Debreceni Egyetem
Informatika Kar

Multi-modális ember-gép kapcsolat digitális képfeldolgozási aspektusa

Témavezető:
Dr. habil. Fazekas Attila
egyetemi docens

Készítette:
Faddi Viktor
programtervező matematikus

Debrecen
2008

1 Bevezető

Dolgozatom célja az egyetemen elindított Török 2 Sakkozó program mellett egy olyan projekt megvalósítása, ami hasonló funkcionalitást lát el.

A Török 2 egyfajta multi-modális ember-gép kapcsolatot megvalósító program, ami képes lejátszani egy teljes sakkjátszmát élőben, valós sakktáblán.

A rendszer két kamerát használ, egyik a táblát figyeli, és megpróbálja megállapítani az aktuális állást, a másik a játszó embert figyeli, és igyekszik felismerni az érzelmeit.

A lépéseket egy sakk szerver dolgozza fel, ami alapján előállít a gépi játékosnak egy lépést, amit egy robotkar hajt végre.

A játék közben egy talking head jelenik meg a képernyőn, ami képes beszédet, és érzelmeket is imitálni. Az elmondott szövege a képernyőn is megjelenik, és élőszóban is elmondja.

Célom volt egy ehhez hasonló táblás játék elkészítése, ami felhasználja a kamerákat, a robotkart, és a talking head-et.

Mivel a robotkar csak sakktáblán való használatra volt tervezve, ezért egy olyan játékot kellett találnom, ami szintúgy ezen a táblán játszódik, és viszonylag ismert. Úgy gondoltam, hogy erre a legalkalmasabb a dáma játék.

Sajnos, a projektet teljes mértékben nem sikerült megvalósítanom. A talking head-et nem használtam fel végül, mivel ez lett volna az utolsó lépés, de nem maradt már rá idő. Ennek következtében a talking head felhasználásának leírása nem képezi részét a dolgozatomnak se.

A dolgozatom első fejezetében áttekintem, hogy milyen szórakoztatóipari termékek jelentek meg eddig, és milyen fejlődés alatt álló projektek vannak, amik a multi-modális ember gép kapcsolaton alapulnak. Ezek között megemlítem a szórakoztató robotok terén elért fejlődéseket, beszélek a virtuális valóságról, illetve a játékok irányítására szánt eszközökről is. Ezenkívül kitérek még a Török 2 vázlatos ismertetésére, valamint az áttekintés végén ismertetem a dámajáték szabályait és történetét.

A következő fejezetben a rendszer struktúráját ismertetem, illetve, hogy milyen eszközöket használtam fel. Ettől a fejezettől kezdve figyelembe fogom venni az implementációs nyelv specialitásait is, vagyis a C# programozási filozófiáját, és eszközrendszerét.

Ezután a rendszerhez illesztett, mások által írt részeket ismertetem egyesével. Itt kerül bemutatásra maga a robotkar, és az azt vezérlő program. Az SVM ismertetése, amit az állás felismerésénél használok fel. Végezetül a DSSamplert mutatom be, ami a kamerából érkező képsorozatok feldolgozásában segített.

A következő fejezetek az egyes nagy problémaköröket, és annak megoldásait tartalmazzák, melyekbe a projekt folyamán ütköztem.

Ezek közül elsőként a mesterséges intelligencia részét tárgyalom. Mesterséges intelligencia alapjául az iskolai tanulmányaim során megismert általános módszert ismertetem, és hogy hogyan használom fel, illetve beszélek az egyes lépések meghatározásáról is.

Következőként a legproblémásabb részt tárgyalom, az állásfelismerést. Ez két részből áll. Első részben a lépés érzékelését mutatom be, a második részben pedig az új állapot feldolgozásának menetéről írok. Mivel nehéz hibátlanul érzékelni az új álláson, hogy hol vannak a bábok, SVM-et használtam fel a jobb felismerés érdekében, illetve egy általános felismerési menetet adtam, amire implementáltam az általam választott módszert, ami a HSI transzformáción alapult. Végül erre teszteket végeztem el, amivel vizsgáltam, hogy a példa mezőállapotok (tananyagok) növelésével hogyan nő a helyes állásfelismerés valószínűsége.

Az utolsó fejezetben a különálló részek kommunikációját ismertetem. Ide tartozik a robotkar és a vezérlő közötti, illetve a táblafigyelő és vezérlő közötti kommunikáció. A robotkar, és vezérlő közti kommunikáció esetén csak a vezérlőnek kellett utasításokat küldenie a robotkarnak, ami utasítássorozatokból állt, nem egyedi utasításokból. A másik kommunikáció a vezérlő, és a táblafigyelő rész között zajlott, amit eseménykezelésszerűen kellett megvalósítanom, mivel a figyelő küldött a vezérlő felé üzeneteket, de nem az vezérelte a kommunikációt.

Az összefoglalásban leírom tapasztalataimat és összefoglalom milyen problémák merültek fel, és azokra milyen megoldásokat alkalmaztam. Végezetül projektem továbbfejlesztési lehetőségeiről beszélek.

2 Irodalmi áttekintés

A számítógépen játékot először A.S. Douglas írt 1952-ben. Azóta sokat fejlődtek, és egyre fontosabbá vált, hogy minél inkább részt tudjunk bennük venni, minél inkább multi-modális legyen. Ennek több oka is van, főként az, hogy sokkal szórakoztatóbb legyenek, de terápiás okok miatt hasznos dolog. A számítógépes szórakozás mára nem csak magára a számítógépre korlátozódik, hanem ez kiterjed számítógéppel irányított robotokra. A robotok feladatok nagyon széles skáláját tudja megoldani, ezek közül csak egy nagyon kis rész a szórakoztató robotok. Elsőként a robotok két fajtáját veszem szemügyre, és ezek alkalmazását.

Az egyik fajta robot lényege, hogy minél kevesebb emberi beavatkozás legyen szükséges az irányításához, tudjon önállóan cselekedni. Ezeket főként embertől távol eső helyeken, akár emberre veszélyes környezetben alkalmazzák. Tudniuk kell alkalmazkodni a dinamikusan változó környezethez, döntéseket hozni a váratlanul felmerülő problémákra. A másik alkalmazása az önálló robotoknak a gyártásban van. Itt ugyanis felmerülnek olyan problémák, amik sablonosak, és nagy precizitást igényelnek. Ilyen feladatokra is tökéletesek, mivel gyorsak és pontosak tudnak lenni.

A második szempont, ami szerint robotokat készítenek, nem az önálló működés, hanem pont ellenkezőleg, tudjanak együttműködni az emberekkel, megfelelően reagáljanak a cselekedeteikre. Ilyen eszközöket többféle helyen is alkalmaznak, például egészségügy, háztartás, vagy ép a szórakoztatóiparban.

A legkomolyabb robotok egyértelműen az egészségügyben vannak. Ezeknek a feladatuk például operációk elvégzése az operáló orvos irányításával. Ezeknél az orvos egy irányító eszközt használ, és a robot végzi helyette a műtétet. A jelenlegi legfejlettebb ilyen eszköz a Da-Vinci elnevezésű műtőrobot. Ennek a legutóbbi fejlesztése, hogy az orvos szemével is tudja irányítani a készüléket, illetve képfeldolgozó, érzékelő rendszerének köszönhetően sokkal több információt juttat a műtőorvoshoz, mintha simán kamerával nézi (Agent Portal, Da Vinci újratöltve).

A háztartásokban is hasznos egy-egy segítő robot. Ilyenek például a porszívó, felmosó, és különböző takarító robotok, vagy amik képesek fűnyírásra. Ezeknek tudniuk kell a terepet

felderíteni, akadályokat kikerülni, és mindent bejárni a tökéletes tisztítás érdekében, és utána visszatérni töltőállomásukra.

A robotok fejlesztését maga a szórakoztatóipar is szorgalmazza, ezeknél a robotoknál nem csak az együttműködés, hanem érzelmek produkálása is fontos szerepet játszik. Komoly kutatások folynak ebben az irányba. Ilyen projektre példa a Kismet, vagy a KASPAR nevezetű robot. Forgalomba jött robotra példa a Robopetek, amik valamilyen állat robotmásolata. Ezek jól utánozzák az eredeti állatot, képesek trükkök elvégzésére, értenek a hangparancsból, felderítik a terepet, és tanulni is képesek.

A Számítógépes játék multi-modális ember-gép kapcsolat megközelítésének egy másik módja, a virtuális valóság megteremtése. Míg az előző megközelítés azt célozza, hogy a gép minél inkább együtt tudjon működni az emberrel, addig a virtuális valóság megpróbálja belehelyezni az embert a számítógép által generált környezetbe. Ez a technika persze még eléggé gyerekcipőben van.

Az első ilyen technika egy kamerát, és képfeldolgozási módszereket alkalmaz. Ez a módszer úgy oldja meg, hogy az ember beáll a kamera elé, és a gép meg megpróbálja felismerni. Ha meg van a játékos, megjeleníti a felületen, belehelyezve a virtuális játéktérbe. Ezután általánosan már csak a játékos mozgását követi, ami a legegyszerűbb, és leggyorsabb technika. Mivel az ilyen játékok erősen korlátos processzorú gépekre készülnek, mozgásérzékelésnél bonyolultabb technikát nem igazán alkalmaznak. Erre játékkonzolokhoz készültek is speciális kamerák, amik kifejezetten támogatják az ilyen felhasználást. Ilyen eszköz a Game Boy Camera, a DreamEye , ami Sega Dreamcaston lehetet használni, illetve a legnépszerűbb az Eyetoy, ami Playstation 2-re illetve PSP-re készült.

Az előbb bemutatott technika viszonylag egyszerűen, és olcsóan megoldható, de közel se mondható el, hogy benne vagyunk abba a környezetbe, amit generál nekünk a számítógép. Az lenne az igazi, ha teljes egészében látnánk, és érzékelnénk a virtuális környezetet, abba a valóságba kerülnénk bele. Ezt a másik valóságot virtuális valóságnak nevezik, és elég népszerű kutatási, fejlesztési irány. Ez a technika főként látványban igyekszik elérni a hatást, amire több fajta megoldás is létezik. Ezek közös jellemzője, hogy valamilyen szemüveget használ. A legegyszerűbb ilyen technika egy olyan szemüveget használ, aminek az egyik lencséje vörös, a másik kék szűrővel van ellátva. Ennek lényege, hogy a képet felbontja kék,

és vörös tartományok között, ezeket egymástól eltolva két képet előállítva egymáson. Ez az eltolt kép egyik fele a vörös szűrőn keresztül látszik csak, a másik meg csak a kéken, és ez 3D érzetet kelt. Jómagam is voltam egyszer egy mozielőadáson, ahol egy természetfilmet vetítettek le ilyen technikával, és elmondhatom eléggé érdekes, és izgalmas volt, mikor a tengeri állatok az orrom előtt „úszkáltak a levegőben”.

Vannak további technikák is, amik jobb hatást érnek el, illetve bonyolultabb technikát alkalmaznak Az összes ilyen megoldás bemutatása helyett csak a legmodernebb technikáról beszélek, ami kereskedelmi forgalomban nem is kapható, hanem inkább csak katonai területeken, illetve különböző olyan területeken használnak, ahol ilyen szimulációs technikával tanítják az újoncokat.

Ezek a virtuális szemüvegek két beépített monitorral rendelkeznek, és a 3D hatást a szem látás módjának figyelembe vételével érik el. Az emberi szemek a világot, és a tárgyakat, amit vizsgálnak, különböző szögben teszi. Ebből az agy generálja a valós 3D látást, ennek köszönhető, hogy érzékeljük a távolságot. A 3D szemüvegeknek az a feladata tehát, hogy ezt biztosítsák, amit legjobban sztereó kép képzésével tudják elérni. Ezek az eszközök nemcsak képet közvetítenek, hanem 3D hangot is, sőt voltak kísérletek olyan eszközökre is, amik szagot is tudnak közvetíteni. Ezek a modern eszközök nem csak a látványt teszik lehetővé, hanem azt is, hogy ezzel a virtuális térrel interakcióba lépjünk. Ezt érzékelő kesztyűvel, vagy akár teljes testre szerelhető érzékelőkkel érik el, tehát, ha mozgatjuk a kezünket az látható a virtuális térben, ha forgatjuk a fejünket, azzal körbenézhetünk. Ezek a technikák jelenleg még nagyon drágák, és minden bizonnyal még sokáig az is maradnak. Természetesen vannak a két véglet között köztes technikák, amik kereskedelemben is elérhetőek, de attól függetlenül nem nagyon elterjedtek. Ennek oka, hogy némelyik technikához magát a szoftvert is úgy kell fejleszteni, de a legtöbbjük talán azért nem használt széles körben, mert ezek rendkívüli módon igénybe veszik a szemet.

A játékokkal való interakció jelenleg legelterjedtebb módjai a célra kifejlesztett irányítóeszközzel vannak megoldva. Bár sok különböző kísérlet van és fejlesztés van ezen a területen, mégis eddig a legnépszerűbbek ezek a módszerek. Következőben ismertetem, hogy milyen változatai vannak.

Elsőként a Joystick jelent meg ilyen célra. Ezt a repülőgépek irányítójáról mintázták, tehát egy botkormány volt, akkor még egyetlen gombbal. Ez az eszköz négy irányba való mozgatást, illetve „tüzelést” tett lehetővé. Az eszköz az egyik korai számítógépes játékhoz készült, ami a Spacewar volt. Ez az eszközcsalád manapság már, mind formatervezésben, mind használhatóságban repülőgép szimulátorok irányítására hegyezik ki.

Második, ennek egy leszármazottja, a D-Pad (directional pad rövidítése), ami egy olyan kis kézre álló eszköz, aminek a bal oldalán van a négy irányba mutató gomb, jobb oldalon meg a tüzelő gombok, középen meg select és start gombok szoktak lenni. Ez irányítás szempontjából sokkal könnyebb használni, illetve sokkal kisebb is, elfér a kézben. Az eszközön csak négy irány van, és az átlók ezeknek a kombinációjaként jön létre, így kiegészítve 8 iránnyá. A mai konzolos játékoknál ez a legdivatosabb. Tervezés szempontjából az ergonómiára, illetve minél bővebb funkcionalitásra törekszenek, vagyis hogy minél több gomb legyen rajta úgy, hogy közben tökéletesen kézre álljon.

A joystick egyre inkább a repülőgép szimulátorok felé tolódott, viszont volt egy másik nagyon népszerű játék, méghozzá a versenyautózás. Ennek a realisztikusságának növelése érdekében született meg a játék kormánykerék, és a pedál. A kormány képes érzékelni, hogy milyen szögben áll, illetve a pedál, hogy mennyire van lenyomva, ezzel finom szabályozhatóságot adva a versenyautós játékoknak. A jelenlegi legrealisztikusabb kormánykereke a Logitech G25 elnevezésű terméke, ami a többivel ellentétben nem 2, hanem 3 pedált, és egy 7 sebességes botkormányt is tartalmaz.

Utolsónak az irányítók tekintetében az utóbbi idők legnagyobb szenzációját mutatom be, ez a Nintendo Wii játékkonzol irányítója. Ez az eszköz ugyanis képes érzékelni azt, hogy hogyan mozgatják a levegőben. Ezzel lehetővé tették, hogy egy virtuális eszköz mozgását egy valós eszköz mozgása alapján határozzák meg. Ilyen lehet pl. egy teniszütő, bokszoló kéz, kard, stb. A Wii távirányító a mozgást egy infravörös érzékelővel tudja érzékelni, és a szabad mozgathatóság érdekében bluetoothon keresztül valósították meg a kapcsolatot.

Mint látható komoly kutatások folynak, és eddig is komoly eredményeket ért el a tudomány a multi-modális ember-gép kapcsolat terén, és ebben a Debreceni egyetem is részt kíván venni. Saját projektünk a Turk 2 egy olyan játékot kíván megvalósítani, amivel jó szórakozás táblás játékokat játszani. Ennek megvalósítása érdekében a következő szempontoknak kíván eleget

tenni: Teljes sakkjátszma lebonyolítása valós táblán, legyen valamilyen megjelenése, amivel lehet szóban kommunikálni.

A játék valós táblán való lejátszását 3 lépcsőben éri el. Elsőként egy kamera figyeli felülről a táblát, ami felismeri az állást. A felismert állapotot egy sakkszervernek küldi tovább, ami megállapítja, hogy mi a teendő, milyen lépést kell végrehajtania. Ezt végül elküldi a robotkar vezérlő programjának, ami végrehajtatja a robotkarral a lépést más bábok érintése nélkül.

Ebben a rendszerben az állásfelismerőt Nagy András készítette. Az állásfelismerő a kamerától érkező képsorozatból megállapítja, hogy mikor történt lépés, és ha lépés történt, azt értelmezi. A lépések értelmezése az üres tábla képe, illetve az előző lépés óta történt változások alapján történik. Mivel nincs az a felismerő program, ami felülről meg tudná állapítani, hogy hol milyen bábu van, ezért nem azt ismeri fel, hanem csak annyit, hogy honnét hova történt lépés. A program feltételezi, hogy kezdetben helyesen volt elhelyezve minden bábu, illetve amikor egy lépés történt, akkor a sakk szabályainak megfelelően meg is vizsgálja, hogy helyes volt-e a lépés, és a szervernek már csak ezt küldi el.

A következő rész a vezérlő program, amit Varga Péter készített. Ennek a résznek a feladata, hogy az egész rendszert összekösse, koordinálja. Ez a rész fogadja az állásfelismerőtől a lépést, amit egy sakkszerverhez továbbít. A sakkszerver egy külső komponens, amit úgy szereztek be a projekthez. A szerver által visszaküldött lépést végül közvetíti a robotkar felé, ami végrehajtja. A robotkar Sándor Ákos munkája, és mivel én is felhasználtam, később még részletesebben leírom működését.

Végső rész, ami egy interaktív felhasználói felületet kíván adni a sakkprogramhoz. Ez két részből áll.

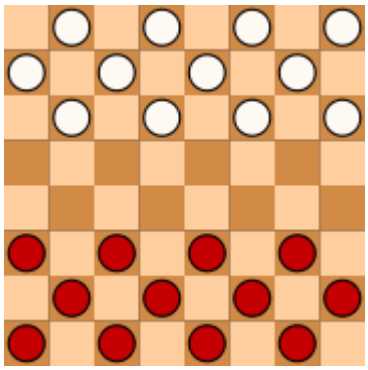
Az egyik a talking head, amit Kovács György készített. Ez a program egy arcot jelenít meg, ami képes gesztusokra is, illetve hangszórón keresztül mondja el üzenetét. Ezenkívül lehet vele hang alapú kommunikációt is folytatni. A másik része egy kamerával figyeli a játékost, és megpróbálja felismerni az érzelmeit. Ez utóbbi Hingyi György, illetve Szabó Péter munkája.

Ezek együtt valósítják meg a kitűzött célt, amit tovább szeretnék vinni egy dámaprogram kidolgozásával, illetve további tapasztalatokkal, ötletekkel gyarapítani a projektet.

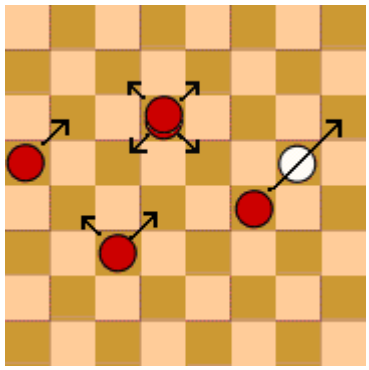
2.1 A dámajáték szabályai

A dámajátéknak szerte a világon nagyon sok változata létezik. A programomban az angol dámajáték szabályait valósítottam meg. (Forrás Wikipedia, English draughts címszó.)

A dámát két játékos játssza egy 8x8-as táblán. A játék kizárólag a fekete mezőkön folyik. A táblát úgy kell elhelyezni, hogy a fekete mezőjű sarok a játékos bal felén legyen. A játék kezdetén mindkét játékosnak 12 korongja van, melyeket gyalognak hívnak, és induláskor a baloldali ábrán látható módon kell felállítani.



Kezdő állás



Példa lépések és ütés

A játék közben a gyalog mindig átlósan előre mozoghat a szomszédos mezőre, ha az a mező szabad. Ezt a lépést sima lépésnek nevezik. Ha a gyalog előtt átlósan az ellenfele egyik korongja van, és az átlón közvetlen utána egy üres mező van, akkor átugorhatja, és leveheti az átugrott bábut. Ezt hívják ütőlépésnek, és az angol szabályok szerint csak előre lehet megtenni a gyaloggal.

A játékosok felváltva tesznek egy-egy lépést, és mindig a sötét fél kezdi. Ütni a dámajátékban kötelező. Ha ütés után az ütő korong tudna még egy ütést tenni, akkor ezt az ütést kötelező szintúgy megtenni, és ezt addig kötelező folytatni, amíg csak lehetséges. Ha több ütés-variáció van, akkor az angol dámajáték szabályai szerint tetszőlegesen lehet választani közülük. Ha egy gyalog eléri az egyik legtávolabbi mezőt (dáma mezők), akkor dámává változik.

A dámára alapvetően ugyanazok a szabályok érvényesek, mint a gyalogra, annyi különbséggel, hogy lépni, és ütni is tud hátrafele. Ha egy gyalog ütés sorozat közben elér egy

dáma mezőt, dámává változik, és így lehetősége nyílik további ütések megtételére, akkor ezt megteheti, sőt ez is kötelező.

A játékot az veszti el, akinek elfogy az összes korongja. A játék véget érhet döntetlennel is, ha a lépő játékosnak nem fogy el az összes korongja, de nem is tud további szabályos lépést megtenni.

2.2 A dáma története

Forrás a Zalaegerszegi dámajáték sportegyesület honlapjáról származik. A dáma története nem egyértelműen ismert, ez a leírás a dáma történetének csak a legnépszerűbb változatát tartalmazza.

2.2.1 Mi az a dámajáték?

A dámajáték kb. 5 ezer éve létezik (ismert volt már az ókori Egyiptomban is), és nem merült a feledés homályába. A szabályok egyszerűsége, a játék mélységének és szépségének elképesztő összefonódása megmagyarázza, miért adták tovább a játékot a különböző országok népei nemzedékről nemzedékre. Természetét nagyon találóan jellemezte J.F.Fomin, egy orosz tudós: „A dáma művészet (és nem helyettesíthető a filmmel vagy a zenével), a dáma sport (és nem könnyebb, mint a súlyemelés vagy a jégkorong), a dáma tudomány (bár nem matematika). A dáma mindez együttvéve. Végezetül a dáma –meglepően egyszerű szabályaival és fantasztikus taktikájával – maga a *dáma*”.

A dámajáték nem csupán remek szórákozás és kellemes időtöltés; az egyszerű szabályok mellett lenyűgözően érdekes, bonyolult pozíciós manőverekkel és kombinációs ötletekkel teli játék. Szellemi mérkőzés, ahol elképzelések csapnak össze, a rövid- és hosszú távú tervezés sikerességét láthatjuk a táblán. Ez a játék ösztönöz is, türelemre nevel, gondolkodásra tanít. Ma is sok helyen játsszák világszerte; az Európai Unió legtöbb tagországában hivatalosan elismert, Magyarországon viszont nem.

2.2.2 A dámajáték rövid története

A dámajáték első tárgyi emlékei (agyagból égetett táblák, hozzá tartozó kúp alakú bábok) még az egyiptomi Ó-birodalom ötödik fáraó-dinasztiájának korából (Kr.e. 2544–2407) származnak. Több történész és egyiptológus írta le a művében, hogy az ókori Egyiptomban sokan szerettek dámázni – szegények és gazdagok, felnőttek és gyerekek. A tehetősebbek

kényelmes karosszékekben ülve, elefántcsontból faragott figurákkal játszottak, a szegények pedig földre rajzolták a dámatáblát, a bábokat: magokat, terméseket, kavicsokat a természet adta.

Az ókori Görögország népei az egyiptomiaktól tanulták el a játékot. Érdeklődők gyűrűje vette körül a játékosokat a köztereken és az utcákon, ahol rendszerint folyt a dámajáték. A görögök nagy becsben tartották az ügyes dámajátékosokat, a legjobbak nevét az utókor is megőrizte: Diodórosz, Teodórosz, León.

A Földközi-tenger partjain élő népek már az ókorban is szoros kapcsolatban álltak egymással. A dámajáték a görög utcákról nemsokára átterjedt Róma lakóházaiba és fórumaira is, persze, más nevet kapott, és kicsit más táblán játszották.

A Római Birodalom légionáriusai az akkor ismert világ szinte minden országába eljutottak, és fegyvereik mellett magukkal vitték a dámajátékot is: a fából, kőből vagy bőrből készült táblát s a hozzá tartozó, viaszból, kőből vagy égetett agyagból készült, félgömb alakú figurákat. Így ismerte meg a játékot Európa, Afrika és Ázsia sok országa.

A középkori Európában már szinte minden országban ismerték és játszották a dámát, mégpedig 64-négyzetes táblán. Közismert, hogy ezekben az időkben Európában a dámajáték főnemesi játékká vált, és a „hét lovagi tevékenységhez" tartozott (tudni kellett lovagolni, úszni, gerelyt vetni, vívni, vadászni, dámajátékot játszani, dalt szerezni és énekelni szíve hölgye tiszteletére).

Bár Portugáliában már a 15. században országos dámajáték-versenyeket rendeztek, a nemzetközi dámajáték szabályait csak jóval később (1740-ben) írta le a francia Laclef „Coups de paries de Dames de la Polonoise" című könyvében.

Az első, főleg a dámajáték angol változatának elméletéről szóló könyv Londonban jelent meg 1756-ban (W.Payne: „Introduction to the Game of the Draughts").

A több ezer év során a dámajátékban is történt fejlődés. Különböző országokban, a tábla méreteiben, a játékszabályokban, bábfigurákban változások mentek végbe, emiatt jelenleg tíznél is több dámajáték-változat (angol, orosz, kanadai, török, német, spanyol, brazil, francia stb.) létezik.

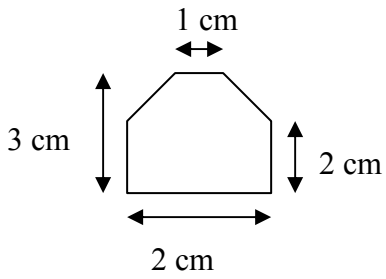
De már majdnem 300 évvel ezelőtt megjelent egy olyan dámajáték-változat, amelyből később a nemzetközi dámajáték is kialakult, hogy a különböző országok játékosai meg tudjanak mérkőzni egymással, és kiderüljön, ki a legjobb a világon.

3 A dámaprogram

3.1 A program felépítése

A dámaprogramhoz a következő eszközöket használtam fel: Egy webkamera, ami felülről figyeli a táblát, és az azon történt lépéseket. A robotkar, ami végrehajtja a gép lépéseit. A tábla, amin a játék folyik. Ezek egy asztalra vannak rögzítve, hogy minden eszköznek meglegyen a pontos helye.

A dámabábok általában korongok szoktak lenni. A gyalogot 1 koronggal, a dámát 2 egymásra helyezett koronggal jelzik. Mivel a robotkar képtelen lenne korongokat megfogni ezért a feladatra speciális bábokat készítettem egy asztalossal. A bábok henger alakúak, és a keresztmetszetük a jobb oldali ábrán látható. Az így kidolgozott bábokat már képes volt megfogni a robotkar.



A sötétbábok sötétvörös anyaggal lettek bevonva, és a világos bábok meg nem lettek színezve egyáltalán. A gyalog bábok a dámabáboktól a felső részükön lett megjelölve más színnel, ezzel elérve, hogy a játékos, és a gép is képes legyen megállapítani, melyik milyen báb.

A szoftveres része 3 programból áll, amik egymással tcp kapcsolaton keresztül kommunikálnak. Ennek következtében a három program akár külön-külön gépen is futhat. A három program: a robotkar vezérlő, a táblafigyelő, és a dámavezérlő. Ezek közül az utóbbi kettőt írtam én.

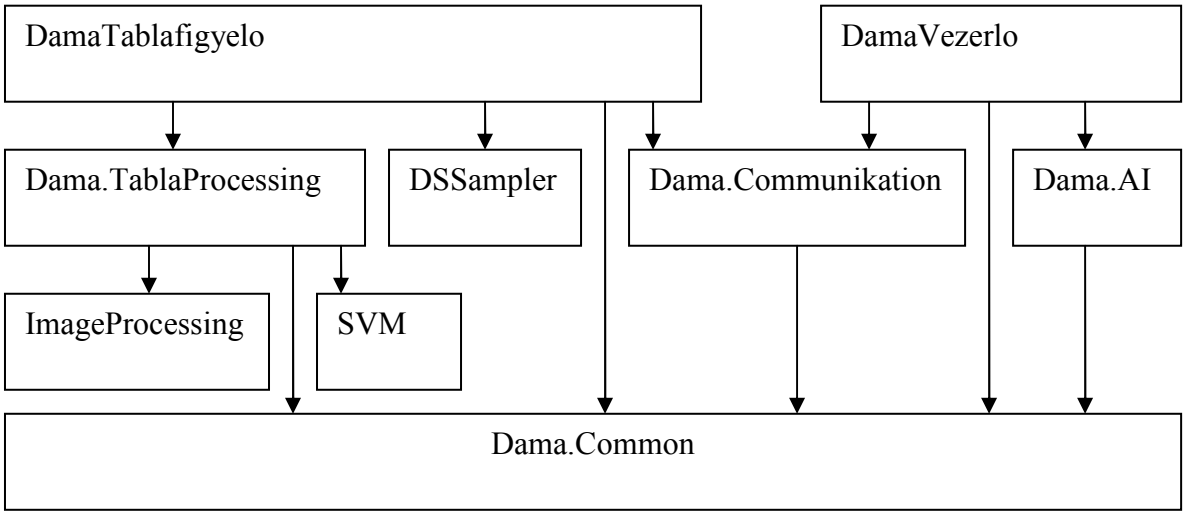
A robotkar vezérlő program. Ennek lehet küldeni a lépéseket, és végrehajtatja a robotkarral.

A táblafigyelő program, ami érzékeli, hogy történt-e lépés, ha történt, akkor feldolgozza, és az eredményt elküldi a vezérlőnek.

A vezérlő, vezérli a működést, és előállítja a gépi lépéseket.

Az egyes részeket C# nyelven valósítottam meg, így az ebben a nyelvben felépíthető programstruktúra mentén fogom elmondani az egyes egységeket is. Ez a nyelv, és a hozzátartozó rendszer legnagyobb egysége, amivel feltördeli a programot, az assembly, ami egyetlen fájlból áll. Az assembly egy logikai egység, ami saját névtér hierarchiával rendelkezik, rendelkezni tud assembly szintű bezárással, és tipikusan egy adott célt (feladatot) megvalósító osztály, és névtér gyűjtemény.

A programom egy közös assembly könyvtárstruktúrát használ, ahol mindegyik assembly egy bizonyos feladatot lát el. Ezek a részek kapcsolatai (Melyik assembly melyik assmeblyt használja fel) az alábbi ábrán látható.



Az egyes assemblyk a következő feladatokat látják el:

Dama.AI: Gép általi lépéseket meghatározó assembly. Részletesen a 3.3 fejezetben foglalkozok vele.

Dama.Common: Ebbe az assemblybe három fontos dolog került.

- A dámatábla megjelenítő Control. Mivel a Táblafigyelőben, és a vezérlőben is ki kell jelezni az állást, ide került a tábla megjelenítéséért felelős control. A tábla tartalmát tetszőleges irányba el lehet forgatni. A kamera nézési iránya az ember nézési irányától balra 90 fokkal el volt forgatva, ezért a megjelenő kép is ennek megfelelő volt, illetve táblán letiltható, és engedélyezhető a mezők tartalmának módosítása.

A táblafigyelőn a valós kép feldolgozott változatát is meg szerettem volna jeleníteni pontosan ugyanúgy, ahogy a képen látszik, ezért a dámatábla megjelenítőn is ugyanúgy elforgatva kellett, hogy látszódjon. A vezérlőn viszont úgy szeretnénk látni, ahogy az ember előtt szokott lenni a tábla (ha a kezdő játékossal játszik), tehát az A1 pozíció a bal alsó sarokban van.

A szerkeszthetőség a táblafigyelőben volt nagyon fontos, hiszen ezzel lehet elérni, hogy tanítsuk a felismerőt. Ha egy pozícióban nem ismeri fel jól akkor bal, illetve jobbklikkel lehet módosítani a mező tartalmát, illetve a középső gomb visszaállítja alapállapotba.

- Adatkonverziók megvalósítása. Mivel az egyes részek különböző formában állítják elő az adatot, illetve más formában várják, ide kerültek azok a konvertáló függvények, amik az átalakítást végzik.

- Azok az interfészek, amik lehetővé teszik az adatátadást az egyes részek között olyan irányba, amerre nincs közvetlen kapcsolat. Ezeket a tábla megjelenítő, és az adatkonvertáló is felhasználja, mivel felsőbb rétegben vannak az adatok kezelő osztályai, és így nincs közvetlen kapcsolat.

Dama.Communication: Összeköttetést létesít a Táblafigyelő, és vezérlő között, illetve a robotkar, és vezérlő között. Részletesen a 3.5 fejezetben foglalkozok vele.

Dama.TablaProcessing: A bemeneti képek feldolgozásáért felelős. Részletesen a 3.4 fejezetben foglalkozok vele.

DSSampler: A kamerától olvassa a bemenetet, és továbbítja feldolgozásra. Részletesen a 3.2.2 fejezetben foglalkozok vele.

ImageProcessing: Általános kép transzformáló algoritmusok gyűjteménye, amit a mezők feldolgozásához használok.

SVM: Support vektor machine. Az egyes mezőkből kinyert adatokat ezzel a tanulóalgoritmussal dolgozom fel. Részletesen a 3.2.3 fejezetben foglalkozok vele.

3.2 Felhasznált komponensek, programok

A dámaprogramban, az általam írt részeken kívül több mások által készített programot, komponenst, és eszközt is felhasználtam, mivel egy eléggé összetett feladat, az egész rendszer megvalósítása.

3.2.1 Robotkar és a vezérlő programja

A felhasznált dolgok közül a legfontosabb a robotkar, mivel nélküle nem lehetne a lépéseket megtenni. Ez az eszköz Sándor Ákosnak köszönhetően jött létre.

Dolgozatomban bemutatom nagyvonalakban a működését, és a vele való kommunikációt.

Ákos által készített robotkar egy un. négy szabadságfokú robotkar. Ez annyit tesz, hogy négy izülettel rendelkezik. Ez az eszköz a következő követelményeknek tesz eleget:

- Képes a sakktábla bármely mezőjét elérni.
- Hozzáfér az egyes bábokhoz, anélkül, hogy a környező bábokat érintené.
- Bármilyen bábot képes megfogni, aminek nagysága elér egy határt. (A dámabábúim voltak az alsó határ.)
- A bábokat képes függőlegesen felemelni illetve lerakni.
- A bábok áthelyezésének ideje megfelelő időn belül történik.
- Képes a táblán kívüli területre pozícionálni.

A bábok mozgatásáért felelős rész összesen három részből áll: a robotkar, a vezérlő, és a program, ezekről röviden:

- A robotkar maga a mechanikus egység, ami a lépéseket végzi. Működéséhez egy olcsóbb megoldásokat használ, méghozzá elektronikus motorokat, és Bowden huzalokat. Ez pontatlanabbá teszi, mintha hidraulikus lenne. Ez nehézségeket okozott a dámabábok esetében. Szerencsére nem okozott akkora pontatlanságot, tehát még mindig mezőn belül helyezte el a bábokat, így lehetséges volt a felismerésük.

- A vezérlő egység még a hardveres részhez tartozik, de a mechanikán kívüli eszköz. Ennek feladata, hogy összekösse a számítógépet a mechanikus robotkarral. A számítógéptől érkező utasításokat átalakítja elektronikus jelekké, ami alapján a robotkar mozogni tud, illetve az onnét érkező jeleket értelmezi, és ezekből adatokat nyer ki, amikkel vissza tudja jelezni a számítógép felé a robot állapotát. A vezérlő ezentúl képes megállapítani, hogy megfelelő szoftvertől érkeznek-e az utasítások, tehát hitelesíti a szoftvert.

- A számítógépes szoftver a kapott utasításokból meghatározza a szükséges mozdulatsorokat. Mivel a vezérlő csak annyit tud, hogy beállítja az adott ízületeket adott pozícióra, és visszaadja az aktuális pozíciókat szükség szerint, a számítógépes szoftvernek kell meghatároznia, hogy mikor melyik ízület mozogjon.

Szempontomból a legfontosabb rész a számítógépes szoftver, mivel azzal kommunikálok.

Indítása „RobotVez.exe beallitas.tbl port” paranccsal történik.

Ebben az utasításban a beallitas.tbl, egy konfigurációs fájl, ami pozícionálási adatokat tartalmaz. Ezek az adatok a globális táblamagasság, bábok elejtési magassága, mezők pozíciója, és fogási magassága. A dámához tartozó beállításokat a dama.tbl fájlba mentettem.

A port, meg a kommunikációhoz szükséges port, számomra a 2001-es.

A szoftverhez tcp kapcsolaton keresztül tudok kapcsolódni, és egyszerű szöveges utasításokat tudok kiadni, amire szintén szövegesen válaszol, mikor végrehajtotta az utasításokat.

Miután csatlakoztam „RobotKar V1.0” választ küld vissza, jelezve, hogy készen áll a parancsok fogadására.

A végrehajtott parancsok után a válasz „OK”, ha sikeresen végrehajtotta, illetve „Hibas parancs”, ha nem megfelelő utasítást küldök.

Az egyes küldhető parancsok (A parancsokba a <param> rész jelöli a paramétereket):

„RobotSakk: Alap” Alaphelyzetbe állítja a robotkart. Ezt célszerű a használatának befejezése előtt megtenni.

„RobotSakk: help” Segédletet küldi vissza.

„RobotSakk: Foleall <mező> <dx> <dy> <várakozás>” A megadott mező fölé állítja a robotkart.

„RobotSakk: Felvesz” Miután a megadott mező fölé van állítva a kar, felemeli a bábot.

„RobotSakk: Letesz <báb>” Leteszi a bábot a mezőre, amelyik fölött áll.

„RobotSakk: Atemel <báb> <startmező> <dx> <dy> <célmező> <dx> <dy> <Kmag> <Amag> <Emag>” A startmező fölé áll, felemeli az adott bábot, a célmező fölé áll, és végül leteszi a bábot. Egyszóval megtesz egy lépést.

„Exit” Robotvezérlő bezárása, kapcsolat bontása.

Az előforduló paraméterek:

<báb>: Ahol a bábot leteszi, ott szükséges a báb típusa, hogy tudja, milyen magasan kell elereszteni. Lehetséges értékei: Paraszt, Bastya, Lo, Futo, Kiraly, Kiralyno, Dama

<mező>: egy mezőpozíciót ad meg a sakk szabályai szerint. Értéke: [a..h][1..8]

<dx>, <dy>: Vízszintes, illetve függőleges irányú eltolás. Ezzel lehet egy adott mezőhöz képest egy pozíciót megadni. Feladata, hogy a bábokat ki lehessen rakni a tábláról, illetve kívülről be lehessen hozni.

<Kmag>: Megközelítési magasság.

<Amag>: Átemelési magasság.

<Emag>: Elejtési pluszmagasság.

Átemeléskor a robot először beáll a startmező fölé, a megadott <Kmag> megközelítési magasságban, majd megemeli azt <Amag> átemelési magasságig, áthelyezi a célmező fölé szintén az átemelési magasságban, majd a báb típusának megfelelő letételi magasságból, hozzáadva az <Emag> elejtési pluszmagasságot, elengedi a bábot.

3.2.2 DSSampler

A második legfontosabb dolog ami kellett a projekt megvalósulásához, az a kamera, ami a táblát figyeli. Mivel ismereteimet nem bővítettem médiaadat ilyen fajta beolvasása felé, és a .Net framework nem tartalmaz erre közvetlen eszközt, illetve sajátot írni is körülményes lett volna, Veres Péter által készített DSSampler nevezetű komponenst használtam.

Ebben a fejezetben röviden ismertetem, hogy mit tud, és hogyan használható.

DSSampler DirectShow eszközeit használja natív hívásokon keresztül. Ennek következtében azonkívül, hogy minden kamerát képes kezelni, általánosabb média feldolgozásra is képes. Tehát azonkívül, hogy kamerától képes olvasni a bemeneti adatot, képes bármely más szabványos videó-bemenettel rendelkező eszközről is olvasni, mint például a TV-kártya. Erre ráadás még, hogy bármilyen videó-fájlból is képes olvasni, amihez van telepítve codec. Bár ezt az utóbbit egy kicsit máskép lehet megtenni.

Az egész eszköz használata elég egyszerű. Alapvetően csak annyi volt a dolgom, hogy példányosítottam a megfelelő osztályt (DSSampler), feliratkoztam a MediaOnline, illetve a FrameComplete eseményre, ezután meghívtam az InteractiveStartup metódusát, amivel beállítja a bemenetet, végül a StartFrameServer metódusal elindítom a feldolgozást.

Az InteraktiveStartup metódus, mielőtt elindítaná a feldolgozást megkérdezi a felhasználót, hogy fájlból, vagy bemeneti eszközről kívánja-e az adatokat olvasni. Ezután a szükséges fájl, vagy eszközválasztás történik, végül az egyéb beállítások megadása.

A MediaOnline esemény akkor következett be, mikor az InteraktiveStartup meghívása után ténylegesen elindult, ezzel inicializációs lehetőséget adva.

A FrameComplete esemény akkor váltódott ki, amikor sikerült egy képkockát teljesen beolvasni. Ezt a beolvasott képkockát átadta Bitmap formában, és addig nem olvasta a következőt, amíg ez fel nem lett dolgozva.

A DSSamplerben a bemutatott feldolgozási módszeren kívül van egy másik feldolgozási módszer is. Ez az eset annyiban különbözik, hogy nem a StartFrameServert kell meghívni, hanem amikor egy képkocára van szükség, akkor RequestFrame metódussal lehet kérni.

3.2.3 SVM – Szupport Vektor Gépek

Az SVM-ek megfigyelés alapú tanuló metódusok, amiket osztályozásra, és regressziószámításra használnak. Az SVMek a kernel módszerek közé tartoznak. Diplomamunkám szempontjából most csak az osztályozó SVM-ek fontosak.

3.2.3.1 Alapprobléma

Vannak pontjaink egy n dimenziós térben, amit két osztályba vannak sorolva.

A feladat, hogy egy új pontról eldöntsük, hogy melyik osztályba sorolható. A két ponthalmazt ehhez egy p-1 felülettel (hiperfelület) el kell egymástól választani. Ezt lineáris osztályozónak hívják. Természetesen a két ponthalmaz között végtelen sok ilyen hiperfelület létezik, ezért egy olyan felületet szeretnénk találni a két ponthalmaz között, ami maximum elválasztást végez a két ponthalmaz között. Ez azt jelenti, hogy olyan hiperfelületet szeretnénk kapni, ami a legközelebbi adatponttól maximum távolságra van. Más szavakkal a legközelebbi pont távolsága a felület egyik felén, és a felület másik felén levő legközelebbi pont távolsága maximális legyen. Ha létezik egy ilyen felület, akkor ezt maximum-határ hiperfelületnek nevezzük, és a hozzátartozó lineáris osztályozót meg maximum-határ osztályozónak.

3.2.3.2 Formalizáció

Adva van n-darab adat. Ezeket fogjuk használni a tanításra:

$$\{(\underline{x}_1, \underline{c}_1), (\underline{x}_2, \underline{c}_2), \ldots, (\underline{x}_n, \underline{c}_n)\}$$

Ahol c_i értéke lehet -1, és 1, attól függően hogy x_i melyik osztályba tartozik. Az x_i-k p dimenziós vektorok, vagyis a tananyagvektorok.

Feladat, hogy megadjunk egy olyan maximum-határérték hiperfelületet, ami elválasztja azokat a pontokat, amik esetében $c_i = 1$ azoktól, ahol $c_i = -1$. Minden hiperfelület, ami felírható x pontok halmazára kielégíti a következő képletet:

$$w \cdot x - b = 0$$

A w vektor normál vektor, és merőleges a hiperfelületre. A b paraméter adja meg a felület eltolását.

Tehát így a feladatunk, hogy olyan w vektort, és b paramétert válasszunk, ami esetében két egymástól maximálisan távol eső párhuzamos hiperfelületet tudunk felírni, amik még szétválasztják az osztályokat. Ez a két hiperfelület a következőképpen írható fel:

$$w \cdot x - b = 1$$
$$w \cdot x - b = -1$$

Akkor tudunk két hiperfelületet választani, amelyek között nincs tananyagpont, ha a tananyag lineárisan szeparálható. Ebben az esetben a két hiperfelület távolsága maximalizálható.

A megfelelő geometriai számításokat felhasználva kiderül, hogy ez a távolság 2/|w|, ebből következik, hogy |w| értéket kell minimalizálni. Mivel azt szeretnénk elérni, hogy a tananyagpontok ne essenek a két hiperfelület közé, a következő megszorításokat kell tennünk:

$w \cdot x_i - b \geq 1$ minden x_i-re ami az első osztályba tartozik, és

$w \cdot x_i - b \leq -1$ minden x_i-re ami a második osztályba tartozik.

Felhasználva a c_i változót, a következőképpen egyszerűsödik az egyenlőtlenség:

$$c_i \cdot (w \cdot x - b) \geq 1, \text{ ahol } 1 \leq i \leq n.$$

Ezután már fel tudjuk írni a megoldandó problémát:

$$c_i \cdot (w \cdot x - b) \geq 1, \text{ ahol } 1 \leq i \leq n$$
$$|w| \text{ -> minimalizálni.}$$

Tehát úgy kell megválasztani b-t és w-t, hogy w vektor hossza minimális legyen.

3.2.3.3 Egyszerű alak

Az előző részben ismertetet optimalizációs feladat kiszámítása közel se olyan egyszerű, mivel függ |w| abszolút értékétől. Ennek az oka, hogy ez egy nem-konvex probléma, amit jóval nehezebb megoldani. Szerencsére |w| helyére behelyettesíthető $\frac{1}{2}\| w \|^2$ úgy, hogy nem változik a megoldás. Ezt egy kvadratikus programozási feladat. Összegezve:

$$c_i \cdot (w \cdot x - b) \geq 1 \text{ , ahol } 1 \leq i \leq n.$$
$$\frac{1}{2}\| w \|^2 \rightarrow \text{ minimalizálni.}$$

Az ½ szorzó matematikai egyszerűsítés miatt van ott. Így már megoldható a probléma standard kvadratikus programozási technikákkal.

3.2.3.4 Kettős alak

Ha az osztályozási szabályt megszorítatlan kettős alakjában írjuk, észrevehetjük, hogy a hiperfelület meghatározása és így az egész osztályozási feladat mindössze a tartóvektorok függvénye, amelyek a határon vannak. Az SVM kettős alakját így írhatjuk fel:

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j c_i c_j x_i^T x_j \text{ , ahol } \alpha_i \geq 0 \text{ , és } \sum_{i=1}^{n} \alpha_i c_i = 0$$

Ahol az α_i változók jelentik a súlyvektort.

3.2.3.5 Könnyű határ

1995-ben Corinna Cortes és Vladimir Vapnik ajánlott egy módosítását a maximum határ ötletnek, ami megereszti a hibás tananyagot. Másképpen szólva, ha nem létezik olyan hiperfelület, ami egyértelműen szétválasztja a tananyagot. A könnyű határos megoldás úgy adja meg a hiperfelületet, hogy a lehető legjobban szétválassza a tananyagot, miközben még mindig maximalizálja a távolságot a legjobb tananyagok esetén.

Ennél a módszernél, új változókat vezetünk be a hiba mérésére. Legyen ξ_i az x_i rossz osztályozásának mértéke, ekkor:

$$c_i(w \cdot x_i - b) \geq 1 - \xi_i \text{, ahol } 1 \leq i \leq n$$

Így már a célfüggvénynek, ami ξ_i-kel nőt, minél nagyobb határérték, és minél kisebb hiba között kell egyensúlyoznia. Ha a hiba lineáris, akkor így módosult az optimalizációs feladat:

$$c_i(w \cdot x_i - b) \geq 1 - \xi_i \text{, ahol } 1 \leq i \leq n$$
$$\frac{1}{2} \| w \|^2 + C \sum_i \xi_i \rightarrow \text{minimalizálni}$$

A képletben C egy konstans, aminek növelése esetén pontosabb határ meghatározása felé tolja el az eredményt, míg csökkentése esetén a nagyobb maximum határ felé, ezzel több hibát megeresztve.

3.2.3.6 Nem lineáris osztályozás

A gyakorlati problémák többsége nem lineáris, emiatt olyan megoldásokat vezettek be, amik áttranszformálják a problémát magasabb dimenzióba, ahol a megoldás már lineáris.

Jelöljük a transzformáló függvényt ϕ-vel, A magasabb dimenziós tért, ahova transzformálunk H-val, ekkor a leképezés:

$$\phi : R^p \rightarrow H$$

A transzformáló függvény megváltoztatja a vektorszorzást is a következőképpen:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

Ezt a K függvényt kernel függvénynek szokás nevezni.

Általánosan használt függvények:

Polinomiális homogén: $K(x_i, x_j) = (x_i \cdot x_j)^p$

Polinomiális inhomogén: $K(x_i, x_j) = (x_i \cdot x_j + 1)^p$

Radiál alapú: $K(x_i, x_j) = \exp(-\gamma \,||\, x_i \cdot x_j \,||^2)$, ahol $\gamma > 0$

Gauss radiál alapú: $K(x_i, x_j) = \exp(-\frac{\|x_i \cdot x_j\|^2}{2\sigma^2})$

Signoid: $K(x_i, x_j) = \tanh(\kappa \cdot x_i \cdot x_j + c)$, néhány (nem mind) $\kappa > 0$, és $c < 0$ ra.

A programomban a polinomiális inhomogén kerneltípust választottam.

3.2.3.7 Felhasznált implementáció

A programomban a libsvm nevezetű implementációt használtam fel.

Ez az eszköz az SVM használatának általános megközelítését valósítja meg.

Általános esetben egy tananyagvektor igen nagy lehet, és igen ritka a nem nulla elem. Emiatt egy tananyagvektor csomópontokból áll, ami megmondja a csomópont vektorbeli pozícióját és az értékét.

A libsvmnek több nyelvre is létezik implementációja. C#-ra (ezt a nyelvet használtam) 2 fajta implementáció is létezik, amelyek közül én Andrew Poh által készített megoldást használtam fel.

Ebben az implementációban a következő osztályok tartalmazzák az adatokat:

Svm_parameter: az svm működésének szabályozásához tartozó beállításokat tartalmazza. A programomban a következő beállításokat használtam:

```
lParam.svm_type = svm_parameter.C_SVC;
lParam.kernel_type = svm_parameter.POLY;
lParam.degree = 3;
lParam.gamma = 1.0;
lParam.coef0 = 0.0;
lParam.eps = 1e-3;
lParam.C = 100;
lParam.nr_weight = 0;
lParam.cache_size = 256.0;
lParam.nu = 0.5;
lParam.p = 0.1;
```

Svm_node: egy csomópontot reprezentál.

Svm_problem: egy komplett tananyagot reprezentál. Tartalmaz egy változót, mely megmondja a tananyag mennyiségét, egy tömböt a referencia értékeknek, és egy vektorok vektorát, ami tárolja a tananyagvektorokat.

Svm_model: Ez az osztály lesz a tanítás eredménye. Becslésnél kell átadni becsülendő vektorral együtt.

A libsvm használatához az SVM osztály statikus metódusai szükségesek. Ezekből a következő kettőt használtam fel:

Svm_train: Meg kell adni a beállításokat, illetve a tananyagot. Ez alapján elvégzi a tanítást, és végül visszaad egy modellt, ami felhasználható becsléshez.

Svm_predict: A megadott modell alapján megbecsüli, hogy a megadott vektor melyik osztályba sorolható.

3.2.3.8 SVM kezelés kiegészítés a programomban

A programomban egy kicsit speciálisabban kell használni az SVM-et, és egyedi problémák merültek fel, amire nem adott megfelelő megoldást az implementáció. Ennek okán írtam egy osztályt (neve Tananyagkezelo), ami megoldja a speciális problémákat, és felhasználja a fentebb ismertetett osztályokat, metódusokat.

Első probléma, amire nem volt megfelelő az alap megoldás, hogy a tananyagállomány mérete dinamikusan növekszik, nem határozható meg egy statikus méret. Erre a problémára a Tananyagkezelo-ben egy Láncolt listában gyűjtöttem a tananyagokat. A láncolt lista mindig tudja a méretét, és dinamikusan változik, tehát tökéletes számomra. Ha tanításra volt szükség, ez alapján le tudtam generálni az svm_train osztályt, és átadtam tanulásra.

Második probléma, hogy a vektorok mindig fix méretűek voltak, sose voltak ritka mátrixok, és nem olyan formában álltak elő, amilyen formában a becslő metódusnak szüksége volt rá. A bemenő vektorból újra és újra legenerálása a becslőnek szükséges formában nagyon költséges.

Szerencsére fix volt minden esetben a vektor mérete, így előre le tudtam generálni a becslőnek szükséges vektor vázát. Ezt becslés esetén feltöltöttem a kapott bemenő vektorból.

Harmadik probléma a tananyag tárolása volt. A libsvm implementációja tartalmaz tárolásra egy megoldást. Ezzel az volt a problémám, hogy nem a tananyagot, hanem a legenerált modellt tartalmazza. Mivel a folyamatosan bővülhet, ezért az eredeti tananyagot kellett letárolni. A saját mentő/töltő funkcióban a tárolt fájl méretének minimalizálását is feladatul tűztem ki, mivel eléggé nagynak találtam a végeredmény fájlok méretét.

Erre a megoldás a tananyagállomány szerkezetének megfelelő megválasztása volt. A minél kisebb méret érdekében a tananyag bináris adatokat tartalmaz, és ezeket ráadásul le is tömörítettem zip tömörítő algoritmussal.

A bináris tananyagállomány szerkezete (ez van tömörítve):

Megnevezés	Méret	Leírás
Jelzőkarakter (t)	1 byte	Tananyagfájl jelölésére szolgál
Vektorméret (k)	4 byte	A fájl által tartalmazott vektorok mérete
Vektorok száma (n)	4 byte	A fájl által tartalmazott vektorok szám
Vektorok	(4+k*8)*n byte	Tananyag

Az egyes vektorok:

Megnevezés	Méret	Leírás
Vektor osztálya	4 byte	A vektor osztályba sorolása
Vektor	k*8 byte	Maga a vektor

3.3 Lépéselőállítás (Dama.AI)

A lépések kiszámítására a Mesterséges intelligencia 1 tárgyból megismert Negamax algoritmusnak készítettem el az AlfaBéta vágással kiegészített változatát. Ezt a módszert generikus módon implementáltam, amihez az állapot reprezentáció maga a dámajáték.

Elsőként bemutatom, hogy mi is az a Negamax algoritmus, illetve ennek az AlfaBéta vágás nevű kiegészítése, illetve ennek milyen heurisztikát készítettem. A következő fejezetben a megtehető lépések meghatározásáról beszélek. Utána a megtehető lépések meghatározásáról írok, ami a dáma lépéseinek bonyolultsága miatt szintén érdekes feladat.

3.3.1 Lépési stratégia meghatározása

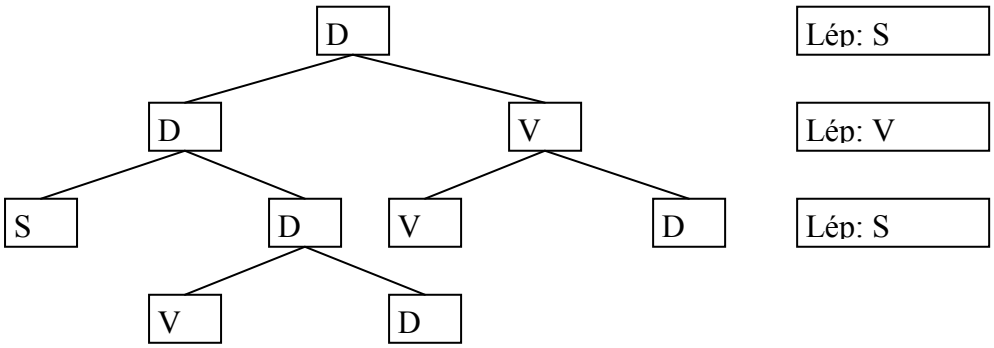
A stratégiai algoritmusok általános elve, hogy a támogatott játékosnak a lehető legjobb lépést ajánlja úgy, hogy közben feltételezi, hogy az ellenfél is tökéletes játékra törekszik. Erre elméleti szempontból az lenne az ideális megoldás, hogy az adott állásból megnézzük az összes lehetséges végkimenetét a játéknak, és ez alapján döntünk. Az összes lehetséges kimenet egy fát állít elő, amelynek csomópontjaiban az egyes állások vannak, az élek a

lépéseket jelölik, illetve a levélelemek jelölik a játék végét. A játék lehetséges kimenetei dáma esetében: sötét nyer, világos nyer, döntetlen.

Az adott játékos szempontjából, mivel legjobb stratégiát akarunk meghatározni, olyan levélelembe kell eljuttatni a játszmát, ahol a támogatott játékos nyer, vagy legalább döntetlennel ér véget a játék.

Ehhez úgymond fordított irányban kell gondolkodni, tehát visszafele kell lejátszani a játékot. Címkézzük meg a fa levélelemeit V-vel ha a világos nyer, S-el ha a sötét nyer, és D-vel, ha döntetlen. Ezután menjünk visszafele a fában, és minden egyes csúcsot címkézzünk meg a következőképpen:

- Ha az adott csúcsban a J játékos jön, és van a csúcsnak olyan gyerekeleme, ahol J címke van, akkor J-vel címkézzük.
- Ha az adott csúcsból csak vesztes állásba, és döntetlen állásba vezető csúcs van J játékos szempontjából, akkor a csúcs D címkét kap.
- Ha az adott csúcsból csak vesztes állásba tud lépni a J játékos, akkor az ellenfél címkéjét kapja a csúcs.



Példa egy játékfára.

Gyakorlatban a teljes játékfát lehetetlenség felállítani, mivel túl nagy, és túl sok időt venne igénybe, így nem a teljes fát szokták meghatározni, hanem csak egy részét, és a hiányzó adatokat egy ún. heurisztikával becslik.

A heurisztika arra szolgál, hogy egy játékállásról megmondja, hogy az egyik játékos szempontjából, milyen jó az állás számára (algoritmusfüggő, hogy melyik). Mivel a heurisztika csak egy becslés, nyerő lépés helyett csak egy elég jó lépést ajánl az algoritmus.

Ezek alapján a negamax algoritmus célja, hogy egy elég jó lépést ajánljon a támogatott játékosnak. Ehhez szüksége van a következőkre:

- A játék reprezentációja, ami tartalmazza az érvényes játékállásokat, ki kezd, a végállásokat, és hogy ki nyer, illetve az érvényes lépéseket.
- A Játékos azon állását, ahol lépni következik.
- Az állások jóságát a soron következő játékos szempontjából becslő heurisztika.
- Mélységi korlát, ami megmondja, hogy milyen mélyen keressünk a fában.

Ezek alapján az algoritmus főbb lépései:

1. A részfa előállítása a megadott korlát mélységig az aktuális állásból.
2. A részfa levélelemeinek megcímkézése a heurisztika segítségével: $jóság(n_b) = h(b)$
3. Szintenként csökkenő sorrendben a részfa nem levél csúcsai jóságainak számítása: ha az n csúcs gyerekei rendre $n_1, \ldots n_k$, akkor
   $jóság(n) \Leftrightarrow \max\{-jóság(n_1), \ldots, -jóság(n_k)\}$

Javaslat: A támogatott játékos egy olyan lépést tegyen meg, amelyik az n_a csúcs jóság értékének -1-szeresével megegyező értékű gyermekébe vezet.

Szóval ezek alapján megvan egy alap algoritmus, amit jól használhatunk lépések ajánlására. Mivel ez a mélység függvényében nagyon gyorsan növekvő fa, szeretnénk valahogy a méretét csökkenteni, tehát kisebb részét kielemezni. Erre szolgál megoldásként az algoritmus egy speciális módosítása, az alfabéta vágás.

Az alfa béta vágás az alap algoritmus gyakorlati implementációját használja ki, és azon módosít úgy, hogy amelyik csúcsot már nem kell kiértékelni a végeredmény érdekében, azt egyszerűen levágja.

Gyakorlati szempontból a kiértékelés úgy néz ki, hogy minden csúcspontban elsőként veszünk egy olyan kis értéket, ami már nem lehet heurisztika értéke, és utána minden egyes aktuális állapotból a megtehető lépésekre, sorra haladunk a következő tevékenységeken:

- Meghatározzuk a lépés heurisztikáját úgy, hogy rekurzívan hívjuk az algoritmust „mélység – 1” mélységi korláttal.

- Ha ennek heurisztikájának -1 szerese nagyobb az eddigi legjobb lépés heurisztikájánál, akkor lesz a legjobb lépés.

Ezt a megoldást az alfabéta vágás úgy módosítja, hogy miután egy új legjobb lépés előállt, megnézi, hogy a szülőcsúcs heurisztikája jelenleg mennyi. Ha az aktuális legjobb lépés -1 szerese kisebb, mint a szülő csomópont hasznossága, akkor befejezi az aktuális csomópont kiértékelését, az eddigi eredményeket adja vissza.

Ez a módszer azért helyes, mert ha belegondolunk, az aktuális csomópont hasznossága a szülőcsomópont szempontjából csökken, és ha a szülő csomópont aktuális hasznosságának szintje alá csökken, akkor már nem fogja kiválasztani, tehát felesleges további energiát pazarolni a kiértékelésére.

A dámaprogramban ennek a megvalósítását készítettem el generikus módon, illetve használtam fel. A dámajáték heurisztikáját eléggé egyszerűre választottam, méghozzá a játék célját vettem alapul, nevezetesen minél több ellenséges báb leütését. Ez szerint egy játékos annál jobb helyzetben van minél nagyobb a következő érték: aktuális játékos bábjainak száma – ellenséges játékos bábjainak száma.

Mivel a gyalog bábu, és a dáma bábu ereje között van különbség, ezt a heurisztikában is kifejeztem úgy, hogy a dámabábu értékét a gyalogbábu értékének 3-szorosát vettem.

A heurisztikában nagyon fontos szerepet töltenek be a végállapotok, mivel ezekben az esetekben egyértelmű, hogy ki nyert. Így a végállapotokat egy nagyon kicsi, illetve nagyon nagy értéknek vettem, amihez még hozzávettem a mélységet. A mélységet hozzávéve úgy

módosul a végállapot, hogy a nyerő játékosnak az legyen a jobb a nyerő lépések közül, amelyik esetében gyorsabban nyer, illetve a vesztő játékosnak az legyen a jobb lépés, amelyik esetében később veszít.

3.3.2 Megtehető lépések meghatározása

A dámában nem is igazán a heurisztika megadása, hanem az egyes állapotokban megtehető lépések meghatározása bonyolult. Általános esetben egy bábuval maximum két átlós irányba lehet lépni. Ez kiegészül még két iránnyal dáma bábu esetén. Ezt mind megváltoztatja ha van legalább egy lehetséges ütés a táblán, mivel akkor a nem ütés lépések már nem tehetők meg, ráadásul az ütés lehetséges, hogy nem egyetlen ütés, hanem ütés sorozat, ami közben akár át is alakulhat a gyalog dámává.

Ezek miatt az egyes állapotokban megtehető lépéseket az új állapot létrejöttekor számoltam ki, és az ai-nak már csak ezeket az előre kiszámolt lépéseket adtam át.

Ezeknek a lépéseknek a kiszámítása a következőképpen történik:
Meghatároztam, hogy ki következik, a tábla bejárása közben csak azoknak a bábutípusait veszem figyelembe.

Minden mezőben, ahol van az adott játékosának bábuja a következőket tettem:
Ha nem volt eddig ütés lépés, akkor megvizsgálom, hogy milyen irányokba léphet, és ezeket felveszem a lépés listába. Ha volt már ütés lépés kihagyom ezt a részt.

Ezután megvizsgálom, hogy milyen ütéseket tud tenni. Ha tud egyet is, akkor kiszedem az összes nem ütést a lépés listából, és beállítom, hogy volt már ütés lépés.

Az ütések vizsgálata egy veremmel történik. Ebbe a verembe belekerül, hogy mi lett ütve, hova lépet az ütő bábu, és hogy átalakult-e gyalogból dámává.

Az ütésvizsgálat menete:
Elsőként belehelyezi a verembe a bábu alaphelyzetét, mint ütést, utána amíg a verem ki nem ürül a következőket csinálja:
Kiveszi a veremből az utolsó ütést.

Megvizsgálja, hogy a célpozícióból milyen irányokba lehetséges további ütést megtenni. Ha egy irányba lehetséges, akkor megvizsgálja, hogy az adott ütéssorozatban le lett-e már ütve (végig megy a vermen). Ha nem akkor hozzáadja a veremhez ennek a bábunak a leütésével megtett lépést. Ha egy ponton már nem lehet tovább ütni, akkor a megtett ütéssorozatot beteszi egy listába, amiben a lehetséges ütéssorozatokat gyűjti. Ez az egész megvalósítás tulajdonképp egy backtrack algoritmus.

A végén ha egyáltalán nem volt megtehető ütés, akkor kiszedi ebből a listából a kezdőlépést reprezentáló ütést (az algoritmus szerkezete miatt ez bekerül).

3.4 Lépések feldolgozása (Dama.TablaProcessing)

A lépések feldolgozása két menetben történik. Elsőként a lépés érzékelése, másodikként az érzékelt lépés feldolgozása történik.

Programozás-technikailag a bemenő képet események sorozataként szolgálja a DSSampler, amit a lépés érzékelő része feldolgoz, és ha lépést érzékel, eseményként küldi tovább az állapot feldolgozó résznek. Ebben az eseményben már csak magát a tábla képét küldi tovább, levágva a felesleges részeket.

A következőkben bemutatom a lépésérzékelést, és az állapot feldolgozást részletesebben.

3.4.1 Lépések érzékelése

A lépések érzékelésénél két dologra kell figyelni. Az egyik, hogy olyan állapotot érzékeljek, ami ténylegesen lépés, a másik, hogy ezt viszonylag kevés processzorterheléssel tegyem. Az utóbbi azért lényeges, mert folyamatosan érkeznek az újabb állapotok, és ezt megfelelő ütemben kell feldolgozni, a minél hamarabbi válaszidő érdekében. Ennek értelmében nem szabad bonyolult transzformációknak alávetni a képet.

A lépés érzékelését alapvetően, mint egy állapotgépet terveztem meg, aminek három állapota lehetséges:

Alapállapot: A kép változatlan, nem történik semmi a tábla fölött.

Lépés történik: Az emberi játékos benyúl a tábla fölé, és egy lépést hajt végre.

Lépés történt: Megtette a lépését az emberi játékos, és elvette a kezét a tábla fölül. Ekkor történik a új rögzített állapot elküldése a feldolgozó résznek.

A mozgás érzékelést úgy végzem, hogy az aktuális képkockából kivonom az előző képkockát, és a létrejövő különbségi kép intenzitásértékeit összegzem. Ha állandó a kép, akkor ez az összeg elméletileg nulla, ami annak felel meg, hogy nem történik semmi. Mivel a kamera eléggé zajos képeket küld, ez nem tud teljesülni. Ennek a problémának megoldására két ötlet kombinációját használtam fel. Az első, hogy ha mindig van egy kis különbség az egyes képkockák között, akkor nulla helyett azt ellenőrzöm, hogy a különbség megfelelően kicsi-e, illetve mozgás esetén elér-e egy megfelelő küszöböt. Sajna ez önmagában nem elég a nagy zajszintnek köszönhetően. A második ötlet a zajszűrésre irányul, méghozzá miután a pixelenkénti különbségeket vettem, az eredményt megvizsgáltam, hogy mennyire tér el a két állapot között. Ha egy bizonyos szint alatt van, akkor úgy vettem, mintha nem is térnének el egymástól, vagyis nulla az eltérés. Ha nagyobb volt az eltérés, akkor csökkentettem a küszöb értékével a különbséget.

Ezek kombinációjával már elég határozottan meg tudtam állapítani, hogy mikor van mozgás a tábla fölött, és mikor nincs. Kísérleteim során kiderült, hogy a pixelenkénti eltérésnek legmegfelelőbb érték a 75, illetve az összeg küszöbének a 200-as érték.

Az előbb ismertetet módszer annak megállapítására, hogy valóban lépés történik-e, még önmagában nem elég, ugyanis mi történik, ha a játékos megállítja a karját a tábla fölött. Ezzel egy állandó állapotot hoz létre az egyes képkockák között, és a módszer lépésnek érzékelné. Ennek elkerülésére kidolgoztam egy másodlagos módszert is, ami a mozgás megszűnése után lép érvénybe. Ekkor nem az egymás utáni képkockák eltérését vizsgálom az előbbi módszer alapján, hanem az utoljára felismert állapotot hasonlítom az aktuális állapottal. Ha ez a különbség is a megadott küszöb alá megy, akkor mondom csak igazából, hogy lépés történt.

Ennél a másodlagos vizsgálatnál alapvető probléma, hogy nem csak az jelent változást az előzőleg felismert állapothoz képest, hogy egy kar van a tábla fölött, hanem az is, hogy egy vagy több bábu pozíciója megváltozott. A sakkban ez a változás a legnagyobb mértékben a sáncolásnál történik, mikor is a bástyával és királlyal egyszerre végzünk lépést. Ekkor összesen négy mező pozíciójában változik meg az előző állapothoz képest a kép, tehát itt beállíthatnánk ennek figyelembevételével egy küszöböt. A dámánál egészen más a helyzet,

mivel ha végrehajtottunk egy sikeres ütést egy bábuval, akkor ugyanazzal a bábuval, ha lehetőség van megint ütni, akkor azt megtesszük, ezzel akár nagyon sok lépést téve, és mezőpozíció állapotát megváltoztatva. Ezt a problémát úgy kerültem ki, hogy kihasználtam, hogy a dámában csak a fekete mezőket használják, a fehérekre sose kerülnek bábuk, tehát az állandó az egyes lépéseknél. Ennek értelmében a másodlagos vizsgálatot leszűkítettem a fehér mezőkre. Ezzel a két módszerrel már egyértelműen meg tudtam állapítani, hogy mikor történik lépés. Miután a lépési folyamatnak vége lépés történt állapotba kerül a figyelő állapotgép, aminek feladata, hogy mentse az állapotot, a következő lépés érzékeléséhez, illetve ezt az állapotot elküldje a feldolgozónak állapot-feldolgozás céljából. Ez az állapot egyetlen képkocka erejéig él, utána visszakerül alapállapotba a figyelő.

3.4.2 Új állapot feldolgozása.

A bemeneti kép feldolgozása volt a legnehezebb feladat. Alapvető probléma volt, hogy a kép nagyon zajos, és különböző környezetben nagyon eltérő volt a bemenet jellemzői. Ezt főleg a megvilágítás erőssége befolyásolta. A másik nagy probléma volt, hogy a játék csak a fekete mezőkön játszódik, és itt a sötét bábok érzékelése nagyon nehézkes volt. A problémának feloldására statikus módszerek helyett adaptív megoldású feldolgozást helyeztem középpontba. Ez annyit jelent, hogy bizonyos jellemzőit mértem az adott mezőnek, ebből előállítva vektorokat, amik leírják a mezőt. Az eredményül kapott vektorokat küldtem tovább feldolgozásra az SVM-nek, hogy megállapítsa az egyes mezők állapotát. Mivel nem mindig sikerül egyértelműen megállapítani, hogy milyen bábuk vannak az egyes mezőkön, ezeket korrigálni kell. Az svm-el mint tanuló algoritmussal meg lehet tenni, hogy ezeket a korrigált értékeket és a hozzátartozó vektort átadva neki tanul a hibából, és legközelebb nem követi el, ráadásul a nem pont ugyanolyan, hanem egy hasonló vektornál is jobban meg tudja állapítani, hogy milyen mezőállapothoz tartozik. Ennek előnye, hogy minden ilyen korrekció után egyre jobban tudja megállapítani, hogy az egyes mezőpozíciók milyen állapotban vannak. Hátránya, hogy ha nem megfelelő a mezők vektorizálása, akkor egyes korrekciók nem fogják tudni javítani a felismerést. Ez a hiba ráadásul még ronthatja is a felismerést, sőt ez odáig is elfajult az egyes kísérleteim során, hogy az új tananyagot már meg se tudta tanulni.

Mivel jó vektorizáló algoritmust nagyon nehéz adni, ezért nyitva hagytam a lehetőséget több megoldás kipróbálására is. Úgy írtam meg ezt a részt, hogy könnyen át lehessen térni más megoldásra is, ha a jelenlegi nem felel meg.

3.4.2.1 Általános állapot feldolgozás menete

Az elemzőhöz már csak a tábla felület képe kerül, a felesleg levágásával, tehát a munkálatok már csak erre vonatkoznak.

- Első lépésként a teljes képen lehet elvégezni esetleges transzformációkat, mint például képsímitást.

- Következő lépés a kép mezőkre bontása, és elhelyezése megfelelő tárban. Ennél a lépésnél figyelembe kell venni a tábla irányát is, és ennek tükrében kell elforgatni, illetve menteni az adatokat.

- Következő a mezőkre bontott képen a fekete mezőkre külön-külön a szükséges kép-transzformációs műveletek végrehajtása, amik akár meg is változtathatják a mezőben található kép méretét.

- Ezután következik a fekete mezők vektorizálása.

- Végezetül a vektorokon SVM becslés végrehajtása, aminek eredményeként megkapjuk, hogy hol milyen bábu van.

Ebben a feldolgozásban az első, a harmadik, illetve a negyedik lépés, ahova tetszőleges módszer behelyettesíthető.

3.4.2.2 Régebbi megoldások

Mielőtt rátérnék a használt felismerési módszerre, bemutatom régebbi, nem teljesen sikeres módszereket, amiken keresztül kialakult a végleges megoldásom.

Legfőbb gondolatom a felismerés módszerével kapcsolatban az volt, hogy a színek alapján kellene végezni, mivel a mező igen sötét volt, és ezen fehér, illetve vöröses színezetű bábu volt található. Így az első módszerem a mező szín-hisztogrammját használta fel a felismerésre. Ez a módszer egész jónak bizonyult állandó fényviszonyok mellett. Viszont mikor változott a fényviszony, akkor eltolódtak az értékek, és így bővíteni kellett a tananyagot. Ez egyre rosszabb felismeréshez vezetett, míg oda nem jutott a felismerő, hogy már nem tudta az újabb tananyagot feldolgozni.

Ennek a módszernek problémája volt még, hogy a vöröses színezetű bábok színe, és a fekete mezők színe alig tért el, így nagyon magas volt a fekete pixelek aránya a képen, ami a hisztogrammon egy eléggé kiegyenlítetlen eloszlást eredményezett az esetek többségében. Ezt kezdetben úgy oldottam meg, hogy a fekete értékű (0) pixelek mennyiségét maximáltam, és ha a maximumon túlnyúlt, akkor levágtam a maximum értékre. Ez könnyítette a feldolgozást, de nem jelentett tökéletes megoldást.

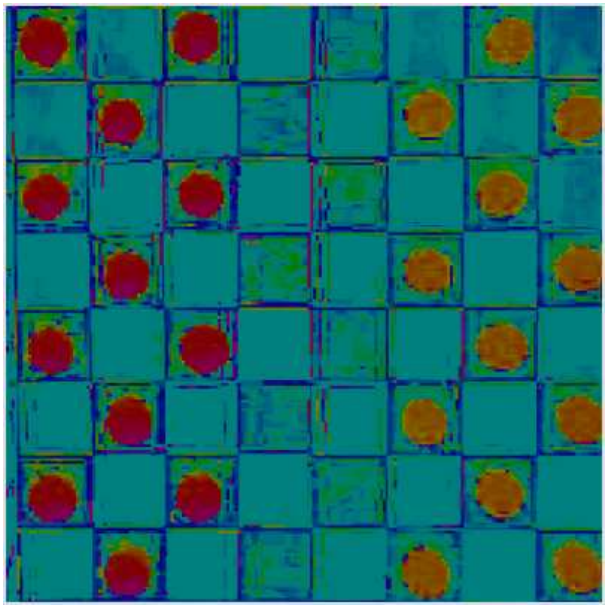
Második próbálkozásom a sötét bábok jobb felismerésére fókuszált. Mivel eléggé egybemosódott a háttért jelentő fekete mezőkkel a sötét bábok, ezért hisztogram kiegyenlítést végeztem rajta. Ennek akkor lett jól látható eredménye, mikor mezőnként külön végeztem a műveletet. Ekkor jól láthatókká váltak a bábok, még a fekete mezőkön is. Ennek problémája az volt, hogy ezzel együtt a színek is teljesen eltorzultak, aminek az eredménye az volt, hogy csak azt lehetett megállapítani, hogy van-e báb a mezőn, vagy nincs.

3.4.2.3 Felismerés HSI transzformációval

A megfelelő bábu-felismeréshez szükséges, hogy a színek ne torzuljanak, illetve jól kivehető legyen, tehát ne függjön a kép az intenzitástól. Ennek megvalósítására jött az-az ötlet, hogy át kell térni RGB színmodellből HSI színmodellbe. Ebben ugyanis szétválik az árnyalat, a telítettség, és az intenzitás. Ebből a hármasból az árnyalatot felhasználva lehetett olyan szín alapú felismerést használni, ahol nincs túlsúlyban a sötét színek, és nincs torzulás se.

A torzulást még azzal igyekeztem csökkenteni, hogy a mezőnkénti transzformáció esetén levágtam a széleit, ezzel elkerülve, hogy fehér mezők belelógjanak a képbe.

Miután HSI transzformációban el lett vetve a telítettség, és az intenzitás, a mellékelt ábrán látható kép jött létre.



HSI transzformáció utáni kép.

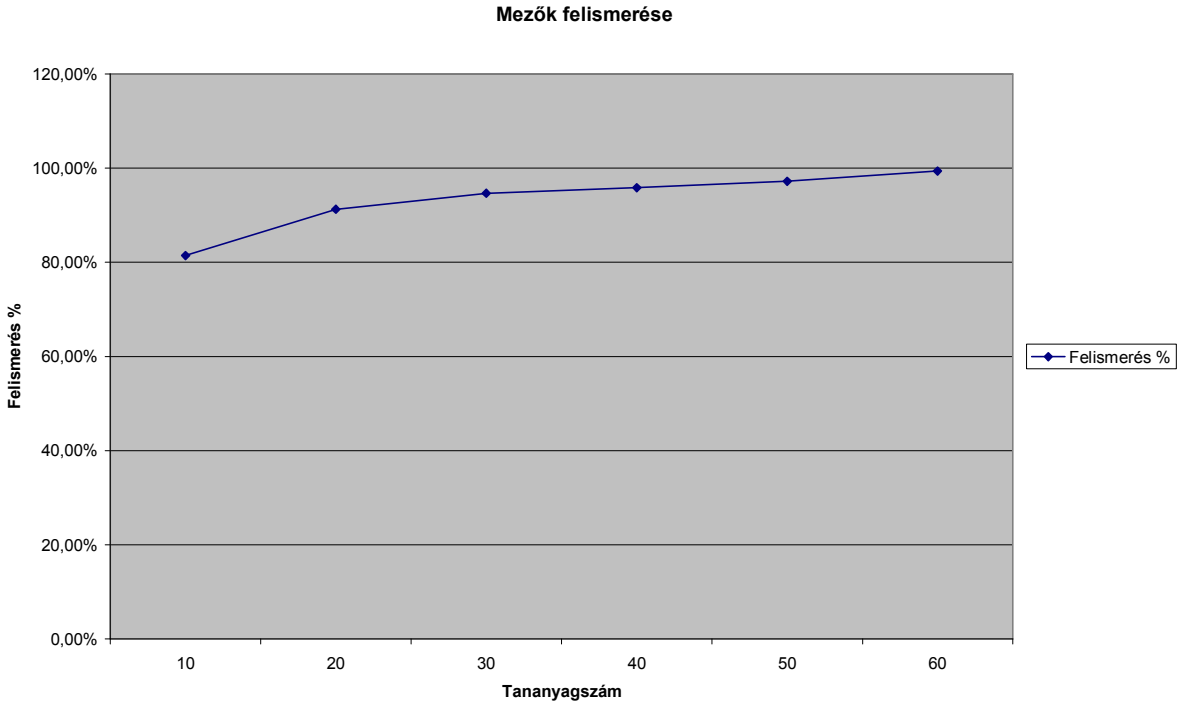
Ezen a képen jól látható, hogy a bábuk jól elválnak a háttértől. A sötét bábok vörös színűek, a világos bábok meg narancssárgás-sárgás színűek lettek, míg a mezők zöldes-kékes színt vettek fel. Persze ez csak akkor volt így, mikor a kamera alapbeállításon volt.

A transzformáció után megfelelő vektorizációra van szükség, amit a következőképp oldottam meg. Elsőként vettem az árnyalatok átlagát, ami általában ki tudja adni, hogy milyen színű bábu van a mezőn. Mivel nem minden esetben olyan egyszerű az átlag alapján, ehhez még hozzávettem a 4 legjellemzőbb színét a mezőnek. Ebből egy 5 elemes vektor állítok elő, ami jól meg tudja állapítani, hogy milyen bábu van a mezőn.

Akkor volt még probléma, amikor a bábuk el voltak csúszva a mező közepétől. Ezt főként a robotkar pontatlanságának volt köszönhető. Ezt a problémát egy kicsivel több tanítás jól feloldotta.

Az algoritmus működéséről tesztet is készítettem. A tesztet mezőnként értelmeztem, és 10 lépésérzékelést végeztem minden esetben véletlenszerűen elhelyezett bábokkal. Mivel egy érzékelés alatt 32 mezőt dolgoz fel, így egy teszt 320 elemet tartalmaz. A tesztet minden 10-edik tananyag hozzáadása után végeztem el újra, és megmértem, hogy hány százalékát

ismerte fel helyesen. Az eredmény a mellékelt grafikonon látható. Ennél a felismerésnél nem csak az a lényeg, hogy hány mezőt ismer fel helyesen, hanem hányszor ismeri fel helyesen mind a 32 mezőt. Ez az érték 50 tananyagszám után kezdett nagymértékben növekedni.



3.5 Kommunikáció az egyes részek között

A kommunikáció vezérléséért alapvetően a dáma vezérlő program felelős, onnét lehet kezdeményezni a kapcsolatok indítását. A kommunikáció tcp kapcsolaton keresztül történik egyszerű szöveges formában. A következő fejezetekben elsőként a vezérlő, és a táblafigyelő közötti kommunikációt tárgyalom ami eseménykezeléshez hasonló megoldással lett elkészítve. A következőben a vezérlő és a robotkar közötti kommunikációt, ami üzenet sorozatokból áll. Utóbbiban nem térek ki magára az egyes parancsokra, mivel azt már a robotkar tárgyalásánál megtettem.

3.5.1 Kommunikáció a vezérlő és a táblafigyelő között

A vezérlő itt leginkább csak hallgató szerepét tölti be. Mindössze csak aktív, illetve passzív állapotba helyezheti a figyelőt, illetve lekérheti az utolsó felismert állást.

A táblafigyelő aktív állapotban figyeli csak a táblán, hogy történik mozgás.

Ha aktív állapotba helyezte a vezérlő a táblafigyelőt, elkezdi várni, hogy a táblafigyelő adatot küldjön, ami egy állás szöveges reprezentációja. Ha helytelen lépés érkezett, jelzi a felsőbb szintnek, és vár újabb állapotokra addig, amíg helyes lépés nem érkezik. Ha helyes lépés érkezett, akkor passzív állapotba helyezi a figyelőt, és elvégzi a további műveleteket (lépésszámítás, robotkarnak parancsok), és mikor a játékoson ismét a sor, újra aktív állapotba helyezi a figyelőt.

Egy állapot reprezentációjára, egy 64 karakteres stringet használtam, ami az egyes mezők állapotát jelenti. A tábla az A1 pozíciótól lett leképezve soronként balról jobbra, illetve az egyes sorok lentről felfelé. Az egyes mezők tartalma a következő karakterek lehetnek:

- e: Az üres mezőnek felel meg.
- b: A fekete gyalog bábunak felel meg.
- g: A fekete dáma bábunak fele meg.
- w: A fehér gyalog bábunak felel meg.
- s: A fehér dámabábunak felel meg.

A táblafigyelő által küldött állás alapján a vezérlőnek el kell döntenie, hogy érvényes lépés történt-e. Ezt úgy tettem meg, hogy elsőként összegyűjtöttem az aktuális állapotra alkalmazható összes lépést, és ezek alapján legeneráltam a lehetséges következő állapotokat. Végül ezeket az állapotokat hasonlítottam a küldött állapotra. Itt probléma volt, hogy a táblafigyelő nem tudta rendesen megállapítani, hogy egy bábu az gyalog, vagy dáma bábu-e. Emiatt a vizsgálatot enyhítettem, és csak azt vizsgáltam, hogy a mező üres, világos, vagy sötét bábot tartalmaz-e.

A kommunikáció implementációjánál a táblafigyelő a szerver, amiben egy aszinkron TcpListener figyeli, hogy van-e bejövő kapcsolatkérés. Ha csatlakozott egy kliens (a vezérlő), akkor addig nem kezd el újra bejövő kapcsolatokra figyelni, míg a kliens be nem zárja a kapcsolatot, vagy megszakad valamilyen hiba miatt. Mivel aszinkron kapcsolatfigyelés van, külön szálra kerül a bejövő adatok fogadása, és a táblafigyelés. Mivel az olvasó szál csak olvasással foglalkozik, meg lehetett szinkron olvasást valósítani. Tehát ez a szál addig blokkolódik, amíg adat nem érkezik. Miután érkezett egy parancs, változókon keresztül

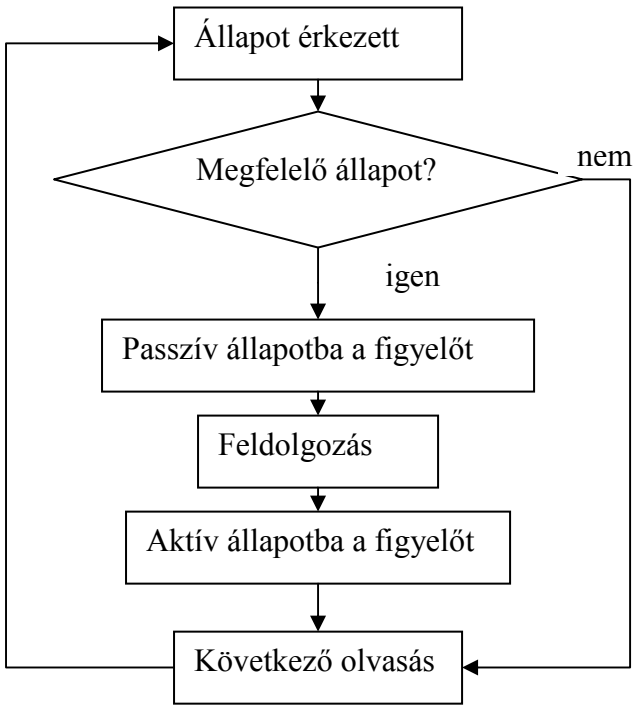
beállítja a szervert a megfelelő állapotba, vagy kiolvassa az utolsó állapotot, és elküldi a parancsnak megfelelően.

Ha a táblafigyelő aktív állapotban van, és csatlakozva van a vezérlő, akkor a létrejövő új állást egyszerűen elküldi a kapcsolaton keresztül.

A vezérlő oldali implementációban aszinkron olvasást valósítottam meg. Miután a kapcsolat felépült, elindítok egy aszinkron olvasást, ami ha új állapot érkezik kivált egy eseményt, amiben feldolgozza a küldött állást, és újra elindítja az aszinkron olvasást.

A kiváltott eseményen belül, ahol történik a feldolgozás, a vezérlő, ha megfelelő állás érkezett, passzív állapotba helyezi a vezérlőt. Ehhez egyszerűen elküldi a megfelelő parancsot. Ezután vezérli a robotkart, végül visszahelyezi aktív állapotba a táblafigyelőt, ezzel elérve, hogy újra küldjön állapotokat.

Ez folyamatdiagramban:



3.5.2 Kommunikáció a vezérlő és a robotkar között

A kommunikáció a robotkarral szintúgy tcp kapcsolaton keresztül folyik, mint a táblafigyelővel. Az utasításokat a vezérlő egyszerű szöveges formában küldi enter karakterrel jelezve a sor végét.

Ez a kommunikáció egyszerűbb volt, mint a táblafigyelővel folytatott kommunikáció, mivel itt csak a vezérlő küld utasításokat, utána vár a visszajelzésre, hogy sikeresen végre lett-e hajtva.

A robotkarral egy utasításban végrehajtható művelet egy bábu áthelyezése egy másik mezőre, bábu kihelyezés, vagy bábu behozás. Ennek következtében, illetve a dámában előforduló lépéssorozat lehetőségeknek köszönhetően, egy lépés nem feltétlenül egy utasításból fog állni, hanem utasítás sorozatból, ráadásul néha az se mindegy, hogy milyen sorrendben hajtja végre a robotkar. Lehet olyan eset (bár nagyon ritka), hogy egy gyalog, ütés sorozaton keresztül, dámává változik, és ugyanoda tér vissza ahonnét indult. Ebben az esetben először kell levenni az eredeti gyalogot, és csak utána helyezhető el a dámabábu.

Az utasítás sorozatok végrehajtására olyan megoldást választottam, hogy elsőként legenerálom a szükséges utasításokat, utána egyenként küldöm el a robotkarnak, és várom, hogy végre lett-e hajtva.

Programozástechnikailag a kezelő osztály megkapja a lépés operátorát (Dama.AI-ban van az Operátor, Dama.Common-ban van az interfész, amin keresztül elküldi). Ez tartalmazza, hogy honnét hova került a bábu, átalakult-e gyalogból dámává, illetve a leütött bábuk listája.

Ebből az adatokból a következőképp állítja össze a parancsépítő (Command Builder) az utasítás sorozatot:

- Ha a bábu dámává lép elő, akkor elsőként kiemeli a gyalogot, utána a célpozícióra beemeli a dámabábut.
- Ha nem lép elő dámává, akkor áthelyezi a bábut a startpozícióról a cél pozícióra.
- Ezután egyenként kiemeli a leütött bábukat a tábláról, ha vannak ilyenek.

Másik probléma volt, amit még itt kellett megoldanom, az a dámabábu behelyezése volt a táblára, ugyanis felmerült a kérdés, hogy honnét helyezzem be. Ennek megoldására egy limitet, és egy megkötést be kellett vezetnem a programba. Méghozzá azt, hogy maximum három dámabábu kerülhet a táblára a sötét játékos részéről, illetve ez a három bábu az f8, g8, illetve a h8 mezők mögött vannak. Így egy Boolean tömbben tudtam tárolni, hogy melyik bábu lett felhasználva, és melyik nem, és ennek megfelelően tudtam utasítani a robotkart, hogy hova nyúljon a táblán kívül.

4 Összefoglaló

A dolgozat elején áttekintettem, hogy milyen irányba fejlődtek a multi-modális technikák, hogyan fejlesztették az ember és gép közötti kapcsolatot, mind vizuális megjelenítésben, mind interakcióban.

Ezután tárgyaltam a dámaprogramomat, ami az ember gép kapcsolatot úgy hivatott megvalósítani, hogy a játékot valós táblán játssza. Ezt úgy érte el, hogy egy kamerával figyelte meg a játékteret, illetve egy robotkarral mozgatta a bábokat. A tárgyalásban leírtam ennek a programnak a struktúráját, illetve tárgyaltam az egyes részleteket, és ezeknek a megvalósítását.

Elsőként a rendszerhez csatlakoztatott, illetve felhasznált eszközöket, komponenseket. Ezek között volt a robotkar, aminek leírtam a kommunikációs protokollját, illetve a működését képességeit nagyvonalakban. Ezek után a kép feldolgozásához használt DSSampler használatát, illetve az SVM-et írtam le. Utóbbinak áttekintettem az elméleti hátterét is, illetve a konkrét implementáció használatát, felhasználását is.

Dolgozatomban ezután következett az általam írt részek három fő részét, a mesterséges intelligenciát, a lépések feldolgozását, illetve az egyes részek közötti kommunikációt.

A mesterséges intelligencia kidolgozásánál bemutattam, hogy hogyan lehet egy ilyen problémát általánosan megoldani, illetve leírtam a konkrét probléma megoldását. A részben külön tárgyaltam a lépések meghatározását, mivel a dáma elég bonyolult szabályrendszert alkalmaz.

A lépések feldolgozásának két része volt egyik az egyes megtett lépések érzékelése, illetve az érzékelt lépések után az aktuális állapot meghatározása. Mivel az állapot meghatározása nagyon nehéz feladat volt, ezért általános megoldási sémára húztam rá a konkrét megoldásom. Itt használtam fel az SVM-et, aminek segítségével pontosabb felismerést tudtam megvalósítani, és ezt hibaarány vizsgálattal is megvizsgáltam.

Végezetül a kommunikációt mutattam be, ami két részből állt, méghozzá a vezérlő és a robotkar közötti, ahol a utasítássorozatok megfelelő kiadását kellett megoldani, illetve a vezérlő és a állásfelismerő közötti, ahol a eseményszerű kommunikációt kellett megoldani.

A projekttel kapcsolatban egy videó is készült, amin részletek láthatók, arról, ahogy a kész programmal játszok. A videón láthatóak a felhasznált komponensek, illetve, hogyan néz ki az egész játéktér. A játékmenet közben láthatóak, hogyan történnek gyakorlatban a lépések, az ütések, illetve, hogy hogyan lett megoldva a gyalogbábok dámává alakulása a robotkar szempontjából.

Bár a projekt nagyrészt megvalósítottam, még nem mondható teljesen kész állapotúnak, további fejlesztésre érdemes.

Az egyik nyilvánvaló továbbfejlesztési lehetőség, amit már a bevezetésben is leírtam az a talking head csatlakoztatása a rendszerhez, ezzel téve teljessé, és maximálisan megoldva a multi-modális célokat.

Implementációs megvalósításban akár általánosításokat, és fejlesztéseket is meg lehet tenni. Az egyik ilyen dolog, hogy az állásfelismerésnél a török 2-ben felhasznált ötletet egyesítve az én projektemben használt ötlettel automatikussá lehetne tenni a tanulást. Alapból azért nem az ott használt ötletet használtam fel, mert el akartam választani az állásfelismerést, és az állás helyességének megállapítását. Ez azért jó, mert így akár visszajelzést is tudna adni a program, hogy hibás lépés történt, és akár tehet is javaslatot, hogy hogyan kéne módosítani a lépést, hogy helyessé váljon.

Az előbbi megoldáson kívül lehet a táblafelismerést úgy átalakítani, hogy az egyes transzformációs módok, amik a vektorokat előállítják, mint plugin csatlakozzanak a rendszerhez. Ez lehetővé teszi, hogy több módszer is kipróbálásra kerülhessen könnyen, a program átalakítása nélkül.

Másik hasonló ötlet, ami megvalósítható lenne, hogy a lépés előállító részt is át lehetne alakítani pluginné, és így nemcsak hogy tetszőleges felismerő, de tetszőleges játék is csatlakoztathatóvá válik a rendszerhez mindenféle átalakítás nélkül.

Persze ez az utóbbi átalakítás több munkát igényelne, mivel a projekt kezdésénél csak dámajáték megvalósítása volt a cél, nem egy általános játékgép. Erre példa az is, hogy a projekthez készült kijelző tábla, csak dámajáték megjelenítésére készült.

Ezek az átalakítások általánossá tudnák tenni a játékot, amin bármilyen táblás játék implementálható lenne sok programozás nélkül, és ezzel egy kiváló multi-modális szórakoztatóiparbeli termékké válna.

5 Függelék

5.1 A dámajáték program ismertetése

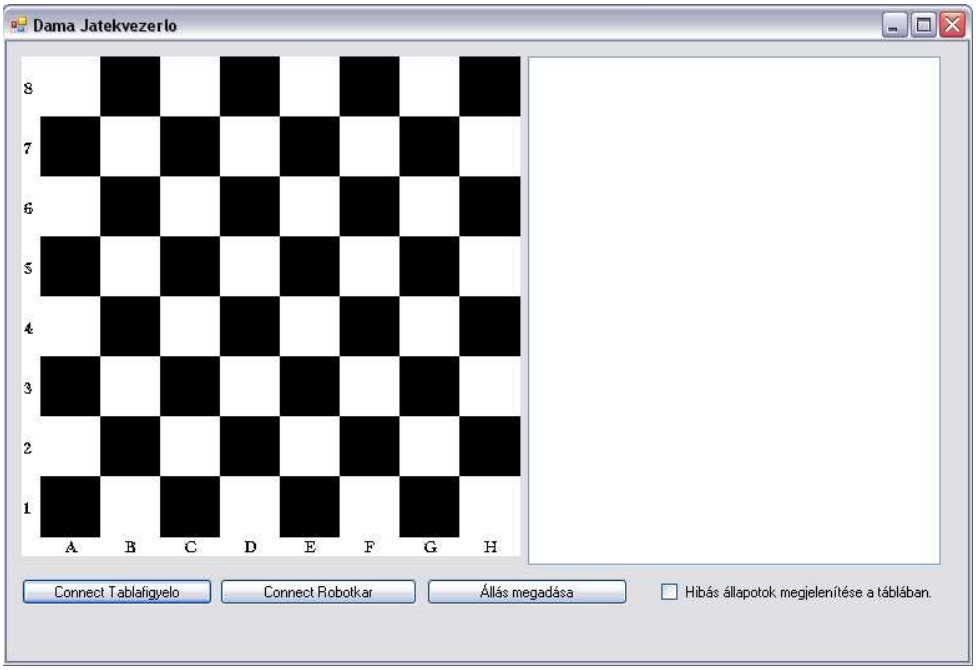
Az dámajáték három alkalmazásból áll. A táblafigyelőből, a dámavezérlőből, illetve Sándor Ákos által írt robotkar-vezérlőből.

A játék elindításához elsőként ellenőrizni kell, hogy a kamera, és robotkar csatlakoztatva van-e, és az utóbbi el van-e indítva. Ezután elsőként a robotkar vezérlő programját kell indítani a dámának megfelelő beállításokkal (robot.bat). Következőnek a vezérlőt kell indítani, és utána azonnal lehet csatlakozni a robotkar programjához, ezzel elérve, hogy a robotkar kifordul. Végezetül a táblafigyelőt kell indítani, felkonfigurálni a megfelelő tananyaggal, illetve kamera beállításokkal. Ezután elkezdi figyelni a táblát, és várni fogja a csatlakozást. Miután a vezérlővel csatlakoztunk, azonnal el kezd várni egy kezdő állapotot. Itt lehetőség van felrakni a bábokat, vagy ha már fel vannak rakva, egy tábla fölé nyúlással jelezni lehet, hogy alapállapotban van. Ha megfelelően fel vannak rakva a bábok, innét megjelenik a vezérlőn az állás, és elindul a játék, lehet megtenni az első lépést.

5.1.1 A dámavezérlő

A vezérlő felületén egy tábla van, ami az aktuális állapotot mutatja, egy lista ami a vezérlőben történt eseményeket mutatja, egy gomb a robotkarhoz való csatlakozáshoz, egy gomb a táblafigyelőhöz való csatlakozáshoz, Állás megadásához egy gomb, illetve egy chekbox, ami

ha ki van pipálva a küldött hibás állapotokat is megjeleníti a mezőn. Ez a mellékelt ábrán látható.



A checkbox, illetve lista a hibák ellenőrzése céljából vannak ott.

Az eseménylista a küldött hibás, illetve helyes állapotokról tájékoztat, ezenkívül a robotkarnak küldött utasításokról.

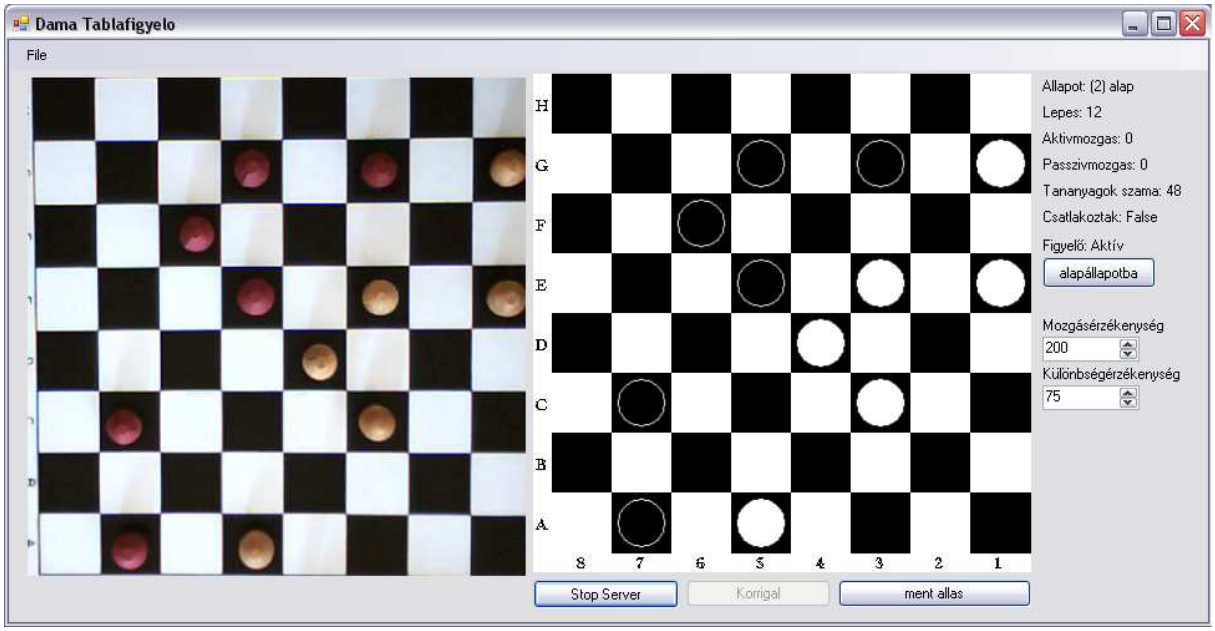
Ha szeretnénk a vezérlőn is látni a figyelő által hibásan felismert állapotokat, akkor a checkbox bepipálásával lehet megjeleníttetni.

A programban lehetőség van nem csak kezdő állapotból indulni. Ehhez az állás megadása gombot meg kell nyomni, és akkor be lehet állítani a táblán a kívánt állást, a mezőkre való többszöri kattintással. Végül a gomb ismételt megnyomásával lehet véglegesíteni az állást.

Ha a vezérlő nem csatlakozik a robotkarhoz, akkor csak a megjelenítőn teszi meg a lépéseket, tehát azt a részt nem kell feltétlenül csatlakoztatni.

5.1.2 A táblafigyelő

Két felülete van, az egyik az indító felület, ahol fel lehet konfigurálni, illetve a figyelő felület, amin figyelemmel kísérhetjük a program működését. A felületek a két mellékelt ábrán láthatóak.



Indító felületén három gomb van: Kamera beállítása, Tananyag megadása, Figyelés indítása. Mivel a Figyelés indításához a kamerát feltétlenül be kell állítani, addig le van tiltva a figyelés indítása, amíg ez meg nem történik.

A tananyag megadása azt jelenti, hogy meg lehet adni egy betanított állásfelismerőt. Ezek a fájlok .tan kiterjesztésűek. Ennek megadása opcionális. Ha nem adjuk meg, akkor új állásfelismerő betanításra lesz lehetőségünk.

A Figyelés indításával a figyelő felületre jutunk, ahol figyelemmel lehet kísérni a táblafigyelést, lehet korrigálni a felismerési hibákat, lehet betanítást végezni, mozgásérzékenységet lehet beállítani, illetve a program állapotát is láthatjuk.

A figyelő felülete a videó-bemenet megjelenítőjéből, az utoljára felismert állapot megjelenítőjéből, állapot beállítókból, illetve állapotkijelzőkből áll.

A tábla alatt található stop frame-szerverrel lehet megállítani a bemenet feldolgozását, és ilyenkor lehet a felismeréssel kapcsolatos dolgokat beállítani. Ekkor a következőket lehet megtenni: Lehet tananyagot kimenteni, illetve betölteni. Lehet korrigálni a hibásan felismert állapotot, amivel új tananyagot ad hozzá az eddig meglévőkhöz.

Ez utóbbiból látszik is, hogy új tananyagot úgy lehet betanítani a felismerőnek, hogy addig korrigáljuk a hibásan felismert állásokat, míg helyesen nem ismeri fel. Megállított állapotban még lehet menteni az utoljára érzékelt állás táblaképét is.

A jobb oldali sávban vannak a mozgásérzékeléssel kapcsolatos kijelzők, illetve beállítók.

Ezek felülről lefelé:

Állapotkijelző, aminek három állapota lehetséges:

(1) lépés történt: ebben az állapotban menti a táblaképet, és felismeri az aktuális állást, illetve elküldi a vezérlőnek, ha csatlakozva van.

(2) alap: Ez az az eset, amikor nem történik semmi a tábla felett.

(3) lépés történik: Éppen a játékos végzi a lépését.

Aktívmozgás: A tábla felett történő folyamatos mozgás intenzitását jelölő érték. Ha ez az érték meghaladja a beállított küszöböt, akkor kerül a (3)-as állapotba a figyelő.

Passzívmozgás: Ez „lépéstörténik” esetben számolódik, amikor éppen befejeződött a mozgás a tábla fölött. Ekkor az utolsó felismert állapothoz tartozó táblaképtől való különbség nagyságát jelöli. Ahhoz hogy megtörténtnek nevezhető legyen a lépés, ennek az értéknek is a küszöb alá kell esnie.

Új sorok száma: a tananyaghoz játék közben történt korrigálások száma.

Csatlakoztak: A vezérlő csatlakozva van-e a figyelőhöz.

Figyelő: Két állapota az aktív, illetve passzív. Tényleges figyelés és lépésfeldolgozás csak aktív állapotban történik.

Alapállapotba gomb: Előfordulhat, hogy valamiért még akkor is lépés történik állapotban van, amikor már befejeződött, ekkor ezzel lehet visszaállítani alapállapotba (megnyomása esetén először lépéstörtént állapotba kerül, és csak utána tér vissza alapállapotba).

Mozgásérzékenység beállító: Az aktívmozgás nagyságának ad küszöböt, amit ha átlép az aktívmozgás, illetve passzívmozgás, akkor állapotot vált a figyelő.

Különbségérzékenység beállító: Zajszűrés erősségét lehet beállítani ezzel. Minél magasabb annál kevésbé érzékeny a zajra. Mivel ha növeljük, a mozgásra is kevésbé érzékeny, tehát nem szabad túl magas értékre állítani.

6 Köszönetnyilvánítás

Elsőként Dr. Fazekas Attilának szeretném megköszönni, hogy ilyen kihívásokkal teli témát adott, illetve vezette, amivel bevont a Debreceni Képfeldolgozó Csoport munkájába. Ezenkívül Lukács Lászlónak, hogy elkészítette a bábokat, Sándor Ákosnak a robotkar, illetve Veres Péternek a DSSampler rendelkezésre bocsátása miatt. Tanácsadás, és háttérmunkálatokban való segítségét meg Sajó Leventének, Kovács Gyurinak, illetve Nagy Andrásnak szeretném megköszönni.

7 Irodalomjegyzék

[1] **Várterész Magda,** „Mesterséges intelligencia 1,” mobiDIÁK Könyvtár, Debrecen 2005, 99-116.

[2] **Zalaegerszegi Dámajáték Sportegyesület,** „Dámajáték története,” http://web.axelero.hu/penkalo/ZDSE_01.htm.
Letöltve: 2008.04.22

[3] **[3] Wikipédia,** „English Draughts,” http://en.wikipedia.org/wiki/Checkers.
Letöltve: 2008.04.22

[4] **Wikipédia,** „Support Vektor Machine,” http://en.wikipedia.org/wiki/Support_vector_machine.
Letöltve: 2008.04.22

[5] **Wikipédia,** „Wii Remote”, http://en.wikipedia.org/wiki/Wii_Remote.
Letöltve: 2008.04.22

[6] **Wikipédia,** „Joystick,” http://en.wikipedia.org/wiki/Joystick.
Letöltve: 2008.04.22

[7] **Wikipédia,** „D-pad,” http://en.wikipedia.org/wiki/D-pad.
Letöltve: 2008.04.22

[8] **Wikipédia,** „Virtual Reality,” http://en.wikipedia.org/wiki/Virtual_reality.
Letöltve: 2008.04.22

[9] **Bellis, M.,** „Computer and Video Game History,” http://inventors.about.com/library/inventors/blcomputer_videogames.htm.
Letöltve: 2008.04.22

[10] **Cynthia Breazeal, Lijin Aryananda, Paul Fitzpatrick Paulina Varchavskaia,** „Kismet,” http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/kismet.html, MIT Letöltve: 2008.04.22

[11] **Agent.ai,** „Da Vinci újratöltve,” 2008, http://www.agent.ai/main.php?folderID=169&articleID=2145&ctag=&iid=. Letöltve: 2008.04.22

[12] **Delira, „**Asimo és Asimov,” 2003 http://nyuz.elte.hu/archiv26/2611/ttt. Letöltve: 2008.04.22

[13] **Sándor Ákos,** Négy szabadságfokú robotkar és vezérlése (Diplomadolgozat), 2006.

[14] **Szabó Péter,** Multi-modális ember-gép kapcsolat digitális képfeldolgozásbeli aspektusa (Diplomadolgozat), 2006