A programming language presented in graphics

Roman Sobkuliak
Faculty of Mathematics and Physics, Charles University in Prague

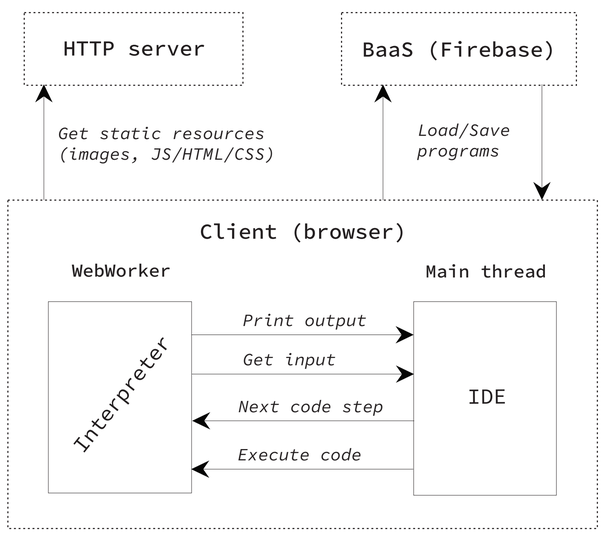
Introduction

We created a programming language with characters and keywords substituted with images and animations (GIFs). We built a web IDE and a client-side interpreter for this language using modern web technologies including WebWorkers, TypeScript and React.

The IDE features code-stepping with information about current location in the source code, environment variables and a call stack. Additionally, there is a support for storing programs on the server and loading them later.

The purpose of the language is educational, e.g., to be used in creative games at programming camps for elementary and high schoolers.

Implementation



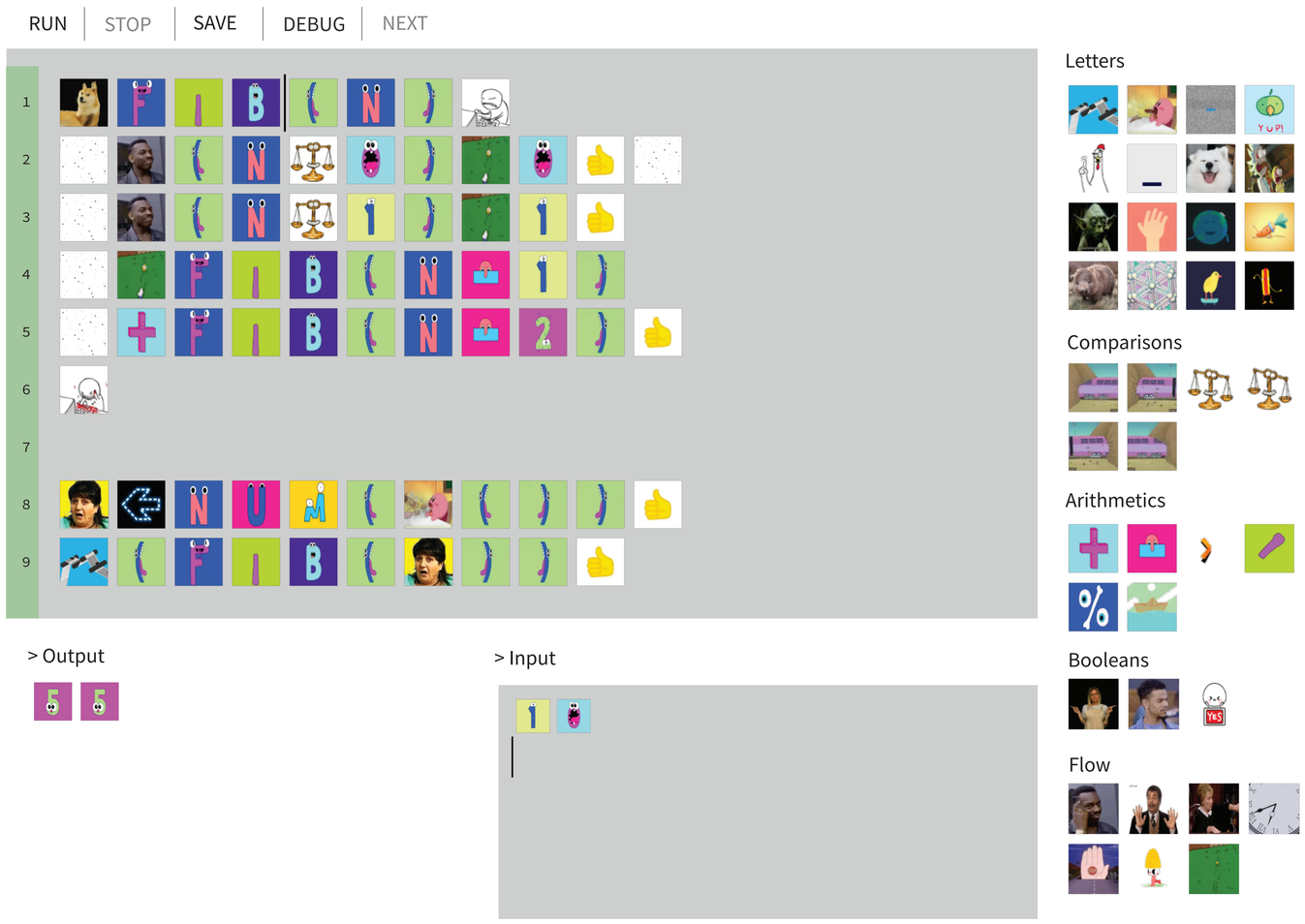
High-level diagram of the implementation

In order to make the IDE highly accessible we chose to implement it for web rather than as a native OS application.
Users can store their programs in the cloud and share it using a unique URL. We implemented this feature using a proprietary cloud service. The HTTP server is very lightweight, serving only static resources.

The frontend consists of two major parts, the interpreter and the IDE. The interpreter performs no optimizations — it executes a parsed AST in a node by node fashion.

For the IDE itself we used the React frontend library. We implemented our own text processor with characters replaced by images.

IDE



Screenshot of the web IDE

The IDE has a standard layout with source code occupying most of the space. Users can provide input interactively, i.e., an interpreted program waits for an input. Similar to this, when **DEBUG** button is pressed the IDE transitions to a code-stepping mode where the interpreter waits before each step until **NEXT** button is pressed.
Possible image "characters" are listed on the right panel. Clicking on an image inserts it at the user's cursor position.

Language

The language we created closely resembles Python. It features:

- Object-Oriented programming contructs — classes, inheritance, polymorphism
- Magic methods for operator overloading
- Functional programming features — anonymous functions and closures

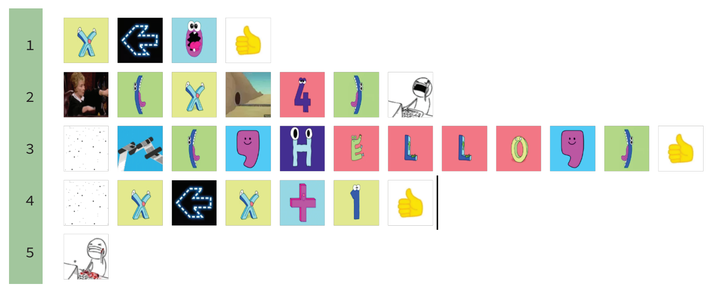
Source code is represented in a textual format where each letter has a one-to-one mapping to a corresponding image or animation.

```
1  x = 0
2  while x < 4:
3    print('HELLO')
4    x = x + 1
```

```
1  x=0;
2  ↻(x<4){
3   λ('HELLO');
4   x=x+1;
5  }
```

Sample Python source code (left) and the same program written in our language (right)

The example above shows a simple Python program represented in the textual form of our language. Variable name **x** was left unchanged, but the keyword **while** is replaced with a single character ↻. Similarly, **print** is replaced with **λ**. Below is the same program displayed in its final graphical form.



Sample program presented in graphics

The fact that every image has a one-to-one mapping to a Unicode character has a nice property regarding the use of the keywords in a string. We wrote the interpreter in JavaScript where each element is considered to be a single UTF-16 code unit. This means that if we only use characters that can be encoded into a single 16-bit code unit, all keywords will take space of a single character in a string.

Object model

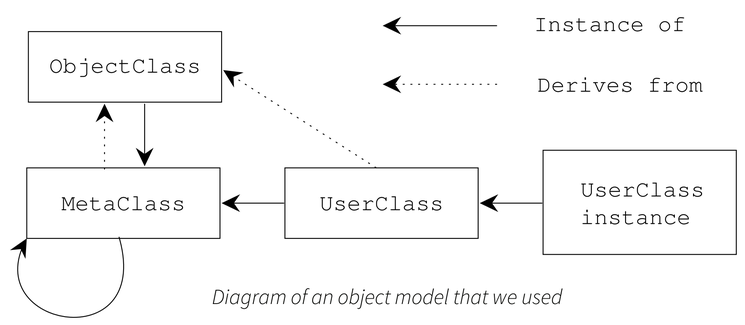


Diagram of an object model that we used

To support OOP constructs in our language we used an object model that resembles Python where everything is an object. As shown on the diagram above, **ObjectClass** is an ancestor of all classes. Since classes are also objects, they are instances of **MetaClass** and **MetaClass** is an instance of itself.
MetaClass is also responsible for creating new instances. Its call operator override takes a class as an argument, spawns an instance and calls a constructor of the given class.

Conclusion

We designed a programming language that can be easily represented using images or animations. We also developed an interpreter and a web IDE for this language. It is going to be incorporated into games at programming camps for elemetary and high schoolers.

Acknowledgements

I would like to give thanks to my supervisor, RNDr. David Bednárek, Ph.D., for his patience and advice.

Author: Roman Sobkuliak
Email: r.sobkuliak@gmail.com
Supervisor: RNDr. David Bednárek, Ph.D.
Department: Department of Software Engineering
Source code: https://github.com/sobkulir/giflang
Submission year: 2020