

FIRACE  
dokumentacja projektowa

Michał Sroka      Sebastian Skałacki

19 czerwca 2007

## **Streszczenie**

Niniejszy dokument stanowi dokumentację do projektu Firace realizowanego na przedmiocie Techniki Mikroprocesorowe na 6 semestrze dziennych studiów na kierunku informatyka, na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki, na Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie.

Celem projektu jest sterowanie mobilnym robotem w wyścigu po zadanej trasie. Zakres obowiązków projektowanego rozwiązania obejmuje komunikację ze sterownikiem RoBOSS2, interpretację fotografii planszy, na której zbudowano trasę wyścigu, oraz bezkolizyjne i możliwie szybkie przeprowadzenie jednego lub kilku robotów przez tę trasę. Projekt został zaimplementowany na platformę Microsoft .NET Framework 1.1 w języku C#.

# Spis treści

<b>1</b>	<b>Wprowadzenie</b>	<b>2</b>
1.1	Wykorzystane narzędzia . . . . .	2
1.2	Słownik . . . . .	2
1.3	Wykaz funkcji matematycznych . . . . .	3
1.4	Autorzy . . . . .	3
<b>2</b>	<b>Geometria w Firace</b>	<b>4</b>
<b>3</b>	<b>Najważniejsze struktury danych</b>	<b>5</b>
3.1	Board . . . . .	5
3.2	FiraceDriver oraz IFiraceRacer . . . . .	5
3.3	Path . . . . .	5
3.4	Track . . . . .	5
<b>4</b>	<b>Stosowane algorytmy</b>	<b>7</b>
4.1	Interpretacja świata . . . . .	7
4.2	Wyznaczanie trasy początkowej . . . . .	7
4.3	Optymalizacja trasy przez symulowane wyżarzanie . . . . .	8
4.4	Sterowanie robotami . . . . .	8
4.4.1	LF . . . . .	9
4.4.2	LFMijak . . . . .	9
<b>5</b>	<b>Opis interfejsu Użytkownika</b>	<b>10</b>
5.1	Graficzny interfejs użytkownika . . . . .	10
5.2	Plik konfiguracyjny . . . . .	10
5.3	Opis świata . . . . .	11
5.3.1	Struktura dokumentu . . . . .	11
5.3.2	Przykładowy dokument . . . . .	11

# Rozdział 1

## Wprowadzenie

### 1.1 Wykorzystane narzędzia

Podczas prac nad projektem wykorzystano wiele narzędzi:

**AnkhSVN** – klient SVN (plugin do MS Visual Studio)

**Kile** – IDE dla języka  $\text{\LaTeX}$ , w którym stworzono niniejszy dokument

**Microsoft Visual Studio 2003 Professional** – IDE, w którym powstawała implementacja projektu. Wykorzystano kopie udostępnione nam na licencji MSDNAA

**NDoc** – narzędzie do generowania dokumentacji API na podstawie wygenerowanej dokumentacji kodu w formacie XML

**NUnit** – framework do przeprowadzania testów jednostkowych

**TortoiseSVN** – klient SVN

### 1.2 Słownik

Na potrzeby czytelności dokumentu definiuje się następujące pojęcia:

**autorzy** – autorzy projektu, patrz: 1.4

**bolid** – patrz: robot

**Firace** – program sterujący robotami wyścigowymi, będący przedmiotem niniejszej dokumentacji i napisany przez *autorów*

**mapa** – bitmapa przedstawiająca świat widziany przez zawieszoną nad nim kamerę bez biorących udział w wyścigu bolidów

**opis wyścigu** – dokument XML przechowujący informacje o wyścigu: mapę i listę punktów kontrolnych

**plansza** – reprezentacja mapy w programie *Firace*

**program** – patrz: *Firace*

**projekt** – aplikacja *Firace* wraz z dokumentacją: niniejszym dokumentem i dokumentacją kodu

**punkt kontrolny** (check point) - punkt, w pobliżu którego przebiega trasa wyścigu

**robot** – robot zgodny ze specyfikacją FIRA biorący udział w wyścigu

**ścieżka** – zdyskretyzowana trasa, ciąg punktów (elementów), które kolejno powinien osiągać bolid; w każdym punkcie zaczepiony jest wektor określający optymalną prędkość bolidu

**świat** – istniejąca rzeczywistość bądź w symulatorze RoBOSS płaszczyzna wraz z leżącymi na niej przeszkodami oraz bolidami, na której przeprowadzany jest wyścig

**trasa** – krzywa, po której powinien przejechać robot

### 1.3 Wykaz funkcji matematycznych

Na potrzeby niniejszego dokumentu definiujemy następujące funkcje matematyczne:

1.  $d(A, B)$  – odległość pomiędzy punktami  $A$  i  $B$
2.  $crossable(A, B)$  – prawda, jeżeli punkty  $A$  i  $B$  leżą na planszy, a odcinek ograniczony nimi nie przechodzi przez obszary nieprzejezdne, w przeciwnym wypadku fałsz

### 1.4 Autorzy

Autorami projektu są dwaj studenci dzienni informatyki na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki, na Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie, rocznik 2004:

- Michał Sroka – [msmisiu@gmail.com](mailto:msmisiu@gmail.com)
- Sebastian Skalacki – [sebastian.skalacki@gmail.com](mailto:sebastian.skalacki@gmail.com)

Projekt został zrealizowany pod opieką Panów:

- dr inż. Wojciech Zaborowski
- mgr inż. Wojciech Turek

## Rozdział 2

# Geometria w Firace

Na potrzeby projektu zaimplementowano klasy i struktury realizujące geometrię:

**Coordinates** (struktura) – punkt w przestrzeni dwuwymiarowej

**Geometry** – dostarcza statycznych funkcji realizujących najważniejsze operacje geometryczne

**Vector** (struktura) – wektor

Szczegółowe informacje na temat tych klas i struktur znajdują się w dokumentacji API.

## Rozdział 3

# Najważniejsze struktury danych

### 3.1 Board

Board jest implementacją planszy. Jest to dwuwymiarowa tablica prostokątnych obszarów, które są uznawane za przeszkody, przejezdne lub nieprzejezdne (położone zbyt blisko przeszkody), zawiera również listę punktów kontrolnych. Board wypełniany jest danymi na podstawie opisu wyścigu. Singleton.

### 3.2 FiraceDriver oraz IFiraceRacer

FiraceDriver opakowuje FIRADriver. Jest wysokopoziomowym sterownikiem do komunikacji z systemem *RoBOSS*. IFiraceRacer steruje pojedynczym robotem.

FiraceDriver odpowiada za ustanawianie i zamykanie połączenia z RoBOSem oraz możliwie częstą wymianę informacji z nim. Oprócz tego zarządza sterownikami robotów (IFiraceRacer): kontroluje kojarzenie ich z konkretnym bolidem, synchronizuje ich pracę między sobą i sterownikiem FIRADriver.

### 3.3 Path

Path realizuje ścieżkę, czyli zdyskretyzowaną postać trasy. Path jest wektorem PathElementów — par punktów, które powinien odwiedzić bolid oraz wektorów optymalnych prędkości w tych punktach. Path jest wykorzystywany przez sterowniki robotów, jako że jest wygodniejszy od Tracka.

### 3.4 Track

Track reprezentuje trasę, po której ma jechać bolid. Track jest generowany przez algorytm znajdowania trasy początkowej (zobacz 4.2). Ze względu na swój analityczny charakter jest wykorzystywany przez algorytm optymalizujący trasę (zobacz 4.3). Track jest podstawą do tworzenia Patha.

Track jest określany przez ciąg TrackPointów  $(t_0, t_1, \dots, t_n)$ . Każdy Track-Point ma przypisane położenie  $P_k$  i promień skrętu bolidu  $r_k$ . Track jest krzywą określoną w następujący sposób:

$$T = \left( \bigcup_{k=0}^{n-1} L_k \right) \cup \left( \bigcup_{k=1}^{n-1} A_k \right) \quad (3.1)$$

gdzie  $A_k$  jest najkrótszym istniejącym łukiem o promieniu  $r_k$  oraz stycznym do  $\overline{P_k P_{k-1}}$  i  $\overline{P_k P_{k+1}}$ , zaś  $L_k$  jest odcinkiem leżącym na prostej  $\overline{P_k P_{k+1}}$  zawartym pomiędzy:  $P_0$  i punktem styczności  $A_1$  z prostą  $\overline{P_0 P_1}$  dla  $k = 0$ ,  $P_n$  i punktem styczności  $A_{n-1}$  z prostą  $\overline{P_n P_{n-1}}$  dla  $k = n$ ,  $P_0$  lub punktami styczności łuków  $A_k$  i  $A_{k+1}$  z prostą  $\overline{P_k P_{k+1}}$  dla  $k = 0$ .

## Rozdział 4

# Stosowane algorytmy

### 4.1 Interpretacja świata

Program otrzymuje dokument XML zawierający opis świata (patrz 5.3). Na jego podstawie budowana jest lista punktów kontrolnych oraz dwuwymiarowa tablica zmiennych boolowskich reprezentująca przejezdne i nieprzejezdne obszary planszy. Jednemu pikselowi fotografii świata odpowiada jeden obszar planszy. Obszar  $O$  jest nieprzejezdny wtedy i tylko wtedy, gdy istnieje obszar  $O'$  taki, że piksel odpowiadający  $O'$  na fotografii świata jest w jasnym kolorze<sup>1</sup> oraz odległość pomiędzy  $O$  i  $O'$  nie jest większa niż `SafeDistance`.

### 4.2 Wyznaczanie trasy początkowej

Algorytm złożony jest z trzech kroków:

1. Wyznaczamy trasę początkową zmodyfikowanym algorytmem  $A^*$ <sup>2</sup>. W odróżnieniu od oryginału, graf przeszukiwany jest tylko w czterech, a nie ośmiu kierunkach — połączenia są tylko pomiędzy obszarami posiadającymi wspólny bok. Ponadto w tym kroku algorytmu za nieprzejezdne uznawane są obszary oddalone od przeszkód o nie więcej niż `SafeDistance + InitialTrackDistanceFromObstacle`<sup>3</sup>. Doświadczenie pokazało, że takie podejście owocuje mniejszą liczbą `TrackPoint`ów w otrzymanej na końcu trasie.
2. Jeżeli odcinek łączący dwa `TrackPoint`y  $A$  i  $B$  nie przechodzi przez żaden nieprzejezdny obszar, usuwamy wszystkie `TrackPoint`y leżące pomiędzy  $A$  i  $B$ .
3. Jeżeli dwa punkty leżą bardzo blisko siebie (są oddalone od siebie o co najwyżej 2 pixele), usuwamy jeden z nich.

---

<sup>1</sup>Poziom jasności piksela, jaki jest wymagany, by obszar został uznany za przeszkodę, określany jest w pliku konfiguracyjnym

<sup>2</sup>[http://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm)

<sup>3</sup>Obie wartości są specyfikowane w pliku konfiguracyjnym

### 4.3 Optymalizacja trasy przez symulowane wyżarzanie

W celu zoptymalizowania trasy bolidu zaimplementowano algorytm symulowanego wyżarzania<sup>4</sup>. Prawdopodobieństwo przejścia z konfiguracji (trasy) A na B wynosi:

$$p = \begin{cases} e^{-\frac{E(A)-E(B)}{kT}} & E(B) < E(A) \\ 1 & E(B) \geq E(A) \end{cases} \quad (4.1)$$

gdzie  $T$  oznacza aktualną temperaturę układu, a  $k$  stałą Boltzmanna. Po każdej iteracji temperatura obniża się według następującego wzoru:

$$T_{i+1} = \frac{T_i}{c} \quad (4.2)$$

gdzie  $c$  jest współczynnikiem chłodzenia (*cooling factor*), który powinien być nieznacznie większy od 1.

Trasa jest optymalizowana pod względem promieni zakrętów. Funkcja energii wynosi:

$$E(T) = \prod_{k=1}^n \frac{1}{T_k} \quad (4.3)$$

gdzie  $T_k$  oznacza promień  $k$ -tego zakrętu trasy  $T$ .

Funkcja poszukująca nowej konfiguracji przeprowadza eliminację zbędnych TrackPointów. Niech  $K$  oznacza największy istniejący zbiór trackpointów trasy  $T = (p_1, p_2, p_3, p_4, \dots, p_n)$  takich, że usunięcie dowolnego z nich nie naruszy poprawności trasy. Jeżeli  $K \neq \Phi$ , z trasy usuwany jest dowolny z punktów ze zbioru  $K$ .

Eliminacja zbędnych TrackPointów jest przezroczysta dla samego wyżarzania. Modyfikowana jest trasa przekazana do funkcji jako argument. Upraszcza to znacznie funkcję energii, która nie musi zależeć od ilości zakrętów, a jedynie ich jakości.

### 4.4 Sterowanie robotami

Niniejsza sekcja prezentuje proponowane przez nas algorytmy sterujące robotami. Architektura prezentowanego przez nas rozwiązania pozwala na łatwe dodawanie kolejnych algorytmów oraz sterowanie różnych robotów różnymi algorytmami w jednym wyścigu. Architektura wymaga, aby algorytmy sterujące miały charakter iteracyjny — każda iteracja ma miejsce pomiędzy dwoma kolejnymi krokami FIRADrivera. Kroki te są dokonywane „tak często, jak to tylko możliwe”. W celu stworzenia własnego sterownika robota należy zaimplementować interfejs IFiraceRacer (zobacz 3.2).

<sup>4</sup>[http://en.wikipedia.org/wiki/Simulated\\_annealing](http://en.wikipedia.org/wiki/Simulated_annealing)  
[http://www.gnu.org/software/gsl/manual/html\\_node/Simulated-Annealing-algorithm.html](http://www.gnu.org/software/gsl/manual/html_node/Simulated-Annealing-algorithm.html)

#### 4.4.1 LF

Jeżeli pierwszy element ścieżki robota znajduje się odpowiednio blisko, jest on usuwany.

Niech  $R$  oznacza obecne położenie robota,  $A$  i  $B$  zaś dwa pierwsze elementy ścieżki robota. Algorytm najpierw określa punkt  $D$ , na który ma się kierować robot:

$$K = \begin{cases} \Phi & \text{jeżeli } B \text{ nie istnieje} \\ \{P : d(P, R) = l \wedge P \in \overline{AB}\} & \text{w przeciwnym wypadku} \end{cases} \quad (4.4)$$

$$D = \begin{cases} A & \text{jeżeli } K = \Phi \\ \delta \in K : \inf_{k \in K} d(R, k) = d(R, \delta) & \text{w przeciwnym wypadku} \end{cases} \quad (4.5)$$

gdzie  $l$  jest stałą określoną w algorytmie i wyznaczoną doświadczalnie.

Następnie wyliczane są prędkości, jakie należy podać na koła. Tu również  $a$  i  $b$  są wyznaczone doświadczalnie:

$$v_{wewn} = \frac{2\Pi}{a} d(D, R) \quad (4.6)$$

$$v_{zewn} = \frac{2\Pi + \psi}{a} (d(D, R) + b) \quad (4.7)$$

natomiast  $\psi$  oznacza kąt pomiędzy wektorem prędkości robota, a prostą przechodzącą przez punkty  $R$  i  $D$ .

#### 4.4.2 LFMijak

Algorytm ten bazuje na LF (zobacz 4.4.1). W odróżnieniu od niego próbuje omijać inne bolidy.

Niech  $r$  oznacza pozycję robota sterowanego przez algorytm,  $R$  zaś zbiór pozycji wszystkich pozostałych robotów. Wówczas  $P$  oznacza zbiór pozycji robotów, które należy ominąć:

$$P = \{\rho \in R : d(r, \rho) < l \wedge \text{crossable}(r, \rho)\} \quad (4.8)$$

gdzie  $l$  jest stałą określoną w algorytmie. Jeżeli  $P = \Phi$  to nie zachodzi potrzeba omijania i robot sterowany jest według algorytmu LF. W obecnej wersji algorytm LFMijak omija tylko najbliższego robota:

$$p = \pi \in P : \inf_{\rho \in P} d(r, \rho) = d(r, \pi) \quad (4.9)$$

Wyznaczane są proste  $a$  i  $b$  takie, że:

$$r \in a \quad (4.10)$$

$$b \ni a \in b \quad (4.11)$$

$$b \perp a \quad (4.12)$$

Następnie robot kieruje się na punkt  $t$  taki, że:

$$T = \{\theta \in b : d(\theta, p) = m \wedge \text{crossable}(r, \theta)\} \quad (4.13)$$

$$d(t, N) = \inf_{\theta \in T} d(\theta, N) \quad (4.14)$$

gdzie  $m$  jest stałą określoną w algorytmie,  $N$  zaś pierwszym punktem ścieżki sterowanego robota. Prędkości na kołach wyznaczane są wg wzorów 4.6 i 4.7.

## Rozdział 5

# Opis interfejsu Użytkownika

### 5.1 Graficzny interfejs użytkownika

Główne okno programu podzielone jest na dwie części. Dolna to pole statusu, na którym wyświetlane są komunikaty związane z pracą Firace. Górna, podzielona na zakładki, umożliwia dostęp do funkcjonalności aplikacji. Autorzy wierzą w inteligencję oraz doświadczenie użytkowników, powstrzymują się zatem od rozwlekłego opisywania obsługi aplikacji. Nadmieniam jedynie, że poszukiwanie trasy początkowej, wyżarzanie oraz sterowanie robotami uruchamiane są w osobnych wątkach, aby zapobiec utracie interakcyjności interfejsu i dostarczyć lepszą kontrolę nad tymi procesami.

### 5.2 Plik konfiguracyjny

Plik konfiguracyjny w formacie XML szczegółowo parametryzuje działanie programu.

**SafeDistance** – odległość w metrach, o jaką zostaną poszerzone przeszkody w planszy

**StepInterval** – aktualnie nie używane

**RacerAcceleration** – maksymalne wypadkowe przyśpieszenie bolidu, używane przy ocenie poprawności trasy

**MaxPathElementsDistanceOnStraight, MaxPathElementsDistanceOnArc** – określa największą dopuszczalną odległość pomiędzy elementami ścieżki odpowiednio na prostych i łukach

**InitialTrackDistanceFromObstacle** – stała używana przy znajdowaniu trasy początkowej (patrz 4.2)

**WheelsDistance** – rozstaw kół bolidu

## 5.3 Opis świata

Program interpretuje świat na podstawie dostarczonego dokumentu XML. Dokument ten zawiera ścieżkę do bitmapy będącej fotografią świata, informacje na temat skali bitmapy oraz położenia początku układu współrzędnych na niej, a także listę punktów kontrolnych.

### 5.3.1 Struktura dokumentu

Korzeniem dokumentu jest element `RaceDefinition`, w nim znajduje się element `Track` posiadający dwa atrybuty: `name` zawierający nazwę trasy oraz `loop` zawierający wartość logiczną, w chwili obecnej żaden z nich nie jest używany.

Pierwszym dzieckiem elementu `Track` powinien być element `ObstacleBitmap`, precyzujący mapę świata. W atrybucie `filename` znajduje się nazwa pliku z bitmapą. Atrybuty `pixel_0_x` i `pixel_0_y` precyzują, w którym pixelu zaczepiony jest początek układu współrzędnych. Atrybuty `size_x` i `size_y` precyzują rozmiar świata w metrach.

Kolejnymi dziećmi elementu `Track` są elementy `Checkpoint` określające kolejne punkty kontrolne. Atrybutami są `id`, określający kolejny numer punktu kontrolnego, `x` i `y`, precyzujące jego współrzędne od początku układu współrzędnych oraz `radius`, aktualnie nie używany.

### 5.3.2 Przykładowy dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<RaceDefinition>
  <Track name="the track 01" loop="true">
    <ObstacleBitmap filename="track_01.bmp" pixel_0_x="359"
      pixel_0_y="341" size_x="5.3" size_y="5.1" />
    <Checkpoint id="1" x="2.0" y="-0.2" radius="0.35" />
    <Checkpoint id="2" x="-0.40" y="1.0" radius="0.3" />
    <Checkpoint id="3" x="-2.3" y="-1.0" radius="0.25" />
    <Checkpoint id="4" x="0.3" y="-1.55" radius="0.3" />
  </Track>
</RaceDefinition>
```

Powyższy dokument definiuje wyścig przebiegający przez punkty kontrolne o współrzędnych  $(2; -0, 2)$ ,  $(-0, 4; 1)$ ,  $(-2, 3; -1)$ ,  $(0, 3; -1, 55)$ . Mapa wyścigu znajduje się w pliku `track_01.bmp`. Świat jest prostokątem o wymiarach  $5,3 \times 5,1$ , punkt  $(0, 0)$  zaczepiamy w pixelu  $(359, 341)$ .