Systemy Robotów Autonomicznych

Wykład nr 2 – Symulator Robot Soccer

dr inż. Andrzej Opaliński
KISiM, WIMiIP AGH

Kraków, 1.10.2016

Plan wykładu

1. Rozgrywki i federacje robotów
2. Przepisy gry – Simurosot federacji FIRA
3. Prezentacja symulatora Robot Soccer v.1.5
4. Programowanie strategii w Visual Studio

Zawody robotów - historia

1. Federacja FIRA (www.fira.net)

- 1995 - pierwszy turniej Micro-Robot World Cup Soccer – KAIST, Korea

- 1996 – założenie FIRA (Federation of International Robot-soccer Association), turniej MiroSot (KAIST, Korea)

- 1997 – MiroSot World Tour (Australia, Brazylia, Kanada, Niemcy, Włochy, Meksyk, Hiszpania, U.K., USA)

- FIRA Cup - 1998 – Paryż, Francja; 1999 – Campinas, Brazylia; 2000 – Rockhampton, Australia;

2001 Pekin, Chiny; 2002 - Seul, Korea; 2003 – Wiedeń, Austria; 2004 – Busan, Korea; 2005 – Singapur;

2006 – Dortmund, Niemcy; 2007 – San Francisco, USA; 2008 – Quingdao, Chiny; 2009 - Inchelon, Korea;

2010 - Bangalore, Indie; 2011 – Kaohsiung, Tajwan, 2012-Bristol, UK, 2013-Shah Alam, Malezja,

2014 – Pekin, Chiny ; 2015 – Daejeon, Korea, 2016 – Pekin, Chiny ;

2. RoboCup (www.robocup.org)

Pre-RoboCup-96 - Osaka, Japonia;

1997 – Nagoya, Japonia; 1998 – Paryż, Francja; 1999 – Stockholm, Szwecja; 2000 – Melbourne, Australia;

2001 – Seattle, USA; 2002 – FukuOka, Japonia; 2003 - Paduva, Włochy; 2004 – Lizbona, Portugalia;

2005 – Osaka, Japonia; 2006 – Brema, Niemcy, 2007 – Atlanda, USA; 2008 – Suhozu, Chiny;

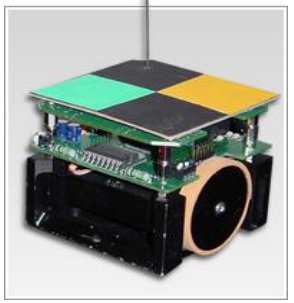
2009 – Graz, Austria; 2010 – Singapur; 2011 – Istanbul, Turcja. 2012- Mexico City, Meksyk,

2013 – Eindhoven, Holandia, 2014 – Joao Pessoa, Brazylia, 2015 - Hefei – Chiny, 2016 – Lipsk, Niemcy

Federacja FIRA - kategorie

AndroSot

waga robota do 600g, przekątna do 50cm, zdalne sterowanie, boisko 4x6m, 3 vs 3



HuroSot

dwunożne roboty humanoidalne – do 150cm i do 30kg, boisko 3x4m, 1 robot

MiroSot

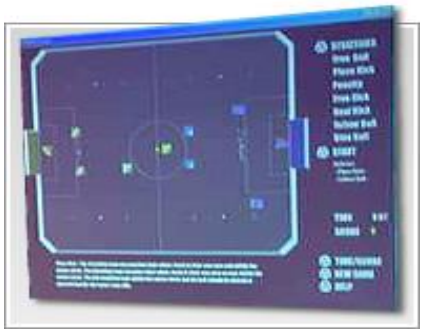
roboty 7,5 x 7,5 x 7,5 cm, 5 vs 5 na boisku 2,2m x 1,8m lub 11 vs 11 na boisku 4m x 2,8m

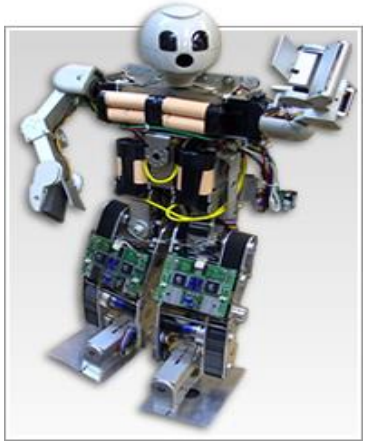
RoboSot

roboty o przekątnej 20cm bez limitu wysokości, 3vs3 na boisku 2,6m x 2,2m

SimuroSot

liga symulacyjna – **5 vs 5** lub 11 vs 11 zawodników





RoboCup - kategorie

Soccer: piłka nożna robotów, współpraca zespołów wieloagentowych i wielorobotowych

- Humanoid – roboty humanoidalne różnych rozmiarów

  (KidSize 30-60cm 3vs3; TeenSize 100-120cm 2vs2; AdultSize 130 i większe 1v1)

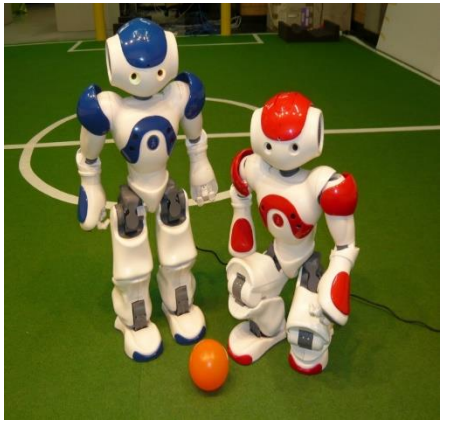
- Middle Size – roboty o średnicy do 50 cm grające „normalną” piłką
- Simulation – liga symulacyjna (2D i 3D)
- Small Size – 5vs5 robotów F180 (15x15x18cm) na boisku 6x4m.
- Standard Platform – 7,4x5,4m, zunifikowane roboty humanoidalne Nao, 3vs3

Rescue – akcje ratunkowe (Robot league/ Simulation league)

@Home – testy użyteczności technologii wsparcia życia codziennego w dziedzinach:
(Human-Robot-Interaction and Cooperation, Adaptive Behaviors, Behavior Integration, Navigation and Mapping in dynamic environments, Object Manipulation, Ambient Intelligence Computer Vision and Object Recognition under natural light conditions)

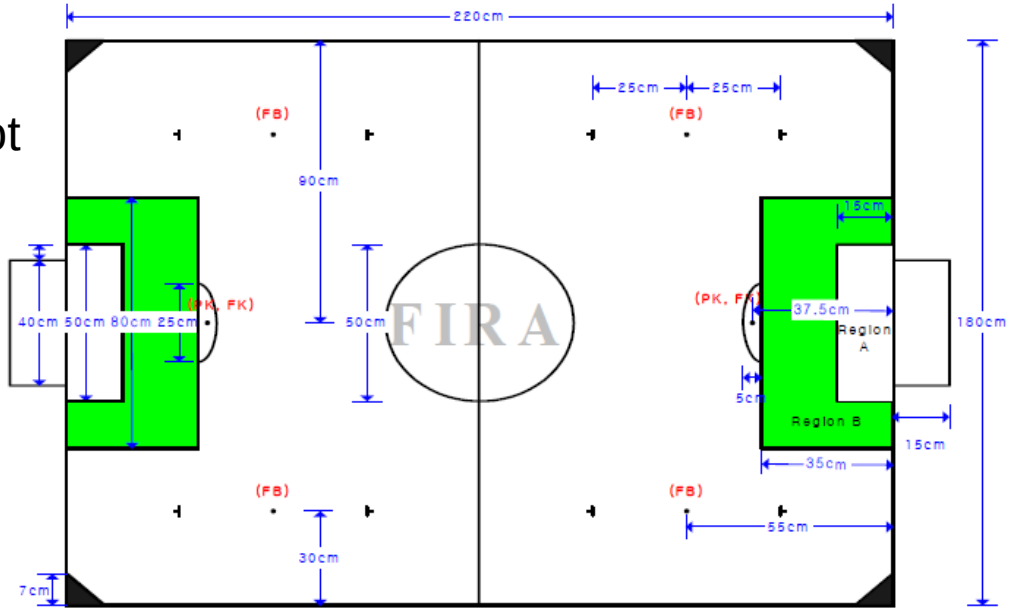
Junior – skierowany do uczniów szkół średnich

Kategorie (Soccer, Dance, Rescue)



SimuroSot Federacji FIRA – przepisy 1/4

1. Obowiązująca wersja symulatora – 1.5
2. Przepisy zgodne z Middle League Mirosot
3. Pole gry



Czarne, bez refleksów, 220x180cm,

Bandy o wysokości 5cm i szerokości 2,5cm.

W narożnikach trójkąty 7x7cm zapobiegające blokowaniu piłki.

Sprecyzowane są grubości, kolory linii i krzywizny łuków.

4. Czas gry

Dwie połowy po 5 minut, 10 minut przerwy między połowami.

Czas przy zmianach i przerwach na żądanie.

Dwie 2-minutowe przerwy na żądanie w ciągu meczu dla każdej z drużyn

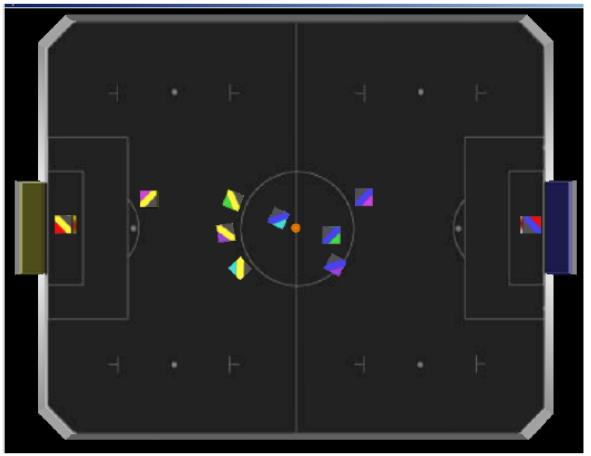
5. Przerwy w grze – następują gdy:

Padł gol lub został popełniony faul

Sędzia odgwizdał rzut karny, rzut wolny, rzut bramkowy lub wolną piłkę.

Nastąpił błąd symulatora lub sprzętu

6. Robot w strefie bramkowej jest uważany za bramkarza
7. Przed początkiem gry losowana jest drużyna rozpoczynająca, mająca możliwość wyboru kolorów markerów



8. Początek połowy

Drużyna rozpoczynająca przed rozpoczęciem połowy ustawia swoje roboty dowolnie na swojej połowie boiska oraz w kole. Drużyna przeciwna ustawia roboty na swojej połowie z wyłączeniem koła środkowego. W drugiej połowę meczu drużyny zamieniają się stronami. Po zmianie połów oraz po strzeleniu gola gra wznawiana jest od środka. Należy rozpocząć podaniem na własną połowę.

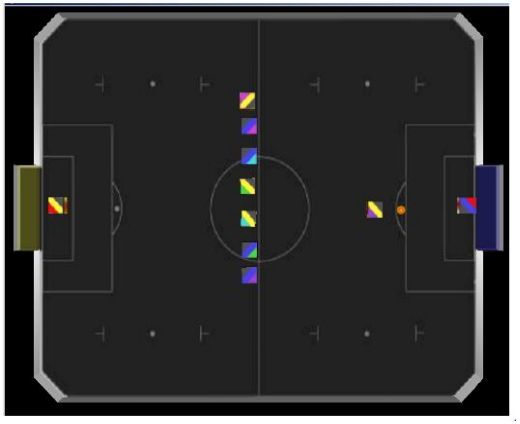
9. Sędzia

Dopilnowuje przestrzegania reguł gry.

Może zatrzymać grę z uwzględnieniem zasad i sytuacji zaistniałej na boisku i wokół niego.

10. Punktacja

Bramka zostaje zdobyta gdy cała piłka przekroczy linię bramkową. W wypadku remisu po 2 połowie następuje 5 minut przerwy i 3 minutowa dogrywka na zasadzie złotej bramki. Jeśli w ciągu 5 minut nie padnie gol następują rzuty karne. Każda drużyna wykonuje po 3 rzuty karne. Po gwizdu sędziego bramkarz może opuścić pole bramkowe. Rzut karny jest wykonany jeśli piłka opuści pole karne, lub minie 30 sekund od gwizdka sędziego.



11. Faule powodujące rzut karny (penalty kick)

Obrona więcej niż jednym robotem w polu bramkowym.

(więcej niż 50% robota w polu karnym)

Obrona więcej niż trzema robotami w polu karnym. (50% j.w.)

12. Pozycja piłki i robotów w czasie rzutu karnego

jak na rysunku, szczegółowo w przepisach.

13. Przewinienia skutkujące rzutem wolnym (free kick)

Kolizja z robotem przeciwnika wpływająca na grę lub mogąca uszkodzić robota przeciwnika

Dopuszcza się przepychanie piłki i robota przeciwnika wstecz w wypadku gdy robot przeciwnika ma ciągły kontakt z piłką.

14. Pozycja robotów i piłki podczas rzutu wolnego

Drużyna atakująca dowolnie.

Drużyna broniąca – jak na rysunku, szczegóły w przepisach.

Gra rozpoczyna się po gwizdku w normalnym trybie.

Dopuszczalny strzał lub drybling.

15. Wolna piłka (free ball)

Gdy przez 10 sekund wystąpi zakleszczenie poza polem bramkowym.

Piłka zostaje umieszczona na miejscu „FB” w odpowiedniej ćwiartce.

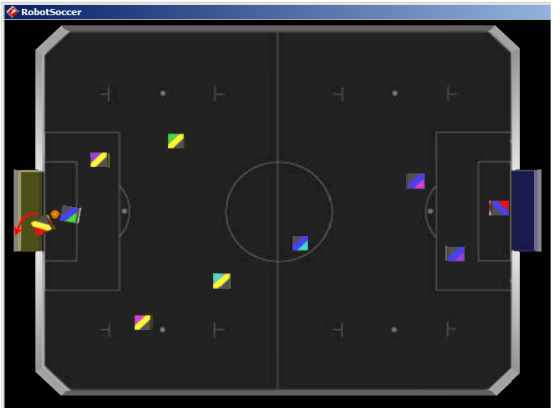
Po jednym robocie z każdej drużyny umieszczane jest w odległości 25 cm od piłki.

Pozostałe roboty ustawiane są dowolnie, poza ćwiartką w której wykonywana jest wolna piłka.

Drużyna broniąca pierwsza rozstawia swoje roboty.

SimuroSot, przepisy 4/4

16. Wybicie od bramki (tzw. „piątka”) (goal kick)



Faul, wybicie od bramki

Atakowanie więcej niż jednym robotem w polu bramkowym.

Robot atakujący przepycha lub blokuje bramkarza co ma wpływ na przebieg gry.

Robot atakujący wpycha bramkarza do bramki gdy piłka jest pomiędzy.

Robot atakujący przepycha bramkarza „za pomocą piłki” utrudniając obronę bramki.

W polu bramkowym następuje zakleszczenie przez min. 10 sekund.

Kontakt z bramkarzem nie uniemożliwiający mu blokowania bramki, nie jest przewinieniem.

Jest dozwolone przepychanie bramkarza w polu bramkowym, jeśli piłka jest między robotami, jednak niedozwolone jest wpychanie robota do bramki, ani wypychanie z pola bramkowego.

O wszystkich sytuacjach spornych decyduje sędzia.

17. Pozycja robotów i piłki dla wybicia od bramki

Jedynie bramkarz może znajdować się w polu bramkowym. Piłka można znajdować się w dowolnym miejscu pola bramkowego.

Pozostałe roboty drużyny wybijającej powinny znaleźć się poza polem bramkowym tak jak na rysunku.

Roboty przeciwnika powinny znaleźć się na własnej połowie boiska.

Sytuacja dozwolona

Symulator Robot Soccer

1. The Robot Soccer Simulator, Middle League (5 vs 5), wersja 1.5. - RSS Development Tean

Symulator robotów „Yujin Robots”. Do pobrania ze strony: http://www.fira.net/?mid=simurosot

Napisany w środowisku (Adobe) Macromedia Director i języku Lingo. http://www.adobe.com/support/director/lingo.html

2. Dokumentacja

file:///C:/Program Files (x86)/Robot Soccer v1.5a/SoccerHelp.htm

3. Minimalne wymagania sprzętowe

Pentium III 600 MHz
256 MB ram
Grafika TNT2 3d, 32 MB ram
24x CD-ROM
Rozdzielczość ekranu: of 800 x 600
16 bitowa karta dźwiękowa
Microsoft Windows 98
Direct X 8.0
10 MB miejsca na dysku twardym

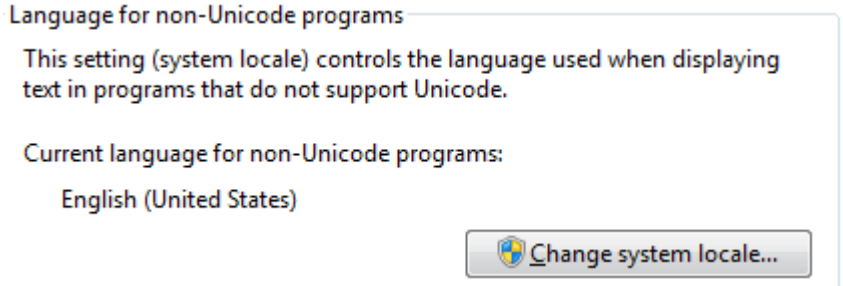
4. Instalacja

Zalecana instalacja na dysku C:

Utworzy katalog C:\Strategy\ (niezbędne prawa administratora).

Konieczna zmiana ustawień regionalnych znaków na English(US)

(WIN 7: Control Panel -> Region and Languages -> Administratives)



Robot Soccer – interfejs użytkownika 1

Pozycje robotów na boisku można ustawiać za pomocą kursora myszy.

Obrót robota – za pomocą klawiszy strzałek.

Ładowanie strategii dla obydwu zespołów

Faule i posiadanie piłki - dla sędziego

Rozpoczęcie/wznowienie gry

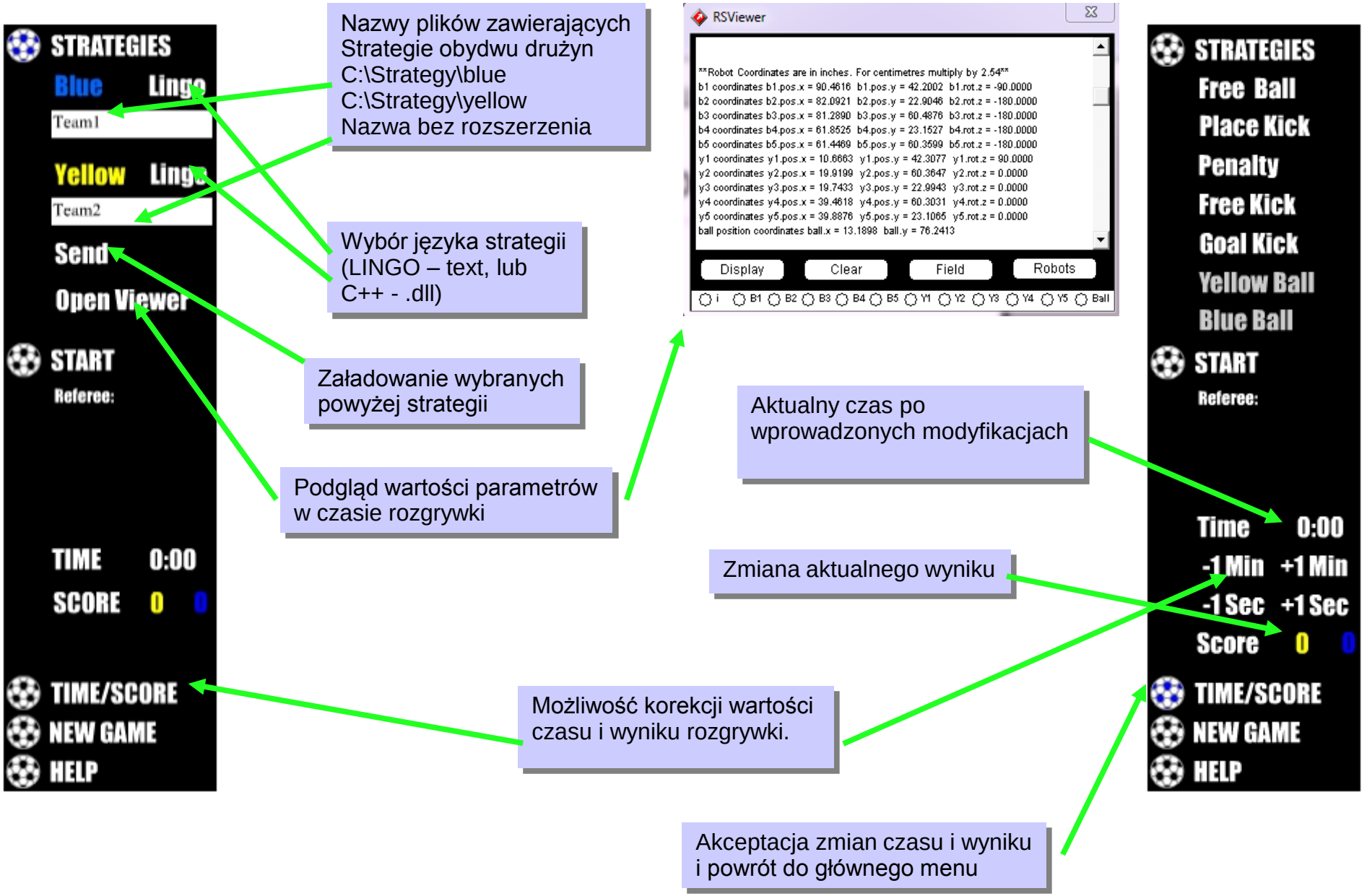
Aktualny czas i wynik meczu

Zmiana wyniku i czasu rozgrywki przez sędziego

Reset ustawień i rozpoczęcie nowej rozgrywki

Dostęp do pliku pomocy w wersji .html

Robot Soccer, interfejs: strategie, czas, wynik



Robot Soccer: przebieg gry, powtórki

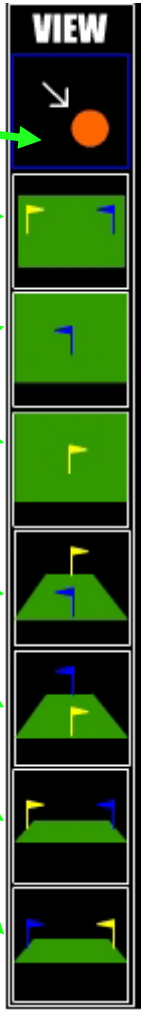
Po uruchomieniu gry („START”)

Przejście do menu powtórek

Wstrzymanie gry

Zatrzymanie gry i powrót do menu w celu podjęcia decyzji przez sędziego

Ustawienie kamery



Podążanie za piłką po boisku

Widok domyślny, z góry na dół

Widok w kierunku bramki drużyny niebieskiej / żółtej

Widok całego boisko zza bramki drużyny niebieskiej / żółtej

Perspektywiczny widok z boku

Sterowanie odtwarzaniem powtórek

Przewinięcie do początku i rozpoczęcie odtwarzania

Odtwarzanie z normalną szybkością

Opuszczenie menu powtórek

Przewinięcie 300 klatek wstecz i rozpoczęcie odtwarzania

Odtwarzanie w zwolnionym tempie

Odtwarzanie w trybie „klatka po klatce”

Anulowanie gola, jeśli menu replay uruchomione po golu

Robot Soccer – uruchomienie symulacji

1. Wybór strategii dla obydwu zespołów

2. Wybór języka oraz wpisanie nazwy plików zawierających strategie

3. Wybór „Place kick” oraz drużyny rozpoczynającej

4. Początkowe ustawienie robotów na boisku. (mysz i klawisze strzałek)

5. Start rozgrywki

Programowanie strategii, kodowanie

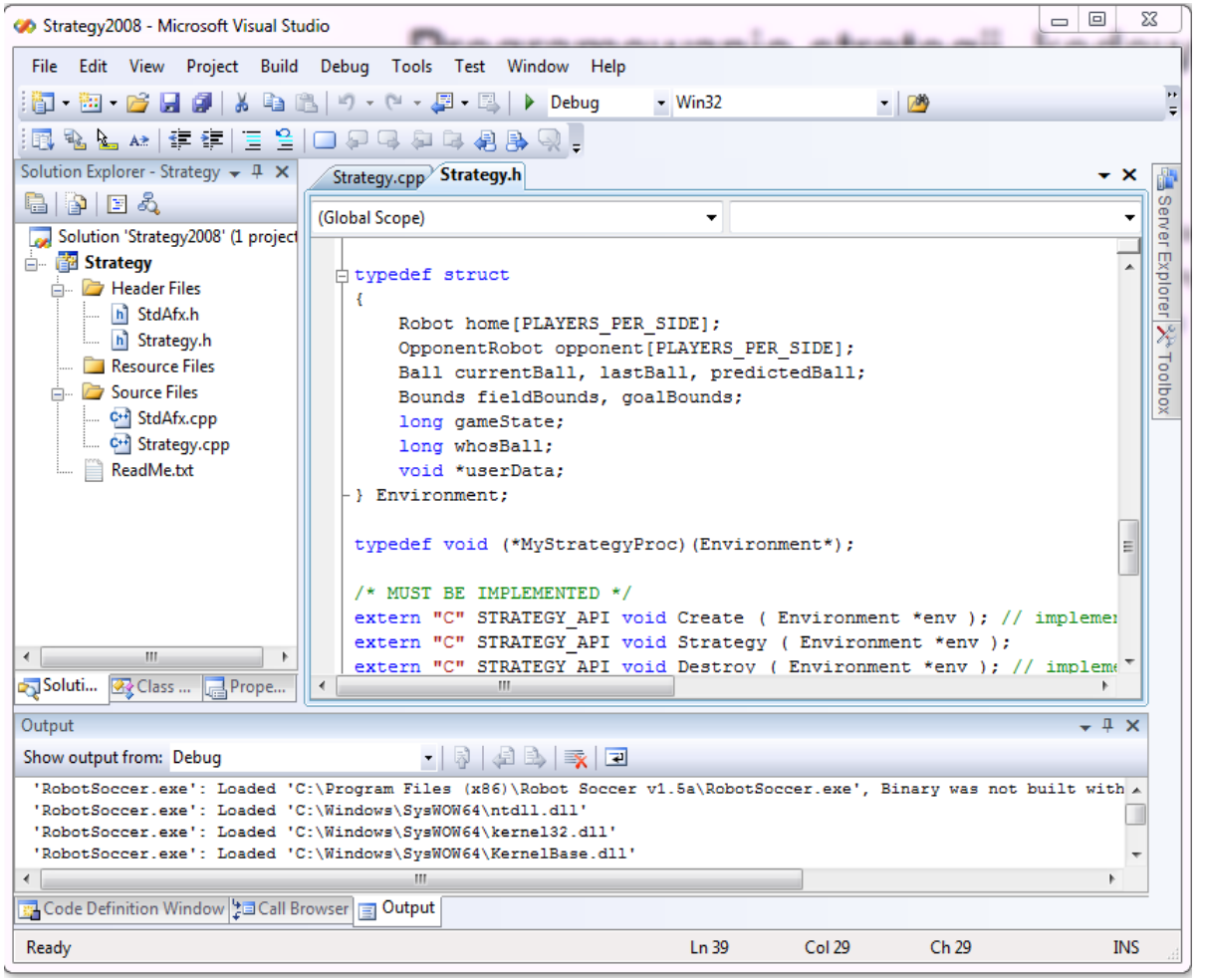
Przykładowy plik ze strategiami w języku Lingo i VC++ 6.0 dostarczony z symulatorem.

Projekt w C++ dla Visual Studio 2008 umieszczony na stronie przedmiotu. (http://tempus.metal.agh.edu.pl/~opal/sysra/)

Najważniejsze pliki i składowe projektu / strategii:

- Strategy.h- deklaracja zmiennych, struktur i głównych funkcji używanych w rozgrywce

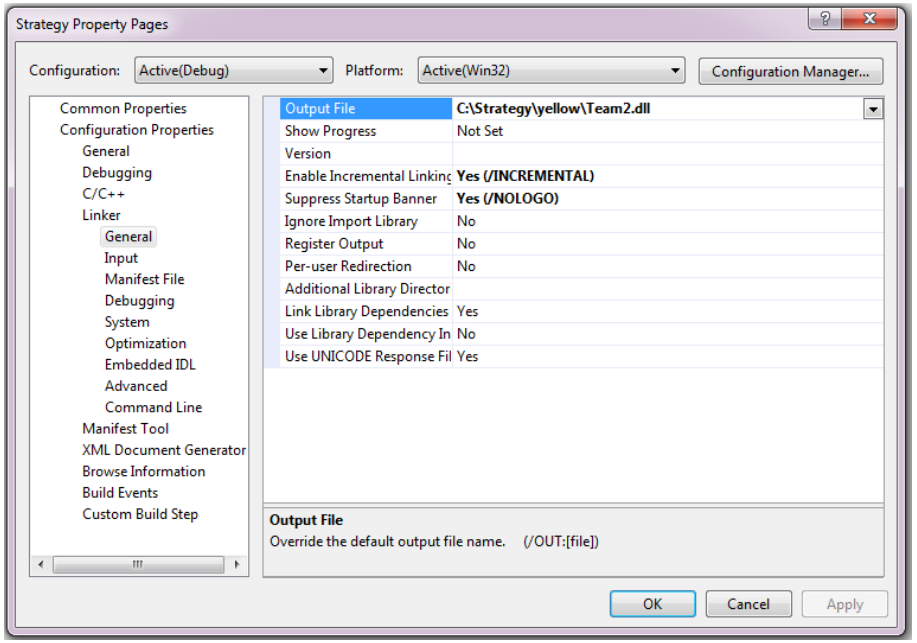
- Strategy.cpp – definicja funkcji głównych oraz dodatkowych



Programowanie strategii, ułatwienia

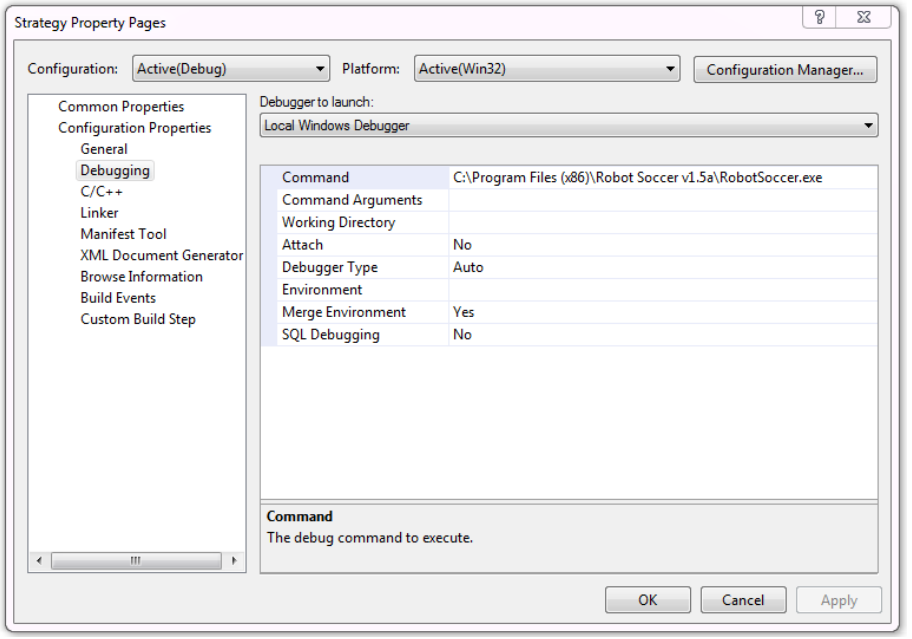
W ustawieniach linkera:

- katalog docelowy strategii
- nazwa strategii Team2.dll



W ustawieniach kompilatora:

- aplikację RobotSoccer.exe do testów



Kodowanie strategii, plik nagłówkowy, zmienne

Współrzędne boiska (zawierają wartość jednego parametru x lub y)

FTOP	współrzędna „y” górnej bandy boiska
FBOT	współrzędna „y” dolnej bandy boiska
FRIGHTX	współrzędna „x” prawej bandy boiska
FLEFTX	współrzędna „x” lewej bandy boiska
GTOPY	współrzędna „y” górnego początku bramki (dla obydwu bramek)
GBOTY	współrzędna „y” dolnego końca bramki (dla obydwu bramek)
GRIGHT	współrzędna „x” prawego końca prawej bramki
GLEFT	współrzędna „x” lewego końca lewej bramki

```
const double FTOP = 77.2392;
const double FBOT = 6.3730;
const double GTOPY = 49.6801;
const double GBOTY = 33.9320;
const double GRIGHT = 97.3632;
const double GLEFT = 2.8748;
const double FRIGHTX = 93.4259;
const double FLEFTX = 6.8118;
```

Stan gry: zmienna „GameState”

1 = Free ball

2 = Place kick(Kickoff)

3 = Penatly kick

4 = Free kick

5 = Goal kick

Posiadanie piłki: zmienna „WhosBall”

0 = Anyones ball

1 = Blue ball

2 = Yellow ball

Wszystkie wartości w calach, w centymetrach * 2,54

Rozmiary boiska: 220cm x 180cm
to jest: 86,6 in x 71 in

Kodowanie strategii, plik nagłówkowy, struktury

```
typedef struct {                        wektor położenia
double x, y, z;
} Vector3D;

typedef struct {                        struktura opisująca ograniczenia prostokąta
long left, right, top, bottom;
} Bounds;

typedef struct {                            robot własny
Vector3D pos;                           pozycja
double rotation;                        kąt obrotu
double velocityLeft, velocityRight;     prędkość lewego, prawego koła
} Robot;

typedef struct {                            robot drużyny przeciwnej
Vector3D pos;                   pozycja
double rotation;                kąt obrotu
} OpponentRobot;

typedef struct {                            piłka
Vector3D pos;                   pozycja
} Ball;

typedef struct {                                        Środowisko gry
Robot home[PLAYERS_PER_SIDE];                   roboty drużyny własnej
OpponentRobot opponent[PLAYERS_PER_SIDE];       roboty drużyny przeciwnika
Ball currentBall, lastBall, predictedBall;      położenie piłki, aktualne, ostatnie, przewidywane
Bounds fieldBounds, goalBounds;                 wymiary (współrzędne) boiska, bramki
long gameState;                                 stan gry
long whosBall;                                  posiadanie piłki
void *userData;                                 dane własne użytkownika
} Environment;
```

Roboty, podział na role, dostęp do zmiennych

Drużyna niebieska, żółta


B0, Y0 - bramkarz



B1, Y1 – pierwszy obrońca



B2, Y2 – drugi obrońca



B3, Y3 – pierwszy napastnik



B4, Y4 – drugi napastnik


Parametry położenia robotów:

&env->home[0]->pos.x – współrzędna X bramkarza drużyny własnej

&env->opponent[2]->pos.y – współrzędna Y drugiego obrońcy drużyny przeciwnej

&env->home[4]->rotation – kąt obrotu drugiego napastnika drużyny własnej

```
void MoonAttack ( Robot *robot, Environment *env )
{
    PredictBall ( env );
    Position(robot, env->predictedBall.pos.x, env->predictedBall.pos.y);
}

void MoonFollowOpponent ( Robot *robot, OpponentRobot *opponent )
{
    Position(robot, opponent->pos.x, opponent->pos.y);
}

void Velocity ( Robot *robot, int vl, int vr )
{
    robot->velocityLeft = vl;
    robot->velocityRight = vr;
}
```

Kodowanie strategii, plik Strategy.cpp

Główna metoda strategii – 60 razy na sekundę

```
extern "C" STRATEGY_API void Strategy ( Environment *env )
```

Metody tworzące i usuwające zmienne środowiskowe użytkownika

```
extern "C" STRATEGY_API void Create ( Environment *env )
extern "C" STRATEGY_API void Destroy ( Environment *env )
```

Metoda systemowa

```
BOOL APIENTRY DllMain ( HANDLE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)
```

Metody składowe strategii:

```
void PredictBall ( Environment *env );
void Goalie1 ( Robot *robot, Environment *env );
void NearBound2 ( Robot *robot, double vl, double vr, Environment *env );
void Attack2 ( Robot *robot, Environment *env );
void Defend ( Robot *robot, Environment *env, double low, double high );
```

Strzał na bramkę

```
void MoonAttack (Robot *robot, Environment *env );
```

Podążanie za przeciwnikiem

```
void MoonFollowOpponent (  Robot *robot, OpponentRobot *opponent );
```

Metody podstawowe:

```
void Velocity ( Robot *robot, int vl, int vr );          - ustawienie prędkości na obydwa koła
void Angle ( Robot *robot, int desired_angle);           - ustawienie robota pod danym kątem
void Position( Robot *robot, double x, double y );       - przemieszczenie robota do danej pozycji
```

Kodowanie strategii, podsumowanie

Główna idea:
Podjąć decyzję o wyborze strategii na podstawie zmiennych ze środowiska

- gameState – stanu gry
- whichBall – stanu posiadania piłki
- **environment** – informacji o **położeniu robotów własnych, przeciwnika oraz piłki**

Realizacja za pomocą istniejących metod podstawowych

```
void Position( Robot *robot, double x, double y ) {
        ...
        Velocity(robot, vl, vr);
}

void Velocity ( Robot *robot, int vl, int vr ) {
        robot->velocityLeft = vl;
        robot->velocityRight = vr;
}
```

Cel pracy na laboratorium:

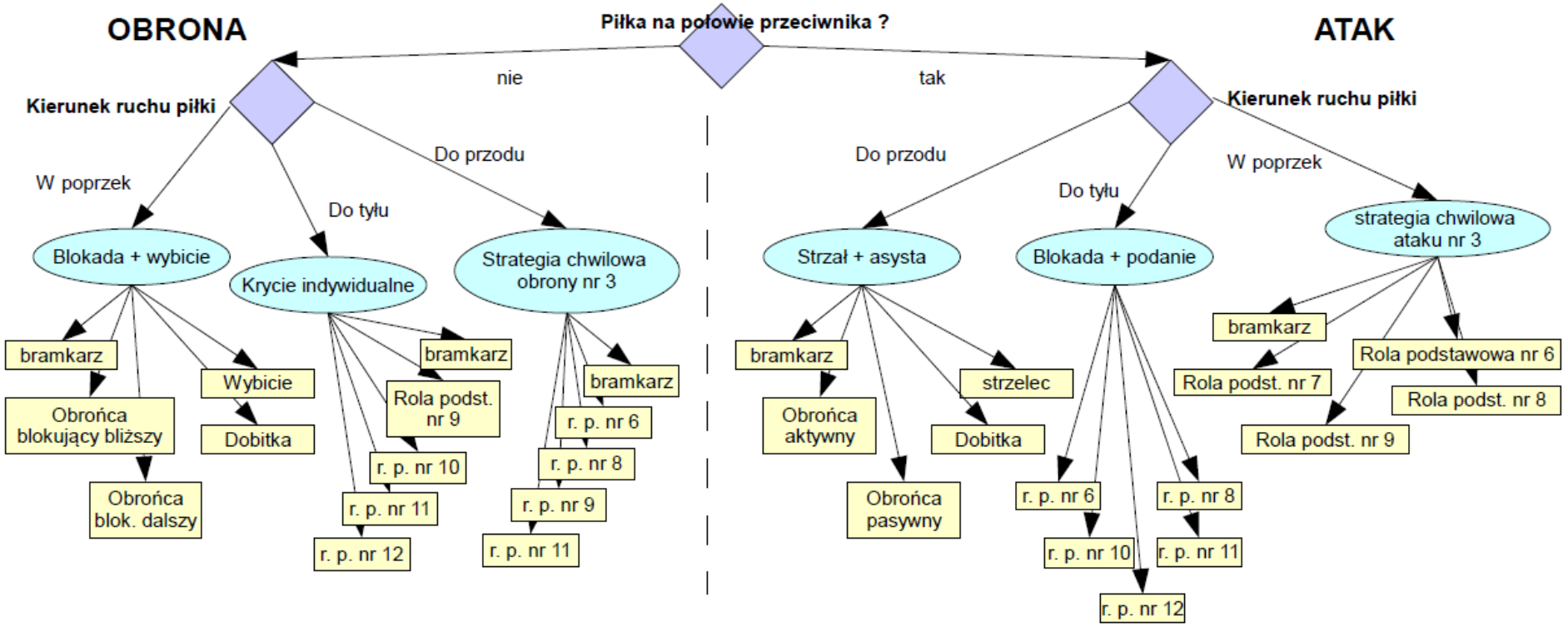
- implementacja własnych wersji istniejących metod podstawowych
- implementacja nowych metod takich jak.:
- dojazd do punktu po innej trajektorii
- omijanie przeszkód
- wybór lepszego strzelca
- inteligentne blokowanie strzału, podania
- strzał na bramkę
- rozbicie strategii ataku/obrony na podstrategie

Procedura testów:

Zakodować własną strategie
Przebudować projekt (utworzyć .dll)
Wkopiować do katalogu strategii (lub konf. opcji)
Wczytać w symulatorze
Testować w rozgrywce ze strategią
- podstawową
- własną
- kolegi

```
extern "C" STRATEGY_API void Strategy ( Environment *env ) {
switch (env->gameState) {
        case 0:
                // default
                MoonFollowOpponent ( &env->home [1], &env->opponent [2] );
                MoonFollowOpponent ( &env->home [2], &env->opponent [3] );
                MoonFollowOpponent ( &env->home [3], &env->opponent [4] );
                MoonAttack ( &env->home [4], env );
                Goalie1 ( &env->home [0], env );break;
        case FREE_BALL:
                // Follow opponent guy
                MoonFollowOpponent ( &env->home [1], &env->opponent [2] );
                MoonFollowOpponent ( &env->home [2], &env->opponent [3] );
                MoonFollowOpponent ( &env->home [3], &env->opponent [4] );
                // attack
                MoonAttack ( &env->home [4], env );
                // Goal keeper
                Goalie1 ( &env->home [0], env );break;
        case PLACE_KICK:
                MoonAttack ( &env->home [2], env );break;
        case PENALTY_KICK:
                switch (env->whosBall) {
                case ANYONES_BALL:
                        MoonAttack ( &env->home [1], env );break;
                case BLUE_BALL:
                        MoonAttack ( &env->home [4], env );break;
                case YELLOW_BALL:
                        MoonAttack ( &env->home [0], env );break;
                } break;
        case FREE_KICK:
                MoonAttack ( &env->home [0], env ); break;
        case GOAL_KICK:
                MoonAttack ( &env->home [0], env );break;
 }
}
```

Projektowanie strategii, schemat



Warunki wyboru strategii chwilowych

- Kierunek ruchu piłki
- Położenie piłki (połowa)
- Aktualny wynik meczu
- Położenie robotów wzgl. piłki
- Potencjalna możliwość wykonania akcji
- ...

Cel: Na podstawie warunków chwilowych **wybrać najlepszą strategię chwilową** w aktualnym momencie rozgrywki

Strategie chwilowe

- Strzał + asysta
- Blokada + podanie
- Blokada + wybicie
- Krycie indywidualne
- ...

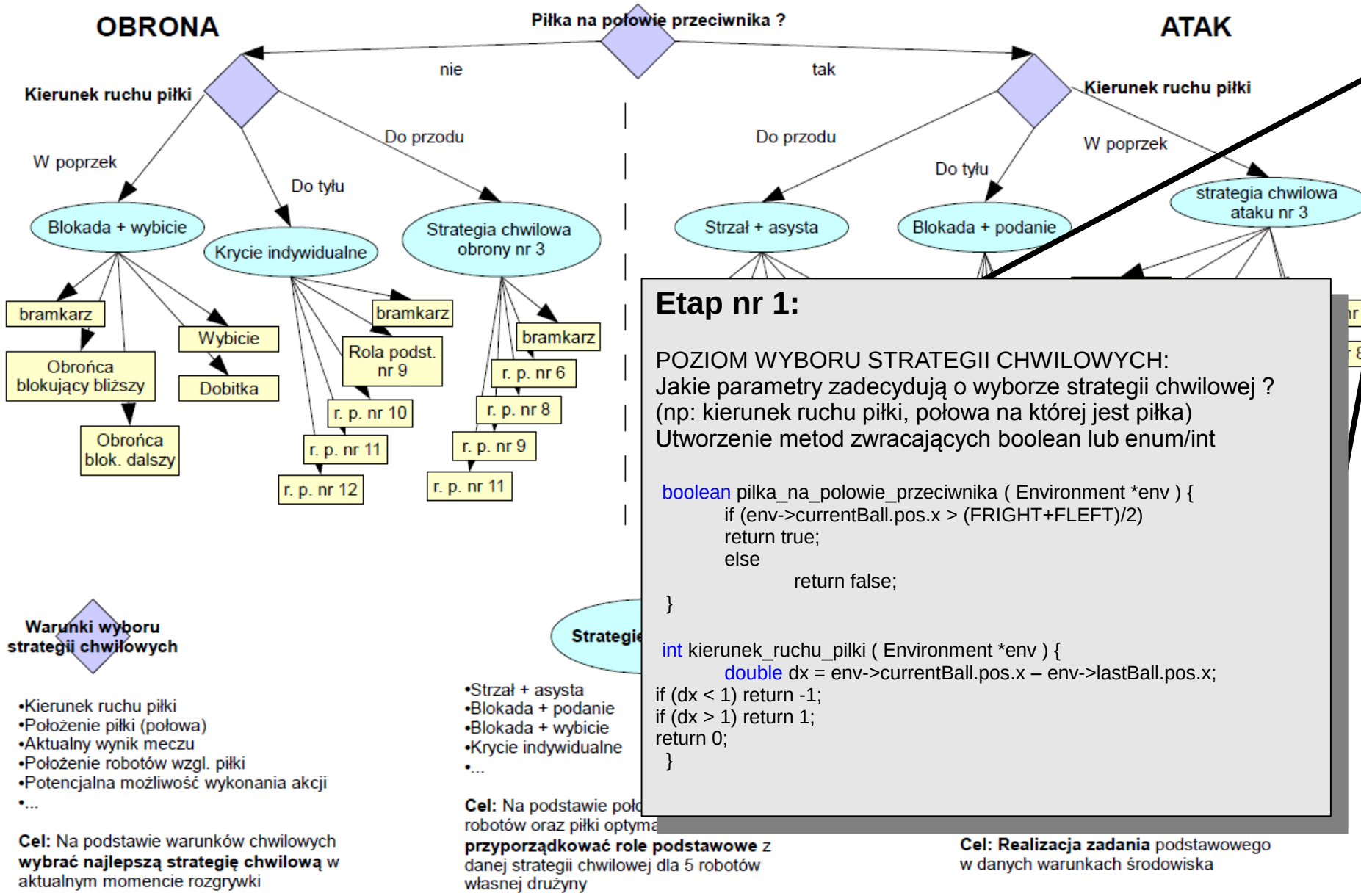
Cel: Na podstawie położenia wszystkich robotów oraz piłki optymalnie **przyporządkować role podstawowe** z danej strategii chwilowej dla 5 robotów własnej drużyny

Role podstawowe

- Bramkarz
- Obrońca pasywny
- Obrońca blokujący bliższy
- Atakujący
- Podający
- Dobijający
- ...

Cel: Realizacja zadania podstawowego w danych warunkach środowiska

Projektowanie strategii, schemat



Etap nr 1:

POZIOM WYBORU STRATEGII CHWILOWYCH:
Jakie parametry zadecydują o wyborze strategii chwilowej ?
(np: kierunek ruchu piłki, połowa na której jest piłka)
Utworzenie metod zwracających boolean lub enum/int

```
boolean pilka_na_polowie_przeciwnika ( Environment *env ) {
        if (env->currentBall.pos.x > (FRIGHT+FLEFT)/2)
        return true;
        else
                return false;
 }

int kierunek_ruchu_pilki ( Environment *env ) {
        double dx = env->currentBall.pos.x – env->lastBall.pos.x;
if (dx < 1) return -1;
if (dx > 1) return 1;
return 0;
 }
```

Warunki wyboru strategii chwilowych

- Kierunek ruchu piłki
- Położenie piłki (połowa)
- Aktualny wynik meczu
- Położenie robotów wzgl. piłki
- Potencjalna możliwość wykonania akcji
- ...

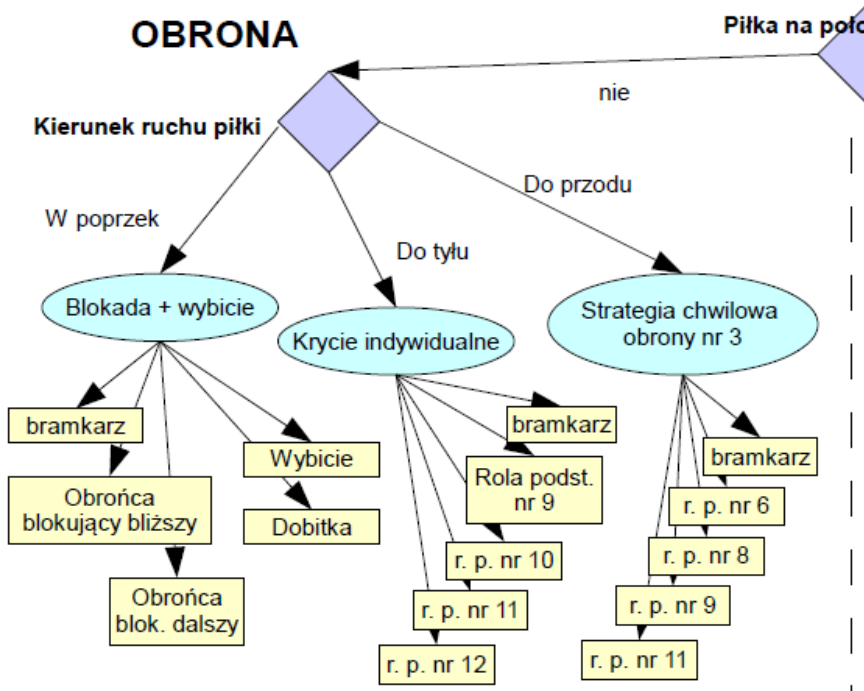
Cel: Na podstawie warunków chwilowych **wybrać najlepszą strategię chwilową** w aktualnym momencie rozgrywki

- Strzał + asysta
- Blokada + podanie
- Blokada + wybicie
- Krycie indywidualne
- ...

Cel: Na podstawie poł... robotów oraz piłki optyma... **przyporządkować role podstawowe** z danej strategii chwilowej dla 5 robotów własnej drużyny

Cel: Realizacja zadania podstawowego w danych warunkach środowiska

Projektowanie strategii, schemat



Etap nr 2:

POZIOM DEFINICJI STRATEGII CHWILOWYCH:
Utworzenie strategii chwilowych (czyli podziału ról dla wszystkich robotów własnych) oraz pomocniczych funkcji decyzyjnych dla wszystkich możliwych sytuacji

```
 void atak_dynamiczny ( Environment *env ) {
Bramkarz(&env->home [0]);
ObroncaAktywny(&env->home [1]);
ObroncaPasywny(&env->home [2]);
Strzelec_czy_pomocnik(&env->home [3],&env->home [4]);
 }
 void Strzelec_czy_pomocnik ( Robot *rob1,  Robot *rob2) {
if (&rob1->position.x > &rob2->position.x) {
        Strzelec(&rob1);
        Pomocnik(&rob2);
}else{
        Strzelec(&rob2);
        Pomocnik(&rob1);
}
 }
```

1
2
3

Warunki wyboru strategii chwilowych

- Kierunek ruchu piłki
- Położenie piłki (połowa)
- Aktualny wynik meczu
- Położenie robotów wzgl. piłki
- Potencjalna możliwość wykonania akcji
- ...

Cel: Na podstawie warunków chwilowych **wybrać najlepszą strategię chwilową** w aktualnym momencie rozgrywki

Strategie chwilowe

- Strzał + asysta
- Blokada + podanie
- Blokada + wybicie
- Krycie indywidualne
- ...

Cel: Na podstawie położenia wszystkich robotów oraz piłki optymalnie **przyporządkować role podstawowe** z danej strategii chwilowej dla 5 robotów własnej drużyny

Role podstawowe

- Bramkarz
- Obrońca pasywny
- Obrońca blokujący bliższy
- Atakujący
- Podający
- Dobijający
- ...

Cel: Realizacja zadania podstawowego w danych warunkach środowiska

Projektowanie strategii, schemat

Etap nr 3:

POZIOM DEFINICJI METOD PODSTAWOWYCH:
Implementacja wszystkich ról podstawowych (Bramkarz, Obrońca, Atakujący, ...) na podstawie:
- informacji o środowisku

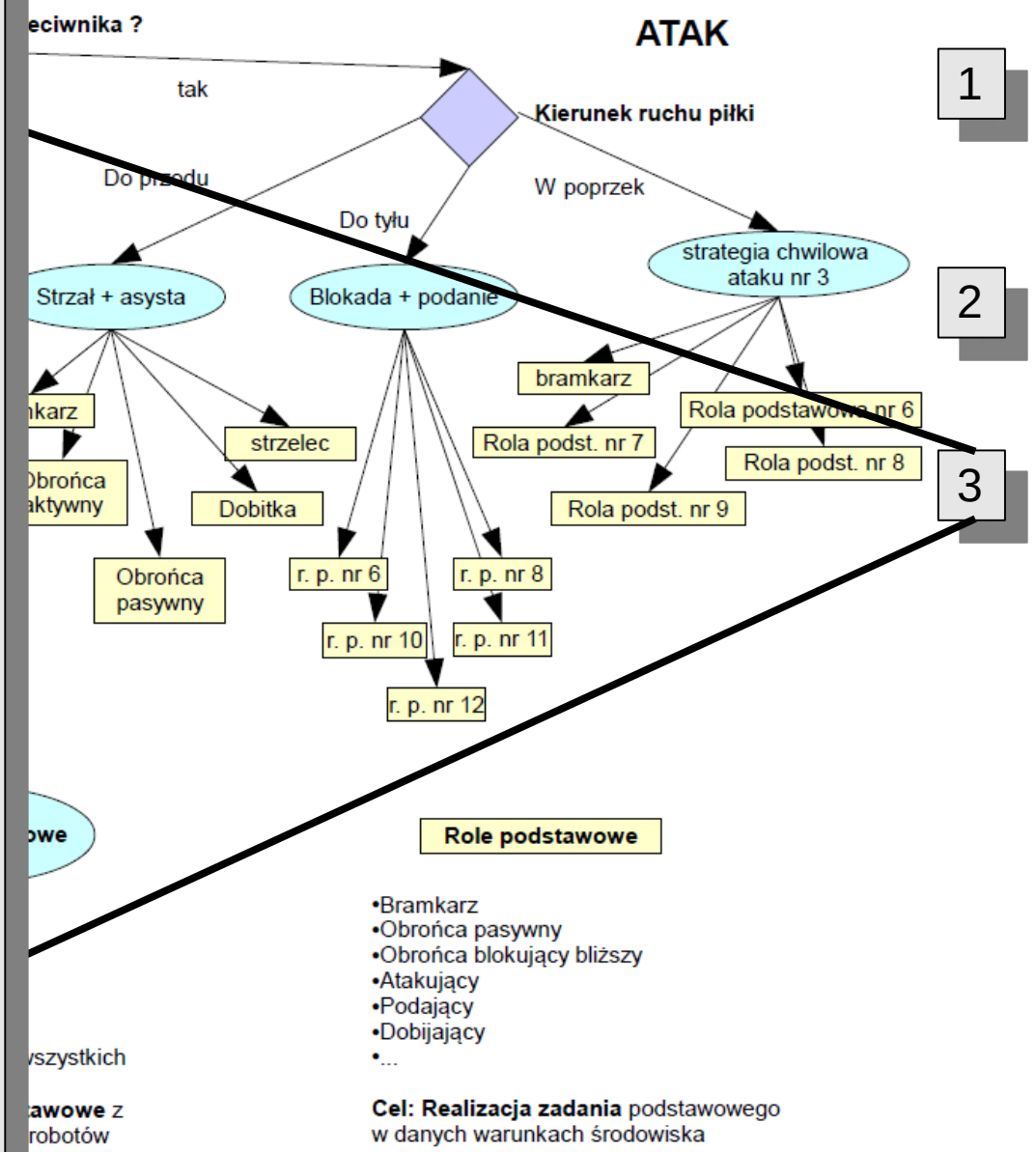
```
typedef struct {
        Robot home[PLAYERS_PER_SIDE];
        OpponentRobot opponent[PLAYERS_PER_SIDE];
        Ball currentBall, lastBall, predictedBall;
        Bounds fieldBounds, goalBounds;
        long gameState;
        long whosBall;
        void *userData;
} Environment;
```

- dostępnych funkcji pomocniczych.

```
void Velocity ( Robot *robot, int vl, int vr ){
        robot->velocityLeft = vl;
        robot->velocityRight = vr;
}
void Position( Robot *robot, double x, double y ) {
        ... // 55 lini kodu
        Velocity(robot, vl, vr);
}
void MoonAttack ( Robot *robot, Environment *env ) {
        PredictBall ( env );
        Position(robot, env->predictedBall.pos.x, env->predictedBall.pos.y);
}
void MoonFollowOpponent ( Robot *robot, OpponentRobot *opponent ) {
        Position(robot, opponent->pos.x, opponent->pos.y);
}
```

Przykład (najprostszy bramkarz):

```
 void Bramkarz (Robot * rob, Environment *env ) {
        double y_ball = env->currentBall.pos.y;
        Position(rob, 10, y_ball);
 }
```



Najczęstszy błąd

Traktowanie strategii jako sekwencyjnego zbioru instrukcji

Niepoprawnie:

```
typedef struct {
        Robot home[PLAYERS_PER_SIDE];
        OpponentRobot opponent[PLAYERS_PER_SIDE];
        Ball currentBall, lastBall, predictedBall;
        Bounds fieldBounds, goalBounds;
        long gameState;
        long whosBall;
        void *userData;
} Environment;

void Velocity ( Robot *robot, int vl, int vr ){
        robot->velocityLeft = vl;
        robot->velocityRight = vr;
}
void Position( Robot *robot, double x, double y ) {
        ... // 55 lini kodu
        Velocity(robot, vl, vr);
}

void MoonAttack ( Robot *robot, Environment *env ) {
        PredictBall ( env );
        Position(robot, env->predictedBall.pos.x, env->predictedBall.pos.y);
}


void doSquare( Robot *robot, double xTopLeft, double yTopLeft ) {

        Position(robot, xTopLeft, yTopLeft);
        Position(robot, xTopLeft, yTopLeft-10);
        Position(robot, xTopLeft+10, yTopLeft-10);
        Position(robot, xTopLeft+10, yTopLeft);

}
```

Poprawnie:

```
typedef struct {
        Robot home[PLAYERS_PER_SIDE];
        OpponentRobot opponent[PLAYERS_PER_SIDE];
        Ball currentBall, lastBall, predictedBall;
        Bounds fieldBounds, goalBounds;
        long gameState;
        long whosBall;
        void *userData;
} Environment;

public  int sqrState = -1;
int dest[4][2] = { {0,0},{0,-10},{10,-10},{10,0} };

…. (jak po lewej stronie)

Bool closeTo(Robot *robot, double xPos, double yPos) {
   if ((abs(robot.pos.x-xPos)<1) && (abs(robot.pos.y-yPos)<1))
        return true
   else return false;
}

void doSquare( Robot *robot, double xTopLeft, double yTopLeft ) {

  if (sqrState<0 || sqrState>3)
      sqrState=0;

  if (closeTo(robot,xTopLeft+dest[sqrState][0], yTopLeft+dest[sqrState][1]) )
      sqrState+=1;
  else
      Position(robot, xTopLeft+dest[sqrState][0], yTopLeft+dest[sqrState][1]);
}
```

Kodowanie strategii, podsumowanie

Wybór strategii i przydział ról odbywa się za każdym razem (60 razy na sekundę)

Metoda `void Strategy ( Environment *env )` **musi zakończyć się ustawieniem prędkości na kołach dla wszystkich własnych robotów**

Trzy-etapowa struktura tworzenia strategii drużyny:
- wybór strategii chwilowej na podstawie warunków środowiska
- przydział ról w wybranej strategii chwilowej
- realizacja ról podstawowych

Uwagi:
Uniwersalność strategii (blue/yellow) – zmienna boolean identyfikująca drużynę
Predykcja kierunku toru piłki

Literatura, materiały

www.fira.net

www.robocup.org

FIRA Middle League Simurosot Game Rules (Mar 2006) (http://fira.net/?mid=simurosot)

Robot Soccer Simulator Documentation (file:///C:/Program Files (x86)/Robot Soccer v1.5a/SoccerHelp.htm)

Pytania ?