Universitatea "Politehnica" București

Facultatea de Automatică și Calculatoare,
Catedra de Calculatoare

PROIECT DE DIPLOMĂ

Mediu grafic de simulare pentru un robot autonom

Conducător Științific:
Prof.dr.ing. Nicolae Țăpuș
As.drd.ing. Dan Tudose

Autor:
Oana-Alexandra Todiruță

București, 2011

Gândul meu se îndreaptă spre toți cei care prin cercetările lor au contribuit la cunoașterea și punerea în practică a descoperirilor înaltei tehnici a roboticii.
Doresc să îi mulțumesc pentru sprijinul acordat în elaborarea lucrării, domnului prof.dr.ing. Nicolae Țăpuș, coordonator al lucrării, care cu rigurozitate și cu tact m-a călăuzit spre desăvârșirea proiectului de diplomă.
În tot acest timp, mi-a fost alături și domnul as.drd.ing. Dan Tudose, căruia îi mulțumesc, întrucât prin dânsul am reușit să realizăm o reală coeziune a echipei, atât de necesară în munca noastră prezentă și viitoare.

Rezumat

Până nu demult domeniul roboticii era dedicat exclusiv cercetării, însă dezvoltarea continuă a microprocesoarelor și a sistemelor incorporate a dus la o scădere semnificativă a prețurilor, făcând-ul din ce in ce mai atractiv, atât pentru cercetători cât și pentru amatori.
Au luat astfel naștere o serie de competiții de robotică pentru amatori, atât la nivel național cât și la nivel internațional. Concursurile de robotică autonomă au tematici dintre cele mai diverse, de la lupte de sumo și rezolvarea labirinturilor, până la jocuri complexe, ce necesită proiectarea unor sisteme din ce în ce mai inteligente.
Scopul acestei teze este să descrie o aplicație cu ajutorul căreia se pot planifica, într-o manieră facilă și flexibilă, comportamentul și traiectoria unui robot autonom mobil.
Aplicația oferă utlizatorului un mediu grafic detaliat cu care poate interacționa într-un mod simplu și intuitiv. Printre facilitățile oferite de aplicație se numară: proiectarea și planificarea unui traseu oricât de complex prin compuneri de traiectorii rectilinii și curbilinii, specificarea comportamentului robotului în orice punct al traseului, specificarea acțiunilor robotului la apariția diverselor evenimente, simularea traseului proiectat.
S-a pus accentul pe reprezentarea cât mai exactă, la dimensiuni reale, atât a robotului cât și a mediului înconjurător. Mai mult decât atât, vizualizarea se poate face atât în 2 dimensiuni, cât și în 3 dimensiuni, din diferite perspective. Robotul, împreună cu toate elementele mediului înconjurător pot fi proiectate în sisteme CAD populare. Structura mediului de simulare este descrisă într-un format ce poate fi extins cu ușurință.
Pe parcursul proiectării aplicației s-a avut în vedere posibilitatea refolosirii acesteia în cât mai multe proiecte, astfel că sistemul a fost conceput modular și flexibil. Introducerea de noi funcționalități sau schimbarea celor existente se va putea face foarte simplu urmărind structura aplicației.
Mai mult decât atât, inclusiv interfața cu utilizatorul se poate modifica ușor, aceasta fiind descrisă în fișiere externe.
Aplicația ce va fi prezentată poate fi folosită pentru mai multe competiții de robotică și sper că va reprezenta un punct de plecare pentru generațiile următoare de studenți pasionați de domeniul roboticii.

Cuprins

Listă de figuri

Listă de tabele

Capitolul 1

Introducere

Termenul de robotică se referă la ştiinţa care se ocupă de studiul şi utilizarea roboţilor. Cuvântul robot provine din limba cehă (robota) şi înseamnă muncitor, sau rob. Înțelesurile sale au fost îndelung discutate și analizate, astfel că au apărut mai multe definiții, printre care putem enumera:

- enciclopedia Webster definește un robot ca fiind "un dispozitiv automat, care execută funcţii normal atribuite oamenilor, sau o maşină cu asemănare de om";
- institutul de Robotică din America definește un robot ca fiind "o maşină reprogramabilă, multifuncţională creată pentru a muta diverse materiale, bucăţi, unelte sau alte dispozitive specializate prin diverse mişcări programate, pentru realizarea unei varietăţi mari de sarcini";
- enciclopedia Britanică definește un robot ca fiind " orice mașină autonomă care înlocuiește efortul uman, deși nu poate acționa în felul în care poate să acționeze un om"
- Merriam-Webster descrie un robot ca fiind "o mașină care se aseamănă cu un om și care poate să întreprindă anumite acțiuni complexe (ca mersul sau vorbitul) la fel ca o ființă umană"
- o altă definiţie acceptată pe scară mai largă este: un robot este un sistem inteligent care interacţionează cu mediul fizic înconjurător, prin intermediul unor senzori efectori.

Una dintre cele mai importante acțiuni pe care trebuie să le îndeplinească un robot este să fie capabil să interacționeze cu mediul înconjurător. În acest sens arhitectura sa poate fi comparată cu organismul uman. Așa cum un om are nevoie de organe pentru a putea trăi, așa și un robot are nevoie de mai multe piese pentru a se putea mișca. Luând ca exemplu omul se observă o asemănare între organele corpului uman și piesele robotului. De exemplu:

- creier - microprocesor, microcontroller;
- ochi - camere video;
- urechi - senzori de sunet, senzori cu ultrasunete;

- gura - receptoare şi transmiţătoare pentru video/date/sunet;
- schelet - structură mecanică;
- muşchi - actuatori hidraulici/electrici/pneumatici;
- mâncare - sursă de curent, baterii;
- membre - roţi, picioare, şenile.

Isaac Asimov, autor de literatură ştiinţifică, a propus următoarele legi:

1. Un robot nu are voie să provoace vreun rău umanităţii, sau prin inactivitate, să permită vreun rău umanităţii;
2. Un robot nu are voie să rănească o persoană umană, sau să permită rănirea unei persoane umane prin inactivitatea acestuia, cu excepţia cazului când această lege contravine cu vreo lege anterioară;
3. Un robot trebuie să respecte toate ordinele date de o persoană umană, cu excepţia acelor reguli care intră în conflict cu vreo lege anterioară;
4. Un robot trebuie să-şi protejeze propria existenţă atâta timp cât această activitate nu intră în conflict cu legile anterioare. Deşi iniţial acestea nu au fost luate în serios, mai târziu ele au constituit principiile de bază pentru existenţa unui robot.

Termenul de robot descrie un domeniu destul de vast, cauză din care roboții sunt sortați în multe categorii:

- roboți mobili – sunt roboți care au capabilitatea să se miște într-un mediu fără intervenția umană. Roboții mobili sunt de obicei găsiți în armată și în securitate. De asemenea sunt folosiți și pentru entertainment sau să execute anumite sarcini (de exemplu aspiratorul);
- roboți autonomi - sunt roboți care au capabilitatea să îndeplinească anumite taskuri fără intervenția umană;
- roboți umanoizi – sunt roboți care copiază înfățișarea și comportamentul oamenilor;
- roboți industriali – nu sunt de obicei mobili, domeniul lor operațional fiind restrâns;
- roboți casnici – sunt roboții care lucrează autonom în gospodărie, de exemplu: robot aspirator, robotul de tuns gazonul, robot de spălat ferestre;
- roboți exploratori – sunt roboți care operează în locații greu accesibile și periculoase teleghidați sau parțial autonomi;
- roboți de jucărie.

În această teză se va face referire doar la roboții mobili.
Un robot mobil ar putea fi descris ca un robot care trebuie să se deplaseze într-un anumit mediu, fără ajutorul unui operator uman şi să execute anumite sarcini date de acesta. Pentru a se deplasa ei au nevoie de una din următoarele componente: roţi, șenile, picioare, perne de aer sau perne magnetice.

O altă clasificare a roboților mobili este după modul în care sunt ghidați, astfel că există două tipuri de roboți mobili:

- roboți ghidați, care urmăresc anumite traiectorii predefinite;
- roboți neghidați care acționează în funcție de împrejurări.

1.1 Descrierea proiectului

Mișcarea autonomă a roboților este un domeniu care în ultimii ani a cunoscut un progres științific și tehnic remarcabil. Cu toate acestea nici până acum nu s-au găsit niște algoritmi de inteligență artificială care să rezolve toate problemele apărute în deplasarea unui robot autonom. Rezultatele obținute sunt fie prea lente, fie sunt imprevizibile, lucru care nu este de dorit de cele mai multe ori. Din această cauză domeniul de căutare a trebuit să se extindă și în alte direcții.
O alternativă bună la mișcarea autonomă folosind algoritmi de inteligență artificială și învățare automată s-a dovedit a fi generarea și simularea unor trasee dintr-o aplicație capabilă să îndeplinescă aceste tipuri de acțiuni.

1.1.1 Scopul proiectului

O cerință obligatorie în dezvoltarea oricărei aplicații este performanța rezultatelor. În cazul mișcării autonome a roboților cel mai important factor care poate decide performanța aplicației îl constituie timpul necesar parcurgerii unei anumite distanțe de la sursă la destinație. Cu cât timpul este mai mic cu atât aplicația este mai performantă. Astfel simularea mișcării roboților rezolvă acest tip de problemă prin reducerea timpului de deplasare și a timpului acordat hardware-ului.
Scopul acestei aplicații este testarea și simularea diferitelor strategii de mișcare a roboților. Pentru a construi un traseu cât mai eficient, robotul trebuie să fie capabil să urmărească atât linii cât și curbe. Rolul curbelor este să micșoreze timpul unei întoarceri pe loc a robotului.
Simularea traseului este de asemenea importantă pentru observarea posibilelor coliziuni și inconveniențe ce pot apărea în timpul mișcării, dar și pentru a putea vizualiza virtual traseul robotului.
Simulatorul este folositor și pentru etapa de testare a unui robot, de exemplu pentru testarea algoritmilor PID [1]. Pentru testarea algoritmului de PID se stabilește mai întâi un punct de început și un punct ideal de sfârșit, apoi se așteaptă rezultatele obținute de robot și se compară rezultatele. Făcând mai multe măsurători se poate calcula eroarea pătratică medie putând fi astfel îmbunătățit algoritmul PID astfel încât să producă rezultate optime.
Simulatorul trebuie de asemenea să ofere posibilitatea construirii unei baze de date alcătuită din mai multe trasee eficiente. Formatul fișierelor de ieșire trebuie să poată fi cât mai ușor de parsat de către microcontroler [2]. Traseele trebuie să conțină informațiile

[1] http://en.wikipedia.org/wiki/PID_controller
[2] http://en.wikipedia.org/wiki/Microcontroller

necesare robotului pentru a se deplasa de la un punct de start la un punct final. Scopul construirii acestei baze de date este ca robotul să aibe posibilitatea să aleagă la un moment dat între mai multe trasee în funcție de obstacolele întâlnite și de alte caracteristici.

1.1.2 Obiectivele proiectului

Unul dintre cele mai importante obiective ale acestui proiect este planificarea traiectoriei unui robot autonom mobil. Traseul pe care trebuie să îl urmărească robotul trebuie să fie unul eficient, astfel ca drumul de la punctul de start la punctul final să fie unul optim.
Reducerea timpului necesar parcurgerii traseului este o altă cerință pe care această aplicație trebuie să o îndeplinească. Pentru aceasta robotul trebuie să poată executa atât linii cât și curbe și să existe mereu posibilitatea de alegere a tipului de segment pentru o anumită porțiune.
Capacitatea de a simula virtual traiectoria parcursă de robot este de asemenea unul dintre obiectivele principale ale acestei aplicații. Cu ajutorul simulării putem să ne dăm seama și să evităm anumite erori sau necorespondențe, de exemplu putem să evităm obstacole sau anumite porțiuni.
Un alt obiectiv important este proiectarea modulară, în așa fel încât acest proiect să poată fi folosit și pentru alte proiecte asemănătoare. Spre exemplu acest simulator a fost folosit pentru concursul Eurobot, Capitolul 2, a cărui temă anul acesta a fost *Chess'Up*. În anul următor tema se va schimba și de aceea este de dorit să se poată refolosi cât mai mult din ceea ce se face in anii precedenți.

Alte obiective importante ar fi:

- reprezentarea cât mai exactă a mesei de joc și a componentelor;
- stabilirea unui format de ieșire ușor de parsat de programul de pe microcontroler;
- modificarea și adăugarea ușoară de funcționalități noi;
- schimbarea și customizarea ușoară a interfeței grafice;
- simularea mișcării robotului după traseul generat;
- reprezentarea 2D și 3D a scenei de joc;
- oferirea mai multor moduri de vizualizare a scenei.

1.1.3 Motivație

Unul dintre motivele cele mai importante pentru care am ales realizarea unui simulator pentru generarea traseului unui robot a fost lispa unor aplicații simple care să rezolve acest tip de task. Aplicațiile existente sunt fie prea generale, fie prea specifice.
Odată realizat un robot capabil să se deplaseze la o poziție dată, se pune problema generării unui traseu optim. De aceea traseul pe care trebuie să îl urmarească robotul trebuie să fie compus atât din linii cât și din cercuri.

Un alt motiv foarte important pentru care era necesară realizarea unui simulator care să genereze trasee este că de multe ori generarea unui traseu optim pe un microcontroller se poate dovedi anevoioasă.
Cu ajutorul acestui simulator se poate testa și algoritmul de PID. Acest proiect putând fi refolosit și în alte scopuri.
Simulatorul poate asigura strategii câștigătoare pentru concursul Eurobot, Capitolul 2.

1.2 Analiza soluțiilor existente

1.2.1 Webots

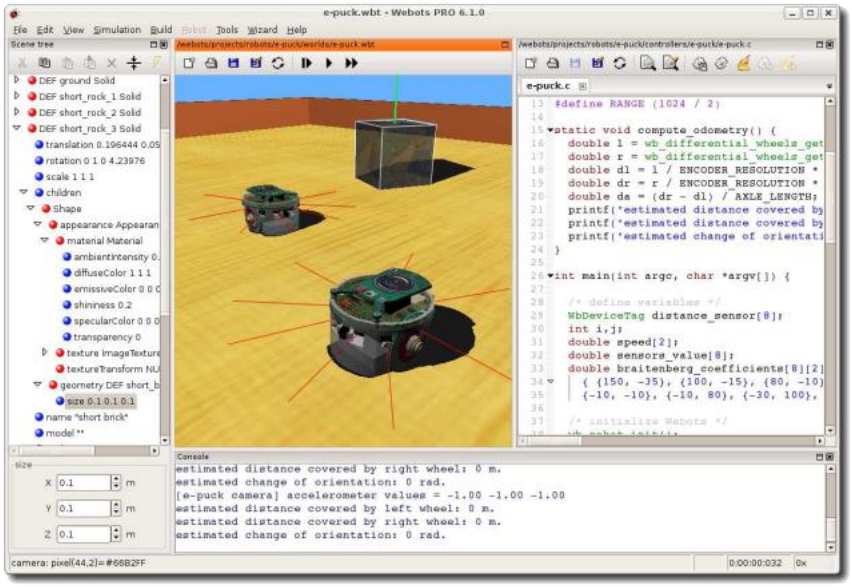

Figura 1.1: Webots

Webots este un mediu de dezvoltare folosit pentru modelarea, programarea și simularea mișcării roboţilor.
Cu ajutorul Webots utilizatorul poate să proiecteze medii complexe, constituite din unul sau mai mulți roboți ce au caracteristici similare.
Proprietățile fiecărui obiect, de exemplu: forma, culoarea, textura, masa, etc. pot fi alese de către utilizator.
În alcătuirea robotului utilizatorul poate să aleagă o gamă variată de echipamente ca:

- senzori de distanță;
- senzori de atingere;
- senzori de lumină;
- camere;
- accelerometre;
- receivere;

- encodere pentru roți;
- motoare;
- led-uri;
- monitoare LCD.

Cu Webots se poate crea un mediu grafic complex de simulare a mișcării unui robot. Se pot folosi tehnici avansate de grafică pe calculator, de exemplu în scenă pot fi introduse: lumini, umbre, texturi, etc.

Avantaje:

- robotul poate fi programat direct din interfața aplicației;
- comportamentul robotului poate fi testat în medii realiste;
- aplicația este compatibilă cu diverse tipuri de roboți existenți pe piață;
- se poate modela și simula orice robot mobil, cu diverse mijloace de locomoție: roți, șenile, picioare și roboți zburători;
- include o librărie completă pentru senzori si actuatori;
- permite programarea roboților în mai multe limbaje de programare: C/C++, Java, Python, URBI, MATLAB;
- suportă mai multe platforme: Windows, Mac OS X și Linux;
- crează filmulețe AVI sau MPEG pentru simulări.

Dezavantaje:

- nu este o aplicație gratuită;
- este mult prea generală;
- se concentrează prea mult pe elementele avansate de grafică pe calculator și de multe ori acestea fac aplicația prea lentă;
- nu se poate construi un traseu format din linii și curbe, și nu este permisă nici editarea acestuia;
- este o aplicație orientată pe bussines, pe prezentări;
- fiind prea generală, aplicația are o interfață complexă făcând greoi procesul de învățare;
- codul sursă nu este public.

1.2.2 Simbad

Simbad este un simulator 3D realizat în Java în scop științific și educațional.
Este dedicat în special cercetătorilor și programatorilor care doresc o platformă simplă pentru studierea inteligenței artificiale și a învățării automate, în contextul roboților și a agenților autonomi.

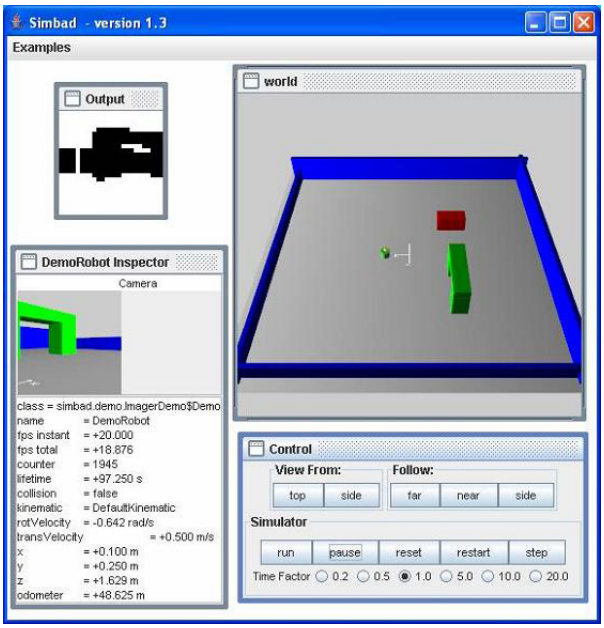

Figura 1.2: Simbad

Comenzile pe care le poate introduce utilizatorul sunt comenzi simple, scopul său fiind să fie cât mai ușor de folosit.
Simbad permite programatorilor să scrie propriul cod de controlare a robotului, să modifice mediul și să folosească senzorii disponibili.

Avantaje:

- este o aplicație gratuită;
- interfața cu utilizatorul este simplu de înțeles;
- posibilitatea de vizualizare 3D a scenei;
- simularea 3d a mișcării robotului;
- codul sursă este public.

Dezavantaje:

- este o aplicație simplă, nu implementează foarte multe funcționalități;
- nu se poate construi un traseu format din linii și curbe, și nu este permisă nici editarea acestuia;
- aplicația se concentrează pe simulare și nu pe generarea unor strategii de mișcare;
- înțelegerea unei aplicații și adăugarea de noi funcționalități este mai dificilă decât implementarea uneia noi, mai ales dacă aceasta este complexă;
- nu există multe forumuri dedicate acestei aplicații;
- lipsa documentației.

	Webots	Simbad
Simulare 3D	da	da
Vizualizare 3D	da	da
Surse disponibile	nu	da
Ușor de folosit	nu foarte ușor	destul de ușor

Tabela 1.1: Comparație aplicații

Alte aplicații asemănătoare:

- simulatorul echipei Eurobot Franța, însă codul și o descrierea completă a acestei aplicații nu este disponibilă;
- simulatorul Player / Gazebo, însă sunt greu de înțeles și necesită documentare riguroasă.

Capitolul 2

Eurobot

2.1 Prezentare generală

Figura 2.1: Logo Eurobot

Creat în 1998, *Eurobot* [1] este un concurs internaţional de robotică deschis echipelor formate din tineri sau studenți.
Concursul nu numai că pune accentul pe competiție, dar încurajează jocul cinstit, solidar, împărtășirea cunoștințelor și nu în ultimul rând încurajează managementul proiectului de către echipă.
Eurobot este mai mult un pretex pentru a lăsa imaginația tehnică a inginerilor să se concentreze pe idei noi, să le discute în echipă, să le impărtășească și chiar să le aplice. Creativitatea este absoult necesară pentru participarea la acest concurs.
După cum spune și numele concursului, *Eurobot* este un concurs care se desfășoară în Europa și este destinat echipelor din întreaga lume. În cazul în care o țara are mai mult de 3 echipe participante, atunci se organizează și un concurs național la care se selectează primele 3 echipe ce vor merge mai departe.
Pentru ca toate echipele să abiă șanse egale, regulile și tema concursului se schimbă în fiecare an. Totuși unele reguli rămân aceleași: un meci durează 90 secunde, robotul

[1] http://www.eurobot.org/

Figura 2.2: Scenă Eurobot

trebuie să se încadreze într-un cub care să nu depășească 30 cm - 40cm, tabla de joc să aibă dimensiunea de 2.1m x 3m.
Concursul se desfășoară într-o atmosferă prietenoasă și deschisă, fiind un prilej prin care ingineri strălucițti din întreaga lume să se întâlnească și să schimbe idei.

2.2 Eurobot 2011

2.2.1 Tema concursului

Tema concursului *Eurobot 2011* este Chess'Up.
Scopul acestui concurs este să aduni cât mai multe piese de șah și să le depozitezi pe

Figura 2.3: Temă Eurobot

culoarea echipei tale.
Un joc durează 90 secunde, și la un moment dat pe tabla de șah există două echipe: o echipă ce reprezintă culoarea roșie și o echipă ce reprezintă culoarea albastră.
Culoarea fiecărei echipe, precum și configurația tablei de șah este aleasă random la începutul fiecarei runde. După ce au fost alese echipele, se așează roboții in zonele de start asociate culorilor atribuite, se aranjează piesele de șah, în așa fel încât inițial ele

să fie poziționate într-o zonă comună ambelor echipe, iar apoi se pornește jocul.
Se declară câștigătoare echipa care acumulează la sfârșitul rundei mai multe puncte ca echipa adversă.

2.2.2 Piesele de șah

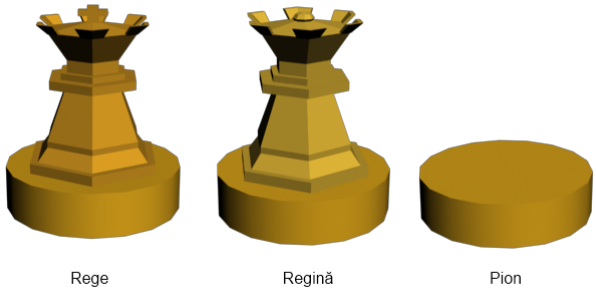

Figura 2.4: Piese joc

În joc există trei tipuri de piese:

1. rege;
2. regină;
3. pion.

Toate aceste piese sunt vopsite în galben și au forma unui cilindru cu diametrul de 190 și înălțimea de 50.
Pentru a putea fi diferențiate, regii și reginele, au lipită la bază o etichetă cu un cod de

Figura 2.5: Cod bare piese joc

bare ("king", respectiv "queen" scris în code 39 Figura 2.5).

2.2.3 Tabla de șah

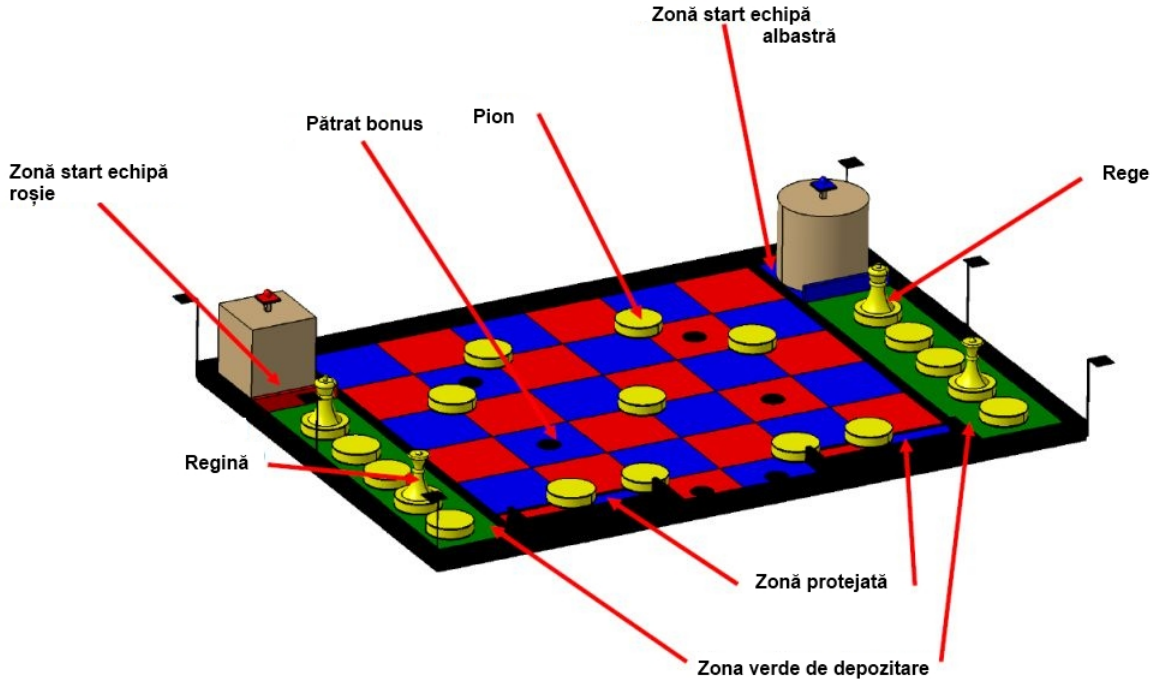

Figura 2.6: Tabla de joc

Tabla de joc este total plană și are dimensiunea de 2.1m x 3m. După cum se poate vedea și în Figura 2.6 în acest an tabla de joc este sub forma unei table de șah.
Tabla este împărțită în mai multe zone:

- zonele de plecare de 400 x 400 - pătrate de culoare roșie, respectiv albastră;
- zona de joc ce conține 6 x 6 pătrate de culoare roșie/albastră ce au dimensiunea de 350mm x 350mm;
- două zone protejate;
- zonele de depozitare, cele pe culoarea verde;
- șase pătrate bonus, care au un cerc negru în centru.

2.2.4 Dimensiuni robot

Constrângeri:

- perimetrul - să nu fie mai mare de 1200mm;
- înălțimea - să nu fie mai mare de 350mm;

- surpotul pentru beacon trebuie situat la înălțimea de 430mm.

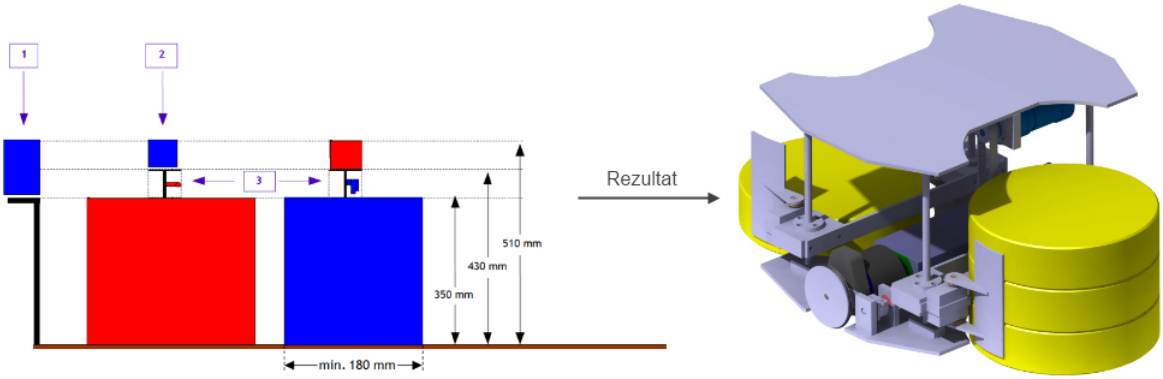

Figura 2.7: Dimensiuni robot

2.2.5 Punctaj

Rege	30
Regină	20
Pion	10

Tabela 2.1: Punctaj

Alte precizări:

- dacă regina sau regele sunt așezați deasupra unui pion li se dublează valoarea;
- dacă regina sau regele sunt așezați deasupra a doi pioni li se triplează valoarea;
- orice altă combinație de turn, 2 pioni fără regină/rege, 3 pioni, ș.a.m.d. este considerată incorectă și se punctează la valoarea unui singur pion;
- dacă există piese într-un pătrat bonus, se acordă 30 puncte pentru acel pătrat, indiferent ce piese se află în el(pion, rege, regină, turn);
- dacă la sfârșitul rundei robotul se află în zona de plecare deasupra unui pion se acordă un bonus de 50 puncte;
- orice piesă din zona protejată nu mai poate fi mutată iar ea valorează 10 puncte.

Capitolul 3

Specificații proiect

3.1 Specificații de proiectare

Suport hardware	laptop, PC
Suport software	Windows XP / Vista / 7, Ubuntu 9.04 sau mai nou
Mediu de dezvoltare	Eclipse
Limbaj de programare	Java[4]
Librării	SwiXML, Processing

Tabela 3.1: Specificații de proiectare

SwiXML [1] este o bibliotecă gratuită folosită pentru parsarea fișierelor XML [2] și transformarea acestora în obiecte javax.swing.
Interfața grafică cu utilizatorul este descrisă în fișiere XML[8, 7] ceea ce asigură separarea modulelor importante ale aplicației.
Pe lângă avantajul modularizării unei aplicații, această bibliotecă este foarte utilă celor care sunt deja obișnuiți cu javax.swing (numele claselor este transformat în nume de tag-uri, numele metodelor este transformat în nume de atribute).
SwiXML se diferențiază de celălalte parsăre prin faptul că se concentrează să mențină cât mai mult sintaxa javax.swing.
Pe lângă toate acestea biblioteca SwiXML este mai rapidă deoarece nu trebuiesc adăugate layere suplimentare deasupra obiectelor Swing.

Processing[3] este o platformă gratuită pentru reprezentarea facilă a scenelor 2D, cu suport și pentru scene 3D.
Processing[1, 2, 6] este o alternativă la aplicațiile grafice scumpe, și este folosit în special de către studenți pentru proiecte școlare și personale, dar nu numai, ci și de mii de companii, artiști, designeri, arhitecți și cercetători.
Processing a apărut în anul 2001 și de atunci s-a format o comunitate de oameni pasionați care au ajutat la dezvoltarea acestei platforme prin: publicarea de cod online,

[1] www.swixml.org
[2] http://en.wikipedia.org/wiki/XML
[3] www.processing.org

răspunsuri și întrebări pe forumul dedicat, construirea de librării care facilitează prelucarea de imagini, electronică, muzică, și aplicațiile de rețea.
Codul rulează pe mai multe platforme ca: Mac, Windows și GNU/Linux.

3.2 Diagramă de utilizare

O diagramă *use case* este una din diagramele folosite în UML [1] pentru a modela aspectele dinamice ale unui program alături de diagrama de activităţi, diagrama de stări, diagrama de secvenţă şi diagrama de colaborare.

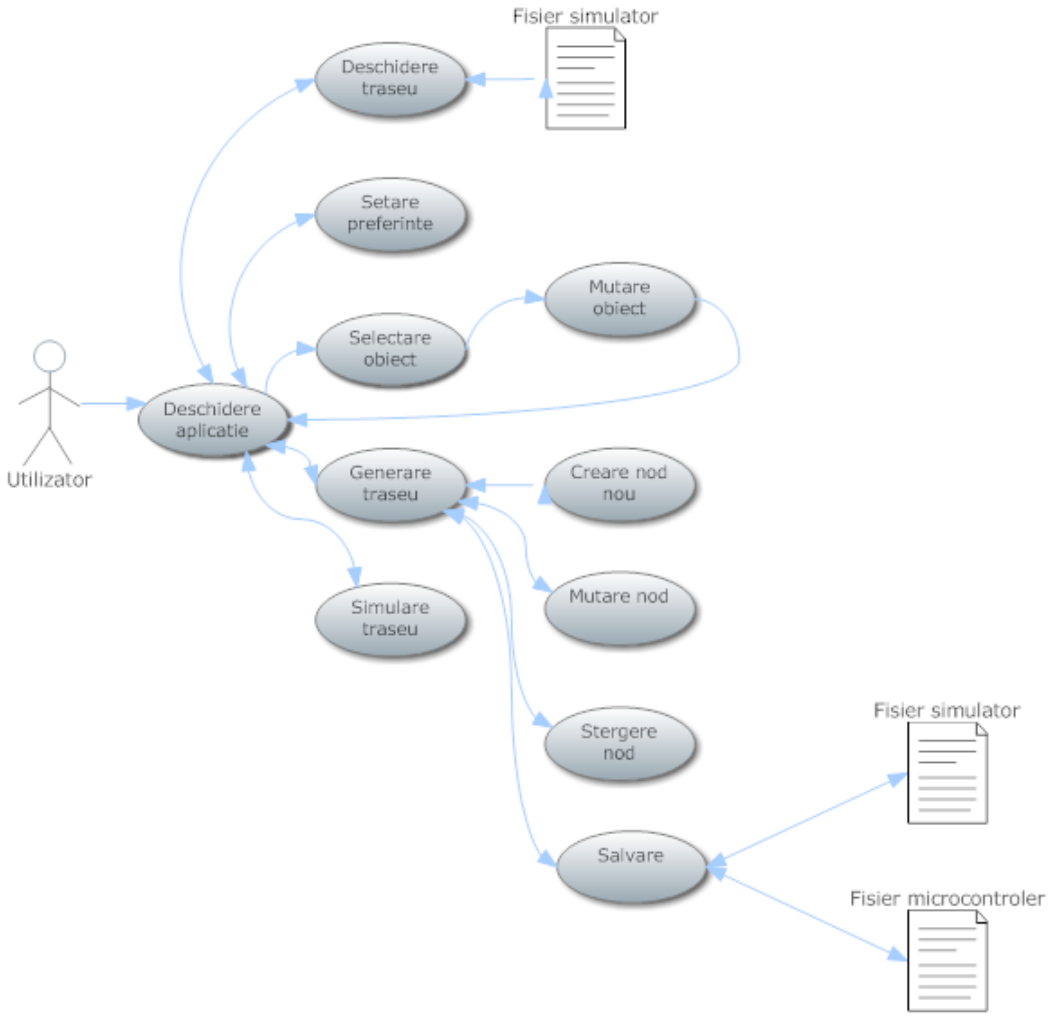

Figura 3.1: Use case

[1] http://en.wikipedia.org/wiki/Unified_Modeling_Language

Elementele componente ale unei diagrame use case sunt:

- use case-uri
- actori (în acest caz actorul este utilizatorul care folosește aplicația)
- relaţiile care se stabilesc între use case-uri, între actori şi între use case-uri şi actori.

Un use case [1] (caz de utilizare) reprezintă cerințe ale utilizatorilor. Este o descriere a unei mulțimi de secvențe de acțiuni pe care un program le execută atunci când interacționează cu entitățile din afara lui (actori) și care conduc la obținerea unui rezultat observabil și de folos actorului.
Un use case descrie ce face un program sau subprogram, dar nu precizează nimic despre cum este realizată o anumitã funcționalitate.
Se va prezenta în continuare o diagramă use case a aplicației și vor fi descrise acțiunile pe care le poate întreprinde actorul.

3.2.1 Deschiderea aplicației

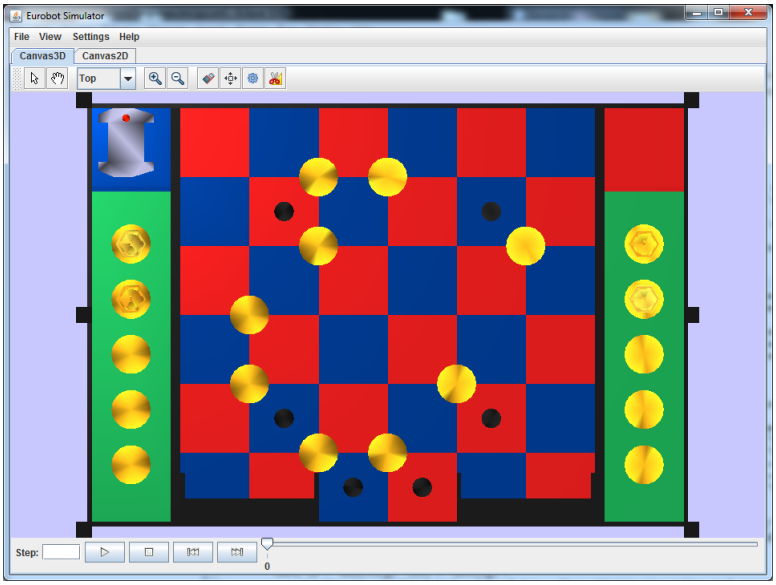

Figura 3.2: Fereastră start

Aplicația poate fi compilată și rulată folosind un fișier Ant, build.xml.
După rularea aplicației, utilizatorul ar trebui să vadă o fereastră ca în Figura 3.2.
Din această fereastră utilizatorul poate alege diferite opțiuni, opțiuni ce vor fi descrise în continuare.

[1] http://en.wikipedia.org/wiki/Use_case

3.2.2 Deschidere traseu

Opțiunea deschidere traseu permite utilizatorului să încarce un traseu salvat anterior.

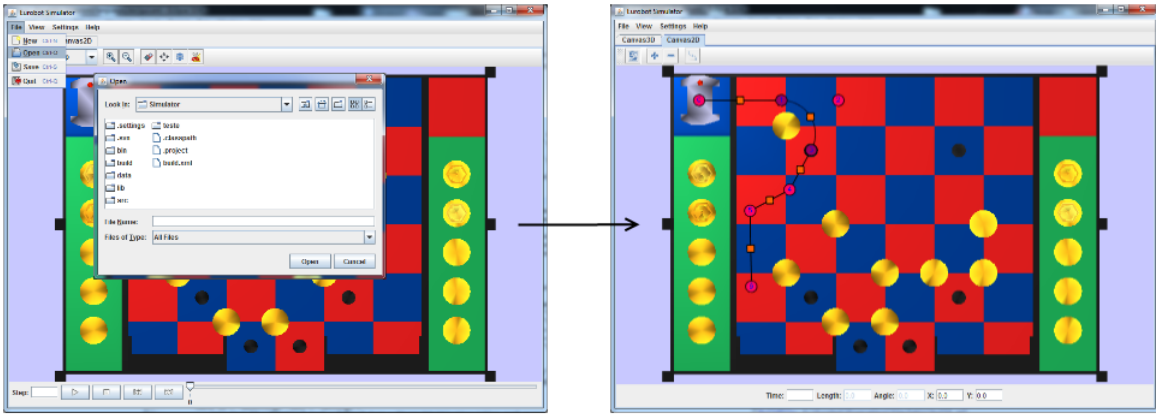

Figura 3.3: Fereastră deschidere fișier

Odată deschis un traseu, acesta poate fi editat, salvat și modificat.
După încărcarea traseului, aplicația se întoarce la ecranul principal.

3.2.3 Setare preferințe

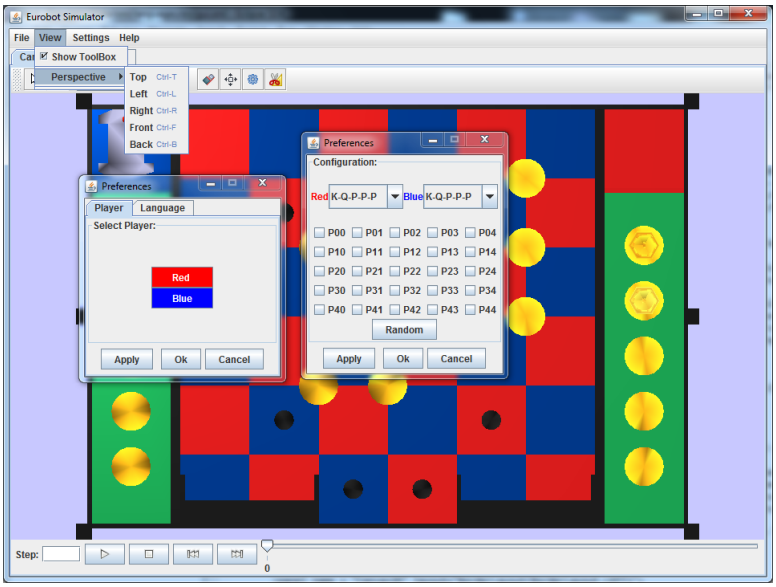

Figura 3.4: Fereastră preferințe

Opțiunea setare preferințe permite utilizatorului să aleagă între diferite opțiuni disponibile:

- setarea culorii din care face parte echipa;

- setarea limbii în care va fi descris meniul, această opțiune va fi aplicată odată cu redeschiderea aplicației;
- setarea configurației tablei de joc;
- setarea tipului de vizualizare: vedere sus, vedere stânga, vedere dreapta, vedere față, vedere spate.

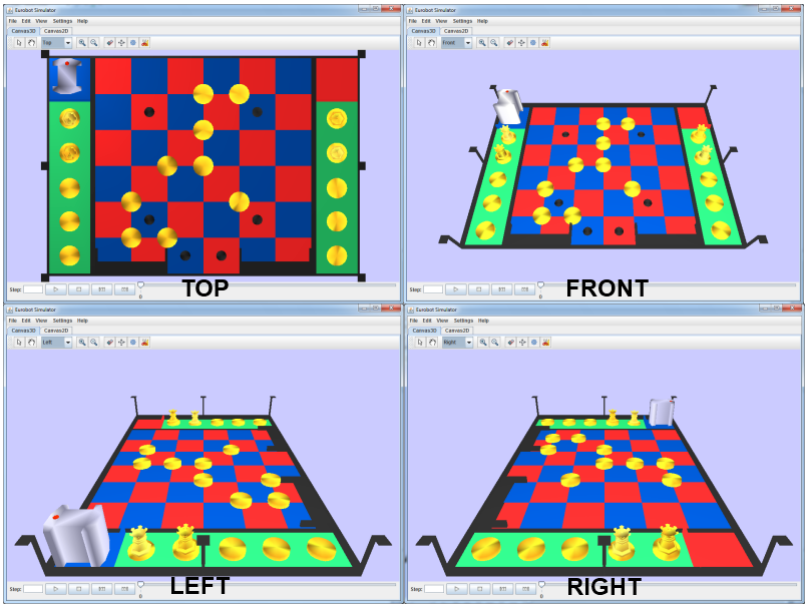

Figura 3.5: Fereastră preferințe - perspective

Există mai multe butoane:

- *Ok* - setează preferințele curente și se întoarce la fereastra principală;
- *Cancel* - anulează preferințele curente și se întoarce la fereastra principală;
- *Apply* - memorează preferințele curente și așteaptă ca unul din butoanele *Ok* sau *Cancel* să fie apăsate.

3.2.4 Selectare obiect

Figura 3.6: Selectare obiect

Opțiunea selectare obiect permite utilizatorului să selecteze un obiect de pe tabla de șah (robot, pion, regină sau rege) și să îl mute la o anumită poziție.
Obiectul poate fi selectat atunci când este activată mânuța, și poate fi deselectat la alegerea săgeții.
Odată selectat un obiect, se poate alege mutarea acestuia. Alegând opțiunea de mutare a unui obiect va apărea o fereastra ca în Figura 3.7.

În această fereastră se poate vedea poziția curentă a obiectului și se poate modifica noua poziție.
Odată mutat un obiect, acesta va rămâne selectat până când altă opțiune este aleasă.

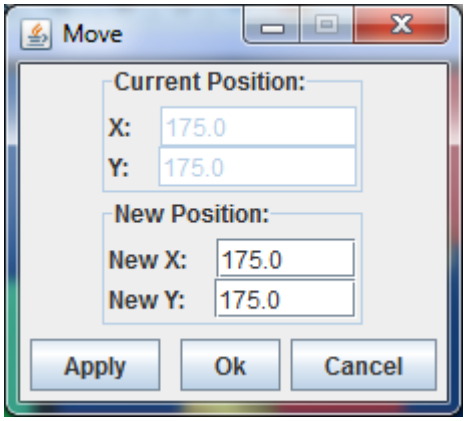

Figura 3.7: Mișcare obiect

3.2.5 Generare traseu

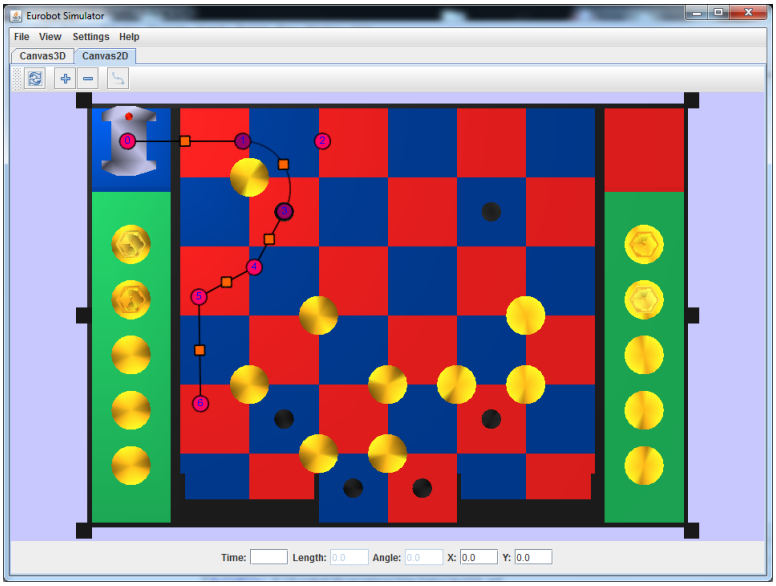

Figura 3.8: Generare traseu

Fereastra *Generare traseu* permite utilizatorului să creeze, să modifice și să salveze un traseu.
Opțiunile disponibile sunt:

- creare nod - este creat un nod nou la apasarea butonului + din bara de sus a meniului. Poziția noului nod va fi generată random și se va încadra în dimensiunile mesei de șah;

- selectare nod - se poate selecta un nod prin apăsarea cu mouse-ul pe cercul încadrator nodului, pentru a se putea deosebi de celălalte noduri la selectare se schimbă culoarea;
- editare nod - se pot edita câmpurile corespunzătoare unui nod: x, y;
- mutare nod - mutarea unui nod se poate face în două moduri: fie în mod grafic prin selectarea și tragerea nodului respectiv pe masa de șah, fie prin setarea coordonatelor (x, y) din bara de jos a ferestrei;
- ștergere nod - un nod poate fi șters prin apăsarea butonului -;
- generare curbă - există posibilitatea de a genera o curbă formată din trei noduri. Mai multe despre curbe și algoritmi de generare a curbelor în Secțiunea 4.2;
- editare segment - se pot edita câmpurile corespunzătoare unui segment: time, angle și length;
- salvare traseu - la salvarea traseului sunt create două fișiere:
  - un fișier în care sunt salvate datele ce vor fi preluate de către microcontroler test1.c;
  - un fișier în care sunt salvate datele necesare pentru reîncărcarea în mod grafic a traseului test1.

Capitolul 4

Implementare

4.1 Arhitectura proiectului

Pe parcursul proiectării aplicației s-a avut în vedere posibilitatea refolosirii acesteia în cât mai multe proiecte, astfel că sistemul a fost conceput modular și flexibil. Introducerea de noi funcționalități sau schimbarea celor existense se va putea face foarte simplu urmărind structura aplicației.
Astfel aplicația a fost proiectată în mai multe module ce vor fi explicate în continuare.

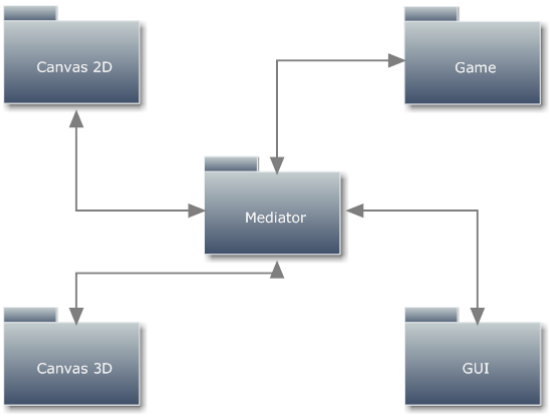

Figura 4.1: Arhitectura proiectului

4.1.1 Componenta Mediator

Mediatorul este componenta centrală a acestei aplicații cu ajutorul căreia se realizează comunicația între celălalte părti ale aplicației.

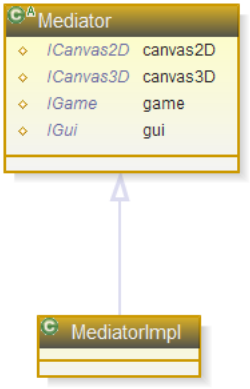

Figura 4.2: Componenta mediator

După cum se poate vedea și în Figura 4.2 pachetul este compus dintr-o clasă abstractă: clasa Mediator și o clasă care o implementează: clasa MediatorImpl.
Rolul acestor clase este să primească cereri de la o anumită componentă și să le pasese către o alta, fără ca una din ele să știe întreaga arhitectură a aplicației. În acest mod se poate realiza o separare clară a modulelor importante ale proiectului.

4.1.2 Componenta Game

Game este componenta care definește obiectele existente in aplicație.
În dezvoltarea acestui proiect s-a avut în vedere reprezentarea scenei atât 2D cât și 3D.

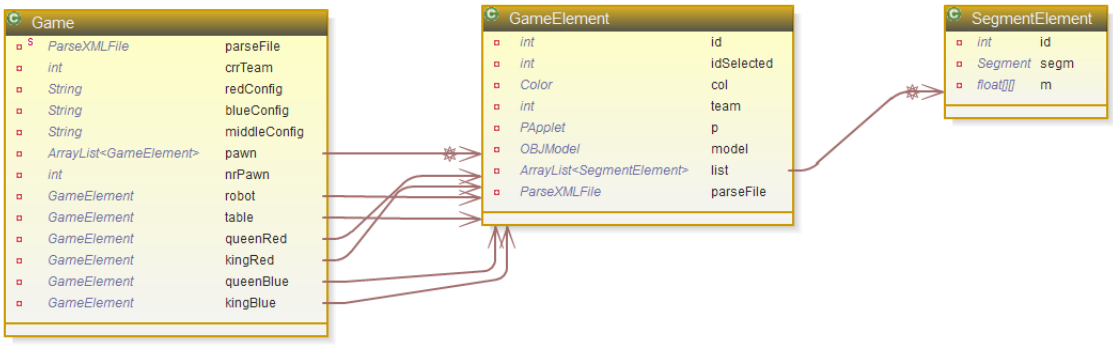

Figura 4.3: Componenta game

Pentru aceasta obiectele scenei sunt construite in 3d Sudio Max [1], iar dimensiunile și

[1] http://usa.autodesk.com/3ds-max

culorile sale sunt cele prevăzute în regulamentul jocului.
Această componentă a fost concepută să fie izolată de celălalte deoarece este de dorit ca această aplicație să poată fi folosită și pentru edițiile următoare ale concursului Eurobot. Astfel în cazul în care tema jocului se va schimba, modificările să se facă doar în acest pachet, celălalte module să fie independente la modificările ce se produc aici.
Pachetul conține trei clase: calsa Game care definește jocul de șah și conține elementele de joc, clasa GameElement care definește o piesă de șah și clasa SegmentElement care conține segmentele din care este compus un element.
Configurația jocului se găsește într-un fișier XML [1] și poate fi modificată foarte ușor.
Obiectele se regăsesc în fișiere .obj [2] generate de 3d Studio Max, iar texturile sunt în fișiere .mtl.
Caracteristicile obiectelor, poziția acestora percum și localizarea lor sunt stocate tot in fișiere XML.
Un exemplu de fișier ce conține un obiect poate fi găsit în king.xml.

Relațiile dintre aceste clase pot fi vizualizate în Figura 4.3

4.1.3 Componenta Canvas3D

Canvas3D este componenta care se ocupă de randarea scenei 3D și este cea care folosește biblioteca processing [3].
În Canvas3D se menține o listă cu obiectele 3D care sunt desenate în metoda draw.

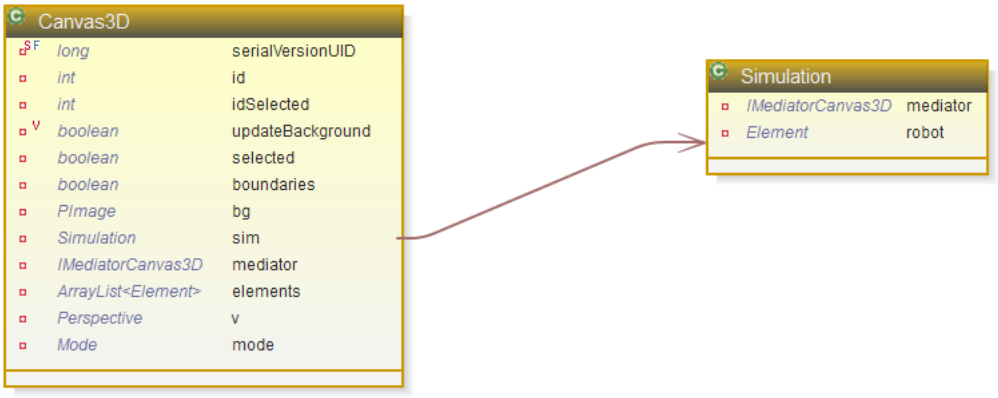

Figura 4.4: Componenta canvas3D

Pentru a beneficia de avantajele graficii 3D există mai multe moduri de vizualizare a aplicației și anume: vizualizare față, vizualizare spate, vizualizare din stânga, vizualizare din dreapta și vizualizare de sus. Toate acestea sunt realizate prin apelarea funcționalităților oferite de biblioteca processing.

[1] http://en.wikipedia.org/wiki/XML
[2] http://en.wikipedia.org/wiki/Wavefront_.obj_file
[3] http://processing.org/

Canvas3D este responsabilă și pentru selectarea și deselectarea obiectelor.
Pachetul conține două clase: clasa Canvas3D al cărei comportament a fost descris mai sus și clasa Simulation.
Rolul clasei *Simulation* este să modifice coordonatele robotului in funcție de opțiunea aleasă de utilizator(înainte / înapoi / flux continuu).

4.1.4 Componenta Canvas2D

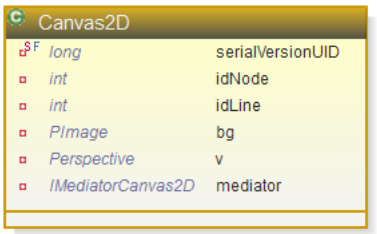

Figura 4.5: Componenta canvas2D

Canvas2D este componenta care se ocupă cu desenarea traseului pe care îl va urmări robotul.
Această componentă a fost concepută individual pentru a se putea face o delimitare între procesul de selecție a obiectelor (robot, pion, regină, rege) și procesul de selecție a nodurilor / a segmentelor răspunzătoare de generarea traseului.
În acestă fereastră nu este permisă decât proiecția de sus a scenei întrucât o altă proiecție nu ar fi ajutat la alcătuirea traseului. Pentru a optimiza această etapă de generare a traseului, randarea obiectelor 3D este oprită.
În acestă fereastră se pot crea, selecta și muta atât linii cât și noduri. Clasa care se ocupă de generarea și desenarea traseului este calsa Path. Această clasă conține o listă de segmente care vor fi descrise în continuare.

Un segment este compus din:

- un câmp *id* care identifică segmentul respectiv
- două *noduri* care definesc acel segment
- un câmp *time* care specifică perioada de timp în care robotul va parcurge segmentul respectiv (este un câmp opțional)
- un câmp *selected* care marchează segmentul ca fiind selectat, desenându-l altfel față de celălalte segmente
- un câmp *enabled* care specifică dacă segmentul respectiv va fi desenat sau nu

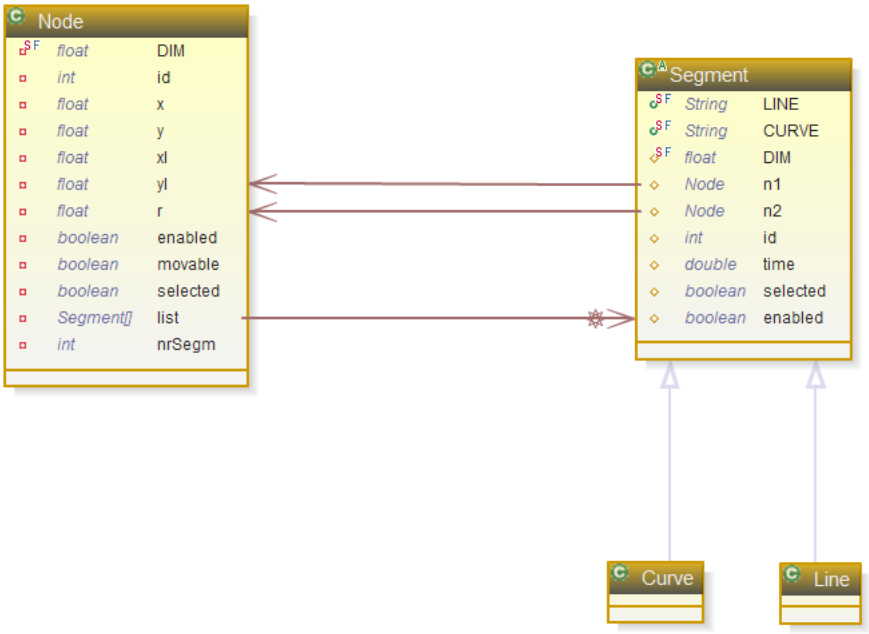

Figura 4.6: Segment

După cum se poate vedea și în Figura 4.6 un segment poate fi atât linie cât si curbă. Pentru a beneficia de avantajele programării orientate obiect clasa Line și clasa Curve implementează clasa Segment în care sunt descrise acțiunile comune celor două tipuri de segmente.
Mai multe despre segmente în Secțiunea 4.2.

Un nod este compus din:

- un câmp *id* care identifică segmentul respectiv
- două coordonate *(x, y)* care definesc poziția acelui nod pe tabla de joc
- două coordonate *(xI, yI)* care reprezintă centrul cercului înscris triunghiului format de segmentele care se intersectează în acel punct, mai multe detalii în Secțiunea 4.2
- un câmp *r* care semnifică raza cercului înscris, mai multe detalii în Secțiunea 4.2
- un câmp *time* care specifică perioada de timp în care va parcurge robotul segmentul respectiv (este un câmp opțional)
- un câmp *selected* care marchează nodul ca fiind selectat, desenându-l altfel față de celălalte noduri
- un câmp *moveable* care specifică dacă nodul respectiv poate fi mutat sau nu
- un câmp *enabled* care specifică dacă nodul respectiv va fi desenat sau nu

- un câmp *list* care reprezintă lista de segmente care conțin nodul respectiv
- un câmp *nrSegm* care reprezintă numărul de segmente din listă

Segmentele cât și nodurile sunt desenate cu ajutorul bibliotecii processing.
Pentru adăugarea altor funcționalități la generarea traseului, aceasta este singura componentă în care se vor face schimbări.

4.1.5 Componenta GUI

GUI este componenta care asigură interacțiunea aplicației cu utilizatorul.
Etapa de dezvoltare a interacțiunii utilizatorului cu aplicația este de cele mai multe ori partea care necesită cel mai mult timp datorită complexității și a numărului mare de acțiuni pe care utilizatorul trebuie să le întreprindă.
De cele mai multe ori apar complicații datorate nerespectării unui anumit pattern pentru schimbarea stărilor, iar greșeala poate fi de multe ori dificil de găsit. De aceea este de dorit să se găsească o metodă care să simplifice lucrurile.
Pentru eliminarea acestor probleme, precum și pentru a putea adăuga și alte funcționalități care altfel nu ar fi fost posibile, am ales ca descrierea interfaței cu utilizatorul sa fie realizată în fișiere XML [1].
Aceste fișiere sunt parsate și transformate in obiecte javax.swing de către biblioteca

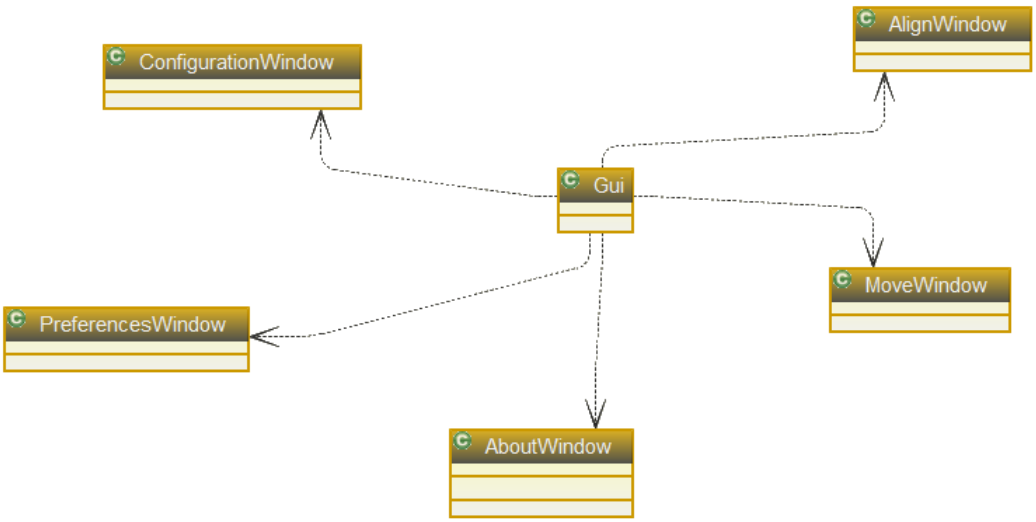

Figura 4.7: Componenta GUI

SwiXML, descrisă în Secțiunea 3.1.

[1] http://en.wikipedia.org/wiki/XML

Printe avantajele acestei metode se pot enumera:

- schimbarea ușoară a stilului aplicației fără cunoștințe avansate de programare
- modularizarea aplicației
- aplicația nu conține mult cod pentru tratarea interacțiunii cu utilizatorul
- se poate schimba limba doar prin traducerea conținutului unui fișier
- este mai ușor de înțeles și simplu de folosit
- marile aplicații folosesc un astfel de procedeu
- se poate extinde ușor prin adăugarea de noi funcționalități

Un exemplu de fișier ce descrie interfața cu utilizatorul poate fi găsit în move.xml.

4.2 Algoritm de generare a traiectoriei

Printre facilitățile oferite de aplicație se numară și proiectarea și planificarea unui traseu oricât de complex prin compuneri de traiectorii rectilinii și curbilinii.
Acestă combinare a traiectorilor rectilinii cu traiectorile curbilinii ajută la reducerea timpului necesar robotului să ajungă de la un punct de start la un punct final.
Vor fi prezentate și explicate în continuare formulele necesare trasării acestor tipuri de traiectorii.

4.2.1 Traiectorii rectilinii

Pentru desenarea unei traiectorii rectilinii am apelat la funcțiile oferite de biblioteca processing [1].
Funcția de desenare a unei linii este:

```
1 // n1 - este primul punct de pe line
2 // n2 - este ultimul punct de pe line
3 line(n1.x, n1.y, n2.x, n2.y);
```

Listing 4.1: Desenare traiectorie rectilinie

În acest caz o dreaptă [2] este definită de două puncte: punctul de start și punctul final.
Ecuația dreptei:

$$y = (m * x + b) \tag{4.1}$$

unde
m - este panta dreptei, adică valoarea funcției tangentă a unghiului dintre dreaptă și sensul pozitiv al abscisei (axa orizontală, Ox).

[1] www.processing.org
[2] http://ro.wikipedia.org/wiki/Dreaptă_(matematică)

b - este ordonata la origine (distanța măsurată pe axa verticală, 0y, dintre punctul de intersecție al dreptei cu axa 0y și originea sistemului de coordonate.
x - este variabila independentă

Alte formule importante:

$$d_x = (n_1x - n_2x) \tag{4.2}$$
$$d_y = (n_1y - n_2y) \tag{4.3}$$
$$len = \left(\sqrt{d_x * d_x + d_y * d_y}\right) \tag{4.4}$$
$$ang = \left(\arctan \frac{d_y}{d_x}\right) \tag{4.5}$$

unde
len - este lungimea dreptei
ang - este unghiul dintre dreaptă și sensul pozitiv al abscisei

Clasa care descrie o linie este prezentată în Line.java.
Un exemplu de traiectorie rectilinie este prezentat în Figura 4.8.

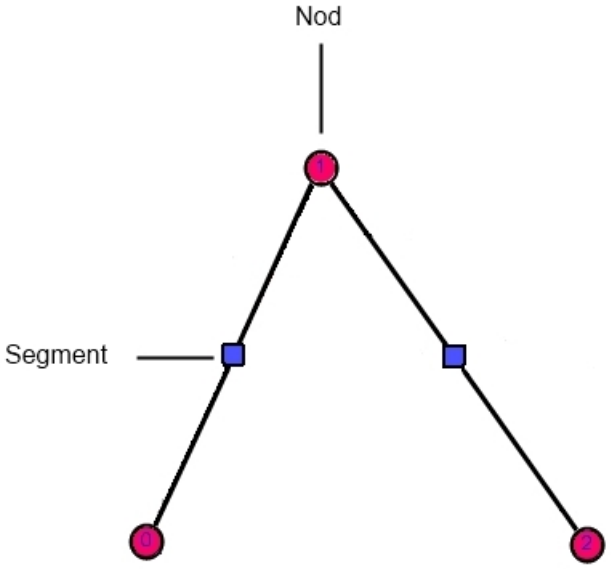

Figura 4.8: Traiectorie rectilinie

4.2.2 Traiectorii curbilinii

Trasarea traiectorilor curbilinii a necesitat mai multă documentație și mai multe investigații. Soluția găsită va fi explicată în detaliu în această secțiune.

Pentru trasarea unei traiectorii rectilinii sunt necesari următorii pași:

1. selectarea a trei puncte care să definească curba

Figura 4.9: Traiectorie rectilinie - Punctele ce descriu curba

unde
A - este punctul de start
B - este punctul ce definește forma curbei
C - este punctul de sfârșit

2. trasarea unui triunghi format din acele puncte

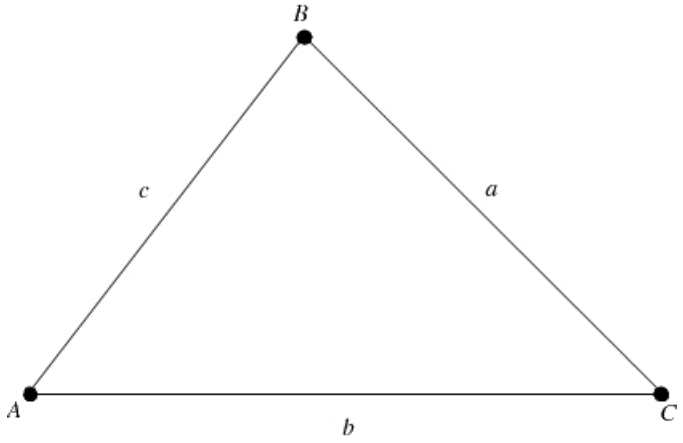

Figura 4.10: Traiectorie rectilinie - Triunghiul format din cele trei puncte

$$a = \left(\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}\right) \quad (4.6)$$

$$b = \left(\sqrt{(x_C - x_B)^2 + (y_C - y_B)^2}\right) \quad (4.7)$$

$$c = \left(\sqrt{(x_C - x_A)^2 + (y_C - y_A)^2}\right) \quad (4.8)$$

unde
a - reprezintă lungimea laturii BC
b - reprezintă lungimea laturii AC
c - reprezintă lungimea laturii AB

3. trasarea cercului înscris triunghilui

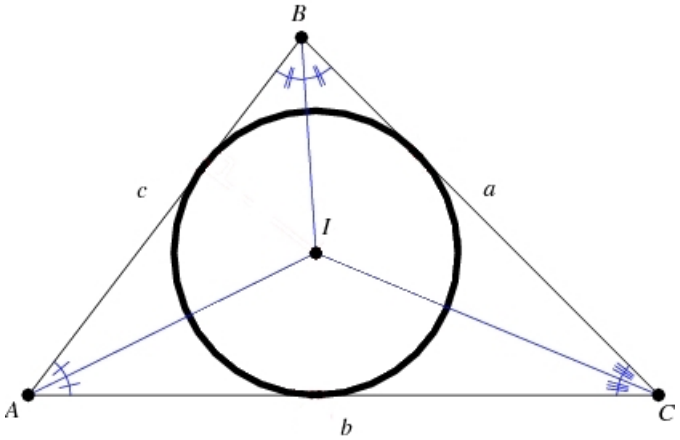

Figura 4.11: Traiectorie rectilinie - Cercul înscris triunghilui

Definiție: Cercul înscris într-un triunghi este cercul care are centrul în interiorul triunghiului şi este tangent laturilor triunghiului.
Centrul cercului înscris în triunghi se află la intersecția bisectoarelor unui triunghi. Bisectoarea unui unghi este semidreapta cu originea în vârful unghiului, care împarte unghiul în două unghiuri congruente.
Raza cercului înscris într-un triunghi este egală cu S/p, unde S este aria triunghiului şi p este semiperimetrul triunghiului.

$$sp = \left(\frac{a+b+c}{2}\right) \quad (4.9)$$

$$S = \left(\sqrt{sp * (sp-a) * (sp-b) * (sp-c)}\right) \quad (4.10)$$

$$r = \left(\frac{S}{sp}\right) \quad (4.11)$$

unde
sp - reprezintă semiperimetrul triunghiului ΔABC

S - reprezintă aria triunghiului ΔABC calculată cu formula lui Heron [1]
r - raza cercului înscris triunghiului ΔABC

Coordonatele centrului cercului înscris triunghiului ΔABC:

$$x_I = \left(\frac{a * x_A + b * x_B + c * x_C}{a + b + c}\right) \tag{4.12}$$

$$y_I = \left(\frac{a * y_A + b * y_B + c * y_C}{a + b + c}\right) \tag{4.13}$$

4. găsirea punctelor tangente la triunghi

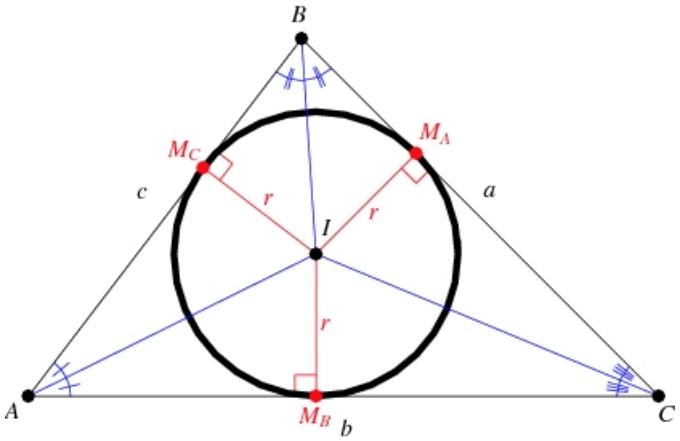

Figura 4.12: Traiectorie rectilinie - Punctele tangente la cerc

Punctele tangente la cerc sunt obținute aplicând formulele:

- pentru punctul M_C:

$$d = \left(\sqrt{(x_B - x_I)^2 + (y_B - y_I)^2}\right) \tag{4.14}$$

$$t = \left(\frac{d}{c}\right) \tag{4.15}$$

$$xM_C = (x_A * t + x_B * (1 - t)) \tag{4.16}$$

$$yM_C = (y_A * t + y_B * (1 - t)) \tag{4.17}$$

- pentru punctul M_B:

$$d = \left(\sqrt{(x_C - x_I)^2 + (y_C - y_I)^2}\right) \tag{4.18}$$

$$t = \left(\frac{d}{a}\right) \tag{4.19}$$

$$xM_B = (x_B * t + x_C * (1 - t)) \tag{4.20}$$

$$yM_B = (y_B * t + y_C * (1 - t)) \tag{4.21}$$

[1] http://en.wikipedia.org/wiki/Heron's_formula

- pentru punctul M_A:

$$d = \left(\sqrt{(x_A - x_I)^2 + (y_A - y_I)^2}\right) \quad (4.22)$$

$$t = \left(\frac{d}{a}\right) \quad (4.23)$$

$$xM_A = (x_C * t + x_A * (1 - t)) \quad (4.24)$$

$$yM_A = (y_C * t + y_A * (1 - t)) \quad (4.25)$$

Pentru desenarea unei traiectorii curbiliniii am apelat tot la funcțiile oferite de biblioteca processing [1].
Funcția de desenare a unei curbe este:

```
1 // xI, yI - sunt coordonatele centrului cercului inscris
     triunghiului
2 // r - este raza cercului inscris triunghiului
3 // alpha - este unghiul pe care il face xMC cu axa ox
4 // beta - este unghiul pe care il face xMA cu axa ox
5 arc(xI, yI, 2 * r, 2 * r, alpha, beta);
```

Listing 4.2: Desenare traiectorie curbilinie

Clasa care descrie o curbă este prezentată în Curve.java.
Un exemplu de traiectorie curbilinie este prezentat în Figura 4.13.

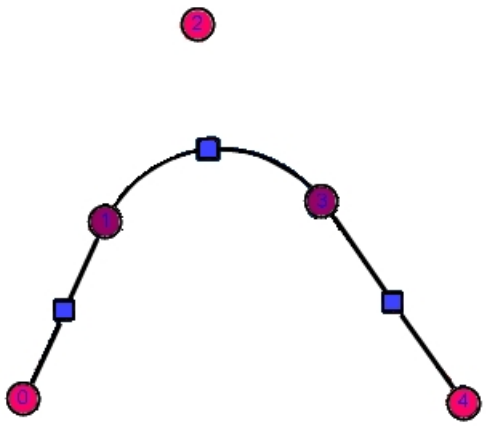

Figura 4.13: Traiectorie curbilinie

[1] www.processing.org

Capitolul 5

Concluzii

Vor fi prezentate în continuare: rezultatele aplicației, problemele apărute în timpul dezvoltării și îmbunătățirile posibile.

5.1 Rezultate

După cum a fost specificat și mai sus aplicația a fost utilizată și testată la concursul național de robotică *Eurobot România*, mai multe despre concurs în Capitolul 2.
Aplicația a reușit să își atingă scopul și să ne ofere un mediu grafic de simulare și testare a mișcării unui robot autonom complex.
După cele cinci runde în care am concurat cu mai multe echipe foarte bine pregătite din România, am reușit să obținem locul 4 și un scor de aproximativ 230 puncte.
Fiind primul an pentru mulți dintre membrii echipei, considerăm rezultatul obținut a fi un rezultat foarte bun, și dorim ca anul următor să fim noi cei care vor concura cu marile echipe din Franța.

5.2 Probleme

Pe tot parcursul dezvoltării aplicației au apărut mai multe probleme precum:

- biblioteca processing este specializată pentru dimensiuni 2D, și nu 3D. Din această cauză a trebuit să implementez eu anumite funcționalități care nu existau sau nu erau bine realizate, funcționalități precum: selectarea obiectelor și a segmentelor obiectelor, încărcarea obiectelor 3D, diferite tipuri de proiecții;
- inițial aplicația nu a fost prevăzută să implementeze toate funcționalitățile descrise mai sus, astfel inițial structura sa fusese mult mai simplă. Din această cauză adăugarea de funcționalități noi era destul de dificilă. Aplicația a fost reproiectată astfel încât structura sa să fie modulară, iar adăugarea, modificarea sau ștergerea de funcționalități să se facă foarte ușor;

- lipsa de documentație privind traiectorile curbilinii și transformarea din traiectorie rectilinie în traiectorie curbilinie. Această problemă a fost rezolvată prin citirea mai multor resurse de matematică;
- testarea aplicației s-a desfășurat destul de târziu din cauza lipsa unei componente esențiale, motorul, fără de care robotul nu se putea deplasa.

5.3 Îmbunătățiri posibile

Fiind o aplicație modulară, adăugarea de funcționalități noi se poate face foarte ușor. Aplicația a fost proiectată astfel încât să poată fi un punct de start pentru edițiile următoare ale concursului *Eurobot*.
Îmbunătățirile posibile care ar putea fi aduse aplicației sunt:

- implementarea unui modul separat de inteligență artificială[9] care să genereze automat traseul, în care ar putea fi folosit un algoritm de căutare A* [1]. Pentru ca acest modul să fie util vor fi necesare și alte aplicații externe. De exemplu este necesară o aplicație de stereo vision care să pună la dispoziție robotului configurația mesei în orice moment al jocului. În acest fel aplicația ar putea să ofere utilizatorului generarea automată sau manuală a celui mai bun traseu;
- o altă posibilă îmbunătățire ar fi găsirea unei metode mai eficiente de generare a traiectorilor curbilinii. O posibilă soluție ar fi transformarea unei curbe Nurbs [5, 3] în primitive: linii, curbe. Această variantă nu a fost încă implementată și testată, din această cauză performanțele nu sunt cunoscute;
- alte funcționalități ce ar putea fi aduse acestei aplicații ar putea fi: afișarea punctajului acumulat, posibilitatea de a muta piesele, încărcarea pieselor din interfața grafică și alte acțiuni ce ar ajuta utilizatorul să folosească mai ușor aplicația[10].

[1] http://en.wikipedia.org/wiki/A*_search_algorithm

Anexa A

Compilare și rulare aplicație

A.1 Rulare aplicație

```
1  <?xml version="1.0"?>
2  <project name="EurobotSimulator" basedir="." default="run">
3     <!-- Properties -->
4     <property name="src.dir" value="src"/>
5     <property name="build.dir" value="build"/>
6
7     <path id="classpath">
8        <fileset dir="lib">
9           <include name="*.jar"/>
10       </fileset>
11       <pathelement location="data"/>
12       <pathelement location="${build.dir}"/>
13    </path>
14
15    <!-- Targets -->
16    <target name="compile">
17       <mkdir dir="${build.dir}"/>
18       <javac srcdir="${src.dir}" destdir="${build.dir}"
            classpathref="classpath"/>
19    </target>
20    <target name="run" depends="compile">
21       <java classname="EurobotSimulator" fork="true"
            classpathref="classpath"/>
22    </target>
23    <target name="clean">
24       <delete dir="${build.dir}"/>
25    </target>
26 </project>
```

Listing A.1: build.xml

Anexa B

Testare aplicație

B.1 Fișier ieșire microcontroler

```
#include "action.h"
const action_t test1[] = {
        // {distanta parcursa roata stanga, distanta parcursa
           roata dreapta}
        {581.0, 581.0},
        {183.9642, -183.9642},
        {320.0, 320.0},
        {280.31482, -280.31482},
        {317.0, 317.0},
        {-194.66461, 194.66461},
        {538.0, 538.0}};
```

Listing B.1: test1.c

B.2 Fișier ieșire simulator

```
//tip_segment timp_parcurgere_segment id_segment nod1_x nod1_y
    nod1_xI nod1_yI nod1_r nod1_id nod1_sePoateMisca
   nod1_esteDisponibil nod2_x nod2_y nod2_xI nod2_yI nod2_r
   nod2_id nod2_sePoateMisca nod2_esteDisponibil
linie 0.0 0 189.55273 166.61896 0.0 0.0 0.0 0 true true
   770.1319 166.61896 770.13184 406.02374 239.40483 1 false
   true
curba 0.0 1 770.1319 166.61896 770.13184 406.02374 239.40483 1
    false true 980.662 520.005 770.13184 406.02374 239.40483 3
    false true
linie 0.0 2 980.662 520.005 770.13184 406.02374 239.40483 3
   false true 828.3363 801.35956 0.0 0.0 0.0 4 true true
```

```
5 linie 0.0 3 828.3363 801.35956 0.0 0.0 0.0 4 true true
     549.3739 950.9481 0.0 0.0 0.0 5 true true
6 linie 0.0 4 549.3739 950.9481 0.0 0.0 0.0 5 true true 557.4597
      1488.6583 0.0 0.0 0.0 6 true true
7 NONE 1171.9856 166.61896 0.0 0.0 0.0 2 true false
```

Listing B.2: test1

Anexa C

Implementare

C.1 Fișier descriere element joc

```
1  <!-- Setarile regelui -->
2  <component>
3     <object fileObject="objects/king.obj"/>
4     <!-- Dimensiunile regelui -->
5     <dimensions height="200" length="50" width="200"/>
6     <!-- Transformarile regelui -->
7     <team>
8        <red>
9           <transformations>
10             <translate tx="0.0f" ty="0.0f" tz="0.0f"/>
11             <rotate rx="0.0" ry="0.0" rz="0.0"/>
12             <scale sx="-1.0" sy="1.0" sz="1.0"/>
13          </transformations>
14
15          <center cx="0.0f" cy="0.0f" cz="0.0f"/>
16       </red>
17       <blue>
18          <transformations>
19             <translate tx="0.0f" ty="0.0f" tz="0.0f"/>
20             <rotate rx="0.0" ry="0.0" rz="0.0"/>
21             <scale sx="1.0" sy="1.0" sz="1.0"/>
22          </transformations>
23
24          <center cx="0.0f" cy="0.0f" cz="0.0f"/>
25       </blue>
26    </team>
27 </component>
```

Listing C.1: king.xml

C.2 Fișier descriere interfață

```
1  <frame id="frame" title="Move" size="300,␣300" visible="false"
       Resizable="false" AlwaysOnTop="true" defaultCloseOperation
      ="JFrame.HIDE_ON_CLOSE">
2
3     <label id="lbError" visible="false">No objects are selected
          or the object is not valid!</label>
4     <panel id="pnlCenter" layout="GridBagLayout" constraints="
         BorderLayout.CENTER">
5        <vbox>
6           <panel id="pnlCenterCrrPosition" Border="TitledBorder
               (Current␣Position:)" layout="BorderLayout"
               constraints="BorderLayout.CENTER">
7              <vbox>
8                 <hbox>
9                    <label labelfor="cbiMoveX" text="X:"/>
10                   <textfield id="cbiMoveX" text="" columns="6"
                         ActionCommand="AC_MOVEX" enabled="false"
                        />
11                </hbox>
12                <hbox>
13                   <label labelfor="cbiMoveY" text="Y:"/>
14                   <textfield id="cbiMoveY" text="" columns="6"
                         ActionCommand="AC_MOVEY" enabled="false"
                        />
15                </hbox>
16                <hbox>
17                   <label labelfor="cbiMoveZ" text="Z:" visible
                        ="false"/>
18                   <textfield id="cbiMoveZ" text="" columns="6"
                         ActionCommand="AC_MOVEZ" enabled="false"
                         visible="false"/>
19                </hbox>
20             </vbox>
21          </panel>
22          <panel id="pnlCenterNewPosition" Border="TitledBorder
               (New␣Position:)" layout="BorderLayout" constraints
               ="BorderLayout.CENTER">
23             <vbox>
24                <hbox>
25                   <label labelfor="cbiMoveNewX" text="New␣X:"/
                        >
26                   <textfield id="cbiMoveNewX" text="0.0"
                        columns="6" ActionCommand="AC_NEWX"/>
27                </hbox>
28                <hbox>
```

```
29                   <label labelfor="cbiMoveNewY" text="New␣Y:"/
                        >
30                   <textfield id="cbiMoveNewY" text="0.0"
                        columns="6" ActionCommand="AC_NEWY"/>
31                </hbox>
32                <hbox>
33                   <label labelfor="cbiMoveNewZ" text="New␣Z:"
                        visible="false"/>
34                   <textfield id="cbiMoveNewZ" text="0.0"
                        columns="6" ActionCommand="AC_NEWZ"
                        visible="false"/>
35                </hbox>
36             </vbox>
37          </panel>
38       </vbox>
39    </panel>
40
41    <panel id="pnlSouth" layout="FlowLayout" constraints="
         BorderLayout.PAGE_END">
42       <button id="b1" text="Apply" ActionCommand="AC_APPLY"/>
43       <separator/>
44       <button id="b2" text="Ok" ActionCommand="AC_OK"/>
45       <button id="b3" text="Cancel" ActionCommand="AC_CANCEL"/
            >
46    </panel>
47 </frame>
```

Listing C.2: move.xml

C.3 Segment

```
1  package simulator.segment;
2
3  import java.awt.Point;
4
5  import processing.core.PApplet;
6
7  public abstract class Segment
8  {
9          public static final String LINE = "linie";
10         public static final String CURVE = "curba";
11
12         protected static final float DIM = 50.0f;
13
14         protected Node n1;
15         protected Node n2;
```

```
16
17          protected int id;
18          protected double time;
19          protected boolean selected, enabled;
20
21          /**
22           * Constructor
23           *
24           * @param n1
25           *             primul punct de pe segment
26           * @param n2
27           *             al 2-lea punct de pe segment
28           * @param id
29           *             id-ul segmentului
30           */
31          public Segment(Node n1, Node n2, int id)
32          {
33                  setNode1(n1);
34                  setNode2(n2);
35
36                  this.id = id;
37                  this.enabled = true;
38                  this.selected = false;
39          }
40
41          /**
42           * @return id-ul
43           */
44          public int getId()
45          {
46                  return id;
47          }
48
49          /**
50           * @param id
51           *             id-ul ce se va seta
52           */
53          public void setId(int id)
54          {
55                  this.id = id;
56          }
57
58          /**
59           * @return timpul
60           */
61          public double getTime()
62          {
```

```
63                  return time;
64          }
65
66          /**
67           * @param time
68           *             timpul ce se va seta
69           */
70          public void setTime(double time)
71          {
72                  this.time = time;
73          }
74
75          /**
76           * @return the nodul 1
77           */
78          public Node getNode1()
79          {
80                  return n1;
81          }
82
83          /**
84           * @param n1
85           *             nodul 1 ce se va seta
86           */
87          public void setNode1(Node n1)
88          {
89                  this.n1 = n1;
90                  n1.setSegment2(this);
91          }
92
93          /**
94           * @return nodul 2
95           */
96          public Node getNode2()
97          {
98                  return n2;
99          }
100
101         /**
102          * @param n2
103          *             nodul 2 ce se va seta
104          */
105         public void setNode2(Node n2)
106         {
107                 this.n2 = n2;
108                 n2.setSegment1(this);
109         }
```

```
110
111
112         /**
113          * Functie de desenare
114          *
115          * @param p
116          */
117         public abstract void draw(PApplet p);
118
119         /**
120          * Intoarce un punct de pe segment
121          *
122          * @return
123          */
124         public abstract Point getPoint(double t);
125
126         /**
127          *
128          * @return unghiul pe care il face dreapta cu ox la
                momentul t
129          */
130         public abstract float getAngle(double t, double step);
131
132         /**
133          *
134          * @return unghiul pe care il face dreapta cu ox
135          */
136         public abstract float getAngle();
137
138         /**
139          *
140          * @return lungimea segmentului
141          */
142         public abstract float getLength();
143
144         /**
145          * @return true daca segmentul este selectat, false
                altfel
146          */
147         public boolean isSelected()
148         {
149                 return selected;
150         }
151
152         /**
153          * @param selected
154          *             seteaza campul selected true sau false
```

```
155           */
156          public void setSelected(boolean selected)
157          {
158                  this.selected = selected;
159          }
160
161
162          /**
163           * @return true daca segmentul este enabled, false
                 altfel
164           */
165          public boolean isEnabled()
166          {
167                  return enabled;
168          }
169          /**
170           * @param enabled
171           *             seteaza campul enabled true sau false
172           */
173          public void setEnabled(boolean enabled)
174          {
175                  this.enabled = enabled;
176          }
177
178
179
180          @Override
181          public String toString()
182          {
183                  String str = null;
184
185                  if (this instanceof Line)
186                          str = LINE;
187                  else if (this instanceof Curve)
188                          str = CURVE;
189
190                  str += "␣" + Double.toString(time) + "␣" +
                        getId() + "␣"
191                                      + n1.toString() + "␣" + n2.
                                           toString();
192                  return str;
193          }
194  }
```

Listing C.3: Segment.java

C.4 Linie

```
1  package simulator.segment;
2
3  import java.awt.Point;
4
5  import processing.core.PApplet;
6
7  public class Line extends Segment
8  {
9
10         /**
11          * Constructor
12          *
13          * @param n1 primul punct de pe segment
14          * @param n2 al 2-lea punct de pe segment
15          * @param id id-ul segmentului
16          */
17         public Line(Node n1, Node n2, int id)
18         {
19                 super(n1, n2, id);
20         }
21
22         @Override
23         public void draw(PApplet p)
24         {
25                 float x1 = n1.getX();
26                 float y1 = n1.getY();
27
28                 float x2 = n2.getX();
29                 float y2 = n2.getY();
30
31                 if (!isEnabled())
32                         return;
33                 p.stroke(0);
34                 if (isSelected())
35                         p.stroke(255, 255, 255);
36                 p.line(x1, y1, x2, y2);
37                 p.fill(255, 100, id);
38
39                 Point pct = getPoint(0.5);
40                 p.rect(pct.x - DIM / 2, pct.y - DIM / 2, DIM,
                      DIM);
41         }
42
43         @Override
44         public float getAngle(double t, double step)
```

```
45          {
46                  return getAngle();
47          }
48
49          @Override
50          public float getAngle()
51          {
52                  float x1, y1, x2, y2, c1, c2;
53                  float ang = 0.0f;
54
55                  x1 = n1.getX();
56                  y1 = n1.getY();
57
58                  x2 = n2.getX();
59                  y2 = n2.getY();
60
61                  c2 = x2 - x1;
62                  c1 = y2 - y1;
63
64                  if (c2 != 0)
65                          ang = (float) Math.atan(c1 / c2);
66                  else
67                          ang = (float) (Math.PI / 2);
68
69                  return ang;
70          }
71
72          @Override
73          public float getLength()
74          {
75                  float x = n1.getX() - n2.getX();
76                  float y = n1.getY() - n2.getY();
77
78                  return Math.round(Math.sqrt(x * x + y * y));
79          }
80
81          @Override
82          public Point getPoint(double t)
83          {
84                  int x = (int) (n1.getX() * (1 - t) + n2.getX()
                        * t);
85                  int y = (int) (n1.getY() * (1 - t) + n2.getY()
                        * t);
86
87                  return new Point(x, y);
88          }
89
```

```
90 }
```

Listing C.4: Line.java

C.5 Curbă

```
1  package simulator.segment;
2
3  import java.awt.Point;
4
5  import processing.core.PApplet;
6
7  public class Curve extends Segment
8  {
9
10         /**
11          * Constructor
12          *
13          * @param n1
14          *             primul punct de pe segment
15          * @param n2
16          *             al 2-lea punct de pe segment
17          * @param id
18          *             id-ul segmentului
19          */
20         public Curve(Node n1, Node n2, int id)
21         {
22                 super(n1, n2, id);
23         }
24
25         @Override
26         public void draw(PApplet p)
27         {
28                 float alpha = getNode1().getAngle();
29                 float beta = n2.getAngle();
30
31                 float xI = n1.getxI();
32                 float yI = n1.getyI();
33                 float r = n1.getR();
34
35                 if (!isEnabled())
36                         return;
37                 p.stroke(0);
38                 p.noFill();
39                 if (isSelected())
40                         p.stroke(255, 255, 255);
```

```
41
42                  p.beginShape();
43
44                  if (Math.abs(beta - alpha) < Math.PI)
45                          p.arc(xI, yI, 2 * r, 2 * r, Math.min(
                               alpha, beta), Math.max(alpha,
46                                         beta));
47                  else
48                  {
49                          p.arc(xI, yI, 2 * r, 2 * r, Math.max(
                               alpha, beta),
50                                         (float) (2 * Math.PI))
                                            ;
51                          p.arc(xI, yI, 2 * r, 2 * r, 0, Math.
                               min(alpha, beta));
52                  }
53
54                  p.endShape();
55
56                  p.fill(255, 100, id);
57                  Point pct = getPoint(0.5);
58                  p.rect(pct.x - DIM / 2, pct.y - DIM / 2, DIM,
                       DIM);
59          }
60
61          @Override
62          public float getAngle(double t, double step)
63          {
64                  Point p1 = getPoint(t);
65                  Point p2 = getPoint(t + step);
66
67                  Node n1 = new Node(p1.x, p1.y, 100, false);
68                  Node n2 = new Node(p2.x, p2.y, 100, false);
69
70                  Segment s = new Line(n1, n2, 100);
71                  return s.getAngle();
72          }
73
74          @Override
75          public float getAngle()
76          {
77                  float alpha = n1.getAngle();
78                  float beta = n2.getAngle();
79                  float angle = beta - alpha;
80
81                  if (Math.abs(beta - alpha) > Math.PI)
82                          angle = (float) (2 * Math.PI + angle);
```

```
83
84                  return angle;
85          }
86
87          @Override
88          public float getLength()
89          {
90                  return Math.round(2 * Math.PI * n1.getR()
91                                  * Math.abs(Math.toDegrees(
                                       getAngle())) / 360.0f);
92          }
93
94          @Override
95          public Point getPoint(double t)
96          {
97                  float angle = (float) (n1.getAngle() +
                       getAngle() * t);
98
99                  int x = (int) (n1.getxI() + n1.getR() * Math.
                       cos(angle));
100                 int y = (int) (n1.getyI() + n1.getR() * Math.
                       sin(angle));
101
102                 return new Point(x, y);
103         }
104
105 }
```

Listing C.5: Curve.java

Bibliografie

[1] Ben Fry Casey Reas. *A Programming Handbook for Visual Designers and Artists.* 2007.

[2] Ben Fry Casey Reas. *Getting Started with Processing.* 2010.

[3] J. A. Adams D. F. Rogers. *Mathematical Elements for Computer Graphics.* 1990.

[4] Bruce Eckel. *Thinking in Java.* 2006.

[5] G. Hera S. Petrescu M. Zaharia Florica Moldoveanu, Z. Racovita. *Grafica pe Calculator.* 1996.

[6] Ira Greenberg. *Creative Coding and Computational Art.* 2007.

[7] O. Constantin Stefan Trausan-Matu, C. Raibulet. *Interfatarea evoluata om-calculator.* 2000.

[8] O. Constantin Stefan Trausan-Matu, C. Raibulet. *Prelucrarea documentelor folosind XML si Perl.* 2001.

[9] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach.* 2010.

[10] Kostas Terzidis. *Algorithms for Visual Design Using the Processing Language.* 2009.