Evolution of Emergent Behaviors for Shooting Game Characters in Robocode

Jin-Hyuk Hong
Dept. of Computer Science, Yonsei University
134 Sinchon-dong, Sudaemoon-ku
Seoul 120-749, Korea
hjinh@sclab.yonsei.ac.kr

Sung-Bae Cho
Dept. of Computer Science, Yonsei University
134 Sinchon-dong, Sudaemoon-ku
Seoul 120-749, Korea
sbcho@cs.yonsei.ac.kr

Abstract- **Various digital characters, which are automatic and intelligent, are attempted with the introduction of artificial intelligence or artificial life. Since a character's behavior is designed by a developer, the style can be static and simple. Even complex patterns designed by a developer cannot satisfy various users and easily make them feel tedious. A game should maintain various and complex character's behaviors, but it is not easy for the developer to design them. In this paper, we adopt genetic algorithm to produce various and excellent behavior-styles for characters especially focusing on Robocode which is one of promising simulators for artificial intelligence**

I. INTRODUCTION

With the development of video games, the interest on the intelligence of game characters is rapidly increased in recent years [1]. Although many techniques for games just focused on the graphics in its early stage, the techniques have changed to the development of game intelligence. Many players prefer much higher level of intelligence, and moreover the intelligence gets to be one of crucial factors of the success of a game [2].

Artificial intelligence controls characters and environments to be more realistic and attractive [3,4]. The research on the generation of character's behaviors starts on robotics. With the recent growth of computer animations and various game industries, it also includes studies to control avatars or animation characters [1]. While robotics focuses on the automation and accuracy of a robot's behavior, the research on character's behaviors for entertainment considers diversity, creativity and aesthetics of them as important.

Digital characters should not repeat the same behavior but have various and creative behaviors to provide players with news and curiosity. In particular, game characters should act diverse behaviors for various environments so as not to be easily understood by players. However, traditional characters designed by human are not complex so players understand them without difficulty. In addition, when the environment is complex, it takes more efforts to design behaviors. The traditional approaches are hard to develop diverse and remarkable behaviors to fill up the player's requests [1,2,5].

In this paper, we focus on the generation of diverse and good behaviors of a character. The genetic algorithm is used for the purpose to create emergent behaviors by combining primitive behaviors of a character.

II. RELATED WORKS

A. Character Behaviors

Game player controls virtual characters to move and act in virtual environment. Since the time should be short in selecting a behavior and the process should be relatively simple at games, the finite state machine is commonly adopted to design the behavior. The finite state machine is usually constructed with 10 or fewer states. At each state the behavior is defined statically with simple operations for fast processing. As mentioned before, however, the character's behavior is easily understood because of its simplicity [2].

There are many other artificial intelligence techniques for designing behaviors, such as case-based reasoning, decision tree, neural network, fuzzy logic, and so on [1,4]. Since most of them are based on manual design, they leave developer some difficulties to design behaviors in all cases. The manual design is apt to restrict the diversity and creativity of behaviors. For the case of neural network, it does not need to design the behavior directly, but the inputs and outputs are explicitly defined so as to hardly expect extra behaviors of characters. In games, extra behaviors cause an interest to players, but it is hard to construct various and extra styles of the behavior with these traditional approaches. In order to solve these limitations, genetic algorithm is applied to some games but not to various kinds of computer games [5].

Artificial life is one of promising techniques for designing characters [3,4]. It is a study of systems which imitate the behavioral characteristics of a living thing. It considers an 'emergent behavior', which is the result of interactions among low-level actions, as important. Artificial life technique divides a huge AI which decides behaviors into a set of small actions so as to generate complex and whole behaviors with interactions between low-level simple rules. Usually it is called as 'emergent'. Artificial life is a useful technique to generate complex

0-7803-8515-2/04/$20.00 ©2004 IEEE

and diverse behaviors, even if it is based on primitive actions. In this paper, we define primitive actions of characters and combine them using genetic algorithm to generate various styles of behaviors.

B. Robocode [7]

In order to demonstrate the proposed method, we adopt Robocode which arouses an interest recently as battle simulator for artificial intelligence. It is Java-based tank battle simulator developed by IBM Alphaworks. A tank programmed by user makes moves in the battle field and evades the opponent's attack and assaults it. As visualized simulator, it is easy to observe the behavior of tanks. In addition, since it is currently on-line, it can be possible to compare various other tanks that are programmed by others.

In battlefield a tank should be survived from a battle among the opponents. It gathers various information on itself and the opponents, and decides its behavior based on the information. It is necessary to consider complex and various factors to win the game and to execute the proper strategy of behaviors. Movement against the opponents and the efficient usage of an energy should be also considered, while the success of an attack leads to the win. A user manually designs and programs a tank, so that it acts accurately but is apt to have simple behaviors.

Tanks start games with limited energy, and it consumes the energy by shooting bullets or bumping somethings. If the energy becomes 0, the tank gets to be destroyed. The energy can be increased as a compensation when it hits the other tanks. The tank consists of gun, radar, and body (vehicle). Each part can operate independently. The gun fires a bullet, and the radar scans opponents and obtains their information, and the body is charged with movement. Each part has various operating functions.

The basic actions of the tank are conducted by a set of commends documented in the Javadoc of the Robocode API, especially in robocode.Robot class. Most of them are related with the movement of robot, gun, and radar. Table I shows some of basic actions defined for designing a tank. None of these actions will return control to the simulator until they are completed.

TABLE I. BASIC ACTIONS FOR MOVMENT

turnRight(double degree) and turnLeft(double degree) turn the tank by a specified degree.
ahead(double distance) and back(double distance) move the tank by the specified pixel distance.
turnGunRight(double degree) and turnGunLeft(double degree) turn the gun, independent of the vehicle's direction.
turnRadarRight(double degree) and turnRadarLeft(double degree) turn the radar on top of the gun, independent of the gun's direction and the vehicle's direction.

It is possible to move the gun and the radar simultaneously with turning the vehicle, unless it needs differently to call methods as shown in Table II.

TABLE II. BASIC ACTIONS FOR GUN AND RADAR

setAdjustGunForRobotTurn(boolean flag): If the flag is set to true, the gun will remain in the same direction while the vehicle turns.
setAdjustRadarForRobotTurn(boolean flag): If the flag is set to true, the radar will remain in the same direction while the vehicle (and the gun) turns.
setAdjustRadarForGunTurn(boolean flag): If the flag is set to true, the radar will remain in the same direction while the gun turns. It will also act as if setAdjustRadarForRobotTurn(true) has been called.

In order to decide the strategy of a tank's behavior, the tank should collect information on the battle. For the purpose, Robocode provides many methods obtainting information about the robot, and Table III shows a short list of frequently used.

TABLE III. BASIC ACTIONS FOR OBTAINING INFORMATION

getX() and getY() get the current coordinate of the tank
getHeading(), getGunHeading(), and getRadarHeading() get the current heading of the vehicle, gun, or radar in degrees.
getBattleFieldWidth() and getBattleFieldHeight() get the dimension of the battlefield for the current round.

Once the tank moves and uses its associated weaponry, it should be considered to adequately fire a bullet. As mentioned before, each tank starts out with a default energy, and is destroyed when its energy falls to zero. A fire use up to three units of energy. The more enery supplied to the bullet, the more damage it will inflict on the target tank. fire(double power) and fireBullet(double power) are basic methods for firing with the specified energy.

The tank moves in according to several events. The basic Robot class has default handlers for all of these events. The developer programs movements for some crucial events, and when an event occurs the corresponding movement is conducted. Table IV shows some events frequently used.

TABLE IV. EVENTS OCCURED IN THE BATTLE

ScannedRobotEvent (onScannedRobot()); this method is called when the radar detects a tank.
HitByBulletEvent (onHitByBullet()); this method is called when the tank is hit by a bullet.
HitRobotEvent (onHitRobot()); this method is called when your tank hits another tank.
HitWallEvent (onHitWall()); this method is called when your tank hits a wall.

As a battle starts, the tank moves based on the basic behavior programmed for the main loop, and an additional behavior is conducted when the event occurs. In this paper, ScannedRobotEvent, HitByBulletEvent, HitRobotEvent, HitBulletEvent (when the bullet hits an other bullet), and HitWallEvent are used for designing the tank.

III. Evolved Strategies Generation

A. Genetic Algorithm

Genetic algorithm, which is one of evolutionary computations based on the evolution theory such as genetic programming, evolutionary programming, and evolution strategies, is applied to optimization and classification problems [4,6]. There are some differences between them in the representation of individuals and in evolutionary operations, but the fundamental principle is the same. In this paper, we generate the strategies of the tank's behavior using genetic algorithm.

Genetic algorithm proposed by John Holland in the beginning of 1970s is an optimization method imitating the mechanism of nature's evolution such as crossover, mutation, and the survival of the fittest. The common procedure of genetic algorithm is as follows.

- step 1: initialize the population
- step 2: evaluate each individual's fitness of the population
- step 3: generate new population in proportion to the fitness of each individual
- step 4: execute genetic operators such as crossover and mutation
- step 5: repeat steps 2~5 until the stop condition is satisfied

Genetic algorithm is very efficient for solving general problems and provides an optimization mechanism, so that it applies to the optimization of gas pipe-line arrangement, travel salesman problem, the behavior evolution of robots, learning neural network, and the optimization of fuzzy membership functions, and so on.

B. Encoding of Characters

We desgin primitive behaviors classified into move-strategy (MS), avoid-strategy (AS), shoot-strategy (SS), bullet-power-strategy (BPS), radar-search-strategy (RSS), and target-select-strategy (TSS). Each strategy has various kinds of primitive behaviors as shown in Table V, and Table VI shows some examples programmed for move-strategy. The design of these primitive behaviors is mostly simple to understand and implement. Even if each of them is very simple, there might be various and complicated behaviors by the composition of them.

TABLE V. Basic Actions of a Tank

MS	randon, simple linear, randon linear, simple circular, anti-gravity, stop, bullat-avoid
AS	use, not-use
SS	Constant, linear, complex-linear
BPS	distance-based, light-fast, powerful-slow, medium, hit-rate-based
RSS	always-turns, target-focus, target-scope-focus
TSS	weak-robot, focus-attack, defense, nearest-robot

TABLE VI. Example-Codes of Move-Strategy

Move-strategy	Code

Random	switch(Math.random()*2){
	case 0: setAhead(Math.random()*500);
	break;
	case 1: setBack(Math.random()*500);
	break;}
	switch(Math.random()*2){
	case 0: setTurnRight(Math.random()*90);
	break;
	case 1: setTurnLeft(Math.random()*90);
	break;}
	execute();

Linear	ahead(100); setBack(100);

Circular	setTurnRight(5000);
	setMaxVelocity(5);
	ahead(5000);

The tank basically moves based on the flow as shown in figure 1. Each behavior has 6 kinds of strategies of Table V and the encoding is shown in figure 2. The values indicate the corresponding primitive behaviors. The strategies for events are decided by the selection of primitive behaviors and genetic algorithm generates the optimal composition of them. There are 17,280 (=6×8×2×3×5×3×4) possible combinations of the primitive behaviors. It is not quite large to search, but the complexity increases rapidly when the number of the primitive behaviors increases.

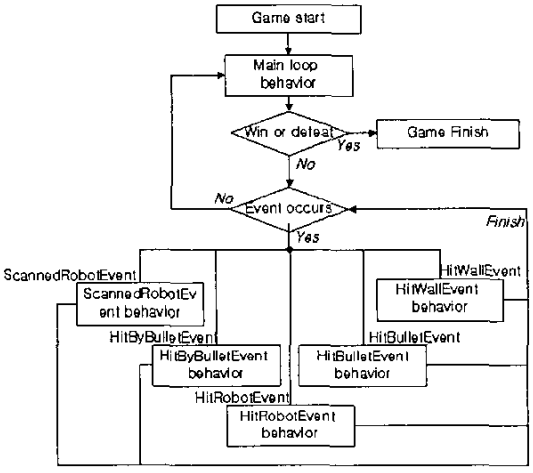


Figure 1. Behavior flow of a tank

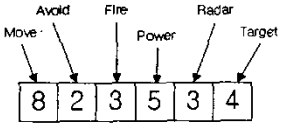


Figure 2. The encoding of a behavior

IV. EXPERIMENTS

A. Experimental Environment

We verify the proposed method matching against several tanks provided by Robocode such as wall-Robot, corner-Robot, and fire-Robot. Wall-robot maintains itself following walls, and corner-Robot stays at a corner of the battle field, and fire-Robot stays a place and fires a bullet when it scanned an opponent. We also evolve a tank against BigBear programmed by Daniel Johnson, which is one of the champs of Robocode Rumble in 2002 [7].

The parameters are set as shown in Table VII. For genetic operation, 1 point crossover, a simple mutation for the real number, and elitism are used. Roulette wheel selection is adopted as selection mechanism. The fitness of each individual is measured by the result of 3 battles among the target tanks. And the fitness of an individual is calculated by the following formula which is the rate of 3 battles' score with the target tank.

$$f = \frac{Score_{own}}{Score_{own} + Score_{opponent}}$$

TABLE VII. PARAMETERS OF EXPERIMENTAL ENVIRONMENT

Genetic operator	Value
Population size	25
Maximum generation	100
Selection rate	0.6
Crossover rate	0.6
Mutation rate	0.1

B. Results of Evolved Strategies

In this paper, we focus on the generation of various and emergent behaviors using genetic algorithm, while the behavior should be also good enough to compete with the target tanks. Table VIII shows the result of the experiment. Evolving a tank against corner-Robot is not difficult as can be seen, because corner-Robot has a simple behavior pattern. Fire-Robot and wall-Robot have a more complex pattern, so the system searches fewer solutions within the same generations. Because the proposed method did not find any solution against BigBear in 50 generations, we have tried 150 generations to search solutions and it generates a few of good strategies to defeat BigBear. The values of the right two rows of the table are different, which indicates that good strategies are different corresponding to target tanks.

TABLE VIII. RESULTS AGAINST TARGET TANKS

Target tank	Number of discovered behaviors (150)	Most frequent behavior	
		Main loop	Scanned-RobotEvent
Corner	23	000310	001003
Fire	13	000200	012413
Wall	7	400302	412223
BigBear	0 (4)	000313	011112

TABLE IX. EMERGENT STRATEGIES AGAINST WALL-ROBOT

Emergent strategy	Description	Behavior
Rambo	Firing at random	012312
Master shot	Tracing the opponent accurately and shooting	412422
Bumping king	Not shooting, only bumping the opponent	401022
Runaway	Avoiding the opponent's attack until the opponent becomes exhausted	210003
Upset driver	Moving the battle filed incomprehensibly	012112

The proposed method has generated many interesting strategies. There are many different ways to defeat wall-Robot, and Table IX shows some of the outstanding emergent strategies beating wall-Robot, specially bumping king and runaway strategies not predicted. The last row indicates the critical values for the behaviors. figure 3 shows the fitness of the best individual among population for wall-Robot, while figure 4 shows the motion trajectories of some emergent strategies.

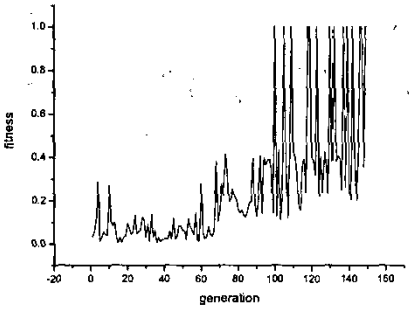
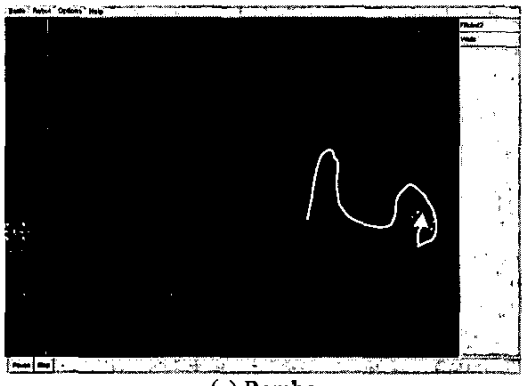


Figure 3. Fitness of the best individual

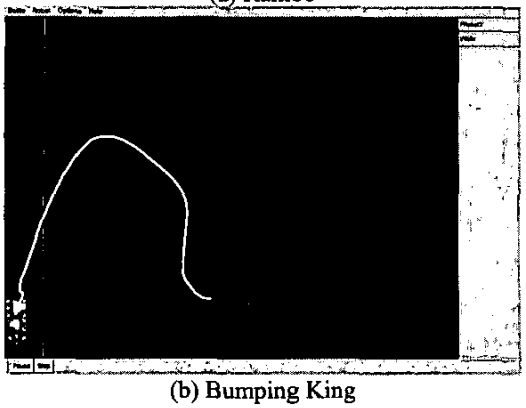
V. CONCLUSION

Artificial life and genetic algorithm are useful to generate various styles of behaviors. Game characters should keep many strategies of their movements, but it is a difficult problem for developers to design diverse behaviors. In this paper, we have tried to generate many interesting strategies for game characters using genetic algorithm. Especially applying to Robocode, we have demonstrated its utility to design various behavior styles.

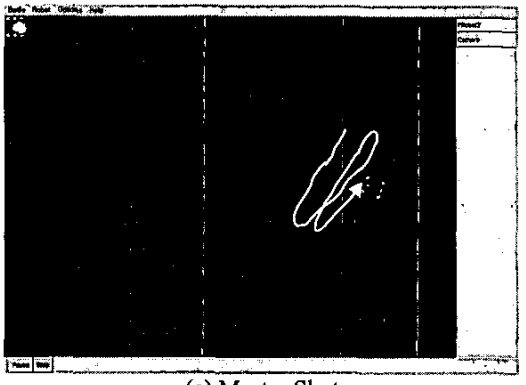
We used a set of predefined actions and combined them to generate the high level of behaviors. As a future work, we will use genetic programming, which is more flexible and representative than genetic algorithm, to generate the structure of behavior and for better movement to calculate the proper value of many parameters such as distance, degree, power, etc.



(a) Rambo



(b) Bumping King



(c) Master Shot

Figure 4. Evolved emergent stategies for wall-robot

VI. ACKNOWLEDGEMENTS

This research was supported by MOST Frontier project in Korea.

VII. REFERENCES

[1] J. Laird and M. Lent, "Human-level AI's killer application: Interactive computer games," *AI Magazine*, vol. 22, no. 2, pp.15-26, 2001.

[2] S. woodcock, "Game AI: the state of the industry," *Game Developer Magazine*, pp. 28-35, August 1999.

[3] C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, "Artificial life 2," *in Sante Fe Institute Studies in the Sciences of Complexity*, Addison-Wesley, pp. 511-547, 1992.

[4] M. Mitchell and S. Forrest, "Genetic algorithms and artificial life," *Artificial Life*, vol. 1, no. 3, pp. 267-289, 1994.

[5] D. Johnson and J. Wiles, "Computer games with intelligence," *IEEE Int. Fuzzy Systems Conf.*, pp. 1355-1358, 2001.

[6] K. Chellapilla and D. Fogel, "Evolution, neural networks, games, and intelligence," *Proc. of the IEEE*, vol. 87, no. 9, pp. 1471-1496, 1999.

[7] S. Li, Rock 'em, sock 'em robocoe!, 2002. http://www-106.ibm.com/developerworks/java/ library/j-robocode