ÓBUDAI EGYETEM
Neumann János Informatikai kar
Mérnök informatikus BSc

Vizuális, interaktív programozás oktató rendszer moduláris megvalósítása

Projektmunka dokumentáció

Ráncsik Áron

2020. május 21.

Tartalomjegyzék

fejezet 1

Bevezetés

Az oktatásban manapság elterjedt gyakorlat az oktatott anyag játékos megfogalmazása [23], idegen szóval a *gamification* [6]. A módszer lényege, hogy a megtanítani kívánt ismeretet egyszerű, nyers forma helyett, játékos formában tesszük érthetőé a tanulók számára. Dolgozatomban a programozást szeretném a tanulni vágyok elé tárni, olyan formában, hogy az ne okozzon nehézséget, illetve a tanulás emléke inkább egy élmény legyen, egy unalmas óra helyett.

Az alábbi célokat tűztem ki. Programozás egyik motivációjának azt szeretném, hogy a felhasználók által készített különböző programkódok, ágensek, egymás ellen tudjanak versenyezni. A programozáshoz ne legyen szükséges konkrét programozási nyelv ismerete. Legyen lehetőség kézvezérléssel végezni a programozást. Az ágensek megmérettetését akár már létező, ismert számítógépes játékokhoz csatlakozva, vagy saját egyedi játékban is lelehessen bonyolítani. További tervezett funkció, hogy az írt kódok összecsapását, kiterjesztett valóságon keresztül is legyen lehetőség követni. Úgy gondolom, hogy a programozásoktatást egyik legszemléletesebb területe a robotprogramozás, ezért azt célt is kitűztem, hogy az egyetemen elérhető da Vinci robotot is lehessen a rendszer által vezérelni. Magas-szintű program utasítások biztosításának a segítségével könnyen, érthetően lehet szemléltetni a robotika területét is, és a programozáson belül az orvostudomány iránt is érdeklődést kiváltani a fiatalokban.

Röviden összefoglalva, a következő irodalomkutatást készítettem a fent megfogalmazott célok ügyében. Első lépésként, a kitűzött célok megvalósítsa érdekében, a 2.1.1 részben a vizuális programozási környezet elkészítésének lehetőségeit részletezem, mely abban segít, hogy egy konkrét programozási nyelv tanulásának kezdeti nehézségein átugorva vizuális eszközökkel teszi elérhetővé a programozást, akár laikusok számára is. A Módszertan szekció 2.2.2 részben azt vizsgálom, milyen lehetőségek vannak a Da Vinci robot egyszerű maga-szintű programozására. A 2.3 szekcióban a napjainkban népszerű kiterjesztett valóság megvalósításának lehetőségeit részletezem. A Módszertan utolsó szekciójában 2.4 a kézvezérlés funkció megvalósításához szükséges ismereteket igyekszem analizálni.

fejezet 2

Módszertan

A korában megfogalmazott célok elérése érdekében végzett kutatómunkát részletezem, ebben a fejezetben.

2.1. Vizuális programozási környezet

2.1.1. Alternatív megoldások vizuális programozásra

Mivel a dolgozatban vizuális programozást is szeretnék biztosítani, ezért megvizsgálom milyen konkurens megoldások léteznek erre a célra. Az alábbiakban bemutatok és elemzek több kutatást, korábbi kész alkalmazásokat és programkönyvtárakat melyek mindegyike a vizuális blokk alapú programozást biztosítja.

Scratch ♦ Nagyon népszerű, oktatásban méltán közkedvelten használt teljes környezet [15, 22, 24], mely vizuális programozást tesz lehetővé. Az alapvető célja olyan média alkalmazások mint pl.: animáció, köszöntések, történet mesélés, zeneklip elkészítésének folyamata közben megtanítani a programozást. Az egyedüli, intuitív tanulás is előnyben részesíti. Főleg kisebb 8-16 éves korosztály számára készült.

Az elkészült program alapértelmezetten egy saját „színpad" környezetben futtatható. Esemény vezérelt programozást is lehetővé tesz, ebben az esetben az egyszerre fellépő konkurens utasítások között esetleges versenyhelyzetet kezeli, de nem minden esetben úgy ahogy azt elsőre a használó gondolná. A programból *broadcast* utasításokkal, saját parancsok, szkriptek végrehajtása lehetséges. A *broadcast* utasítássokkal lehetséges a külvilággal történő kommunikáció is, de körülményes. Kiegészítőkkel bővíthető, de így sem nem lehet az alap működésre kiható szignifikáns módosításokat végrehajtani.

Előnyei *
- Szabadon elérhető, nyílt forráskódú.
- Kész megoldásról beszélve, sokkal kevesebb munka segítségével lehet használni.
- Egyedi működés *broadcast* utasításokkal lehetséges.
- Az alap környezet többnyire magas szintű előre definiált utasításokban gazdag pl.: animációk
- Kezdők számára is egyszerűen érthető felület.
- Tapasztalatok alapján nagyon ismert az általános iskolások körében.

– Érthető, naprakész dokumentáció áll rendelkezésre

Hátrányai *
– A *broadcast* utasításon kívül máshogy nem megoldható egyedi utasítás létrehozása.
– Az elkészült program csak a saját környezetén belül futtatható, nem lehetséges egyedi környezetben futtatni.
– Az elkészült program egyszerűen nem alakítható valódi programkóddá. (Van harmadik féltől származó kész megoldás)
– Monolitikus, bővítésre nézve többnyire zárt rendszer.
– Ugyan van webes felület, de saját oldalba nem ágyazható.

Snap ♦ Új, még kevésbé ismert, teljes megoldás [9], segítségével vizuálisan van lehetőségünk programozni. A Scratch-el szemben több szabadságot biztosít a személyre szabhatóság tekintetében. Sokkal bonyolultabb problémák megoldására is alkalmas, az objektum orientált paradigmát is támogatja. A környezet kevésbé kezdő barát. *Url* utasításokkal webes csatlakozó felületen keresztül tud fogadó és küldőként is könnyen kommunikálni a külvilággal. A felhő alapú gépi tanulás szolgáltatásokhoz csatlakoztatva akár „AI" fejlesztésre is alkalmas [12]. A párhuzamos ciklusokat időosztásos módszerrel futtatja párhuzamosan „yield" utasítások segítségével, de erre is igaz. hogy mindezt igyekszik elrejteni a felhasználók elől és nem igényel különösebb beavatkozást az alapvető működéshez hozzá lehet szokni.

Előnyei *
– Szabadon elérhető, nyílt forráskódú.
– Kész megoldásról beszélve, sokkal kevesebb munka segítségével lehet használni.
– Egyedi működés *url* utasításokkal lehetséges. Sokkal kényelmesebben mint Scratch esetén.
– Az alap környezet előre definiált utasításokban gazdag pl.: *sprite*-ok vezérlése.
– Bonyolultabb problémák megoldására is alkalmas.
– Érthető, naprakész dokumentáció áll rendelkezésre
– Van webes felülete, de saját oldalba nem ágyazható.
– Egyedi blokkok létrehozását támogatja.
– Objektum orientált paradigmát támogatja

Hátrányai *
– Az elkészült program csak a saját környezetén belül futtatható, nem lehetséges egyedi környezetben futtatni.
– Az elkészült program nem alakítható valódi programkóddá.
– Nem biztosít felhasználható könyvárat, csak a projekt forrását felhasználva lehet egyedi alkalmazásokban használni.
– Jelenleg (2020. május 21.) béta állapotban van.
– Monolitikus, bővítésre nézve többnyire zárt rendszer.
– A objektum orientáltság különböző megkötésekkel érhető el.

GP ♦ Általános célú blokk alapú [19] [16] programozási nyelv. Egy valódi programozási nyelv mely blokk és kód alapú programozási lehetőséggel is rendelkezik. Sok alacsony szintű

funkciók érhetőek el benne. Sokkal közelebb áll a valóságos programozáshoz a korábban említett lehetőségekhez képest, rendelkezik webes felülettel és a külvilággal internetes *get*, *put* utasítás lehetséges a kommunikáció egyedi rendszerekkel. Támogatja az objektum orientált paradigmát. Tartalmaz bonyolultabb adatszerkezeteket is.

Előnyei *
- Kész megoldásról beszélve, sokkal kevesebb munka segítségével lehet használni.
- Egyedi működés *get, put* utasításokkal lehetséges.
  Sokkal kényelmesebben mint Scratch esetén.
- Az alap környezet többnyire alacsony szintű előre definiált utasításokban gazdag pl.: *set pixel* utasítás blokk.
- Bonyolultabb problémák megoldására is alkalmas.
- Moduláris módon készült.
- Érthető, naprakész dokumentáció áll rendelkezésre.

Hátrányai *
- Az elkészült program csak a saját környezeten belül futtatható, nem lehetséges saját környezetben futtatni.
- Az elkészült program nem alakítható valódi programkóddá.
- Ugyan van webes felület, de saját oldalba nem ágyazható.

Alice ♦ Interaktív 3D Animációs környezet [4]. Az idézett kutatás által megfogalmazottan a célja egy 3D-s környezet interaktív fejlesztése melyben szabadon lehet felfedezni az elkészült alkotást. Főleg szkript alapú prototípus fejlesztő környezet.

Előnyei *
- Kész munka elvileg kevesebb munka lehet átalakítani, felhasználni
- Adott 3D környezet
- Esemény vezérelt programozás tanítására is alkalmas

Hátrányai *
- Az elkészült program csak a saját környezeten belül futtatható, nem lehetséges saját környezetben futtatni.
- Nem vizuális egy sajátos egyedi szkriptnyelvvel rendelkezik.
- Az elkészült program nem alakítható valódi programkóddá.
- Nincs webes felülete, futtatható állományok telepítését igényli.
- Kissé régi, idejét múlt, már kevésbé támogatott.

Egyéb kész megoldások ♦ A fenti részben az általam, a dolgozatommal legjobban összehasonlítható szempontok alapján fontosnak tartott kész megoldásokat mutattam be. Természetesen rengeteg egyéb kész alkalmazás létezik mely vizuális programozási lehetőséget biztosít. Felsorolás szintjén itt leírok pár szerintem említésre méltó alkalmazást.
- Game Maker [10] 2D játék készítő, vizuális és saját szkriptnyelvvel is rendelkezik. A teljes verzió pénzbe kerül.
- Stencyl [14] 2D Játék készítő alkalmazás melyben vizuálisan lehet programozni.

A továbbiakban kész alkalmazások helyett, vizuális programozásra készült programkönyvtárak bemutatásával fogom folytatni az választható megoldások listáját.

Blockly ♦ Google által fejlesztett, vizuális programozást támogató program könyvtár [3] [20]. A vizuális megjelenésért felelős, nem egy programozási nyelv, csak egy vizuális leíró nyelv, mely könnyedén exportálható vallós programozási kóddá. Több létező programozási nyelv kódot lehet a vizuális környezetből generálni, többek között: Javascript, Python, Lua, Dart nyelvű kódokat is. A Blockly könyvtárat sok kész alkalmazás használja a saját egyedi vizuális környezetének megvalósítására. Például a korábban említett Scratch, Snap is használja vizuális könyvtárként, de az ezután részletezett MakeCode (PXT) környezet is Blockly-t használ a vizuális megjelenítésre.

Előnyei *
- Szabadon elérhető, nyílt forráskódú.
- Kezdők számára is egyszerűen érthető felület.
- Érthető, naprakész dokumentáció áll rendelkezésre.
- Program könyvtár, könnyen bővíthető egyedi utasításokkal
- A program futtatási környezetére nincs megkötés.
- Nemzetközi. Több mint 40 (köztük magyar) nyelven elérhető.
- Bármilyen futtatási környezetbe könnyen integrálható.
- Rengeteg programkód generálható.

Hátrányai *
- Csak egy vizuális programozást támogató „drag & dropp" programkönyvtár.
- Viszonylag régi technológiákkal készült, nehézkes modern környezetben használni.

MakeCode (PXT) ♦ Microsoft által fejlesztett, összetett, vizuális programozói környezet támogató, nyílt forráskódú programkönyvtár [25]. A Blockly könyvtárat felhasználja a blokk alapú vizuális programozás biztosítására. A blokk alapú programozáson túl biztosít lehetőséget a szöveges programozásra is, egy beépített weboldalon futamítható fejlesztő környezetben. Abban különbözik a korábban alternatív lehetőségektől, hogy egyben nyújt egy teljes fejlesztő környezetet, amihez könnyen hozzá lehet csatolni egy saját *target* modult egyedi utasításokkal. [7] Kutatás részletesen bemutatja, hogyan lehet használni beágyazott rendszerekhez. A fejlesztő környezethez alacsony szintű mikrokontrollerhez való fordítót csatlakoztatva.

Előnyei *
- Szabadon elérhető, nyílt forráskódú.
- Kezdők számára is egyszerűen érthető felület.
- Érthető, naprakész dokumentáció áll rendelkezésre.
- Program könyvtár, könnyen bővíthető egyedi utasításokkal
- A program futtatási környezetére nincs erős megkötés.

Hátrányai *
- Integrálása korlátozott, alkalmazkodni kell a felépítéséhez, ettől eltérni nem lehetséges könnyen.

– Béta verzióban van a fejlesztés.

Választott vizuális programozási környezet

Kipróbáltam, teszteltem a legtöbb felsorolt megoldást, továbbá a korábbi elemzés összevetése után a Blockly környezet használata mellet döntöttem. A dolgozat igényeit nem lehet kész megoldásokkal pl. Scratch, Snap stb. kielégíteni, mert túlságosan zárt működésük nem teszik lehetővé, hogy egyedi környezethez lehessen programozni ezekben az alkalmazásokban. A Blockly kellőképpen módosítható és könnyen beilleszthető, vizuális programozást biztosító programkönyvtár, mely megfelel a feladat részéről támasztott elvárásoknak. A Blocky kellőképpen egyszerűen használható és könnyen egyedi igényeknek megfelelően alakítható.

2.2. Da Vinci

A célok között szerepelt a da Vinci robot egyszerű vezérlése. Az elkövetkező részben, igyekszem körbejárni a lehetőségeket és elemezni előnyeit hátrányait, melyek segítségével lehet kapcsolódni a da Vinci rendszerhez. Az egyetemen egy 2010-ben bemutatott első generációs da Vinci classic rendszerhez van lehetőség hozzáférni, ezért ehhez készül az alkalmazás. A továbbiakban mindig erről a modellről lesz szó.

2.2.1. Robot Operating System

Annak érdekében, hogy a fejlesztés során, ne kelljen folyamatosan egy fizikai da Vinci rendszerrel dolgozni, egy olyan környezetre van szükség ami képes biztosítani da Vinci robotot virtuálisan. Több rendszer elérhető mely robot programozás vezérlő környezetet biztosít, csak felsorolás szintjén néhány [18, 17], azonban az egyik legkorábbi és legnépszerűbb lehetőség a Robot Operating System [21] továbbiakban csak ROS. A ROS órási és növekvő közösséggel rendelkezik, *de facto* standard az iparban és kutatásban. Dolgozatomban több szempont miatt is a legjobb választásnak a ROS rendszert bizonyult. Attól a hátrányától eltekintve, hogy nem multiplatform azaz csak Ubuntu GNU/Linux és annak is csak bizonyos verzióihoz támogatott hivatalosan, rengeteg előnnyel rendelkezik. Sok robotvezérléssel kapcsolatos alapfunkciót tartalmaz, amit így nem kell újból megírni, biztosít egy jól elszeparált keretet külön a kommunikációra és az üzleti logikára.

Előnyei ♦
- Szabadon elérhető, nyílt forráskódú
- Kiforrót
- Népszerű
- Moduláris
- Kutatási célokhoz megfelelő
- Érthető, naprakész, jó dokumentáció áll rendelkezésre

Hátrányai ♦
- Nem multiplatform, csak Ubuntu GNU/Linux és annak is csak bizonyos verzióihoz elérhető

2.2.2. Da Vinci Research Kit

A da Vinci Research Kit továbbiakban dVRK egy nyílt forráskódú rendszer ami lehetővé teszi a Da Vinci robot karok programozását [13]. A ROS rendszerhez kapcsolódik, könnyen lehetséges az utasításait használni egy ROS modult biztosít használatra. A célkitűzéseimhez képest viszonylag alacsony-szintű da Vinci robotkarvezérlő utasítások kiadására alkalmas. Ennek a modulnak a segítségivel lehetséges a ROS rendszerrel da Vinci robotot vezérelni, továbbá virtuálisan szimulált robotkar létrehozása is lehetséges. Erre keretrendszerre mindenképp szükség van amennyiben ROS rendszerhez szeretnénk kapcsolnia a da Vinci robotot.

2.2.3. iRob Surgical Subtask Automation Framework

Nagy D. Tamás és Haidegger Tamás által fejlesztett keretrendszer továbbiakban SAF [5]. Alapvetően arra a célra készült, hogy részleges automatizáció segítségével levegye a sebészekről a kognitív terhelést. Részfolyamatok automatizálását teszi lehetővé a keretrendszer. A ROS rendszerben elérhető modulokat biztosít, a dVRK rendszerre épül. Használata során felépít egy hierarchikus modul rendszert, melynek a legmagasabb szintjén magas-szintű utasítások kiadása lehetséges. Számomra a céloknak tökéletesen megfelelő.

Előnyei ♦
- Szabadon elérhető, nyílt forráskódú
- Moduláris
- Kutatási célokhoz megfelelő
- Érthető, naprakész, jó dokumentáció áll rendelkezésre

Hátrányai ♦
- Jelenleg fejlesztés alatt áll.

2.3. Kiterjesztett valóság

2.4. Kézvezérlés

2.5. Gépi látás

2.6. Kamera mozgás becslés

2.6.1. Markerek keresése

fejezet 3

Megvalósítás

3.1. Da Vinci vizuális programozási környezete

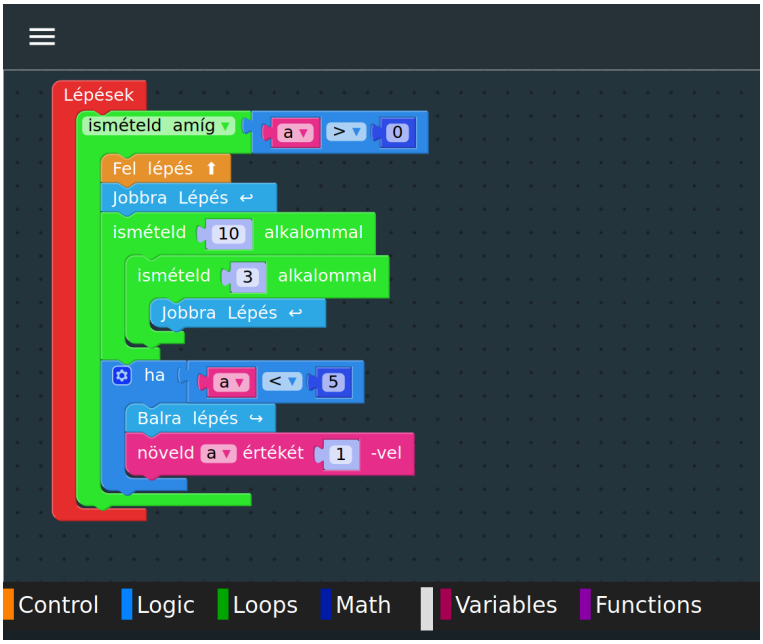
A korábbi irodalomkatatusban arra a következtetésre jutottam hogy a Blockly rendszert fogom használni a grafikus programozás megvalósítására. A da Vinci kar programozásának a példáján keresztül fogom bemutatni a létrehozott vizuális programozási környezetet. A Blockly egy JavaScript könyvtár melyet böngészőből lehetséges használni. Az Blocly használatával készült weboldal kiszolgálásához NodeJS szervert választottam. Úgy gondolom, hogy NodeJS segítségével kellőképpen egyszerűen lehetséges a weboldal kiszolgálása és WebSocket szervert készítése is könnyen megvalósítható. A NodeJS szerver segítségével három fontosabb modult létrehoztam. A vizuális programozást biztosító továbbiakban ”editor„ modul, a felhasználó által létrehozott ágens futtatására alkalmas ”view„ modul és a különböző felhasználók adminisztratív kezelésére szolgáló ”control„ modul.

3.1.1. Control

Ennek a modulnak a segítségével ki tudjuk listázni az alkalmazáshoz csatlakozott felhasználókat. A csatlakozott felhasználók által létrehozott Blockly utasítás struktúrák minden módodaíráskor automatikusan mentésre kerülnek. A ”control„ modul segítségével lehetséges egy felhasználó egy mentett struktúrájának a betöltése a ”view„ modulba. Az teljes alkalmazásnak van olyan funkciója melynek segítségével a felhasználók által létrehozott ágensek küzdhetnek egymással, ennek a kezelése is ebből a modulból érhető el ez keserűbb részletezem. Jelen példában maradunk a da Vinci programozásánál. A da Vinci számára létrehozott algoritmust miután betöltjük a ”view„ modulba akkor ”control„ modulnak van lehetősége azt elindítani a da Vinci rendszeren.

3.1.2. Editor

Ez a módul ténylegesen használja a Blockly könyvtárat. Böngészőből a felhasználóknak van lehetősége ezt a modult használni. Ennek a segítségével vizuálisan lehetséges algoritmusokat létrehozni. A da Vinci rendszert programozásához szükséges volt egyedi utasítások létrehozása, hogy a da Vinci rendszer számára tudjunk magas-szintű utasításokat létrehozni. Jelen esetben még csak tárgyak megfogásának utasítását lehetséges használni.



3.1. ábra. A felhasználó által használható editor felület.

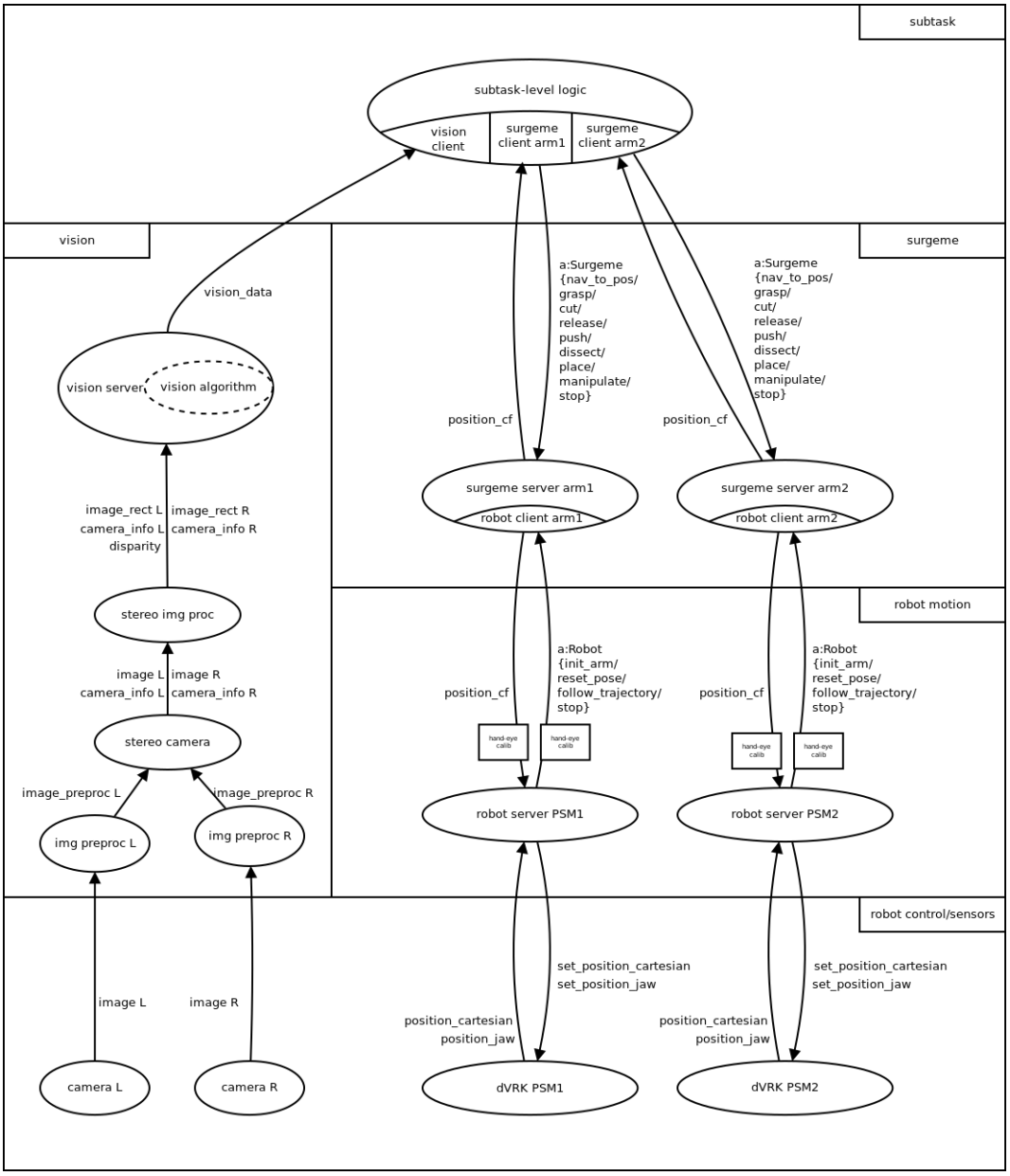
3.1.3. View

Ebbe a modulba töltjük a felhasználók által létrehozott algoritmust, illetve itt van lehetőség annak futtatására. A ”control„ modul tölti be a kiválasztott felhasználó Blockly utasításait, a Blockly utasítások JavaScript nyelvi utasításokra fordulnak ebben a modulban. A különböző Blockly utasítások számára itt lehetséges egyedi JavaScript utasításokat definiálni. A JavaScript-re fordított ágens kódját ”JS-Interpreter„ segítségével futtatom [11]. A ”JS-Interpreter„ egy JavaScriptben írt JavaScript értelmező ”sandbox„ környezet, melyben biztonságosan van lehetőségem elválasztani az ágensek kódjait a böngésző valós JavaScript motorjától. Az ágensek kódja csak és kizárólag egyedileg definiált függvények segítségével tudnak kommunikálni a valós környezettel ezt a dokumentáció ”API hívásnak„ nevezi, jelen esetben a ”JS-Interpreter„-t könyvtárat használó fejlesztő feladata a hivatkozott ”API„ elkészítése. Az utasítások végrehajtása nem valós-időben történik, folyamatosan megvárja a utasítás végrehajtásának sikerességét.

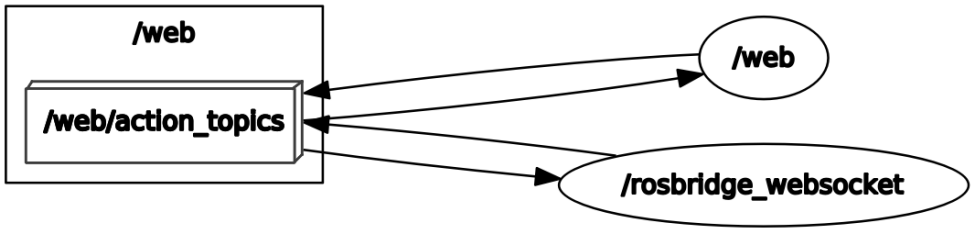
3.2. ábra. A da Vinci számára történő utasítás kiadására alkalmas Blockly vezérlési szerkezet

3.2. Da Vinci webes elérhetősége

Először meg kellet ismerkednem a ROS architektúra és a korábban részletezett 2.2.3 SAF keretrendszer használatával. A SAF felépítését a 3.2 ábra szemlélteti. A SAF ”subtask„ nevű legfelső moduljában található ”megfogás„ művelet külső meghívásának megvalósítását készítettem el. A meglévő SAF kódból leágazva Létrehoztam egy ROS web modult, amely ROS action kezelésére alkalmas. Az új modul a ”web„ témakörre van feliratkozva, innen vár goal definíciókat, amiket kiszolgálhat. Jelenleg csak a megfogás művelet van implementálva, de később kibővíthető a további műveletek elvégzésére is. ROS szolgáltatásként lehetséges elindítani a webes modult. Összefoglalva a következőképpen működik: a ROS web modul a következő részben ismertetett Standard ROS JavaScript könyvtár segítségével lehetséges megszólítani. A webes kérés ROS action formában érkezik. Amennyiben kérés érkezett a SAF iRob Surgical Subtask Automation Framework modult hívja a kérésnek megfelelő utasítással. A mennyiben a kiadott utasítás sikeresen befejeződött ezt egy action alapértelmezett műjkédésének megfelelően egy sikeres callback függvény segítségével visszajelzi a kérést kiadó böngészőnek. A böngésző és a ROS közötti kommunikáció a Standard ROS JavaScript könyvtár működéséből következően WebShocket protokollon keresztül történik. A WebShocket protokol előnye a http-vel szemben, hogy kétirányú kapcsolat könnyű kialakítására alkalmas.



3.3. ábra. A SAF sematikus működése, felépítése [2]



3.4. ábra. ROS web node ki/be meneteli csatlakozásai [2]

3.2.1. Standard ROS JavaScript könyvtár

A Standard ROS JavaScript könyvtár továbbiakban roslibjs a Robot Web Tools része [26]. Alap JavaScript könyvtár, ami a ROS környezettel való böngészőből történő interakció kialakítására készítettek. A Robot Web Tools olyan eszközök fejlesztésének gyűjteménye melyek célkitűzése a ROS rendszer böngészőből történő használatához készítenek, napjainkban is folyamatosan egyre több funkciót lehetséges elérni benne. A könyvtár két részből áll. Biztosít egy JavaScript könyvtárat melyen keresztül kéréseket tudunk megfogalmazni a ROS szerver számára nevezzük ezt most JavaScript könyvtárnak. A Robot Web Tools legfontosabb része a *rosbridge* protokoll ami a kiadott kérések fogadására van. A *rosbridge_server* egy ROS szolgáltatás amit el kell indítani, hogy továbbítsa a kéréseinket megfelelő modulok számára.

JavaScript könyvtár

A segítségével ROS alapértelmezett kommunikációs rendszeréhez lehet kéréseket küldeni, üzenet küldések, szervizhívások, és akcióhívások segítségével. A hívás válaszakor egy eseménykezelő *callback* függvény meghívódik (WebShocket-es esemény hatására), amiben le tudjuk kezelni, hogy mi történjen a kiadott utasítás után. Mint korábban már említettem a hálózati kommunikáció a WebShcoket protokoll [8] használatával történik, ennek a segítségével hálózaton keresztül egyszerű esemény alapú programozást lehet megvalósítani, a JavaScript az ilyen típusú programozás közkedvelt nyelve.

3.3. Da Vinci vizuális programozás összefoglaló

A ROS modulok használata jelenleg Ubuntu 16.04-es verzióval van tesztelve. A Web modulokat én a mindenkori legfrissebb Manjaro Linux alól tesztelem, de elvi szinten bármilyen disztribúció alkalmas ezek használatra, a megfelelő függőségek feltelítése után. Az Ubuntu 16.04-et javaslom virtualizált környezetben futtatni. Az alkalmazás szimulációval történő da Vinci programozás használathoz a következőket kell tenni. A SAF keretrendszer leírásának megfelelően el kell indítani egy dVRK PSM kar szimulációt [**irobotics2020May**]. Majd egy SAF "dummy„ objektumot kell elhelyezni a szimulációban. A szimulált kar állapotát "home„ pozíció kell hozni. "Dummy vision„ elindítás is szükséges. Következő szükséges lépés továbbra is a leírásának megfelelően irob_robot szolgáltatás majd surgeme_server szolgáltatás elindítása. A leírástól eltérve kell folyatni. Ez után a web_actions modul futtatása szükséges. Következő lépésként pedig a " rosbridge_websocket„ futtatása szükséges. Ezek után meg vagyunk ROS oldalról. A következőkben az NodeJS szerver modul leírásának megfelelő utasítások futtatása szükséges [1]. Amennyiben ezzel is megvagyunk az alkalmazás localhost hálózton elindította szolgáltatásait. Melyet modern böngészők segítségivel el lehet érni.

Irodalomjegyzék

[1] aaronrancsik. *thecodergames*. [Online; accessed 20. May 2020]. 2020. máj. URL: https://github.com/aaronrancsik/thecodergames/tree/fork_old_version_for_demo.

[2] Abc-irobotics. *irob-saf*. [Online; accessed 18. May 2020]. 2020. máj. URL: https://github.com/ABC-iRobotics/irob-saf/blob/master/docs/irob-autosurg-blockdiagram.png.

[3] *Blockly | Google Developers*. [Online; accessed 4. Mar. 2020]. 2020. febr. URL: https://developers.google.com/blockly.

[4] Stephen Cooper, Wanda Dann és Randy Pausch. „Alice: a 3-D tool for introductory programming concepts". *Journal of computing sciences in colleges* 15.5 (2000), 107–116. old.

[5] Tamás D. Nagy és Tamás Haidegger. „A DVRK-based Framework for Surgical Subtask Automation". *Acta Polytechnica Hungarica, Special Issue on Platforms for Medical Robotics Research* 16.8 (2019), 61–78. old.

[6] Sebastian Deterding és tsai. „From game design elements to gamefulness". *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek 11*. ACM Press, 2011. DOI: 10.1145/2181037.2181040.

[7] James Devine és tsai. „MakeCode and CODAL: intuitive and efficient embedded systems programming for education". *ACM SIGPLAN Notices* 53.6 (2018), 19–30. old.

[8] Ian Fette és Alexey Melnikov. *The websocket protocol*. 2011.

[9] Brian Harvey és tsai. „Snap!(build your own blocks)". *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, 759–759. old.

[10] Jennifer Jenson és Milena Droumeva. „Exploring Media Literacy and Computational Thinking: A Game Maker Curriculum Study." *Electronic Journal of e-Learning* 14.2 (2016), 111–121. old.

[11] *JS-Interpreter Documentation*. [Online; accessed 20. May 2020]. 2020. febr. URL: https://neil.fraser.name/software/JS-Interpreter/docs.html.

[12] KM Kahn és tsai. „AI programming by children using snap! block programming in a developing country". (2018).

[13] Peter Kazanzides és tsai. „An open-source research kit for the da Vinci® Surgical System". *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, 6434–6439. old.

[14] Jiangjiang Liu és tsai. „Making games a" snap" with Stencyl: a summer computing workshop for K-12 teachers". *Proceedings of the 45th ACM technical symposium on Computer science education*. 2014, 169–174. old.

[15] John Maloney és tsai. „The scratch programming language and environment". *ACM Transactions on Computing Education (TOCE)* 10.4 (2010), 1–15. old.

[16] Jens Monig, Yoshiki Ohshima és John Maloney. „Blocks at your fingertips: Blurring the line between blocks and text in GP". *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE. 2015, 51–53. old.

[17] Michael Montemerlo, Nicholas Roy és Sebastian Thrun. „Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit". *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. 3. köt. IEEE. 2003, 2436–2441. old.

[18] *MRPT – Empowering C++ development in robotics*. [Online; accessed 20. May 2020]. 2020. máj. URL: `https://www.mrpt.org`.

[19] Yoshiki Ohshima, Jens Mönig és John Maloney. „A module system for a general-purpose blocks language". *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE. 2015, 39–44. old.

[20] Erik Pasternak, Rachel Fenichel és Andrew N Marshall. „Tips for creating a block language with blockly". *2017 IEEE Blocks and Beyond Workshop (B&B)*. IEEE. 2017, 21–24. old.

[21] Morgan Quigley és tsai. „ROS: an open-source Robot Operating System". *ICRA workshop on open source software*. 3. köt. 3.2. Kobe, Japan. 2009, 5. old.

[22] Mitchel Resnick és tsai. „Scratch: programming for all". *Communications of the ACM* 52.11 (2009), 60–67. old.

[23] Marc Riar és tsai. „How game features give rise to altruism and collective action? Implications for cultivating cooperation by gamification". *Proceedings of the 53rd Hawaii International Conference on System Sciences*. 2020.

[24] *Scratch - Developers*. [Online; accessed 4. Mar. 2020]. 2019. jan. URL: `https://scratch.mit.edu/developers`.

[25] Pradeeka Seneviratne. „MakeCode Setup Fundamentals". *BBC micro: bit Recipes*. Springer, 2019, 1–28. old.

[26] Russell Toris és tsai. „Robot web tools: Efficient messaging for cloud robotics". *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, 4530–4537. old.

Ábrák jegyzéke