using imperative programming languages like OpenFisca; this forced the programmers to make difficult decisions on whether to encode less than the whole rule, or to do a lot more work to capture its whole meaning.

Summary

Our first experiments convinced us that rules should probably be encoded by the rule-makers (those who know them best) as they are being created. However, most RaC tools are designed for programmers; they aren't easily accessible for legal and policy professionals who rarely have backgrounds in the computer sciences. The federal public service has a unique combination of technical, judicial, and subject matter expertise to fill this user-developer gap. As such, we have set out on a mission to leverage this network towards building open-source and user-friendly RaC tools designed for rule-makers.

This has led us to the following two innovations:

1. A user-friendly RaC tool called Blawx ; and
2. A rule-drafting methodology that incorporates code.



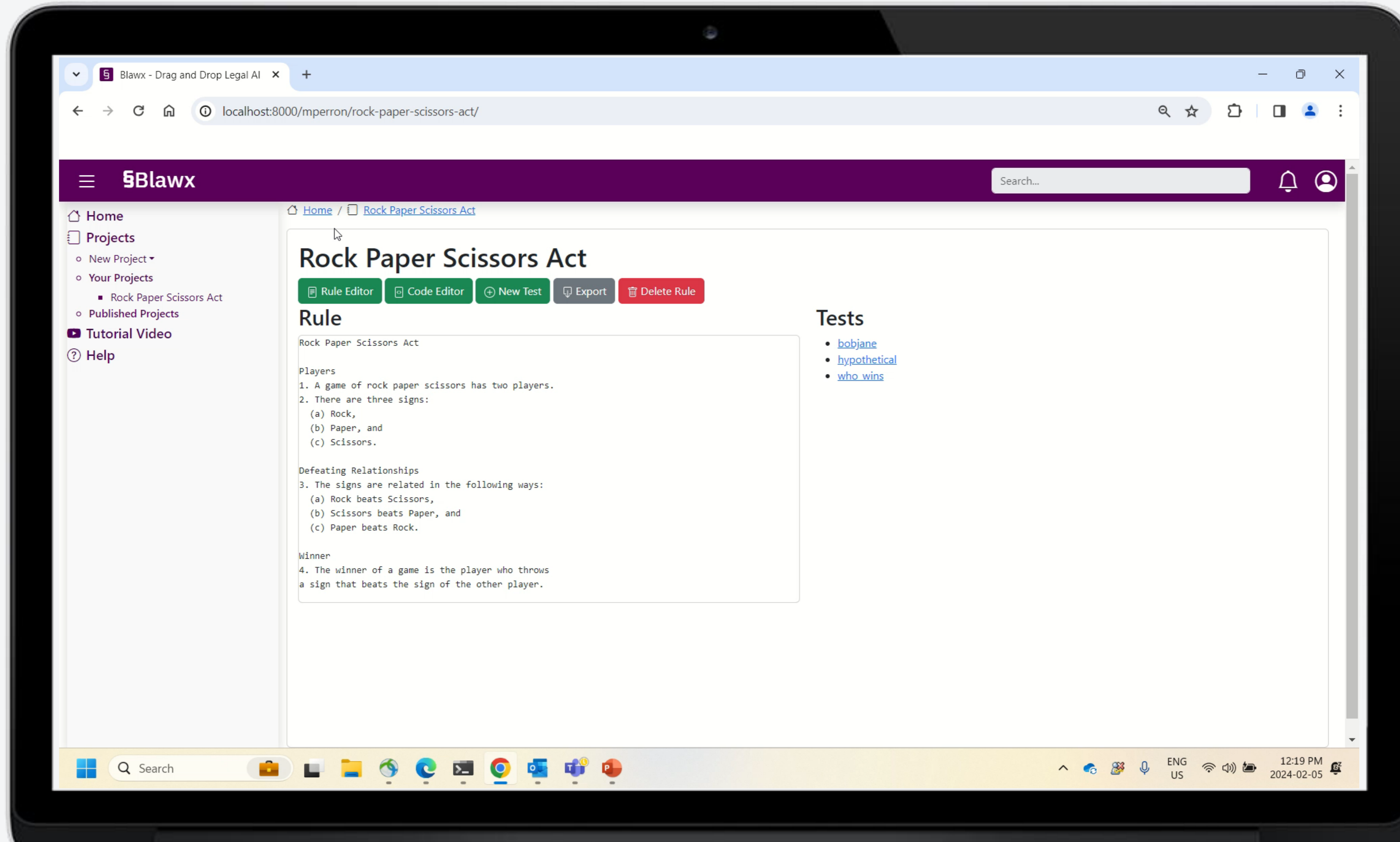
Innovation 1 | Blawx

Blawx

Our first innovation – Blawx – is an open-source and user-friendly programming tool designed specifically to help non-programmers encode, test, and use rules. Blawx is powered by a predicate declarative logic software called s(CASP) and overlaid with a visual programming interface (Blockly). It has a user-friendly simulation interface, provides detailed explanations for answers, and can execute hypothetical reasoning tasks. We are developing Blawx in collaboration with Canadian public service professionals, so that it can help fill the developer-user gap we have identified in the RaC space.



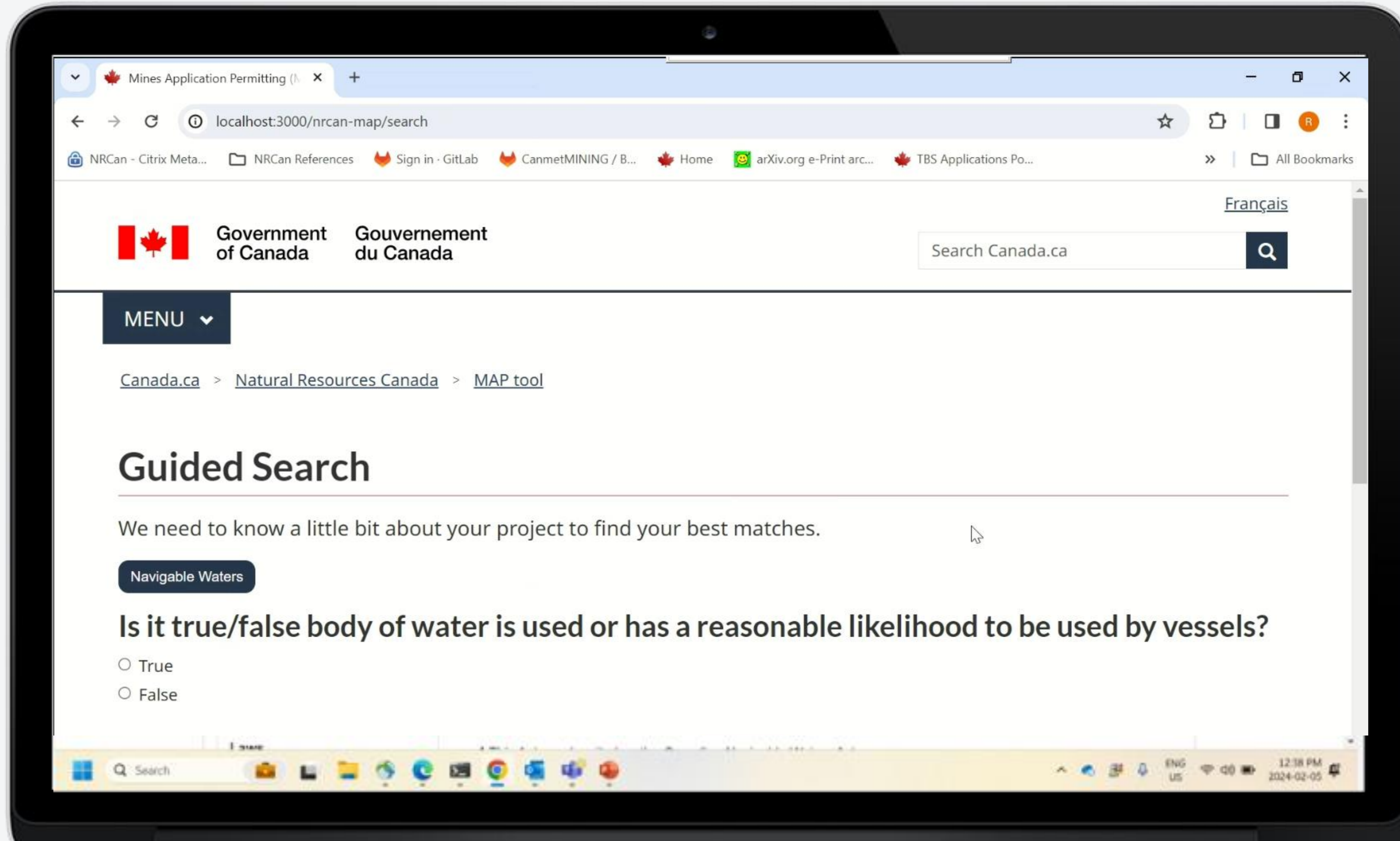
Demo | *The Rock Paper Scissors Act*



Demo | *The Canadian Navigable Waters Act*

The screenshot shows a web browser displaying the Canadian Navigable Waters Act page on the Justice Laws Website. The browser's address bar shows the URL: laws-lois.justice.gc.ca/eng/acts/n-22/page-1.html#h-364602. The page header includes the Government of Canada logo and navigation links for Canada.ca, Services, Departments, and Français. The main navigation menu features categories like Family Law, Criminal Justice, Funding, and Canada's System of Justice. A search bar is located in the top right corner. The breadcrumb trail reads: Home → Laws Website Home → Consolidated Acts → R.S.C., 1985, c. N-22 - Table of Contents → R.S.C., 1985, c. N-22. The main content area displays the title "Canadian Navigable Waters Act (R.S.C., 1985, c. N-22)" and provides links for the full document in HTML, XML, and PDF formats. It also indicates that the act is current to 2024-01-14 and last amended on 2019-10-04. A sidebar on the left lists "Constitutional Documents" including the Canadian Charter of Rights and Freedoms, and various consolidation reports. The page title is "Canadian Navigable Waters Act" and the subtitle is "R.S.C., 1985, c. N-22". The description states: "An Act respecting the protection of navigation in Canadian navigable waters". The "Short Title" section is partially visible at the bottom.

Demo | Blawx API (Application Programming Interface)





Innovation 2 | Code-Assisted Regulatory Drafting

PSSA 3(1) – Definition of Salary Regulation

Our second innovation – a rule-drafting methodology – was developed in collaboration with the Treasury Board of Canada Secretariat (TBS). TBS wanted to apply a RaC approach to help draft a “Definition of Salary” regulation that clearly outlines and integrates desired policy outcomes under s.3(1) of the *Public Service Superannuation Act*. This was an excellent case study for RaC, as these rules were complicated, and applied to different collective agreements and payment codes.

salary means

(a) as applied to the public service, the basic pay received by the person in respect of whom the expression is being applied for the performance of the regular duties of a position or office exclusive of any amount received as allowances, special remuneration, payment for overtime or other compensation or as a gratuity unless that amount is deemed to be or to have been included in that person’s basic pay pursuant to any regulation made under paragraph 42(1)(e), and

(b) as applied to the regular force or the Force, the pay or pay and allowances, as the case may be, applicable in the case of that person as determined under the *Canadian Forces Superannuation Act* or the *Royal Canadian Mounted Police Superannuation Act*; (*traitement*)

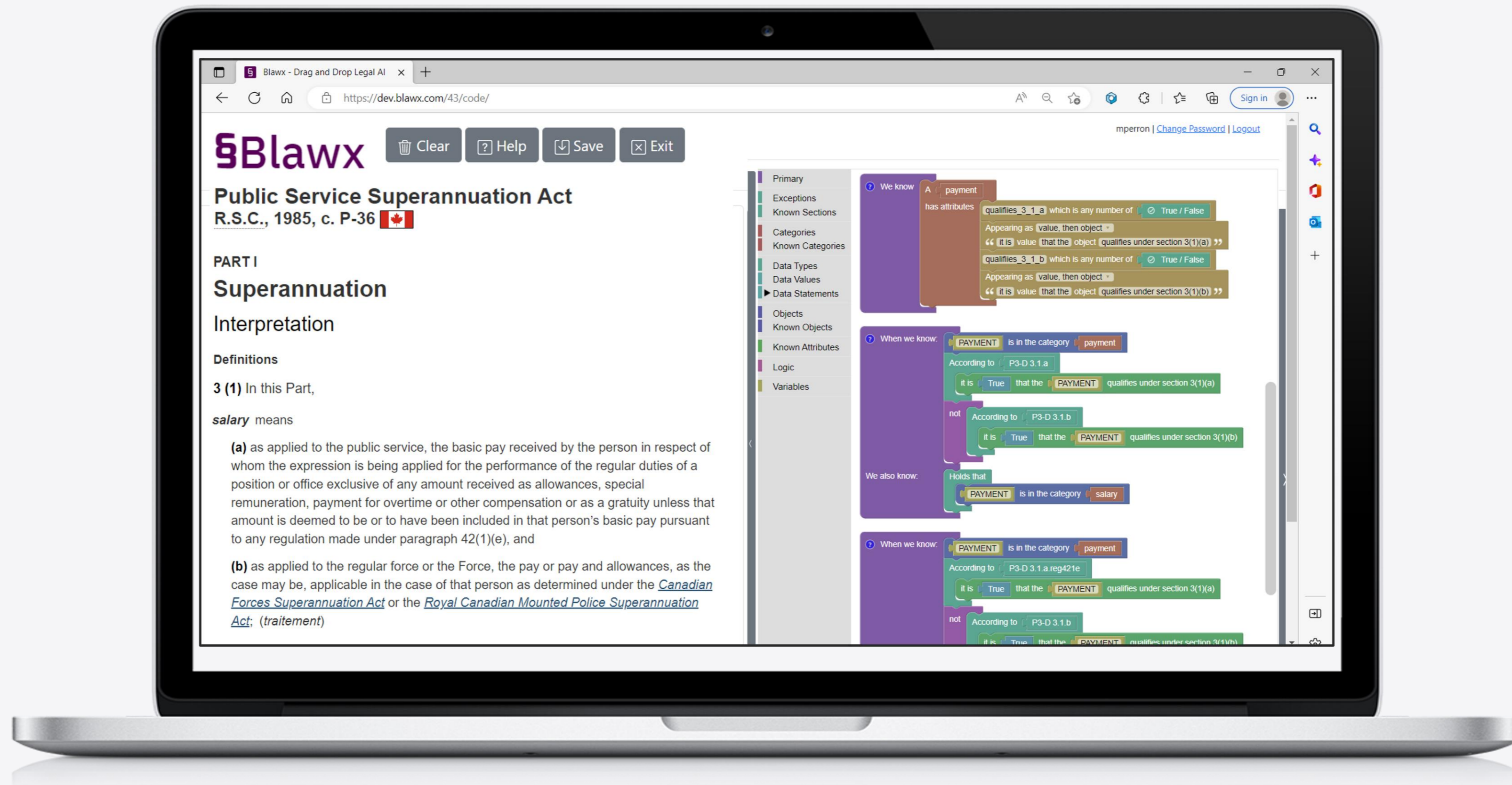
traitement

a) La rémunération de base versée pour l’accomplissement des fonctions normales d’un poste dans la fonction publique, y compris les allocations, les rémunérations spéciales ou pour temps supplémentaire ou autres indemnités et les gratifications qui sont réputées en faire partie en vertu d’un règlement pris en application de l’alinéa 42(1)e);

b) la solde, ainsi que les allocations, payables dans le cadre de la force régulière ou de la Gendarmerie en vertu de la *Loi sur la pension de retraite des Forces canadiennes* ou de la *Loi sur la pension de retraite de la Gendarmerie royale du Canada*. (*salary*)

Writing Drafting Instructions in Code

In the first stage of this experiment, we used RaC to test whether a regulation was the proper policy tool to meet the desired outcomes for TBS. Using our Blawx prototype, we co-drafted the policy proposal into code and tested it with subject matter experts. By making the proposed rules machine-consumable, we could run them through a series of fact scenarios and use our findings to guide the policy work. Overall, this exercise helped us better prepare drafting instructions.



Project Dashboard: Provides the user access to all the relevant tools for encoding and testing a set of rules.

The screenshot shows a web browser window with the URL `localhost:8000/mperron/pssa-reg-v3/`. The application header is purple and contains the Blawx logo, a search bar, and user profile icons. A left sidebar lists navigation options: Home, Projects (with sub-items for New Project, Your Projects, and Published Projects), Tutorial Video, and Help. The main content area is titled "PSSA + Reg (v3)" and includes a toolbar with buttons for Rule Editor, Code Editor, New Test, Export, and Delete Rule. The "Rule" section displays the text: "PSSA 3(1) - Definition of salary with new regulations included", "3. Interpretation", "(1) Salary means," followed by sub-points (a) and (b). Below this is a section for "Regulations -" which is currently obscured by a black redaction box. On the right, a "Tests" section lists various test identifiers such as `3_1`, `3_1_a`, `3_1_a_exception`, `3_1_b`, `reg_1`, `reg_1_a`, `reg_1_b`, `reg_1_c`, `reg_1_d`, `reg_1_e`, `reg_1_f`, `reg_2`, `reg_2_a`, `reg_2_b`, `reg_2_c`, `reg_2_d`, `salary_test`, and `act_test`.

Code Editor: Convey the meaning of each section of law into code using Blawx's drag-and-drop block interface

The screenshot displays the Blawx web application interface. At the top, a browser window shows the URL `localhost:8000/mperron/pssa-reg-v3/code/`. The application header is purple with the Blawx logo on the left and a search bar on the right. A left sidebar contains navigation links: Home, Projects (with sub-links for New Project, Your Projects, and Published Projects), Tutorial Video, and Help. The main content area shows a breadcrumb trail: Home / PSSA + Reg.(v3) / Section. Below this is a tree view of a legal document structure:

- PSSA 3(1) - Definition of salary with new regulations included
 - 3 Interpretation
 - (1) Salary means,
 - (a) as applied to the public service, the basic pay received by the person in respect of whom the expression is being applied for the performance of the regular duties of a position or office exclusive of any amount received as allowances, special remuneration, payment for overtime or other compensation or as a gratuity unless that amount is deemed to be or to have been included in that person's basic pay pursuant to any regulation made under paragraph 42(1)(e), and
 - (b) as applied to the regular force or the Force, the pay or pay and allowances, as the case may be, applicable in the case of that person as determined under the Canadian Forces Superannuation Act or the Royal Canadian Mounted Police Superannuation Act; ("traitment").

Below the tree view, there are tabs for 'Blawx' and 's(CASP)'. The 'Blawx' tab is active, showing a code editor with two columns of blocks. The left column, titled 'We know', contains blocks for defining categories and their attributes:

- salary is a Category
Appearing as: "object is salary"
- person is a Category
Appearing as: "object is a person"
- basic_pay is a Category
Appearing as: "object is basic pay"
- The category person has an attribute public_servant which is of type true / false, appearing as "object is a public servant"
- The category payment has an attribute received_by which is of type person, appearing as object, then value "object is received by value"
- The category payment has an attribute received_for_reg_duties which is of type true / false, appearing as "object was received for the performance of the regular ..."

The right column, titled 'When we know:', contains blocks for defining relationships and rules:

- Person is in the category person
- Payment is in the category payment
- Person is a public servant
- Payment is received by Person
- Payment was received for the performance of the regular duties of a position or office
- We also know that according to P3-D 3.1.a
- Payment is in the category basic_pay
- subject to exceptions (checked) / subject to applicability (unchecked)

Test Questions: Once rules have been encoded, craft a question. This question will be used in the Scenario Editor to test the code.

The screenshot shows the Blawx web application interface. At the top, a browser window displays the URL `localhost:8000/mperron/pssa-reg-v3/test/salary_test/`. The application header is purple and contains the Blawx logo, a search bar, and user profile icons. A left sidebar lists navigation options: Home, Projects (with sub-items: New Project, Your Projects, Published Projects), Tutorial Video, and Help. The main content area shows a breadcrumb trail: Home / PSSA + Reg.(v3) / salary_test. Below this, there are tabs for Blawx, Answers, Problems, and s(CASP). A vertical menu on the left lists various ontology elements: Primary, Exceptions, Known Sections, Categories, Events, Numbers, Dates, Lists, Objects, Known Objects, Known Attributes, Known Relationships, Logic, and Variables. The main workspace contains a question editor with the text "Is it true that:" followed by a blue box containing "Payment" and a dropdown menu set to "salary".

Scenario Editor: Simulate various legal scenarios by providing a set of facts to the test question.

The screenshot shows the SBlawx Scenario Editor interface. The browser address bar displays the URL: localhost:8000/mperron/pssa-reg-v3/test/salary_test/scenario/#. The application header includes the SBlawx logo, a search bar, and navigation icons. The left sidebar contains a navigation menu with options: Home, Projects (New Project, Your Projects, Published Projects), Tutorial Video, and Help. The main content area is titled 'Scenario Editor' and features tabs for 'Facts', 'Answers', 'View', and 'Devel'. Under the 'Facts' tab, a list of facts is shown, each with a delete icon (X):

- amount_X is a payment X
- bob is a person X
- bob is a public servant X
- amount_X is received by bob X
- amount_X was received for the performance of the regular duties of a position or office X

Below the list is a fact construction tool. It consists of several input fields and buttons:

- 'It is' followed by a dropdown menu currently set to 'true that'. A dropdown menu is open showing options: 'true that', 'false that', and 'uncertain whether'.
- A dropdown menu set to 'amount_X'.
- The text 'is received as an allowance'.
- Two checkboxes: a green one with a checkmark and a red one with an X.
- A blue '+Add F' button.
- A green 'Save Default Fact Scenario' button.
- A blue 'Run' button.

Test Answers: If the question can be answered based on the facts provided, you will receive all relevant answers and explanations.

- Home
- Projects
 - New Project
 - Your Projects
 - PSSA + Reg (v3)
 - Published Projects
- Tutorial Video
- Help

Facts | **Answers** | View | Devel

Answers

Answer #1 ^

- Payment: amount_X

Explanation #1 v

Explanation #2 v

Explanations: Each answer includes a detailed breakdown of the legal reasoning for the rule-makers to vet.

Explanation #1

Getting AI Summary...

Details

We know amount_X is salary because

- it holds that according to [section 3 subsection 1](#), amount_X is salary^[i].

We know it holds that according to [section 3 subsection 1](#), amount_X is salary because

- according to [section 3 subsection 1](#), amount_X is salary^[i] and
- there is no evidence that the conclusion in [section 3 subsection 1](#) that amount_X is salary is defeated^[i].

We know amount_X is a payment, which was provided as a fact.

We know it holds that according to [section 3 subsection 1 paragraph a](#), amount_X qualifies under this paragraph because

- according to [section 3 subsection 1 paragraph a](#), amount_X qualifies under this paragraph^[i].

We know it holds that according to [section 3 subsection 1 paragraph a](#), amount_X is basic pay because

- according to [section 3 subsection 1 paragraph a](#), amount_X is basic pay^[i] and
- there is no evidence that the conclusion in [section 3 subsection 1 paragraph a](#) that amount_X is basic pay is defeated^[i].

There is no evidence that it holds that it is false that according to [section 3 subsection 1 paragraph a span exception](#), it is not true that amount_X is basic pay because

- there is no evidence that according to [section 3 subsection 1 paragraph a span exception](#), it is not true that it is not true that amount_X is basic pay^[i].

There is no evidence that according to [section 3 subsection 1 paragraph a span exception](#), it is not true that it is not true that amount_X is basic pay because

- amount_X is a payment^[i]
- there is no evidence that amount_X is received as an allowance^[i]
- there is no evidence that amount_X is received as special remuneration^[i]
- there is no evidence that amount_X is received as overtime^[i]

Results, Outcomes & Impacts

By running simulations with our code, we were able to analyze the rules in a more informative and cohesive way than we would have been able to by relying solely on the natural language versions of the law. It also helped us identify elements that needed correction in the Blawx software. The rule encodings can now be reused in the drafting room to help the subject matter experts communicate with the legal drafters responsible for writing the regulations.

Next Steps

We have completed the policy exercise phase, and our next step is the regulatory drafting phase, where we will test the encodings alongside legal drafters. We believe the time we spent writing out the rules into code during the policy phase will help parties more efficiently and effectively develop comprehensive and clear rules in the drafting room. Furthermore, the encodings could be repurposed by future rule-makers when amendments are made at a later date.

While the initial rule encodings for this project were drafted by a lawyer-turned-developer, the latest regulatory text proposals were encoded by a pensions subject matter expert from the TBS. We are also exploring the possibility of supporting a regulatory drafter with how to respond to the proposed drafting instructions in code, using Blawx.



Key Considerations

Legal Status of the Encodings

Once the rules and their associated encodings have been finalized, we will investigate the feasibility of publishing our rule encodings for consultation in the Canada Gazette. However, it is important to note that the encodings should not be viewed as having equal legal status to the official rules. Rather, they are trustworthy interpretations of rules that can help communicate their meaning to citizens and stakeholders, as governments strive to improve service delivery.

Collaboration with Justice

We hypothesize that encoded rules can help those who implement (or are subject to) those rules more effectively conduct administration, evaluation, and compliance activities. We have started working with Justice's the *LSB AI & RaC Workgroup* to bring legislative expertise to our RaC work, and to mitigate legal risks associated with this practice. We are also exploring questions pertaining to cabinet confidences, solicitor-client privilege, and ways to improve Blawx so that it better serves the needs of legal drafters.

Conditions for Success

If you would like to use existing RaC tools within your organization, it is important to first identify the rules you would like to write into code, and then assemble a multidisciplinary team of professionals with skillsets in the areas of legislative drafting, symbolic AI, and generative AI. It is highly recommended that you get support from leadership in your organization so you can get the required resources for your RaC project. Invest your time in building the right pitch and presenting it to the right people.

Replication

Rules as Code is a relatively new practice, but there are experiments being conducted in this space around the world. However, what makes this case study unique is that we are using a RaC approach to craft new rules, not just converting existing ones. As with all RaC projects, the encodings are designed to be repurposed and reused in the future, making replication easy.

Notable Accomplishments

To our knowledge, this project marks the first time where:

1. Rules as Code has been used by Canada (or in Canada) to support legal/policy decision-making;
2. Answer set programming (ASP) is used for legal/policy decisions;
3. The code was written primarily by a non-programmer;
4. The code was deployed to answer the question "should a regulation be written for X"; and
5. The code is used in a real-world legislative task.

Key Lessons Learned

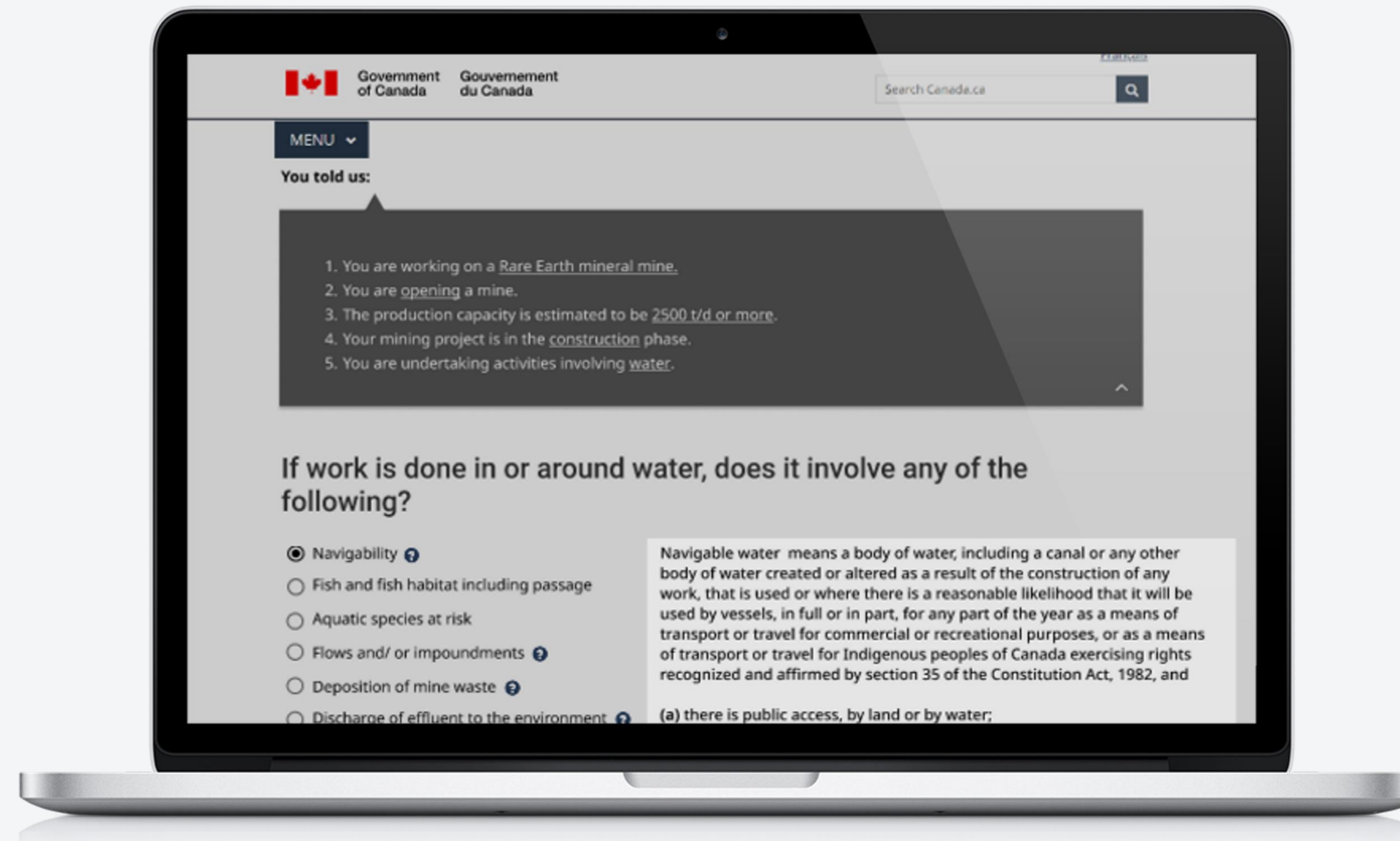
1. Encoded rules make legal automation, simulation, and verification possible.
2. Writing rules into code as they are being developed helps reveal gaps, loopholes, and ambiguities that may otherwise go unnoticed when reading and writing the natural language on its own.
3. Encoded rules are more trustworthy if they are written by the rule-makers, using tools that improve communication between participants.
4. Rules as Code can be conducted by non-programmers, but work is still needed to promote its benefits and make it more appealing to this audience.



Ongoing Projects & Developments

PROJECT | NRCan Canmet Mining MAP Tool

We are currently conducting a RaC experiment with Natural Resources Canada, where rules pertaining to the mining permit process are being converted into code. The Mining Application Permitting Tool (MAP Tool) intends to automate a set of complex legal processes by isolating rule encodings from the user interface that displays them. These rule encodings can then be linked to NRCan's website by the IT/Developer team. If the law changes, the new legal data can be easily integrated into NRCan's website without making major reconfigurations to the user interface.



Analyzing Multiple Laws At Once

The screenshot shows a laptop screen with a web browser displaying the Canadian Navigable Waters Act page. The browser's address bar shows the URL laws-lois.justice.gc.ca/eng/acts/n-22/FullText.html. The page features a blue navigation bar with categories like Family Law, Criminal Justice, Funding, and Canada's System of Justice. A sidebar on the left lists various legal documents and laws. The main content area displays the title "Canadian Navigable Waters Act" and "R.S.C., 1985, c. N-22", along with a short title and interpretation section. The Windows taskbar at the bottom shows the time as 9:35 AM on 2024-02-27.

Canadian Navigable Waters Act (R.S.C., 1985, c. N-22)
Full Document: [HTML](#) (Accessibility Buttons available) | [XML](#) [275 KB] | [PDF](#) [485 KB]
Act current to 2024-02-06 and last amended on 2019-10-04. [Previous Versions](#)

Table of Contents

Canadian Navigable Waters Act
R.S.C., 1985, c. N-22

An Act respecting the protection of navigation in Canadian navigable waters

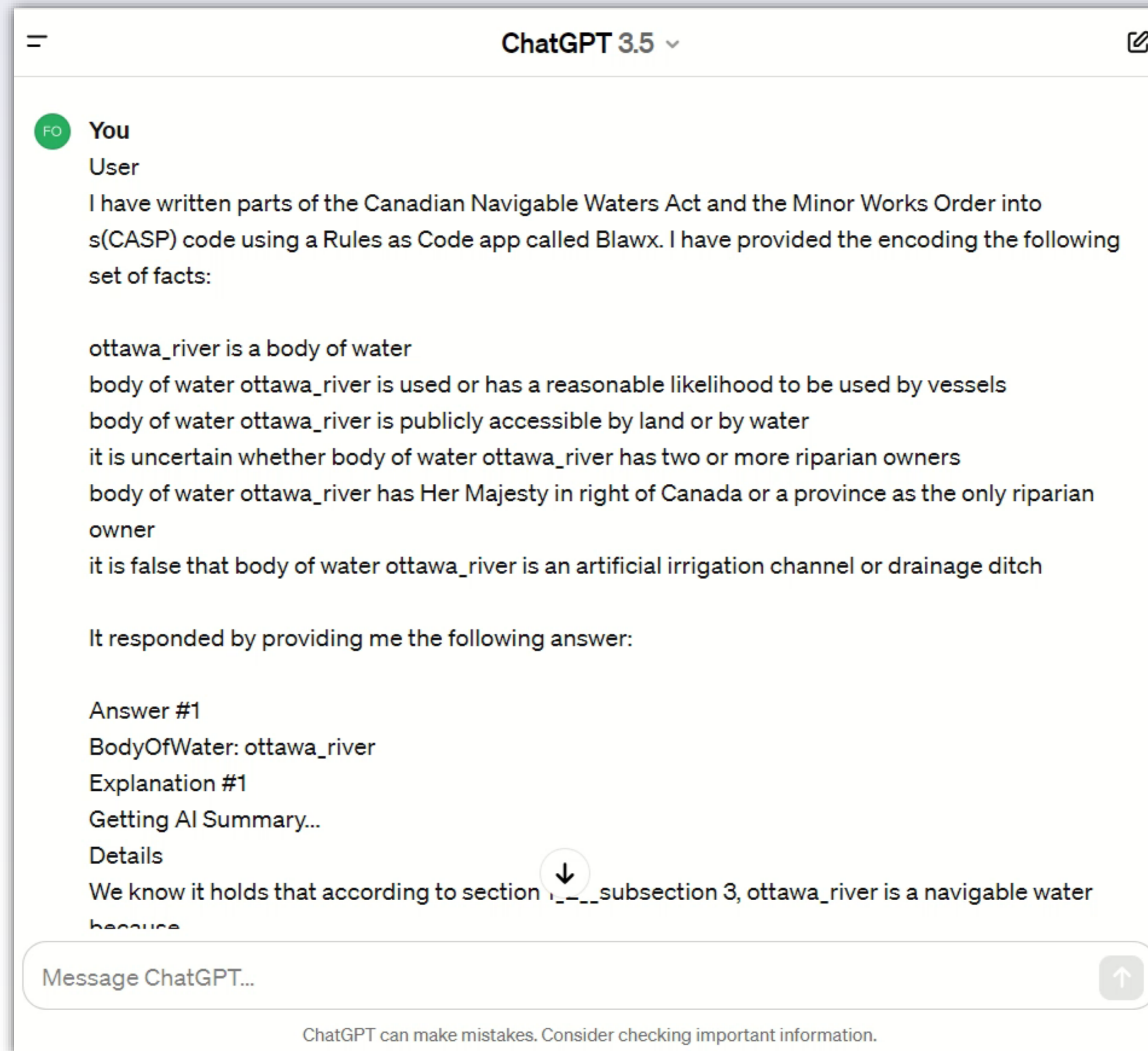
Short Title

Short title

1 This Act may be cited as the *Canadian Navigable Waters Act*.
R.S., 1985, c. N-22, s. 1; 2012, c. 31, s. 316; [2019, c. 28, s. 46](#).
[Previous Version](#)

Interpretation

Combining Symbolic AI with Generative AI | Provide Plain Language Summaries



The screenshot shows a chat window titled "ChatGPT 3.5". The user's message is as follows:

You
User
I have written parts of the Canadian Navigable Waters Act and the Minor Works Order into s(CASP) code using a Rules as Code app called Blawx. I have provided the encoding the following set of facts:

ottawa_river is a body of water
body of water ottawa_river is used or has a reasonable likelihood to be used by vessels
body of water ottawa_river is publicly accessible by land or by water
it is uncertain whether body of water ottawa_river has two or more riparian owners
body of water ottawa_river has Her Majesty in right of Canada or a province as the only riparian owner
it is false that body of water ottawa_river is an artificial irrigation channel or drainage ditch

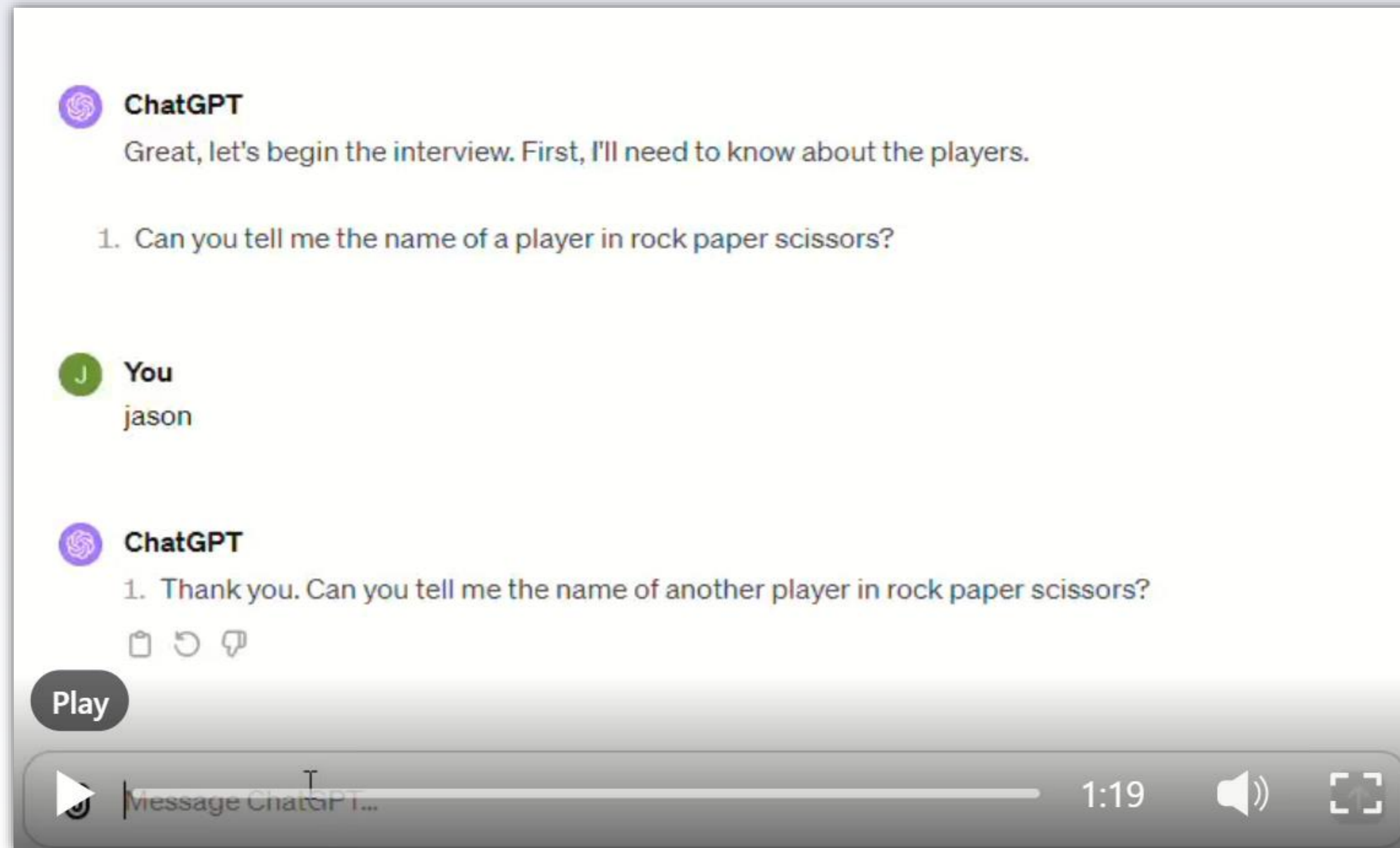
It responded by providing me the following answer:

Answer #1
BodyOfWater: ottawa_river
Explanation #1
Getting AI Summary...
Details

We know it holds that according to section 1_--_ subsection 3, ottawa_river is a navigable water because

At the bottom of the chat window, there is a text input field with the placeholder "Message ChatGPT..." and a send button with an upward arrow. Below the input field, a disclaimer reads: "ChatGPT can make mistakes. Consider checking important information."

Combining Symbolic AI with Generative AI | Prompt Users for Information



The screenshot shows a video player interface. The video content is a chat conversation with ChatGPT. The chat starts with ChatGPT saying, "Great, let's begin the interview. First, I'll need to know about the players." followed by a numbered list item: "1. Can you tell me the name of a player in rock paper scissors?". The user, identified as "You" with the name "jason", responds. ChatGPT then asks, "1. Thank you. Can you tell me the name of another player in rock paper scissors?". Below the chat, there is a "Play" button and a video player control bar. The control bar includes a play button, a progress bar, a timestamp of "1:19", a volume icon, and a full-screen icon. The video title is partially visible as "Message ChatGPT...".

ChatGPT
Great, let's begin the interview. First, I'll need to know about the players.

1. Can you tell me the name of a player in rock paper scissors?

You
jason

ChatGPT
1. Thank you. Can you tell me the name of another player in rock paper scissors?

Play

Message ChatGPT... 1:19

Combining Symbolic AI with Generative AI | Autogenerate Code

- Legal Document Name: Rock Paper Scissors Act
- Legal Document Slug: rpsa
- Legal Document Tags:
 - Law

You can edit this information by clicking the button below.

[✎ EDIT](#) [✕ DELETE](#)

Content

- Rock Paper Scissors Act
- Rock Paper Scissors
- 1. Rock paper scissors is a game played between two players.
- Signs
- 2. There are three signs:
 - a) Rock,
 - b) Paper, and
 - c) Scissors.
- Defeating Relationships
- 3. The signs have the following defeating relationships:
 - a) Rock beats Scissors,
 - b) Paper beats Rock, and
 - c) Scissors beats Paper.
- Winning the Game
- 4. The winner of a game is the player who throws the sign that beats the sign thrown by the other player.

[+ ADD CONTENT](#)



Stay connected



canada.ca/school-ecole



@School_GC | @Ecole_GC

