Neurocomputing 172 (2016) 453–458

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Designing virtual bots for optimizing strategy-game groups

Manuel G. Bedia [a], Luis Castillo [b,c,*], Carolina Lopez [a], Francisco Seron [a], Gustavo Isaza [c]

[a] Department of Computer Science, University of Zaragoza, Spain
[b] Department of Industrial Engineering , National University of Colombia (Manizales), Colombia
[c] GITIR group, Engineering Technical Faculty, University of Caldas, Colombia

ARTICLE INFO

Article history:
Received 22 September 2014
Received in revised form
28 April 2015
Accepted 5 May 2015
Available online 5 August 2015

Keywords:
Genetic algorithms
Swarming
Videogames

ABSTRACT

In the last decade, a permanent increasing in the popularity of videogames has happened. As the cost of the computational power has decreased, graphic games more realistic have been developed. However, although everybody in this industry knew that an intelligent and believable behavior of bots (or, in general, non-playing characters) is an important key to make a game fun to play, the experts have not showed real interests in this issue until recently. In this paper, we propose how a good strategy for optimizing the behavior of a team of bots (with roles between members and communication skills between each other) in the "capture the flag game" domain, could be designed and analyzed using a combination between swarming optimization techniques and mathematical analysis based in Markov models in order to improve the standard strategies that videogames use. This domain presents a particular case of the "exploration vs. exploitation" dilemma, a paradox that appears in numerous situations where systems needs being adaptable and learnable at the same time and solutions of the dilemma are built evolving balances between parts. The environment used to test the proposed model will be the Unreal Tournament virtual world.

1. Introduction

Videogames offer new environments for artificial intelligence (AI) technology and research. Over the last few years the videogame world has become a huge industry and the use of AI techniques to improve NPC behaviours have become increasingly more important to make smart and fun games [1]. Currently, it is completely assumed by the videogame research community that Artificial Intelligence is going to become the core issue in creating realistic experiences in a next generation of video and computer games. The experts in this field assume that future videogames will be characterized by showing novel situations where bots exhibit changing and unpredictable behaviors keeping the interests in the game during long time. It implies to focus on AI techniques that allow us to design a new generation of behaviors in bots. In this paper, we are interested in a game known as "Capture the Flag" that appears in numerous gaming platforms and that consists in a strategy game where players compete to capture the other team's flag and return it to their base.

In order to implement a successful strategy, competitive teams must use a great deal of team-play to generate the combination of individual behaviors that generate a solution in a whole level. From our point of view, the challenge is a particular case of the well-known "exploration vs. exploitation" dilemma – a recurrent paradox that emerge in all systems that try to get a balance between two types of incompatible behaviors. In this paper, we will propose how to design a group strategy that models the behavior of a team-play in a this videogame platform.

The paper is structured as follows: In Section 2 a brief overview is given on the Unreal Tournament platform and the Pogamut gametool, together the reasons why these platforms have been chosen for this research. In Section 3, a combination between Markov chain and genetic algorithm model will be used in order to model (i) the behavior of an individual bot, and (ii) a group strategy of a team of bots where the exploration/exploitation dilemma should be dealt. Furthermore, it will be explained the methodology used in this research: (i) how the optimization strategies will be outlined; (ii) and an in-depth description of the genetic algorithm model and how it works will be explained. In Section 4, the set of experiments that have been done with the algorithms described in the previous section and the parameter settings will be detailed, and after that, the results of the experiments will be discussed. Finally, in Section 5, the most important conclusions will be remarked.

2. Experimental settings and approaching to the problem

Unreal tournament 2004 (developed by Epic Games Inc.) is the name of a very famous First Person Shooter multi-player game

* Corresponding author.
E-mail addresses: mgbedia@unizar.es (M.G. Bedia), luis.castillo@ucaldas.edu.co (L. Castillo), c.lopez@unizar.es (C. Lopez), seron@unizar.es (F. Seron), gustavo.isaza@ucaldas.edu.co (G. Isaza).

http://dx.doi.org/10.1016/j.neucom.2015.05.118
0925-2312/

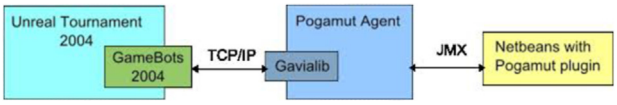

Fig. 1. Pogamut architecture (with UT2004) (source: Pogamut documentation).

where the player has to fight enemies that can be controlled by the computer (i.e. bots) or by other human players in a network structure [2]. Although the engine of Unreal tournament 2004 (UT 2004) is neither completely open nor available for researching, the advantage is that the behavior of the virtual bots (path planning algorithms, routines for moving, attacking, etc.) can be implemented through a platform known as Pogamut. Pogamut is a Java middleware that provides an API to control bots [3]. Most research groups and several experiments have been developed in academic environments using Pogamut (for instance, in Genetic Bots [4], StorySpeak [5], Emohawk [6], etc.). In Fig. 1, readers can see a diagram of the architecture of Pogamut.

In order to explore how to design intelligent strategies of a group of bots, we have chosen the mode named "Capture the Flag", a version in which different teams have to capture the enemy flag from the opponents' base. In this mode, bots must solve tasks they know how to deal, but on the other hand, they should try to discover new possibilities. If bots focus on the first aspect, they only exploit the information available with difficulties to be adaptable to new situations. For that reason, the ideal behavior will be structured by a balance between two opposite strategies. Traditionally, the problem of finding a balance between these two factors has been called the "exploration–exploitation" dilemma (EE). Some authors have remarked that different versions of this dilemma appear in different engineering domains [7].

There are different ways in which the behavior of a bot can be programmed. One of the most basic alternatives is to program it by using if–then structures. Here, the programmers should know a priori a large list of possible situations to implement them, but bots are not going to change its predefined behavior so they cannot deal any unforeseen situations. In particular, and related to our interests in this paper, there exist multiple experiences of using genetic algorithms to let virtual bots learn different tasks and perform different tasks. For instance, in [8] genetic algorithms were used to tune Counter Strike bot behavior and to calculate efficient paths between two endpoints in a landscape; in [9] genetic algorithms were chosen to evolve scripts to solve various tasks in Lemmings; in [10] they were used for implementing pathfinding techniques; in [11] to implement virtual entities and to define both their morphology and the neural circuitry involved, etc. In this paper, we will use them to create a team of bots with capabilities to solve the problem of "capturing the flag", using techniques of long-distance communication, genetic algorithms tools and Markovian chain notions.

2.1. Towards a model adapted to the problem

Multiagent systems (MAS) paradigm is generally considered to be the most adequate paradigm for modeling collective systems. Although being recognized as only one, MAS models capture collective strategies from different perspectives. In general terms, whereas traditional multi-agent systems have the agent in the middle of the model, self-organizing models are focused more on the organization of the system, allowing the designer to focus on the goals without considering how the goals should be fulfilled (eg. using genetic algorithms). The classical, agent-centered system and the organization-centered system, where the organization of the system is modeled (and the role of each agent is generated in a natural way under particular constraints) are the options that a programmer can use for modeling teamwork strategies. When the system has a high structural complexity, there are greater advantages of dividing the implementation into two distinct parts. Sometimes the two approaches overlap. The reason is that there will always be situations where it is not clear whether one system is an advantage over the other. In the videogame field, there is no definite answer to whether using one or other approach would be better. In such situations it is important to realize how complex the game is. If the game consists of one well-defined type of controllable character, a self-organizing model is probably not the best choice. However, the unreal environment is a very open context and we propose in this paper to take the organizational approach. As we will see below, the resulting agents react quite fast to changes in the environment; with short code and only few precisely defined responsibilities. The agents are also able to cooperate in fulfilling their mission.

3. Mathematical model

This project deals with the implementation of a genetic algorithm to optimize the performance of a group of bots in the "capture of the flag" game. The experiment takes place in a scenario where the goal of each team is, firstly, finding the flag and then stand up waiting for the rest of the members to travel where the flag is placed (Fig. 2a). Bots are unable to perform both tasks simultaneously, therefore, we find a particular instance of the exploration–exploitation dilemma explained above. In our approach, the bots of the group have exactly the same characteristics and have the same objective (capturing the flag). However, the variable that characterizes the amount of time that the bot spend in "exploration tasks" vs "the exploitation stage" can be different in every member of the team (it is just this rate of both processes the parameter that will be optimized). All the bots present two regimes of behavior: an exploration phase (or "listening phase", i.e., how much time the bot spends searching or exploring), and (ii) an exploitation phase (or "roving phase", i.e. how much time should the bot be exploiting information in a dynamic environment). In order to be able to optimize these type of bots, some sort of indicator (a measurement that calculates the rate between stages) is needed. We have defined a parameter called index of hyperactivity ("Ih") defined as the percentage of time that a bot is moving (with a value that goes from 0 to 1, with reference to a bias unit of time T_c that characterizes the minimum time interval in both states). In this sense, we have (i) a bot-time for roving: $Ih \cdot T_c$ (ii) a bot-time for listening: $(1-Ih) \cdot T_c$.

In every cycle (again limited by T_c), every bot can walk around the map totally randomly if its status is "roving". Otherwise, in the "listening" stage, the bot stops and waits any information (from other bots) about where the flag is. As we can easily deduce, only two factors can be the dependence variables that guarantee the success of a strategy: (i) The individual probability of any bot to find the flag; (ii) The structure of the team and their communication dependencies. The objective of this model is to validate a model that tries to deal successfully with the EE dilemma, studying the behavior of different alternatives as they can affect directly to the future creation of intelligent strategies in videogame domains.

3.1. First part: how to model an individual bot

In general terms, the global probability for finding the flag will be a general mapping:

$$P(flag) = p(x_0, N, Ih)$$