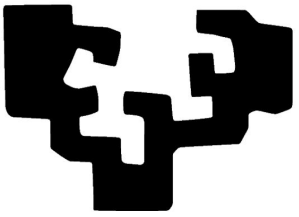
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Techniques and Algorithms in Video Game AI Developing

Advanced Techniques in AI

Walid Boussaboun
Imanol Gomez

October 18, 2017

Abstract: This document explains three techniques used in video games in order to allow non-human agents to react to their virtual environment. These techniques are pathfinding, which allows the agents to trace and follow a route from two positions, flocking, which allows agents to group and share a space as well as move in groups in a realistic way, and dead reckoning, which allows real time network games to be played without noticing network latencies.

Introduction

As everybody knows, video game industry is one of the most important industries nowadays. The first modern video games were created around the 1960s, and they were incredibly simple. As time went by, people created better computers, better algorithms, learnt new techniques and improved artificial intelligence. Thanks to this process, we could say that today's games agents have "intelligence", because they take decisions based on external information, and they do not act randomly, as they used to do in the past.

Also, video games have become more complicated to program, they needed more programmers working on them and also more time to be completed. On the other hand, they became more fun, they started to offer more challenges to players, and can be enjoyed by more people than before.

Video game industry will continue growing, that is a fact, and algorithms and techniques used in games agents will improve too. Taking this into account, we can see the importance of creating efficient algorithms that will be used by virtual agents, so they can give a better game experience to future players.

We liked video games and we played them since our childhood, so we decided to do a work related to them.

We are coursing the Advanced Techniques in AI subject which is about Intelligent Agents, how do they work, which algorithms and logic do they use, what kind of Agents exist, etc… So, according to what we have seen in class and our passion for video games, we decided to do a work based on the three most used algorithms or techniques in video games by agents.

Pathfinding

Background

This algorithm is one of the most used algorithms in video games and it consists in finding the shortest route between two points. It is heavily based on Dijkstra's algorithm for finding the shortest path on a weighted graph, as seen in subjects in the Computer Science career.

Use in video games

In video games, this algorithm is frequently used in the NPCs (agents), and it's purpose is to achieve that each NPC is able to go from its current location to its target location avoiding other NPCs and obstacles in the path. It also takes into account a cost function to take the best path, in other words, if more than one possibilities were available from going to point A to point B, the algorithm would choose the more optimal one (usually this is the fastest one).

Here we can see a very good example on how this works in a common strategy game:

Figure 1: Screenshot of Warcraft III.

In this type of games, there is a huge map to explore, and we can choose one unit or a number of troops to go wherever we want selecting the troops and clicking in the point where we want them go to. As we can see in Figure 1, the map has several routes to travel between cities without entering in the forest. These routes are used by NPCs to travel between points automatically, and they act like edges in a graph, and the little forest clears which joins different routes act like nodes. So, if we want our troops to go from point A to

point B, and this implies a long way that contains many routes and forest clears, they will automatically choose the fastest way.

Here we can see some nodes in red and routes in blue of the map of the game above:

Figure 2: Graph made based on the map.

Related Techniques

But, how do the NPCs know which areas are able to pass pass through, and which areas are not? (like rivers, for example). All the map is covered by invisible meshes called “Navigation Meshes” and thanks to them, NPCs recognize the different areas in the map and how they can interact with them.

For example, in a FPS (First Person Shooter), the Navigation Meshes show where can the NPCs (in this case soldiers) make some actions, such as, walk in the street, take cover from the enemy fire behind a car, climb up some stairs, etc.

Examples

Assassins Creed NPCs walking: https://www.youtube.com/watch?v=U8-BCm-IrcI

Flocking

Flocking is the collective motion of a large number of self-propelled entities. Interaction between simple behaviors of individuals produce complex yet organized group behavior. The component behaviors are inherently nonlinear, so mixing them gives the emergent group dynamics a chaotic aspect (unpredictability), since the position and movement of an entity cannot be predicted in long amounts of time. At the same time, the negative feedback provided by the behavioral controllers tends to keep the group dynamics ordered. The result is *life-like* group behavior.

Rules

Basic models of flocking behavior are controlled by three simple rules:

1. Separation - avoid crowding neighbors (short range repulsion).

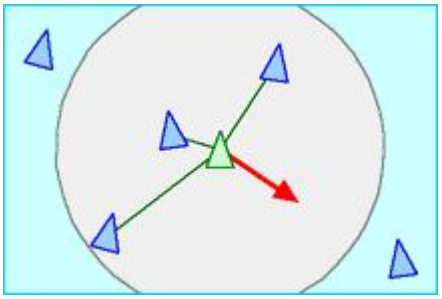


Figure 3: separation.

2. Alignment - steer towards average heading of neighbors.

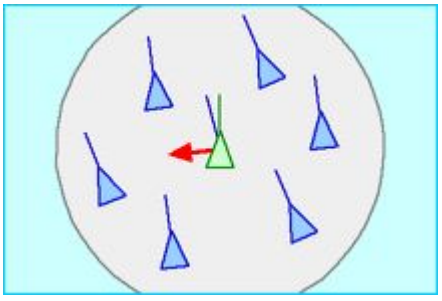


Figure 4: Alignment.

3. Cohesion - steer towards average position of neighbors (long range attraction).

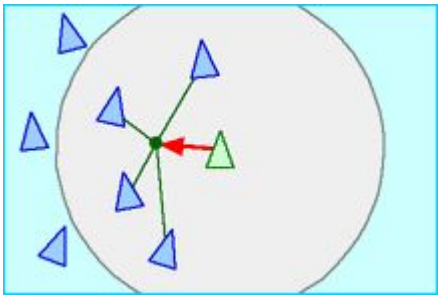


Figure 5: Cohesion.

With these three simple rules, the flock moves in an extremely realistic way, creating complex motion and interaction that would be extremely hard to create otherwise.

Background

In 1986 Craig Reynolds made a computer model of coordinated animal motion such as bird flocks and fish schools. It was based on three dimensional computational geometry of the sort normally used in computer animation or computer aided design. Craig Reynolds called the generic simulated flocking creatures boids. [Craig Reynolds]

Algorithm

Straightforward implementation of the boids algorithm has an asymptotic complexity of $O(n^2)$. Each boid needs to consider each other boid, if only to determine if it is not a *nearby flockmate.* However it is possible to reduce this cost down to nearly O(n) by the use of a suitable *spatial data structure* which allows the boids to be kept sorted by their location. Finding the nearby flockmates of a given boid then requires examining only the portion of the flock which is within the general vicinity.
A way to implement the second algorithm is using, for example, bin-lattice spatial subdivision. In bin-lattice spatial subdivision, a box-shaped region of space is divided into a collection of smaller boxes called "bins". For simplicity the edges of the big box, and the dividing planes which form the faces of the smaller boxes are all aligned with the global axes. At the beginning of the simulation, characters are distributed into bins based on their initial position. Each time they move they check to see if they have crossed into a new bin, and if so, update their bin membership. The big box is selected to surround the area of interest (in this case, the meadow in the pigeon's park). The number of subdivisions along each axis is selected to provide a good tradeoff between more precise localization of characters and less bin switching overhead as characters move.

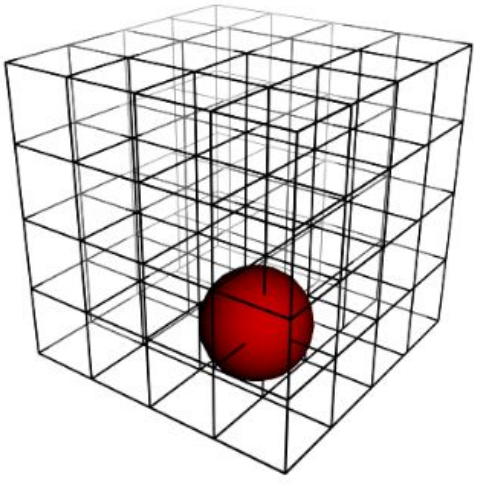


Figure 6: Bin-lattice spatial subdivision.

Implementations

First implementation in Symbolics Common Lisp: http://www.red3d.com/cwr/code/boids.lisp [Craig Reynolds]

Dead Reckoning

Dead reckoning is the process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time and course.

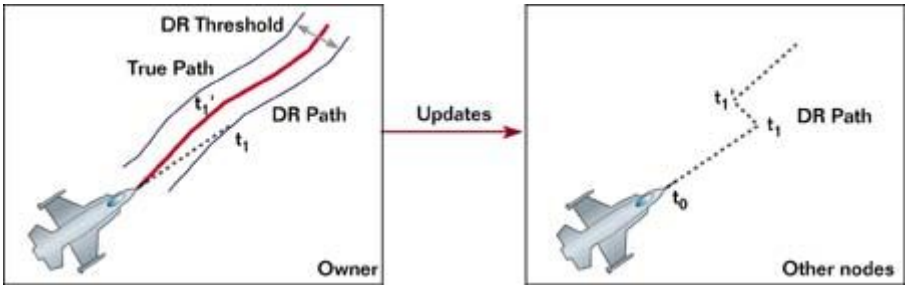


Figure 7: Example of Dead Reckoning, where we see the differences while having data sent over the network with low frequency.

Background

Networked games and simulation tools routinely use dead reckoning to predict where an actor should be right now, using its last known kinematic state (position, velocity, acceleration, orientation, and angular velocity).This is primarily needed because it is impractical to send network updates at the rate that most games run, 60 Hz.

Algorithm

The basic solution starts by projecting into the future using linear physics:

$$Position_{t1} = Position_{t0} \qquad (\#1)$$

$$Position_{t1} = Position_{t0} + \vec{v}(t1 - t0) \qquad (\#2)$$

$$Position_{t1} = Position_{t0} + \vec{v}(t1 - t0) + 1/2\vec{a}(t1 - t0) \qquad (\#3)$$

1. Updates position with the new position.
2. Extrapolates the position based on the current position and the speed.
3. Extrapolates the position using the current position, speed and acceleration.

This formula is used to move the object until a new update is received over the network. At that point, the problem is that there are now two kinematic states: the currently estimated position and the just received, actual position. Resolving these two states in a believable way can be quite complex. One approach is to create a curve (ex cubic Bézier splines, centripetal Catmull–Rom splines, and Hermite curves) between the two states while still projecting into the future. Another technique is to use projective velocity blending, which is

the blending of two projections (last known and current) where the current projection uses a blending between the last known and current velocity over a set time.

This allows smoother game experience as it deals with latency related problems in network games. [Wei Shi, Jean-Pierre Corriveau, and Jacob Agar]

Conclusions

This report has approached 3 of the main techniques used in agents in video games. It describes the different techniques (pathfinding, flocking and dead reckoning), as well as present and explain the algorithms used to implement these, since they solve common problems that programmers face when programming AI for video games.

References

General Ideas

https://prezi.com/sqx5t9bvaznb/agentes-inteligentes-en-los-videojuegos/ [Gabriel Lara]

Pathfinding

https://en.wikipedia.org/wiki/Pathfinding [Wikipedia]
https://visstaralax.wordpress.com/2014/01/27/capitulo-12-mallas-de-navegacion/ [visstaralax]
https://www.youtube.com/watch?v=TD5BGZ-W4-0 [euskopokalipsis]

Dead Reckoning

https://www.gamasutra.com/view/feature/131638/dead_reckoning_latency_hiding_for_.php [Jesse Aronson]
https://www.hindawi.com/journals/ijcgt/2014/138596/ [Wei Shi, Jean-Pierre Corriveau, and Jacob Agar]
https://en.wikipedia.org/wiki/Dead_reckoning#Autonomous_navigation_in_robotics [Wikipedia]

Flocking

https://en.wikipedia.org/wiki/Flocking_(behavior) [Wikipedia]
https://geeks.ms/jbosch/2009/10/26/xna-flocking-o-movimientos-colectivos [Jesús Bosch]
http://www.red3d.com/cwr/boids/ [Craig Reynolds]
http://www.red3d.com/cwr/papers/2000/pip.pdf [Craig Reynolds]