Game Theoretic Methods for Action Games

Ismo Puustinen   Tomi A. Pasanen

Gamics Laboratory
Department of Computer Science
University of Helsinki

Abstract

Many popular computer games feature conflict between a human-controlled player character and multiple computer-controlled opponents. Computer games often overlook the fact that the cooperation between the computer controlled opponents needs not to be perfect: in fact, they seem more realistic if each of them pursues its own goals and the possible cooperation only emerges from this fact. Game theory is an established science studying cooperation and conflict between rational agents. We provide a way to use classic game theoretic methods to create a plausible group artificial intelligence (AI) for action games. We also present results concerning the feasibility of calculating the group action choice.

1 Introduction

Most action games have non-player characters (NPCs). They are computer-controlled agents which oppose or help the human-controlled player character (PC). Many popular action games include combat between the PC and a NPC group. Each NPC has an AI, which makes it act in the game setting in a more or less plausible way. The NPCs must appear intelligent, because the human player needs to retain the illusion of game world reality. If the NPCs act in a non-intelligent way, the human player's suspension of disbelief might break, which in turn makes the game less enjoyable. The challenge is twofold: how to make the NPCs act individually rationally and still make the NPC group dynamics plausible and efficient?

Even though the computer game AI has advanced much over the years, NPC AI is usually scripted (Rabin, 2003). Scripts are sequences of commands that the NPC executes in response to a game event. Scripted NPCs are static, meaning that they can react to dynamic events only in a limited way (Nareyek, 2000). Group AI for computer games presents additional problems to scripting, since the NPC group actions are difficult to predict. One way to make the NPCs coordinate their actions is to use roles, which are distributed among the NPCs (Tambe, 1997). However, this might not be the optimal solution, since it means only the distribution of scripted tasks to a set of NPCs.

Game theory studies strategic situations, in which agents make decisions that affect other agents (Dutta, 1999). Game theory assumes that each agent is rational and tries to maximize its own utility. A Nash equilibrium is a vector of action selection strategies, in which no agent can unilaterally change its strategy and get more utility. A Nash equilibrium does not mean that the agent group has maximal utility, or that it is functioning with most efficiency. It just means that each agent is acting rationally and is satisfied with the group decision.

If a NPC group can find a Nash equilibrium, two important requirements for computer game immersion are fulfilled: each NPC's actions appear individually reasonable and the group seems to act in a coordinated way. This requires that the NPC has a set of goals, which it tries to attain. The NPC gets most utility from the group actions which take it closer to its goals. The NPC's goals are encoded in a utility function. Defining the utility function is the creative part of utilizing game theory in computer game design, since Nash equilibria can be found algorithmically.

In our research we study creating a well-working utility function for a common class of computer action games. We also consider the problems of finding a suitable Nash equilibrium in reasonable time.

2 Game Theory

Game theory helps to model situations that involve several interdependent agents, which must each choose an action from a limited set of possible actions. Choosing the action is called playing a *strat-*

egy. One round of strategy coordination between agents is called a *game*. The game can be represented as a set of *game states*. A game state has an action vector with one action from each agent. Thus a game has k^n game states, if there are n players in the game with k possible actions each. Each agent gets a utility from each game state: the game states are often described as utility vectors in a matrix of possible agent actions. If an agent's strategy is to play a single action, it is called a *pure strategy*. Sometimes an agent gets a better expected utility by selecting an action randomly from a probability distribution over a set of actions; this is called a *mixed strategy*. The set of actions an agent plays with probability $x > 0$ is the agent's *support*.

As stated before, a strategy vector is a Nash equilibrium only if no agent can unilaterally change their action decision and get a better utility. Nash (1950) proved that every finite game has at least one Nash equilibrium. When a pure strategy Nash equilibrium cannot be found, a mixed strategy equilibrium has to exist. If we find a Nash equilibrium, we have a way for all participating agents to act rationally from both outside and group perspective.

Finding a Nash equilibrium from a game state search space is non-trivial. Its time complexity is not known (Papadimitriou and Roughgarden, 2005). The n-player time complexity is much worse than the 2-player case (McKelvey and McLennan, 1996). The problem is that the players' strategy choices are interdependent: whenever you change one variable, the utilities for each other player also change. The problem of finding all Nash equilibria is also much more difficult than the problem of finding a single Nash equilibrium.

How can we know if the Nash equilibrium we found is good enough? It also turns out to be quite difficult. Determining the existence of a Pareto-optimal equilibrium is NP-hard (Conitzer and Sandholm, 2003). Even finding out if more than one Nash equilibrium exists is NP-hard. However, some places in the search space are more likely to have a Nash equilibria, and that heuristic is used in a simple search algorithm to find a single Nash equilibrium (Porter et al., 2004).

The search algorithm is based on the heuristic that many games have an equilibrium within very small supports. Therefore the search through the search space should be started at support size 1, which means pure strategies. In real-world games a strategy is often dominated by another strategy. A dominated strategy gives worse or at most the same utility as the strategy dominating it. Therefore it never makes

sense to play a dominated strategy. The search space is made smaller by using iterated removal of dominated strategies before looking for the Nash equilibrium in the support.

3 Problem Setting

A typical action game can be described as a set of combat encounters between the PC and multiple enemy NPCs. The NPCs usually try to move close to the PC and then make their attack. The abstract test simulation models one such encounter omitting the final attack phase. The purpose of the simulation is therefore to observe NPC movement in a game-like situation, in which the NPCs try to get close to the PC while avoiding being exposed to the PC's possible weapons. A starting point for the simulation is described in Figure 1. The game area consists of squares. In the figure the PC is represented by 'P' and the NPCs by numbers from 1 to 3. 'X' represents a wall square and '.' represents open ground.

```
..............................
..............................
.............XX...............
......1......X.....P..........
.........XXXX.................
...2..XXXXX...................
.......XX.....................
..............................
..............................
....X.........................
....XXX...............XXXXX...
.............X.........XXXX...
........................X.....
.......................3......
```
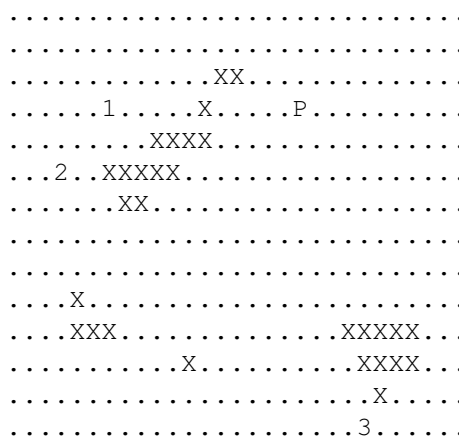
Figure 1: A starting point for test simulation.

The test simulation has a set of rules. It is divided into turns, and the NPCs decide their actions each turn. The PC doesn't move in this simulation. Each NPC has a set of five possible actions, which can be performed during a single turn: { left, right, up, down, do nothing }. A NPC cannot move through the walls, so whenever a NPC is located next to a wall, it's action set is reduced by the action that would move it into the wall. The PC cannot see through walls. Two NPCs can be located in the same square, but a NPC cannot move to the square occupied by the PC.

In the test simulation all NPCs choose their actions

simultaneously from their action sets. However, the Nash equilibrium search is not made by the NPCs themselves, but by a central agency. This is justified, since the chosen strategies form a Nash equilibrium. Even if the computing was left to the NPCs, their individual strategy choices would still converge to a Nash equilibrium.

Levy and Rosenschein (1992) studied the game theoretic solution for the predator-prey problem domain, which has four predator agents trying to encircle a prey agent in a grid. They used game theory as a means to have the predator agents move more precisely when they got near the prey. The game theoretic predators avoided collisions by selecting game states where no collisions occured: these were naturally the Nash equilibria. In our simulation, however, the focus is different. Game theory allows the NPCs to act individually rationally in a way that can be, if necessary, against other agents and their goals. This allows for realistic-looking cooperation, which in turn leads to a greater immersion for the player. It is easy to make a NPC in a computer game to be more dangerous for the PC: the NPC can be made, for instance, faster or stronger. The problem of making the NPCs appear life-like is much more difficult. The simulation aims to provide some insight into how realistic-looking NPC movement might be attained.

All "intelligence" of the NPCs is encoded in the utility function, which means that all targeted behavioral patterns must be present within it. Levy and Rosenschein used a two-part utility function with the predator agents: one part of the function gives the agents payoff for minimizing the distance to the prey agent and the other half encourages circulation by rewarding blocked prey movement directions. Following the same principle, the utility function that we use is made of terms that detail different aspects of the NPCs' goals. Each term has an additional weight multiplier, which is used to control and balance the amount of importance the term has. We found the following terms to be the most important in guiding the NPCs' actions in the simulation: aggression, safety, balance, ambition, personal space and inertia. The final utility function is the sum of all the terms weighted with their weight multipliers. The terms are detailed below.

Aggression means the NPC's wish to get close to the PC. The term's value is $-xD_i$, where D_i is the NPC i's distance from the player and x is an additional constant multiplier representing the growing of aggression when the NPC gets nearer the PC.

Safety represents the NPC's reluctance to take risks. When a NPC is threatened, it gets a fine. The fine amounts to $-r_i/k$, where r_i is the severity of the threat that the NPC encounters and k is the number of NPCs who are suspect to the same threat. If k is zero, no NPCs are threatened, and the term is not used. In the test simulation a NPC was threatened, if it was on the PC's line of sight. The divisor is used to make it safer for one NPC, if several NPCs are subjected to the same threat from the PC.

Balance is the NPC's intention to stay approximately at the same distance from the PC as the other NPCs. The value of balance is $-d_i$, where $d_i = \left| D_i - \frac{\sum D_{-i}}{k} \right|$. $\frac{\sum D_{-i}}{k}$ is the average distance of all other NPCs from the PC. D_i is the NPC i's distance from the PC. $\left| D_i - \frac{\sum D_{-i}}{k} \right|$ gets bigger as the NPC's distance from the PC gets further from the average distance. The term is needed to make the NPCs try to maintain a steady and simultaneous advance towards the PC, and to prevent the NPCs from running to the PC one by one. The term is not used if there are no other NPCs in the game.

Ambition means the NPC's desire to be the first attacker towards the PC. If the NPC is not moving towards the PC, ambition value is 0. If no NPC is going towards the PC, the ambition value is tx, where t it the amount of turns in which no NPC has moved towards the PC and x is constant.

Personal space is the amount of personal space a NPC needs. The term has value x_i, which is the NPC's distance to the nearest other NPC, if the distance is below a threshold value x'. If $x_i > x'$, the term has value x' instead of x_i. This term is needed to avoid the NPCs packing together and encourage them to go around obstacles from different sides.

Inertia is the NPC's tendency to keep to a previously selected action. The term makes the NPCs appear consistent in their actions. If the NPC is moving in the same direction as in the previous game turn, it gets bonus x. If the NPC is moving in an orthogonal direction, it gets the bonus $\frac{x}{2}$. Inertia helps to prevent the NPCs from reverting their decisions: if ambition drives the NPCs from their hiding places, inertia keeps them from retreating instantly back into safety.

Since the utility function provides the information about which actions the NPC values, all other necessary AI functions must be implemented there. The test simulation required implementation of A* algorithm for obstacle avoidance: all measured distances towards the PC or other NPCs are actually shortest-path distances. Game-specific values can also be adjusted in the utility function. For instance, the game designer may want to change the game difficulty level

mid-game by tweaking the game difficulty parameters (Spronck et al., 2004). These adjustments must be made within the utility function, otherwise they have no effect in NPC action selection.

The simple search algorithm is deterministic by nature, and therefore the Nash equilibrium found from any given starting setup is always the same. If a mixed strategy is found, randomness follows implicitly, because the action is randomly selected from the probability distribution. However, the algorithm is biased towards small supports for efficiency reasons, and therefore tends to find pure strategies first. The pure strategies are common with the game simulation setting, since the NPCs are rarely competing directly against one another. The game designer may want to implement randomness in the utility function by adding a new term, *error*, which is a random value from $[0 \ldots 1]$. The weight multiplier can be used to adjust the error range.

4 Results

The simulation yielded two kinds of results: the time needed to find the Nash equilibrium during a typical game turn and the NPCs' actions using the utility function described in section 3. The simulation run began from the setting in Figure 1.

Because the computer game AI is only good if it gives the player a sense of immersion, the evaluation of the utility function's suitability must be done from the viewpoint of game playability. However, no large gameplay tests were organized. The utility function goodness is approximated by visually inspecting the game setting after every game turn.

Figure 2 shows the game board on turn 3. The '+'-signs represent the squares that the NPCs have been in. NPC 2 began the game by moving south, even though its distance to the PC is the same in both the northern and southern route around the obstacle. This is due to the fact that NPCs 1 and 2 were pushed away from each another by the term *personal space* in the utility function.

Figure 3 on turn 10 has all NPCs in place to begin the final stage of the assault. None of the NPCs are on the PC's line of sight. The NPCs 3 and 2 have reached the edge of the open field before NPC 1, but they have elected to wait until everyone is in position for the attack. The term *safety* is holding them from attacking over the open ground.

Game turn 12 is represented in Figure 4. The NPCs have waited one turn in place and then decided to attack. Each NPC has moved simultaneously away from cover towards the PC. In game theoretic sense,

```
..............................
..............................
.............XX...............
......+++1..X.....P...........
.........XXXX.................
...+..XXXXX...................
...+...XX.....................
...+2.........................
..............................
....X.........................
....XXX..............XXXXX...
...........X.........3XXXX...
......................++X.....
.......................+......
```

Figure 2: Turn 3. NPC 1 has begun to go around the obstacle by North and NPC 2 from South. NPC 3 has moved into cover.

```
..............................
...........+++1...............
...........+.XX...............
......++++++X.....P...........
.........XXXX.................
...+..XXXXX...................
...+...XX.....................
...+++++2.....................
..............................
....X.........................
....XXX..............XXXXX...
...........X.........3XXXX...
......................++X.....
.......................+......
```

Figure 3: Turn 10. All members of the NPC team have arrived to the edge of the open ground.

two things might have happened. The first possibility is that term *ambience* makes one NPC's utility from moving towards the PC grow so big that attacking dominates the NPC's other possible actions. Therefore the NPC's best course of action is to attack regardless of the other NPCs' actions. When this happens, the sanction from term *safety* diminishes due to the number of visible NPCs, and it suddenly makes sense for the other NPCs to participate in the attack. The other possibility is that the algorithm for finding Nash equilibria has found the equilibrium, in which all NPCs move forward, before the equilibrium, in which the NPCs stay put.

NPCs succeed in synchronizing their attack be-

```
.............................
...........++++1.............
..........+.XX...............
......++++++X.....P..........
.........XXXX................
...+..XXXXX.................
...+...XX...................
...++++++2..................
............................
....X.......................
....XXX............XXXXX...
...........X.......3+XXXX...
....................++X.....
.....................+......
```

```
.............................
...........++++++++..........
..........+.XX...1...........
......+++++X.....P..........
.........XXXX.....3..........
...+..XXXXX.......+..........
...+...XX2........+..........
...+++++++........+..........
..................+..........
....X.............+..........
....XXX...........+..XXXXX...
...........X......++++XXXX...
....................++X.....
.....................+......
```

Figure 4: Turn 12. NPC team decides to attack. All NPCs move simultaneously away from cover.

Figure 5: Turn 21. NPCs 1 and 3 have reached the PC. NPC 2 has gone into hiding once more.

cause the Nash equilibrium defines the strategies for all NPCs before the real movement. Synchronization leaves the human player with the impression of planning and communicating enemies. If the NPCs had ran into the open one by one, stopping the enemies would have been much easier for the human player, and the attack of the last NPC might have seemed foolhardy after the demise of its companions.

Figure 5 shows the game board on turn 21. NPCs 1 and 3 have moved next to the PC. NPC 2 has found new cover and has decided not to move forward again. The situation seems erroneous, and it is true that the NPC 2's actions seem to undermine the efficiency of the attack. However, this can be interpreted to show that NPC 2 has reexamined the situation and decided to stay behind for it's own safety. One way to resolve the situation is to change the term *safety* to lessen the threat value, if at least one NPC is already in hand-to-hand combat with the PC.

Creating a usable utility function is quite straightforward if agent's goals can be determined. Balancing the terms to produce the desired behavior patterns in different game settings can be more time-consuming. Each different video game needs a specific utility function for its NPCs, since for instance the distance measurements are done in different units. Also the game designer may want to introduce different behavior to different NPC types, which is done by creating a utility function for each NPC class within the video game.

The test simulation used the McKelvey et al. (2006) implementation of the previously mentioned simple search algorithm. The algorithm's time complexity limits the simulation's feasibility when the

number of agents in the game grows larger.

We measured the time needed for finding a single Nash equilibrium in a Macintosh computer equipped with a 2.1 GHz processor. The extra calculations in the utility function (such as A* algorithm) were left out. The needed time was calculated using the three-agent setting described in Figure 1 and the previously detailed utility function. The measurements were also made using six NPCs, whose starting positions are detailed in Figure 6.

```
.............................
.............................
.............XX..P...........
......1.....X...............
........XXXX.................
..2..XXXXX...............5..
.......XX....................
....................4........
.............................
....X........................
....XXX..............XXXXX...
...........X..........XXXX...
....6.................X.....
.....................3......
```

Figure 6: The starting point for the six-player simulation.

Both games were run for 25 turns, and the experiment was repeated ten times. The games had therefore 250 data points each. The test results are presented in table 1.

Agents	min.	max.	avg.	median
3	42 ms	225 ms	71 ms	62 ms
6	97 ms	2760 ms	254 ms	146 ms

Table 1: Experimental results for the time needed to find the first Nash equilibrium in attack game.

The results show that in a three-player game the Nash equilibrium was found in average within 71 milliseconds. This can be still feasible for real video games. In six-player games the worst calculation took almost three seconds, which is far too long for action games. Still, the median in six-player game was only 146 milliseconds, which may still be acceptable. In both games a mixed strategy was never needed: the Nash equilibria were always found in supports of size 1. The worst times were measured in the first turn. When a NPC had several dominated actions or was next to a wall, the search was faster, because the search space was reduced.

5 Conclusion

Using game theoretic methods in action games seems a promising way to do a more plausible NPC group AI. If the agents try to find a Nash equilibrium, their individual decisions are rational. If the agents' utility functions are designed to find cooperative behavior patterns, the group seems to function with a degree of cooperation. Having agents do their decisions based on agents' internal valuations helps agents maintain their intelligent-looking behavior in situations, where a scripted approach would lead to non-satisfactory results. This might make the game designer's work easier, since all encounters between the PC and a NPC group need not be planned in advance.

The problem in this approach is the time complexity of finding Nash equilibria. Finding one equilibrium is difficult, and finding all equilibria is prohibitively expensive. Still, our results indicate that modern computers with a good heuristic might be able to find one Nash equilibrium relatively fast, especially if the NPCs' action sets are limited and the number of NPCs is small. If the Nash Equilibria cannot be found soon enough, it is up to the game designer to decide the fallback mechanism. The scripted approach is one possibility, and another is to use a greedy algorithm based on the utility functions. A greedy algorithm would not waste time on calculating the possible future actions of the other NPCs, but would assume that the NPCs stayed idle or continued with a similar action as in the previous game round.

References

Vincent Conitzer and Tuomas Sandholm. Complexity results about Nash equilibria. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 765–771, 2003.

Prajit K. Dutta. *Strategies and Games: Theory and Practice*. MIT Press, 1999. ISBN 0262041693.

Ran Levy and Jeffrey S. Rosenschein. A game theoretic approach to the pursuit problem. In *Proceedings of the Eleventh International Workshop on Distributed Artificial Intelligence*, pages 195–213, Glen Arbor, Michigan, February 1992.

Richard D. McKelvey and Andrew M. McLennan. Computation of equilibria in finite games. In H. Amman, D. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, volume 1, pages 87–142. Elsevier, 1996.

Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory, version 0.2006.01.20, 2006. URL http://econweb.tamu.edu/gambit/.

Alexander Nareyek. Review: Intelligent agents for computer games. In T. Anthony Marsland and Ian Frank, editors, *Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 414–422. Springer, 2000. ISBN 3-540-43080-6.

John Nash. Equilibrium points in n-player games. In *Proceedings Of NAS 36*, 1950.

Christos H. Papadimitriou and Tim Roughgarden. Computing equilibria in multi-player games. In *SODA*, pages 82–91. SIAM, 2005. ISBN 0-89871-585-7.

Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 664–669. AAAI Press / The MIT Press, 2004. ISBN 0-262-51183-5.

Steve Rabin. Common game AI techniques. In Steve Rabin, editor, *AI Game Programming Wisdom*, volume 2. Charles River Media, 2003.

Pieter Spronck, Ida G. Sprinkhuizen-Kuyper, and Eric O. Postma. On-line adaptation of game opponent AI with dynamic scripting. *Int. J. Intell. Games & Simulation*, 3(1):45–53, 2004.

Milind Tambe. Towards flexible teamwork. *J. Artif. Intell. Res. (JAIR)*, 7:83–124, 1997.