Garrett Fleischer
Jose Moreno
Eiffel Report

Eiffel had it's initial release in 1986. It was developed by Eiffel Software, designed by Bertrand Meyer, with the intention of creating language based support for secure and maintainable code. It currently includes several implementations including: EiffelStudio, EiffelEnvision, LibertyEiffel, and Visual Eiffel. EiffelStudio is available as open source or commercially licensed IDE. Eiffel Vision is a plugin that provides Eiffel support for Visual Studio.

Since Eiffel is intended for heavy use in OOP paradigm our project was chosen to take advantage of polymorphism and the networking libraries provided by the Eiffel. To this end we are making a multi-player roguelike game called Salvos Aptissimum. While Eiffel is statically, and strongly typed the fact that it is object oriented allows us to use Polymorphism extensively to deal with multiple data representations.

Eiffel is a compiled language, and actually leans heavily on the compiler for its runtime efficiency. The goals of the language are long term maintainability and security of the code base, and discards optimization as part of the coding practice. Instead it falls on the compiler to optimize code written in Eiffel. In terms of efficiency it focuses on the long term maintainability of the code rather than the short term writeability of a program. To this end Eiffel uses the uniform access principle which simply states that classes should have uniform access to functions and to data members. The goal being that the person working with that class should not be worried about whether their request is being calculated or simply pulled from memory. Where Eiffel focus most heavily is in security. It includes garbage cleanup, null safety checks, implements assertions, post and pre-condition checking, and allows the creator of a class/interface to specify requirements for a class that must be followed by its descendants.

Bertrand Meyer is often credited with the term open/closed principle. So it is not surprising that Eiffel feels a lot like a forefather to Java. Like Java it is almost exclusively written in classes, with even it's base data types like INTEGERS are a class type. As such all types are defined and you can actually see many of their implementations inside of EiffelStudio. One of the more interesting features of Eiffel is that feature access can be assigned very specifically. It is possible to designate specific classes that are allowed access to the components of the class. For example it would be possible to create a class FOOD which only RABBIT could call a function that affects the class, without having RABBIT inherit from or be related to FOOD. Access can be explicitly defined by listing classes, tagged as (NONE} effectively making access private, or tagged as {ANY} effectively making access to that feature public. It differs from java in that operator overloading is allowed, but name overloading is not. If there are multiple functions of the same name in a class due to inheritance then the function must be renamed in order to eliminate ambiguity.

Eiffel includes Agents which allow you to pass functions in a way similar to C#. These agents can include all the same behaviors present in any function, including the use of Eiffel's

pre and post condition checks. Among some of the issues we discovered is that Eiffel suffers from the same limitations with threads that C has; probably not surprising since Eiffel is generally compiled into C code before then using a C compiler to create the actual program. Strangely, unlike C code, Array's actually start at an index of 1.

In conclusion Eiffel is an interesting approach to long term sustainability and security for code. Though much of what it does can be done through standard documentation in practice this is rarely seen. Considering it's age it is unlikely that it will find the traction necessary to make this a viable option for its purpose. The low spread of Eiffel means that any project significant enough to take advantage of it's primary purpose would likely be refactored into one of the many languages that have since been developed with OOP paradigm and have more established programers.