Inquiry: The University of Arkansas Undergraduate Research Journal

Volume 11 Article 18

Fall 2010

Research Note: Automated Path Finding Service for Second Life

Daniel Starling
University of Arkansas, Fayetteville

RESEARCH NOTE: AUTOMATED PATH FINDING SERVICE FOR SECOND LIFE

By Daniel Starling
Department of Computer Science and Computer Engineering

Faculty Mentor: Craig Thompson
Department of Computer Science and Computer Engineering

Second Life (a product from Linden Labs) is a 3D virtual world platform where one can create custom objects (houses, cars, pets, etc.) and embed logic in them through scripts, giving rise to a rich, interactive world made of user content. Human participants in the world are represented by "avatars" which can wander about freely in "regions" (also known as "islands") that subdivide the world into pieces that are hosted on servers on the Second Life (SL) grid (server farm located at Linden Labs). While humans have the necessary cognitive facilities to navigate this 3D environment, scripted objects do not. In particular, there are no facilities for path finding (e.g., moving from A to B via a set of waypoints that avoid obstacles). Were this feature to exist, it would become easier to implement realistic simulations and avatar-bots (avatars operated by programs instead of humans) inside of SL. As it is, every application needing this functionality implements a special case, primitive navigation for agents that move from place to place with very limited knowledge of their surroundings. This is due in part to resource concerns (e.g., CPU, memory overhead) on SL's servers.

Problem

Part of current research at the University of Arkansas concerns modeling healthcare logistics and medical workflows (e.g., medical procedures) within SL. This work falls under an umbrella project called "Everything is Alive" (EiA) [1]. In EiA, we suppose that pervasive computing exists and that objects are uniquely identifiable. Consequently, we always know where objects are located. The simulations that we carry out in SL often require moving an object (a wheelchair, a box of stents, etc.) using an agent (an avatar-bot) from location A to B. Typical static obstacles include hospital walls and medical equipment. Since SL lacks the ability to determine waypoints that avoid these obstacles (much less provide a reasonably short path), we needed to implement our own mechanism.

Path Finding Service

The solution we pursued is called the "path finding web service" (see [2] for more details). Agents in SL can contact this web service to navigate SL by providing their current location and where they want to go – the web service will return an appropriate list of waypoints that the agent can follow to avoid solid obstacles. Calculating waypoints from a 3D geometry model is a computationally intensive task, which is one reason why it is difficult to implement directly on SL's servers, which are already bustling with thousands of scripted objects. By offloading the task to our own server, we can partially avoid quota restrictions found in SL's Linden Scripting Language.

The Path Finding Algorithm

We use a path finding algorithm known as A*, which incrementally approaches an optimal solution through the use of a "cost" and "heuristic" function as it evaluates path choices. As A* searches through potential paths, it will evaluate a cost function that expresses distance traveled so far and then evaluates a heuristic function that approximates how close we are to the goal (in our case, it is approximated as a direct line between the current location and the destination). We originally used Djikstra's algorithm, which happens to be a special case of A* where there is no heuristic function (that is, the heuristic function would always evaluate to zero). This proved to be costly since the heuristic function is vital in guiding the search and avoiding the need to traverse the entire search space to find a solution – the heuristic function allows A* to "home in" on a solution.

Our algorithm based on A* works as follows:

1. Take a vertical slice of the world, allowing a reduction of 3D to 2D space. This yields rectangles derived from 3D bounding boxes of objects. See Figure 1.

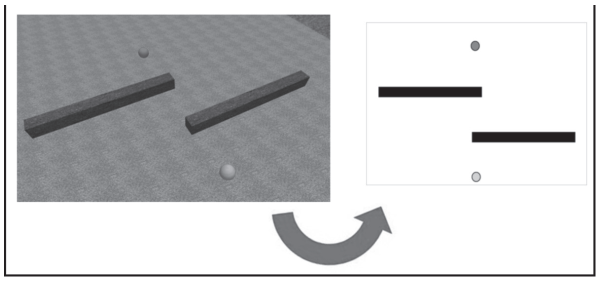


Figure 1. First step of the path finding algorithm: collapsing 3D to 2D space.

2. Expand rectangle sizes by some factor, which generally increases the amount of rectangle overlap. This avoids the problem of adjacent objects (e.g., walls) that have gaps between them (sometimes nearly imperceptible) that the algorithm would normally plot a path through, even though the object or avatar-bot that we are attempting to move through the area could not fit there. Additionally, this performs a similar duty in assuring that the entity we move through the world does not clip too close to corners or attempt to walk inside of a wall as it walks alongside it. See Figure 2.

3. Convert the rectangles into a graph, which is a data structure consisting of vertices (points) and connections between them (edges). This graph consists of the start and end points and every

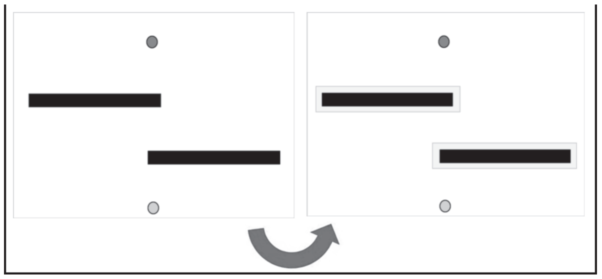


Figure 2. Second step of the path finding algorithm: rectangle size expansion.

vertex of each rectangle. Connectivity of vertices (the introduction of edges) is determined by line-of-sight reachability from one vertex to another. This is a quadratic time algorithm and becomes increasingly CPU-intensive as the number of vertices increases, sometimes rivaling A* in running time. An adjacency matrix, (a data structure that demonstrates efficient memory use for graphs with a large number of edges) stores the graph. See Figure 3.

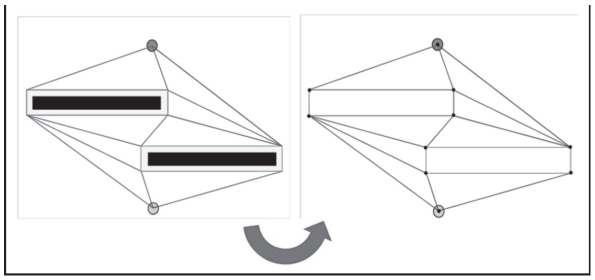


Figure 3. Third step of the path finding algorithm: conversion of rectangles and start/end points to a graph

4. Run the A* algorithm on our graph, which we have just framed as a classic single-pair shortest path problem. See Figure 4.

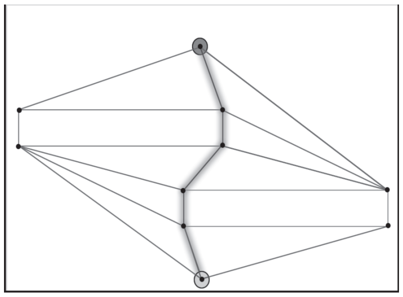
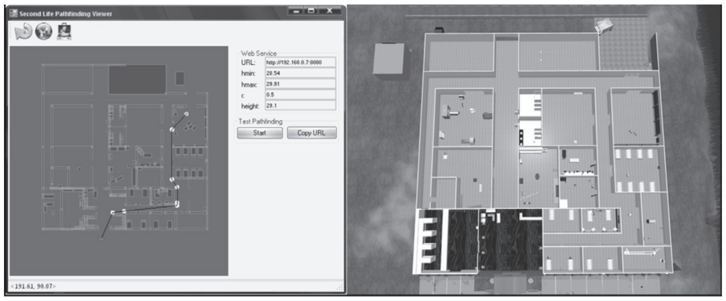


Figure 4. Fourth, final step of the path finding algorithm: apply A* algorithm to derive shortest path.

Use of the Service

The EiA project experiences considerable student churn. Undergraduate students may only be around for one or two semesters but contribute as much to the overall project as graduate students. The initial path finding web service was not very accessible to these students since it was not well-documented, relied on hard-coded parameters, and could not be updated readily as changes to layout in SL place were made (e.g., a new wall was introduced). In essence, it was not easy for new students to quickly pick up and start using the pathfinding web service in projects. Through revisions, we were able to address these issues so that a student who had just learned about Second Life (and its scripting language) could use the web service to compute paths in their own projects.

A tool was designed for student use (pictured on the left in Figure 5) that visually relates the effects of the parameters that they pass to the web server (for example, the rectangle expansion factor used in the overall algorithm). Students can also plot paths to review them for correctness and can also clear out areas of the island from the model. Re-scanning the island (done when changes take place) is just a matter of moving a special "scanner" object into position and activating it, after which it will contact the web service to report 3D bounding boxes in the world.



URL: http://192.168.0.7:8080
hmin: 28.54
hmax: 29.91
r: 0.5
height: 29.1

<191.61, 90.07>

Figure 5. A visual tool for inspecting the bounding boxes with which our web service is generating a path (left) and the corresponding SL hospital on the University of Arkansas island (right).

Our final step was to release this path finding service as a package that users can set up on their own servers and use.

Discussion of Limitations

While generally useful, our design is not without shortcomings. First, to calculate a path in SL, we must be fully aware of the geometry surrounding that navigation. SL places tight restrictions on geometry export – getting a model of an entire SL island out of the SL servers and into an external format is non-trivial. We settled on a less-than-precise method that employs SL's scripted "scanner" functionality to accumulate "bounding boxes" for each object found in the world. These bounding boxes roughly describe the space that the object takes up, although it can be wildly inaccurate in some instances (e.g., L-shaped objects that appear as giant box-like solids when their bounds are inspected). The process we developed requires some "manual effort" – our scripted "scanner" object has to periodically move throughout our region of the world to read in this data and report it back to the server. This process is time consuming due to SL scripting restrictions given limits, for example, on how fast a scanner object can move and query for world geometry. Our chosen approach can be unwieldy at times; objects moved by students or changes in building architecture often require that we be able to re-scan regions. Other issues such as inconsistencies in SL's sensor implementation make scanning in geometry all the more difficult.

The second shortcoming is not so much an issue with SL but rather an area for future work. As mentioned, path finding is a resource-intensive process. To simplify the problem and generate paths in a reasonably short period of time, we do not search for paths in the entire 3D world (or region) but rather collapse a vertical slice of it on the Z-axis into a 2D "overhead view" where we can apply the custom search algorithm to derive the shortest path (see Figure 5). Since our simulations are carried out on leveled-out hospital floors, this works sufficiently well, but our solution is not scalable to more complex geographies that involve vertical movement (e.g., paths that traverse stairs or windows). A better algorithm would implement A* in full 3D geometry with accommodation for picking paths that that are traversable by entities of varying size and limitation (e.g. gravity-bound objects).

Right now, all path computation takes place outside of SL. It is reasonable to assume that implementing path finding directly on Linden's servers is not currently feasible. Even if they were to create native, efficient functions for script-writers to use, the underlying algorithms would necessitate restrictive quotas that would likely render them less useful for the situations described in this paper.

Future Work

To make our functionality more available, we are considering the addition of the path finding service to OpenSimulator, an open-source alternative to Second Life. Hopefully, as Second Life evolves, it will offer features found in the AI component of current-generation 3D game engines, such as path finding. With such features, we would come a step closer to creating the realistic environment that Second Life's name suggests.

Acknowledgement

The author is indebted to undergraduate Nicholas Farrer, who developed the first generation path planner on which the current work is based.

References

[1] For papers on the Everything is Alive project, see http://vw.ddns.uark.edu/index.php?page=docs

[2] Starling, Daniel. "Pathfinding Extensions." <http://www.csce.uark.edu/%7Ecwt/COURSES/2009-08--CSCE-4613--AI/TERM-PROJECTS/FINAL-REPORT--Pathfinding--Starling.doc>.

[3] Farrer, Nick. "Second Life Robot Command Language." <http://vw.ddns.uark.edu/content/2009-02--SL-Robot-Command-Language-v0--Nicholas-Farrer.doc>.

Mentor Comments: Craig Thompson's students' work with Second Life and the Everything is Alive project appear in three locations in this journal, first in Eguchi's award-winning paper, next in Kumar's manuscript, and finally in this Research Note by Starling. Each article demonstrates the diversity of research possibilities with the EiA project.

For the past three years, my research has involved how to use 3D virtual worlds like Second Life to explore what the real world will be like when every physical object is a network object, with its own identity, behaviors, and the ability to communicate with humans and other objects. In the Fall 2009, I taught Artificial Intelligence. Daniel was one of my star students. In a previous semester, one of my students Nick Farrer, since graduated, had developed a virtual world robot assembly language that included software for finding paths from a start location to and end location in the virtual world. The system used a graph of way points and an efficient path finding algorithm for searching for a route. This allowed a robot to be tasked to go from A to B in a single command. Daniel took Nick's extensive code base, understood it (never easy), and then extracted the path finding code, and repackaging it as a separate modular service that can be used for many purposes. The ability to modularly build virtual worlds up from a set of modular services is a hot topic in the virtual world architecture community at present. Daniel went a step further – he also observed that our maps of waypoints become out of date fairly quickly. So he developed a closely related service that scans an area of a virtual world for obstacles that can include walls but also furnishings or doors. We now run this second service periodically to maintain our waypoint graph in a current state. Daniel wrote a paper based on his work "Automated Path Planning for Second Life" for the X10 Workshop on Extensible Virtual Worlds (http://vw.ddns.uark.edu/X10, March 29-30, 2010). Having graduated in May 2010, Daniel went to work for a startup software company to gain real world experience but he is interested in eventually returning to academics for a Masters or Ph.D.