Algorithm Description of the Artificial Intelligence for the Game SolarFighter

Adrian Posor

August 8, 2010

This document describes the algorithm that controls the enemy ships in the new 2D shooter game *SolarFighter* created by *Neo-Digital*. It serves as a specification for the implementation.

Copyright © 2009 - 2010 Adrian Posor. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

1 Introduction

SolarFighter is a 2D shooter game created by *Neo-Digital*. In contrast to typical shoot 'em ups, the protagonist may rotate and move in any direction and the enemies the player faces in this game do not move in predefined attack patterns, but are controlled by the algorithm described in this document. The game features a multiplayer mode, which additionally sets this game apart from its competitors.

2 Requirements

Every ship that falls under the control of the computer shall have his own instance of the AI. There shall be no AI that controls all ships together. Every instance of the AI shall have a list of enemies that reflects all the entities the AI is going to fight against. The AI is expected to support "visual thinking", which means it shall be able to visualize its calculations and decisions.
The AI algorithm shall have the following inputs:

- the position and direction of movement of all relevant objects
- a list of enemies

The AI algorithm shall have the following outputs:

- velocity and direction of movement of the controlled ship

3 Description

The AI shall support the following modes of operation:

Idle: The AI controlled ship flies around, does not shoot, and avoids collisions with objects.

Attack: The AI controlled ship attacks all its enemies by moving towards them and shooting.

Flee: The AI controlled ship increases the distance to its enemies.

Algorithm 1 Determine mode of operation

```
if currentMode = idle then
    if ship is hit then
        currentMode ← attack
    end if
else if currentMode = attack then
    if lifeEnergy ≤ threshold then
        currentMode ← flee
    end if
else if currentMode = flee then
    if no enemy within defined range then
        currentMode ← idle
    end if
end if
```

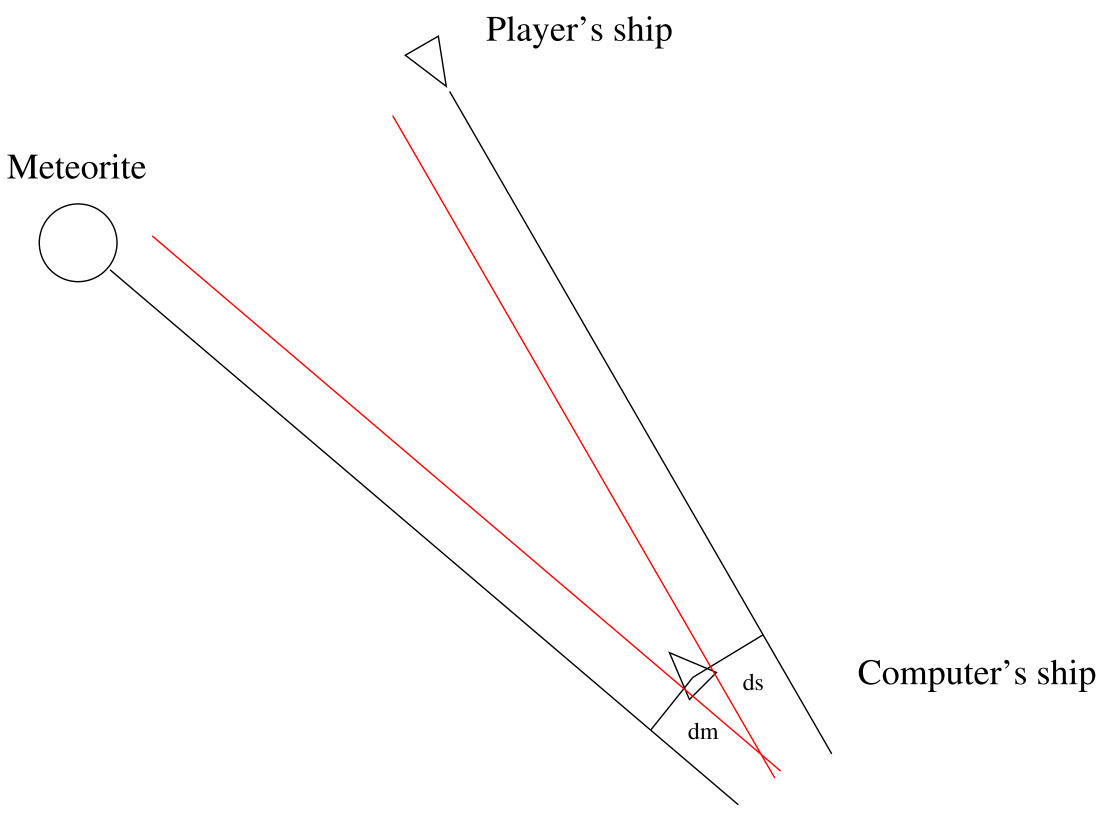

Figure 1: Distance between ship and other objects

Algorithm 2 Calculate new desired coordinates for ship

Ensure: p is a safe position for the ship
for all objects on playing field **do**
$d_n \leftarrow$ distance between ship and object n
end for
for all objects with $d_n \leq threshold_1$ **do**
$trajectory_n \leftarrow$ trajectory of object
$d_{trajectory_n} \leftarrow$ distance between ship and trajectory of object n
end for
if number of elements in $d_{trajectory} = 0$ **then**
return current target position
end if
$d_{min} \leftarrow d_{trajectory_0}$
$trajectory_{min} = trajectory_0$
for all $d_{trajectory_n} \leq threshold_2$ **do**
if $(d_{trajectory} \leq threshold_2) \wedge (d_{min} > d_{trajectory_n})$ **then**
$d_{min} \leftarrow d_{trajectory_n}$
$trajectory_{min} \leftarrow trajectory_n$
end if
end for
if $d_{min} > delta$ **then**
return current target position
end if
$perp \leftarrow$ perpendicular to the $trajectory_{min}$ through the position of the ship
$p \leftarrow$ foot of $perp$
$done \leftarrow$ **false**
$dir \leftarrow 0$
while $\neg done$ **do**
$(q_1, q_2) \leftarrow$ points on the perpendicular with $distance(q, position_{ship}) = distance(p, position_{ship}) + delta$
if $dir = 1$ **then**
$q \leftarrow q_1$
else if $dir = 2$ **then**
$q \leftarrow q_2$
else
if $distance(q1, position_{ship}) < distance(q2, position_{ship})$ **then**
$dir \leftarrow 1$
$q \leftarrow q_1$
else
$dir \leftarrow 2$
$q \leftarrow q_2$
end if
end if
$done \leftarrow$ **true**
for all t in $trajectory$ **do**
if $\neg(t \parallel perp) \wedge distance(t, q) < threshold_2$ **then**
$done \leftarrow$ **false**
end if
end for
$p \leftarrow q$
end while
return p

Algorithm 3 Calculate distance between ship and object

Require: $p = (x_p, y_p)$ and $q = (x_q, y_q)$
Ensure: d = distance between p and q
  $\Delta x \leftarrow |x_p - x_q|$
  $\Delta y \leftarrow |y_p - y_q|$
  $d \leftarrow \sqrt{\Delta x^2 + \Delta y^2}$
  return d

Algorithm 4 Calculate distance between ship (point) and trajectory of object (line) (HNF)

Require: $\vec{p} = (x_p, y_p)$ and trajectory in HNF $\vec{r} \cdot \vec{n} - c = 0$
  $d \leftarrow |\vec{n} \cdot \vec{p} - c|$
  return d

Algorithm 5 Calculate distance between ship (point) and trajectory of object (line) (parameter form)

Require: $\vec{p} = (x_p, y_p)$ and trajectory in parameter form $\vec{x} = \vec{a} + t \cdot \vec{b}$
  $t_0 \leftarrow \frac{(\vec{p}-\vec{a})\cdot\vec{b}}{\vec{b}^2}$
  $\vec{x}_0 \leftarrow \vec{a} + t_0 \cdot \vec{b}$
  return $d \leftarrow |\vec{x}_0 - \vec{p}|$

Algorithm 6 Calculate trajectory of object (point and angle)

Require: position $\vec{p} = (x_p, y_p)$ of object and angle α
Ensure: parameter form $\vec{x} = \vec{a} + t \cdot \vec{b}, t \in \mathbf{R}$
  $\vec{a} \leftarrow \vec{p}$
  $\vec{b} \leftarrow (cos(\alpha), sin(\alpha))$
  return $\vec{a}, \vec{b}$

Algorithm 7 Calculate trajectory of object (two points)

Require: two points $\vec{p}_1, \vec{p}_2$ on trajectory
Ensure: parameter form $\vec{x} = \vec{a} + t \cdot \vec{b}, t \in \mathbf{R}$
  $\vec{a} \leftarrow \vec{p}_1$
  $\vec{b} \leftarrow \vec{p}_2 - \vec{p}_1$
  return $\vec{a}, \vec{b}$

Algorithm 8 Calculate perpendicular of trajectory through position of ship

Require: trajectory in parameter form $\vec{x}_{tr} = \vec{a} + t \cdot \vec{b}, t \in \mathbf{R}$ and position p of ship
Ensure: perpendicular in parameter form $\vec{x}_p = \vec{a}_p + t \cdot \vec{b}_p, t \in \mathbf{R}$

$t_0 \leftarrow \frac{(\vec{p}-\vec{a})\cdot\vec{b}}{\vec{b}^2}$
$x_0 \leftarrow \vec{a} + t_0 \cdot \vec{b}$
$\vec{a}_p \leftarrow \vec{p}$
$\vec{b}_p \leftarrow \vec{x}_0 - \vec{p}$
return $\vec{a}_p, \vec{b}_p$

Algorithm 9 Calculate foot of perpendicular

Require: trajectory in parameter form $\vec{x}_{tr} = \vec{a} + t \cdot \vec{b}, t \in \mathbf{R}$ and position p of ship
Ensure: $\vec{x}_0$ is foot of perpendicular

$t_0 \leftarrow \frac{(\vec{p}-\vec{a})\cdot\vec{b}}{\vec{b}^2}$
$\vec{x}_0 \leftarrow \vec{a} + t_0 \cdot \vec{b}$
return $\vec{x}_0$

Algorithm 10 Check if two lines are orthogonal

Require: two lines in parameter form $\vec{x} = \vec{a} + t \cdot \vec{b}$
return $\vec{b_1} \cdot \vec{b_2} < 0.0001$

Algorithm 11 Check if two lines are parallel

Require: two lines in parameter form $\vec{x} = \vec{a} + t \cdot \vec{b}$
return $\vec{b_1}^{\perp} \cdot \vec{b_2} < 0.0001$