Library on crowd behavior for agents treatment on videogames

Biblioteca sobre el comportamiento de las masas para el tratamiento de los agentes a los videojuegos

M. Proenza and Y. Proenza

Keywords:
Artificial Intelligence, agent, Command Pattern, Steering Behaviors.

Abstract

This paper covers the investigation process and further design and implementation of a library on crowd behaviors for videogames. Three methods were studied in order to pick up one of them, resulting to be Steering Behaviors method the one selected to be adopted. The library supports simulations with a large number of moving agents. It also copes with polygonal obstacles avoidance. Additionally, simulations can be adapted to many contexts by tuning some parameters. The library consists of an original design mostly based on Command Pattern that allows it to be easily usable and extendable.

Palabras clave:
Inteligencia Artificial, agente, modelo de comandos, comportamientos de dirección.

Resumen

Este documento cubre el proceso de investigación y el diseño e implementación de una biblioteca en los comportamientos de multitud de videojuegos. Tres métodos fueron estudiados con el fin de recoger a uno de ellos, el resultado es el método de dirección Comportamientos que se ha seleccionado para su adopción. La biblioteca admite simulaciones con un gran número de agentes móviles. También hace frente a la evitación obstáculos poligonales. Además, las simulaciones se pueden adaptar a diferentes contextos mediante la regulación de algunos parámetros. La biblioteca consta de un diseño original basado principalmente en modelo de comandos que permite que sea fácilmente utilizable y extensible.

I. INTRODUCTION

Some parts of the real world are crowded, busy places, full of people, animals and objects, which are all interacting with each other, moving around in opposite directions. When taking an exhaustive look to Nature, people can contemplate many amazing behaviors performed by flocks of birds, or schools of fish and herds of land animals, which seem to be extremely complex, or randomly arranged, but still beautiful. Surprisingly, some researches reveal that these behaviors conform to more specific rules, which combined can generate very complex results.

One area of interest in computer sciences has always been trying to capture these kinds of movements into some sort of formulas and simulate them. Computer games developers have been interested in this for decades, since they have always tried to achieve a more accurate simulation of reality. In particular, Artificial Intelligence (AI) researchers have been leading this field, coming up with new ideas every day, making games move faster towards more realistic approximations. Developers are investigating and testing new AI technologies in order to be able to build better and smarter games.

When dealing with movement in games, developers often refer to objects that have the ability to move and react to environmental changes, as units, or non-player character (NPC), or agents. From now on, the last approximation is the one that will be adopted by this research to refer to a single object in a group, which is able to somehow be aware of its surrounding environment and take an action (move) according to it.

Lots of games have incorporated some techniques on group behaviors to bring their inner world to life, making them more exotic. The success of these games has been determined in some occasions by the inclusion of a relatively advanced AI, and today it is one of the most discussed topics when trying to take a game into the market and make it sell.

The leadership of computer games industry belongs to the most developed countries and the richest corporations. The Cuban game industry is attempting to get included among the most successful ones in the world, in order to cope with the necessity of exporting software products. The *Universidad de las Ciencias Informáticas (UCI)* is a leading entity in Cuban game industry. Game developers from this university realized that, generally, games require some kind of AI, which is also demanded by nowadays market, which expects for the games to be more realistic and the characters to be more "intelligent" each time. It is needed to include AI in the game products. In spite of that, this goal hasn't been accomplished, most of the time because of the lack of AI resources, resulting in poorly realistic games. Besides, it is needed to speed up development time significantly, especially in the products to come, where AI will play an important role in their acceptance, but the ground is not settled yet. It is also very difficult to incorporate AI to computer games, and this is why most of the products delay their release and deployment and projects get stuck.

Crowd behavior is one part of AI that has been spread and used in many classic games, such as *Star Craft* and *Age of Empires*, rising with impressive effects and realism, what makes these games very attractive. In the UCI, no solution of this kind have been achieved, a fancy one that captures the users. Despite this, some attempts have been made, for example, the implementation of a set of algorithms in a module, which introduced similar techniques, but with no great results because of their simplicity [1].

The main objective to achieve in this research is the implementation of a library for applying realistic crowd behavior to computer games, hoping it can be used by every game developer at the UCI. So, it is expected to have a library to relay in for applying crowd behavior in games, resulting in reusable source code, less implementation time and more realistic and attracting game worlds.

II. METHODOLOGY

For a better understanding of what the research should be led to and for having a way of discerning among the studied methods and algorithms for its adoption, it was stated "major goals" to achieve. These major goals have been divided into two categories: Agent-Level Goals and Library-Level Goals.

They mostly refer to answers to the following questions: What the agents should behave like? How reactive should they be? How efficient must the algorithms be? How real the simulation should be performed? How extendable should the library be?

Agent-Level Goals:

- Agents' behavior should look as realistic as possible
- Agents should move within a continuous (not discrete) space.
- Agents should be able to react to both, internal logic (concerning to environment state, i.e. obstacles, other agents) and external logic (user/player control signals).
- Agents should be capable to avoid collision with static polygonal obstacles.
- Agents should be capable to act like a single unit (not always in a group) and achieve "personal missions".
- Agents should demonstrate a coherent (not chaotic) behavior.
- When in a group, agents should look like having a common goal (army effect).
- Since this is not a problem of path finding, agents aren't meant to find the best (shortest) path to its goal, but a safe one.

Library-Level Goals

- Algorithms and methods should be efficient enough so that games' frame rate is acceptable when using the library.
- The library should allow enhancements for future extensions in games.
- The library should be fast to create (implementation time of about two months).
- The library should be adaptable to new simulation requirements.
- The library must be able to handle 2D movement.

The investigation process explored some concepts and methods for dealing with group behaviors. It was studied how each of these methods can handle groups of agents and were described their limitations, in order to end up picking one of them.

Three methods were evaluated:

- Potential Function Based Movement
- Cellular Automata
- Steering Behaviors

Table 1 summarizes the comparison among the three methods according to six selected hints.

At first, these three methods were narrowed down to two candidates: Potential Field Based Movement and Steering Behaviors.

While Potential Field Based Movement might have been useful for achieving most of the "major goals" dictated previously (see Introduction), it was the high capability for extensions that Steering Behaviors method has, the deciding factor leading to choose it as the method to be adopted in the implementation.

These are the adopted specifications that guided the design and implementation of the library:

- Use Steering Behaviors Method to model the agents' movement.
- Find a method that uses information about objects in the world to keep agents aware of its surrounding environment.
- The method should be robust enough to support a large number of agents.
- Find a design where Locomotion and Steering layers are decoupled.
- Use a 2D approach.

The solution proposal is focused on finding a suitable design for the library, so that it was easily extendable. On the other hand, requisites were not clear at first. They were appearing one by one and at the same time added into the library. The code was refactored after the inclusion of functionalities, so that the current version of the library was stabilized all the time. The programming language was C++ and the project was developed using Visual Studio 2008 Team System.

III. OUTCOMES AND DISCUSSION

The design of the library is basically centered on *Command Pattern*. This design pattern allows you to decouple the requester of an action from the actual

Hints / Methods	Level of Realism	Possibility of Avoiding Polygonal Obstacles	User Control	Algorithms Efficiency	Possibility of Extensions	Ease of implementation
Potential Function Based Movement	High with circular obstacles	Yes, but with bad approximations	High	Medium. Can be accelerated	Yes, but with the constraint that the force must point along the line connecting two objects	Very easy
Cellular Automata	Medium	Yes. Requires finding a function	Low	High	Yes, but need also to extend the transition functions	Easy
Steering Behaviors	High. Requires tweaking	Yes	High	Medium. Can be accelerated	Yes	Easy

object that performs the action. The library takes advantage of that and lets some objects communicate for an effective interaction between the agents and the environment. The specifications for this pattern can be found at [2].

In the library, the implementation of the *Command Pattern* has assumed some modifications and extensions (although its essence remains the same), and has been approached the following way (see Figure 1).

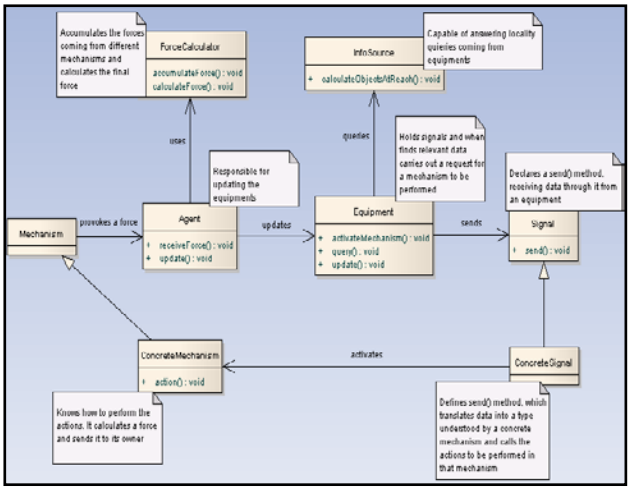

Fig. 1 Report of relevant data and agents' reaction

This figure shows only a reduced part of the classes in the library, but which are considered to be the top level ones.

TABLA I
Comparison among studied methods

The sequence of interactions among these objects is related bellow:

1. The agent gets equipped with all the equipments.
2. The agent constantly updates each equipment it has been equipped with.
3. In each update time, the equipments query the world for information (relevant data).
4. If some relevant data is found, the equipment unchains a mechanism in the agent, which means send a signal (or more than one) with the information collected.
5. The signal contains the knowledge to activate the proper mechanism for the agent to react to the particular information that has been transmitted through it, so, it firstly converts this information so that the mechanism understands it and then activates the mechanism.
6. The mechanism calculates a force to be applied to the agent accordingly and sends it to the agent.
7. The agent accumulates the force received from each mechanism to apply it to itself.
8. The agent responds to the accumulated force.

The mechanisms currently supported in the library

are the following:

- Flocking

The equipment used here is called *X-Circular View Range*, because it appeared useful to model the agents' vision to cope with flocking. Figure 2 shows it graphically.

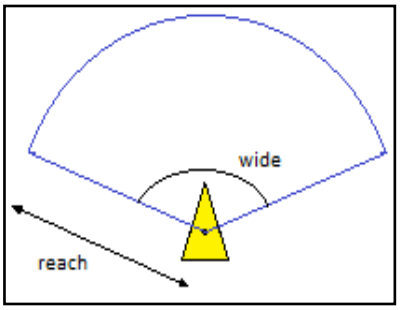

Fig. 2 X-Circular View Range

Agents just flock with those within their view ranges, following three simple rules described in [3], which are: *Separation*, *Alignment* and *Cohesion*.

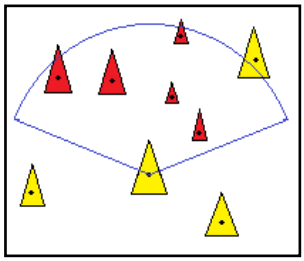

Fig. 3 An agent flocking with others: Separation, Alignment and Cohesion

- Obstacle Avoidance

To model the *Obstacle Avoidance* it was used an equipment named *Box Obstacle Sensor.* A box is a very convenient shape since it can totally wrap the agents so that it can protect them better and keep them safe from bumping into obstacles.

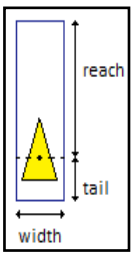

Fig. 4 Box Obstacle Sensor

To detect a collision, three segments are defined in the box, so that obstacle avoidance is reduced to segment-segment interception with the sides of the obstacles.

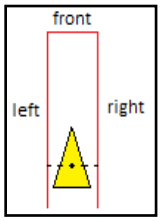

Fig. 5 Interception segments

The avoidance is performed directing the agent towards a direction under a user defined angle according to the normal to the collision point.

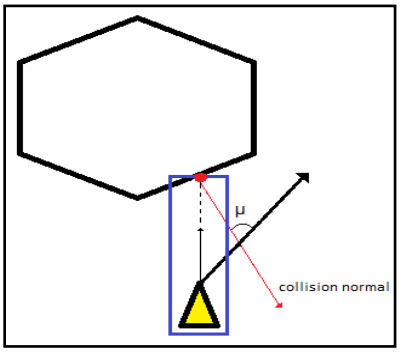

Fig. 6 Obstacle Avoidance

- Avoid Individual

Avoid Individual mechanism uses the same equipment as *Flocking*: *X-Circular View Range*.

Taking evasive action has been approached a way similar to that described in [4]. After having detected the closest approach, if its distance is bellow a distance toleration, the agent deviates itself perpendicularly to its current heading.

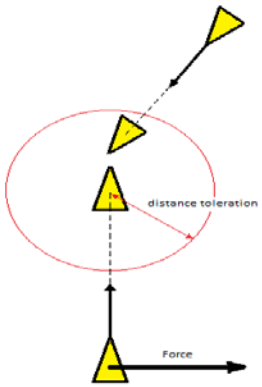

Fig. 7 Avoid Individual

The library also holds some other features:

- Neighbors' Search Acceleration

For this, a grid shaped space partition has been approached, so that complexity is reduced from O (n^2) to a complexity close to O (n). Locality queries are made to this utility in order to be aware of the objects that are a distance away from certain position.

- Movement Smoothing

A smoother utility keeps track of a number of values for the agents' heading and can be asked any time for its average value. This way and by decoupling the agent's heading from its velocity the movement is smoothed so that agents do not make sudden turnings.

- Non-Penetration Insurance

Cero overlapping is also guaranteed in the library. A non-penetration enforcer forces the agents to separate when an overlapping is found.

- Groups Formation

Groups formation is approached in an original way, by assigning category numbers to the agents. A category setter handles the generation of these numbers for the agents, so that forming groups is easily done by "playing" with categories.

The interface provided by the library is very simple to use and let the users work without regards of what happens inside. On the other hand, an important characteristic of the library is its extensibility. The library can be easily extended to cope with certain requirements in a game. This is considered to be the most important contribution of this research.

IV. CONCLUSIONS

This research and the implementation of a library on crowd behaviors for videogames arise with the following results:

- Game developers now count on a resource for applying crowd behaviors for videogames, which is very easy to use, extend and which can be shared among programmers.
- The library supports the following behaviors: Separation, Alignment, Cohesion, Polygonal Obstacle Avoidance and Individual Avoidance.

The library holds some other features, like: Neighborhood Queries Acceleration, Movement Smoothing, Cero Overlapping among Agents and an easy way of Forming Groups.

REFERENCES

[1]Falcón, R. E. "*Módulo de algortimos de locomoción con múltiples Steering Behaviors*". UCI. 2008. Undergraduate Thesis.

[2]Freeman, E., Bates, B. & Sierra, K. *Design Patterns.* [ed.] Head First. 1st. s.l.: O'Reilly, 2004.

[3]Reynolds, Craig W. "*Steering Behaviors for Autonomous Characters*". In Proceedings of SIGGRAPH'99. 1999.

[4]Green, Robin. "*Steering Behaviors*". In Proceedings of SIGGRAPH'00. 2000.

.