Fejlesztői dokumentáció

LISP, REPL, FRUBLD

A projekt elkészítése során számomra a legdöbbenetesebb meglepetést magának a szkript nyelvnek (illetve a generatív magját adó LISP dialektusnak) a megalkotása jelentette. A lehető legelemibb folyamatokból (stream ki- és beolvasás) fokozatosan kívántam ezt a nyelvet felépíteni, amikor szembesültem azzal, hogy tulajdonképpen a LISP alapvetői függvényeit implementálom le, hiszen a string/stream műveletek nagy többsége jól megfeleltethető a lista műveletekkel definiált függvényeknek is. (LISt Processing)
A dolog külön érdekessége pedig az, hogy eredetileg az általam erre a célra a legalkalmasabb tartott Common LISP nyelven terveztem a projektnek ezt a részét megvalósítani, viszont portolhatósági és főleg integrálhatósági nehézségek miatt lemondtam erről.
Persze az alapszintű LISP ismeretek így is nagy hasznomra voltak, hiszen előttem már volt előttem egy kipróbált koncepció, amit csak meg kellett valósítanom egy C++ programon belül, rengeteg próbálkozástól és kudarctól kímélve meg magam.

A megalkotott szkript nyelv szintaxisát tanulmányozva, a LISP-et ismerő programozók viszont így is két jelentős különbséget fedezhetnek fel a LISP és a projekt belső nyelve között:

– a LISP szintaxisában legalapvetőbbnek számító zárójelek szinte teljes hiányát
– a szkript nyelv kis/nagybetű érzékeny lett

Az előbbire magyarázat lehet az, hogy még sem volt szükség egy LISP-hez mérhető funkcionalitású nyelv teljes megalkotására és a feldolgozás továbbra is sztring típusú objektumokon keresztül történik, zárójelekkel definiált és egymásba ágyazott listák helyett.
Másfelől, a leendő célcsoport (a programozásban nem feltétlenül járatos felhasználók) tagjai is valószínűleg inkább igénylik az átláthatóbb, egyszerűbb programkódokat, a tömör és magas absztrakciós szintű nyelvi elemekkel szemben.
Ennek ellenére a LISP nyelvek legnagyobb előnyéről (az önkiértékelő függvényekről az adat és program elemek teljes felcserélhetőségéről) valamint a funkcionális programozásról így sem kellett lemondani.

A kis/nagybetű érzéketlenséget pedig sokkal prózaibb okokból hagytam figyelmen kívül: a LISP nyelveknek legelőször futási környezetet biztosító, mindössze második generációs számítógépeknek még nem volt kapacitása a case sensitive ki- és bevitel kezelésére, később pedig, „történelmi okokból” szinte minden LISP dialektus megőrizte ezt a tulajdonságot… én pedig nem láttam semmi okát annak, hogy miért is kellene nekem is így tennem.

A projekt engine oldala három, jól elkülönített de teljes mértekben egymásra épülő részből áll:

Legbelül található a *topológia*, egy olyan, mindössze egyszer példányosított *szingleton osztály*, ami tulajdonképpen leírja a Rubik kocka struktúráját, leírja a belső állapotterét és annak változásait, az izomorfizmust a Rubik kocka algebrai csoportja és a számítógép CPU-ja által könnyen kezelhető objektumok (integer tömbök) között és a kocka szinte minden elemét lekérdezhetővé teszi az algoritmusok és a fejlesztők számára egyaránt.

A topológiára épül közvetlenül az algoritmikus függvények tere: értelemszerűen a különféle kereső heurisztikák helye.

Legfelül pedig, a LISP-ből is jól ismert *olvasó-kiértékelő-kiíró-ciklus* (Read-Evaluate-Print-Loop vagy röviden: REPL) található, ami egyszerűsége dacára négy alapvető funkciót lát el:

– a stdin -nel szemben ergonómikus eszközt biztosít a felhasználók számára az input/output kezelésére: többek között az input sorok szerkeszthetőek is benne (balra/jobbra mozgatható kurzor), támogatja a command line history -t (fel/le kurzor), az automatikus parancs kiegészítést (TAB), illetve rugalmasan kezeli a különböző input/output forrásokat (billentyűzet, kijelző, fájlok, web-socket stb) és ezeket bármikor át is irányíthatja,, ki is cserélheti
– a felhasználók számára is elérhetővé teszi az alatta lévő rétegeket
– egy rugalmas absztrakciós eszközt (szkript nyelv) biztosít az összetettebb feladatok (például megoldó algoritmusok) implementálására
– rejtetten ugyan, de megvalósítja az úgynevezett *envelope – letter programozási patternt:* mivel a C++ nyelv objektumaiban a *this* adattag konstansnak számít, ezért futási időben nincs közvetlen lehetőség az objektumok kicserélésére, ezért vagy még egy újabb réteget készítünk az objektumok halmaza fölé (akár az osztályban statikusan, akár az osztály fölé), vagy pedig magukat az objektum példányokat „burkoljuk” be megváltoztatható, akárhonnan megcímezhető borítékokba. A REPL függvény (mivel rajta keresztül a teljes objektum elérhető, teljesen „beburkolta”) pontosan ezt teszi: így már van lehetőség akár egyszerre több virtuális Rubik kocka egyidejű kezelésére is.

A szkript nyelvnek köszönhetően definiálhatunk egy újabb absztrakciós szintet is: a külső utasítások terét, vagyis azokat a függvényeket, melyeket már nem a C++ kódba implementálva találunk meg, hanem a szkript nyelvben hoztunk létre.