Real Time Planning for Path-finding in Computer Games

Rubén Tous

Universitat Pompeu Fabra (UPF), Departament de Tecnologia,
Pg. Circumval·lació, 8. E-08003 Barcelona, Spain
ruben.tous@tecn.upf.es

Abstract

The relationship between AI and computer games could be qualified as "bi-directional inverse". Modern AI techniques use old games –chess, bridge, go, etc..- as test fields while modern computer games use old AI techniques as A. The fast evolution of this kind of entertainment has multiplied the AI requirements of the new developments evidencing the necessity to reuse the work already done in the research field. This article discusses what role play and what role could play the last achievements in the field of AI Planning in the resolution of the problem of path-finding in modern real time computer games.*

0. AI and Modern Computer Games

Recent articles published about the state of the art of the commercial computer game's AI [1] reveals that this issue has finally been recognized as an important part of the game design process. In the past this functionality used to be relegated in the projects schedule but now, for many people, game's AI has become as important as for example as the game's graphics engine. The next graph (Fig. 1), making reference to information gathered at the Game Developers Conference 2001 [2] roundtables, illustrates this evolution in terms of human resources dedicated and CPU time reserved.

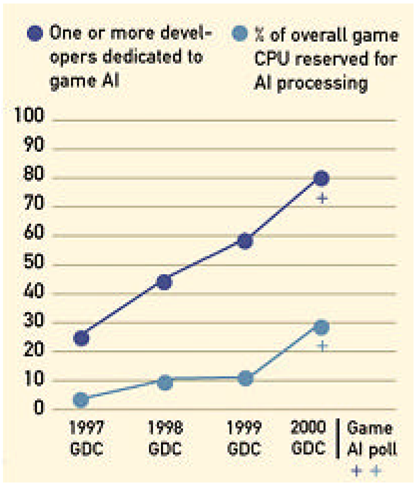

Fig. 1

Numbers reveal that attitudes toward game AI have shifted. One consequence of that is the change on the distribution of computational resources. For example, in prior years AI computations were ever limited "as long as it doesn't affect the frame rate". Now one can hear from a project manager that "new graphics features are fine, so long as they don't slow down the AI".

There are a lot of different applications of AI technologies in computer games, and different trends can be identified. This article focuses on path-finding but before getting to the point let's mention here other applications and other kind of games. The next table, extracted from [4], shows the importance of AI in the different commercial game categories.

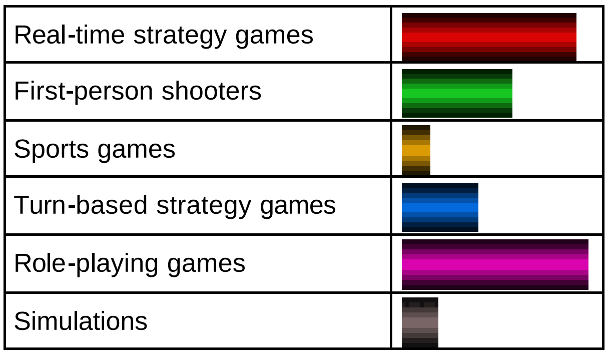

Fig. 2

There are a lot of games not included in the table where AI plays a very important role. For example we have games using some kind of Artificial Life technology. A-Life is perhaps the most evident trend of the last years and has been used in Maxi's *The Sims* or CogniToy's *Mind Rover. T*his approach consists in provide the game characters with a realistic, lifelike behavior. The target of A-Life techniques is to artificially reproduce the way in that real-world driving organisms behave, and that's the reason because in the research area this approach is know as Behavior Oriented Artificial Intelligence. In fact, people of the world of Robotics and Intelligent Agents have already made a lot of work over this issue, as the recent articles about Behavior Oriented Design [3].

1. Path-finding and Terrain Analysis

Path-finding is, of all the decisions involved in game's AI, probably the most common. The problem consists in looking for a good route when moving an entity from *here* to *there*. The entity can be any mobile entity of the game (a virtual character, a vehicle, etc.). A good route means a realistic route (that respects the terrain topology), and a route that minimizes the cost (energy, food, etc.). Here it must be remarked that even we're using the term "path-finding" and "path-planning" in the same way, these two terms mean different things. Path-planning is the action of deciding which route to take, based on and expressed in terms of the current internal representation of the terrain, while path-finding is the execution of this theoretical route, by translating the plan from the internal representation in terms of physical movement in the environment.

Pathfinding can appear in games of any genre (action games, simulators, role-playing, strategy, etc.) where the computer is responsible for moving things around. It could seem a trivial problem, because is as old as games themselves, but questions about path-finding are regularly seen in online game programming forums, and the entities in several nowadays games move in less than intelligent paths.

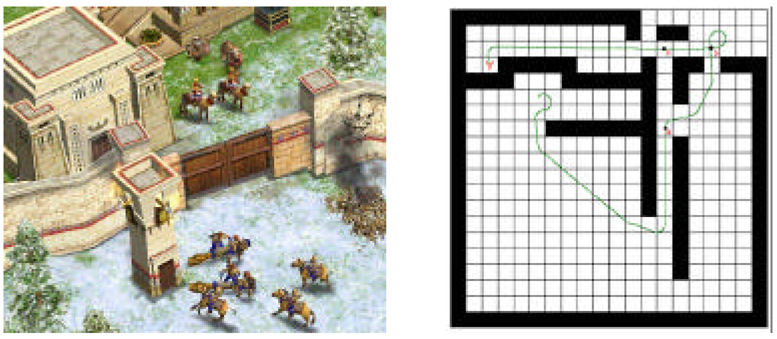

Fig. 3

Recently pathfinding is included in a bigger sack called Terrain Analysis [5]. This term, overall used when referring to RTS games, refers to any kind of strategy that involves units movements, and can go from simple pathfinding in tile maps to more complex issues like area decomposition or influence maps. It reflects the necessity to reduce the gap between the way human player and computers act. Though this issue is very interesting and introduces the concept of abstraction (for example in area decomposition) when facing the pathfinding problem, it's not the target of this article. Here we will focus on remarking that recent research in the field of AI Planning will serve as an alternative way to A* when solving the classic pathfinding problem.

2. Path-finding Sample: Real Time Strategy (RTS) Games

RTS is one of the most popular genres of computer games nowadays. It appeared more than ten years ago with games like Herzog Zwei (1989) or Westwood's Dune II (1992), and consolidated with titles like Blizzard's Warcraft (1994) or Westwood's Command&Conquer (1995). The target of a RTS game is to coordinate a relatively big amount of partially autonomous units (workers, soldiers, vehicles,…) to overcome one or more players, computer or human, with their own groups of units over a shared territory. In a RTS game the units move autonomously, but the player can order certain unit to move to any certain location. Unit movements, autonomous or not, need to be managed by the computer, and here is where comes path-finding. The territory of a RTS game seems continuous, but it isn't, it is divided in cells approximately of the size of an unit. Units can only move according to the cell grid. So this is a classical path-finding problem but with two added complexities: 1) Uncertainty: Other moving entities could block the way and invalidate the previously computed path-plan. 2) Limited CPU: Hundreds or even thousands of units are moving at the same time and need to compute their next move.

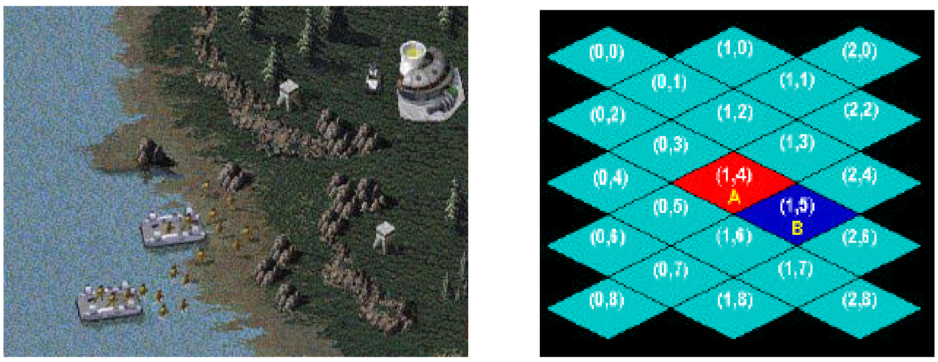

Fig. 4

RTS is only a sample to illustrate an application of path-finding in modern computer games, and it's an example of how this issue hasn't still a definitive solution. Old RTS games like Blizzard's Warcraft suffer of a very poor path-finding algorithms that have desperate more than one human player (seeing how an unit wasn't capable to avoid even a small bunch of trees to reach its destination). But even now games as Ensemble's Age of the Empire's II employ an extraordinary amount of computational time to provide better but not perfect solutions.

3. Limitations of Current Path-finding Solutions

The traditional approach to find a route between two locations in a game terrain is to model this problem as a planning problem in a state model and to apply some kind of search algorithm. Every possible location in the map is considered a different state with a set of possible actions to perform (movements to other locations) that will serve to represent the topology of the terrain. The resulting states graph can be explored from the initial state to the goal state using different strategies to find the best way (this that reduces the cost).

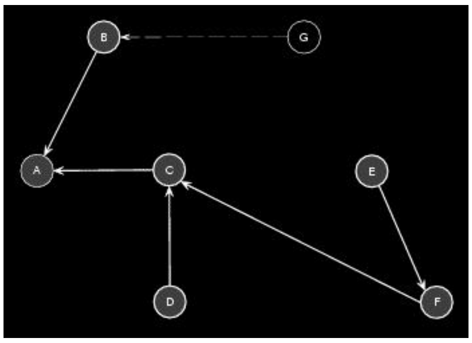

Fig. 5

The different ways the nodes of the graph can be explored motivate the existence of different search algorithms, as Breadth-first search, Depth-first search, Dijkstra's, Iterative-deepening depth-first search or the preferred in the game community, A*. Once we have found an available path between the initial state and the goal we can begin to move. To overcome the problem of moving entities that could interrupt our way we can update our plan by recalculating the path every certain time or just every time the something blocks our way.

There are situations where A* may not perform very well, for a variety of reasons. The more or less real-time requirements of games, plus the limitations of the available memory and processor time in some of them, may make it hard even for A* to work well. A large map may require thousands of entries in the Open and Closed list, and there may not be room enough for that. Even if there is enough memory for them, the algorithms used for manipulating them may be inefficient.

3.1 Real Time Planning

Traditional search methods from artificial intelligence, such as the A* method, first plan and then execute the resulting plan. Real-time (heuristic) search methods (a term coined by Korf), on the other hand, associate values with the states and update them during execution (often, using dynamic programming methods) to prevent cycling and focus the search. They restrict the search to a small part of the domain that can be reached from the current state of the agent with a small number of action executions. This is the part of the domain that is immediately relevant for the agent in its current situation. Real-time heuristic search methods do not attempt to minimize the plan-execution time but rather decrease the planning time or the sum of planning and plan-execution time over that of traditional search methods. This is possible because interleaving planning and plan execution allows real-time heuristic search methods to reduce the planning effort spent on contingencies that could have occurred but did not occur. Examples of real-time heuristic search methods are Korf's Learning Real-Time A* (LRTA*) [8] or ASP [9].

To understand how these kind of algorithms work, let’s take a glance to the LRTA* method:

```
for each neighbor j
        $f(j) = k(i,j) + h(j)$
$h(i) = \min_j f(i)$
move to j with min $f(j)$
```

Initially we have no information about the other states, we only know what are the states available from the initial state and what’s the cost to reach them (function k). The heuristic h(j) for states yet not visited must not overestimate the real cost to go from state j to the goal (h(i) <= h*(i)) and can be set, for example, to the lineal distance in the problem of path-planning. Once we’ve selected the next state we can update the old state h(i). Repeating this process we keep accumulating information of the visited states and their respective h values, that will converge very fast and drive us to the goal.

Let’s see and example:

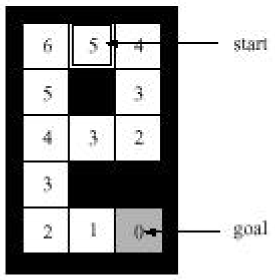

Fig. 6

Initially we start with an admissible h(i) that corresponds to the distance between the start and the goal if no objects were present. The first choice will be the node of the right because it has a smaller h(i) value.

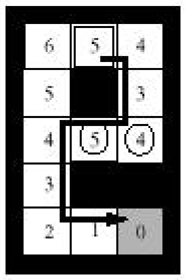

Fig. 7

Fig. 6 show the path achieved to the goal. It’s not the optimal path, but after other attempts the values of the heuristics will converge to it as can be seen in the next figures.

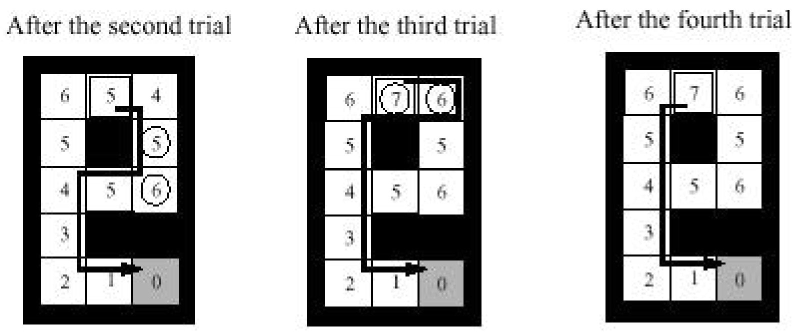

Fig. 8

Real-time (heuristic) search methods have been shown to be efficient alternatives to traditional search methods for a variety of search problems, including traditional search problems, moving-target search problems, STRIPS-type planning problems, robot navigation and localization problems with initial pose uncertainty, robot exploration problems, totally and partially observable Markov decision process problems, and reinforcement-learning problems, among others.

These methods are domain-independent and allow for fine-grained control over how much planning to do between plan executions and thus are any-time contract algorithms. They also amortize learning over several search episodes, which allows them to find plans with suboptimal plan-execution time fast and then improve the plan-execution time as they solve similar planning tasks, until their plan-execution time is optimal. Thus, they always have a small sum of planning and plan-execution time, and minimize the plan-execution time in the long run in case similar planning tasks unexpectedly repeat. This is important since no search method that executes actions before it has solved a planning task completely can guarantee to minimize the plan-execution time right away.

4. Conclusions

The path-finding problem serves as a case-study to show the gap between the research field and the games industry in terms of AI. It's shocking the way in that most game developers ignore the research achievements in the fields of planning, agents and robotics. Companies spend great amounts of resources developing complex solutions trying to extend old planning techniques as A* to adapt them to highly dynamic worlds. These solutions consume lots of computational resources (CPU time and memory) constraining other game features. During the exploration performed before writing this article, and after analyzing dozens of technical reports written by games AI developers of some of the biggest companies, we haven't found no reference to Real Time Planning in none of them.

Real Time Planning has proved to be a good solution to path-finding in the field of robotics, and has all the required features to be also a good solution for the problem of path-finding in commercial computer games.

5. References

[1] Steven Woodcock (Wyrd Wyrks) . "Game AI: The State of the Industry". Game Developer Magazine, August 2001.
http://www.gamasutra.com/features/20001101/woodcock_pfv.htm

[2] Game Developers Conference
http://www.gdconf.com/

[3] Joanna Bryson. "Dragons, Bats & Evil Knights: A Three-Layer Design Approach to Character Based Creative Play". 18 December, 2000
ftp://ftp.ai.mit.edu/pub/users/joanna/charjour.pdf

[4] GameAI
http://www.gameai.com/

[5] Dave C. Pottinger. "Ensemble Studios Terrain Analysis in Realtime Strategy". Studios Programmers Journal
http://www.ensemblestudios.com/news/devnews/terrain1.shtml

[6] Alex J. Champandard . "Path-Planning from Start to Finish".
http://ai-depot.com/BotNavigation/Path.html

[7] W. Bryan Stout . "Smart Moves: Intelligent Path-Finding". Game Developer Magazine. February 12 1999.
http://www.gamasutra.com/features/19990212/sm_01.htm

[8] R. Korf. "Real-time heuristic search. Artificial Intelligence". 1990.

[9] Héctor Geffner, Blai Bonet, Gábor Loernics. "A Robust and Fast Action Selection Mechanism for Planning".

[10] Marco Pinter. "Toward More Realistic Pathfinding". Game Developer Magazine, April 2001.
http://www.gamasutra.com/features/20010314/pinter_01.htm