The Great Wanderer: A Markov Approach to Autonomous Rovers

Wizardry and Steamworks [TBA][1]
e-mail: office@was.fm
[1]Wizardry and Steamworks, Consulting Group (http://was.fm)

December 7, 2014

Abstract

The Great Wanderer is an autonomous constant-memory rover based on weighted probabilistic automata. The rover has been prototyped, for convenience, in a Virtual World. This paper borrows concepts from automata theory, more precisely Markov chains, in order to show that a rover can be made to follow complicated pathways on its own with minimal expert guidance. The suggested mode of operation will hopefully be useful for practical scenarios such as cartography. The semantics are simple to understand while having in mind a symmetric and efficient design that would be easy to fabricate as a real application.

1 Introduction

One of the difficulties of autonomous rovers is that they lack the intuition, or are already too burdened in order to carry devices that allow them to adapt to the environment. A difficult reasoning problem lies in creating an autonomous rover that is able to traverse a space populated with obstacles, without prior knowledge of the placement of the objects and without human intervention. One solution is to keep track of previous coordinates and progressively explore, based on what has not been already visited. However, that implies that the storage space required to store the previous destinations

is proportional to the space that the rover has to explore. This may not be feasible in practical scenarios where the space to explore would require an unknown amount of storage space in order to keep record of all the previously visited locations. An example could be a rover that is designed to explore the bottom of the ocean floor, or a robot that should adapt to obstacles and unforeseeable pathways in space.

This paper describes a rover design, inspired by Markov (Мáрков, 1906) automata, that is able to explore a space containing arbitrarily-placed obstacles without expert intervention (Harmon & King, 1985; Simmons & Koenig, 1995) .We leverage concepts from automata theory (Hopcroft, Motwani, & Ullman, 1979) and statistics (Bar-Shalom, Li, & Kirubarajan, 2004) in order to introduce an automaton that is able to adapt to any surrounding environment. We offer experimental representations of collisions for a Great Wanderer rover that has been set-up to explore an enclosed space while unattended. Going one step further, the article explains that goal-based pathfinding can be implemented by just manipulating the list of probabilities given the relative position between the Great Wanderer and its desired destination.

The Great Wanderer was built in a Virtual World (Eschenbrenner, Nah, & Siau, 2008) (VW) called Second Life (Linden Lab, 2007), which is based on the Havok game engine (Kanode & Haddad, 2009) that implements physics, as well as offering an internal programming language called Linden Scripting Language (LSL). The decision to implement the Great Wonderer in a VW allowed us to abstract away from gravity by canceling out the force exerted on the Great Wanderer. Given the calculations presented in the Background 1 section, the rover does not require that level of abstraction because it is able to handle only a subset of directions. An advantage of LSL is that program states can be observed within the syntax of the programming language by using the `state` clause which makes the language an attractive programming environment. VWs have shown that they are capable of hosting simulator prototypes (L. & E., 2012) as well as providing an easily accessible platform for science and education (Virtual Islands for Better Education, 2011).

Background

The Great Wanderer automaton is inspired by Markov chains and uses probabilities to condition the transitions between states that correspond to the direction.

Definition 1.1. *Given a finite set D containing arbitrary directions:*

$$D = \{d_1, d_2, \ldots, d_n\}$$

and a corresponding set of probabilities P:

$$P = \{p_1, p_2, \ldots, p_n\}$$

such that $|D| = |P|$. The bijective function f:

$$f : P \mapsto D$$

maps elements from the set of probabilities P to the set of directions D in order.

The Great Wanderer automata does not strictly conform to the Markov property of a memoryless process (Saul & Jordan, 1999) because it does use memory to store probabilities. However, the advantage compared to a visitor system is that previous locations are not stored, but rather that the storage is of a fixed size and modified in-place whenever a direction is obstructed. Consequentially, the previously visited locations are stored within the entropy of the list of probabilities. The memory requirements (Olivera & Viera, 2007) are proportional to the directions that the Great Wanderer can move along rather than proportional to the space to be explored. This follows from Definition 1.1 because both the set of directions D and the set of probabilities P are finite and have the same cardinality, with f being an bijective function, such that adding directions to D would only linearly increase the size of the set of probabilities P. The choice of eliminating the requirement to store previously visited locations is more convenient because the size of the space to explore is, in many practical applications, unknown compared to the number of directions of movement that are finite.

By mapping the automata states to axes of movement of the Great Wanderer as shown in Figure 1, we can see the number of states is proportional to the number of directions. Every state `up`, `front`, `right`, `down`, `back` and `left` is accessible from the central `compute` state that decides to what state the next transition is made.

Internally, the Great Wanderer maintains a map of directions to probabilities that change whenever the Great Wanderer senses a collision (Zheng & Hemami, 1985) with an object on either senses of the x, y or z axes. In order to obtain the experimental results illustrated in this paper, the Great

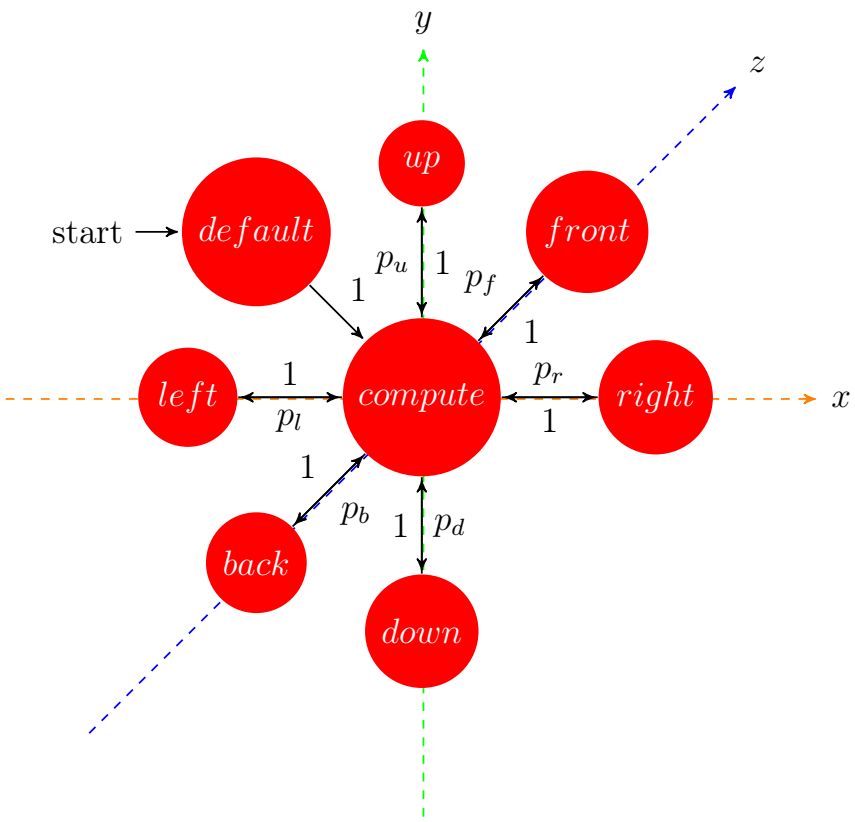

Figure 1: The automata that Great Wanderer uses is illustrated as projected on top of the directional axes x, y, z, representing the possible directions of movement of the Great Wanderer. The `default` state is used for setting up the initial parameters of the Great Wanderer, the `compute` state is used to compute the next direction and there is no end state.

Wanderer conveniently used collisions in order to sense obstacles. In a real world scenario, one can assume that no physical collision with an object is necessary and that sensors (Magori & Walker, 1987) could be used to detect a blocked path rather than damaging the rover.

Automata Design

The Great Wanderer autoamta in Figure 1 will initially divide the probabilities of movement equally between the states describing the possible directions of movement as shown in Figure 2. For six directions of movement all di-

rections are equally weighted, thus their probabilities of being chosen are initially equal.

```
[ up, front, right, down, back, left ]  ↦
[ 0.16, 0.16, 0.16, 0.16, 0.16, 0.16 ]
```

Figure 2: An initial configuration of the Great Wanderer with six possible directions of movement such that all directions are mapped to an initial set of equally weighted probabilities.

After detecting an object, the Great Wanderer returns to the `compute` state where a cumulative probability algorithm is used (Appendix 11) in order to select a direction. The prototype built in Second Life uses the range $[0, 1]$ for all probabilities in P as per Definition 1.2. Depending on the implementation, it is possible to scale the bounds to other values as long as the invariant I is preserved.

Definition 1.2. *Given a list of probabilities P, used by the Great Wanderer:*

$$P \quad = \quad \{p_1, p_2, \ldots, p_n\}$$

all the elements $p_i \in P$ are bounded such that:

$$\forall p_i \in P \quad : \quad \Re \mapsto [0, M]$$

where M is an upper bound constant, such that $M : \Re \mapsto \Re^+$.

The algorithm first sorts both the direction and the probability list using a Quicksort (Foley & Hoare, 1971) implementation (Appendix 10) in decreasing order, picks a random value $\rho : \Re \mapsto [0, 1]$ and then proceeds to add up the smallest probabilities until the randomly picked value ρ is exceeded (Appendix 11).

Definition 1.3. *Given any set of probabilities P:*

$$P \quad = \quad \{p_1, p_2, \ldots, p_n\}$$

sorted in increasing order such that $\forall p_i$, $p_i \leq p_{i+1}$, such that the sum of all elements in P will always add up to a constant value $\sum_i^n p_i = const$.

Definition 1.4. *Let there be a recursive function σ:*

$$\sigma(p_i) = \begin{cases} i,\ p_i \geq \rho \\ \sigma(p_i + p_{i-1}) \end{cases}$$

that will return the index i of the last summed probability p_i to exceed ρ. Based on the index i returned by the function σ and via Definition 1.1, the Great Wanderer will then pick the direction d_i as the next traveling direction.

In case the Great Wanderer collides along a direction, the probability for that direction will be decreased by a factor ϵ and then the remainder will be distributed equally to all other directions (Definition 1.5).

Definition 1.5. *Given a set of directions D:*

$$D = \{d_1, d_2, \ldots, d_n\}$$

and a set of probabilities P:

$$P = \{p_1, p_2, \ldots, p_n\}$$

If an obstacle is encountered along any direction $d_x \in D$, a coefficient $\epsilon : \Re \mapsto [0, p_x]$ is subtracted from p_x and redistributed to the other probabilities in P in order to obtain a new set of probabilities P':

$$\begin{aligned} P' &= \{p_x - \epsilon, p_{x+1} + \frac{\epsilon}{n-1}, \ldots, p_{x+n-1} + \frac{\epsilon}{n-1}\} \\ \epsilon &: \Re \mapsto [0, p_x] \end{aligned}$$

thereby decreasing the probability that the Great Wanderer will travel along the direction d_x.

By distributing the subtracted factor ϵ to the rest of probabilities in P, when a path is blocked, ensures that the sum of all probabilities for all directions will remain constant.

Lemma 1.1. *The Great Wanderer will always preserve the same invariant:*

$$I = \sum_{i=1}^{n} p_i = const$$

regardless of what collisions occur on any direction $d_i \in D$.

It is shown in Proof 1.1, by induction over the number of directions of movement, that the invariant from Lemma 1.1 holds regardless of the direction $d_i \in D$ along which a collision may have occurred.

Proof 1.1. *Given a finite set of directions D:*

$$D = \{d_1, d_2, \ldots, d_n\}$$

and a corresponding set of probabilities P (from Definition 1.1):

$$P = \{p_1, p_2, \ldots, p_n\}$$

Summing up the probabilities in P, we obtain I:

$$I = \sum_{i=1}^{n} p_i = const$$

Suppose that a collision occurs along direction d_1. Following Definition 1.5, we know that the resulting adjusted set P will become:

$$\begin{aligned} P_1 &= \{p_1 - \epsilon, p_2 + \frac{\epsilon}{n-1}, \ldots, p_n + \frac{\epsilon}{n-1}\} \\ \epsilon &: \Re \mapsto [0, p_x] \end{aligned}$$

Now, by summing up the elements of the resulting set P_1, we obtain $I \rightsquigarrow I_1$ after the collision:

$$\begin{aligned} I_1 &= \sum_{i=1}^{n} p_i = \\ &= p_1 - \epsilon + p_2 + \frac{\epsilon}{n-1} + p_3 + \frac{\epsilon}{n-1} + \ldots + p_n + \frac{\epsilon}{n-1} \\ &= p_1 + p_2 + p_3 + \ldots + p_n - \epsilon + (n-1)\frac{\epsilon}{n-1} \\ &= p_1 + p_2 + p_3 + \ldots + p_n \end{aligned}$$

which is equal to I before the collision even occurred. The same calculations apply by using Definition 1.5 for all directions in $d_i \in D$. The invariants I_1, I_2, ..., I_n are obtained and via Definition 1.3 we also know that all the invariants add to a constant, such that:

$$I_1 \equiv I_2 \equiv \ldots \equiv I_n \quad = \quad \sum_{i=1}^{n} p_i = const$$

which shows that Lemma 1.1 holds regardless on what direction a collision occurs.

The Great Wanderer conveniently checks its own correctness after every collision or seeding event in the `compute` state by adding the probabilities in the list and reporting any violation of the invariant to the user. The VW that the prototype was built in implements only mechanics and additionally the API provides a way via the `sensor` events to sense objects. For the built prototype, collisions were chosen instead of using the proximity detection features of the VW in order to simplify the code and reduce the stress imposed on the simulator (ie: by avoiding distance calculations between the Great Wanderer and the obstacles). In a real world scenario various types of sensors could be used to distinguish between various types of obstacles. Given different types of detectors ϵ could represent a composition of multiple parameters depending on the type of the application and the environment that the Great Wander must explore.

After successive collisions on a certain direction, the probability for that direction decreases such that the Great Wanderer is more likely to choose a direction that has experienced less collisions than the current one.

Case Example

Suppose that at some point the map of direction to probabilities is the one illustrated in Figure 3.

```
[ up, front, right, down, back, left ]  ↦
[ 0.23, 0.12, 0.45, 0.01, 0.12, 0.07 ]
```

Figure 3: An arbitrary configuration of the Great Wanderer at some point in time.

The Great Wanderer uses a modified Quicksort algorithm to preserve the mapping of directions to probabilities and to sort both lists as shown in Firgure 4.

$$\texttt{[right, up, front, back, left, down]} \mapsto$$
$$\texttt{[0.45, 0.23, 0.12, 0.12, 0.07, 0.01]}$$

Figure 4: An arbitrary configuration of the Great Wanderer at some point in time.

Now that the map is sorted, the Great Wanderer picks a random number ρ in the interval $[0, 1]$. Let us suppose that value $\rho = 0.32$. The Great Wanderer starts to compare and add-up the smaller probabilities until they exceed the value of ρ as can be seen in Figure 5.

$$\begin{aligned}
\texttt{[down]} \mapsto \texttt{[0.01]} &\not\geq \rho = 0.32 \\
\texttt{[left]} \mapsto \texttt{[0.01 + 0.07]} &\not\geq \rho = 0.32 \\
\texttt{[back]} \mapsto \texttt{[0.01 + 0.07 + 0.12]} &\not\geq \rho = 0.32 \\
\texttt{[front]} \mapsto \texttt{[0.01 + 0.07 + 0.12 + 0.12]} &= \rho = 0.32
\end{aligned}$$

Figure 5: The Great Wanderer adds-up the smaller probabilities following the sorted map from Figure 4 and stops when the value exceeds a randomly generated number in the range $[0, 1]$ in order to determine the next direction.

Since the last direction `front` is greater or equal to ρ, the Great Wanderer would, in this case, commute to the `front` state and move front until the next occurring collision. When the next collision occurs, the front probability is decreased (Definition 1.5) in the map between directions and probabilities and the Great Wanderer commutes to the `compute` state and the same procedure repeats.

The reason for choosing the suggested cumulative probability algorithm from Definition 1.4 is that the directions with the smallest probabilities are still given a chance to be selected. This is particularly useful because it allows the Great Wanderer to overcome obstacles while still maintaining the current direction of travel. A good example thereof would be an object placed in the current traveling direction of the Great Wanderer. When a collision occurs, if the Great Wanderer would select only the highest probability, then the Great Wanderer would move away from the obstacle in any opposed

sense. By giving directions with lower probabilities a chance to be selected, the resulting behavior is that the Great Wanderer attempts to overcome the obstacle by moving along its surface.

2 Pathfinding

In order to make the Great Wanderer reach a precise destination the list of probabilities can be seeded in order to exert an overall tendency of movement.

Definition 2.1. *Inducing a tendency of movement along a direction is the converse of a collision along that direction. Given a finite set of directions D:*

$$D = \{d_1, d_2, \ldots, d_n\}$$

and a corresponding set of probabilities P (from Definition 1.1):

$$P = \{p_1, p_2, \ldots, p_n\}$$

in order to induce a tendency along any direction $d_x \in D$, a coefficient $\epsilon : \Re \mapsto [0, p_x]$ is added to p_x and subtracted from all the other probabilities in P, such that after the collision the new set of probabilities $P \rightsquigarrow P'$ becomes:

$$\begin{aligned} P' &= \{p_x + \epsilon, p_{x+1} - \frac{\epsilon}{n-1}, \ldots, p_{x+n-1} - \frac{\epsilon}{n-1}\} \\ \epsilon &: \Re \mapsto [0, M] \end{aligned}$$

Definition 1.5 is the converse of Definition 2.1, however the former refers to collisions and the latter refers to course-corrections. The advantage of contextually splitting collisions from seed events is that an overall movement tendency can be imposed while still allowing the Great Wanderer to avoid objects and autonomously perform fine course-corrections.

Lemma 2.1. *The Great Wanderer will always preserve the same invariant:*

$$I = \sum_{i=1}^{n} p_i = const$$

regardless of what seeding events are requested for any direction $d_i \in D$.

Proof 2.1. *Given a finite set of directions D:*

$$D = \{d_1, d_2, \ldots, d_n\}$$

and a corresponding set of probabilities P (from Definition 1.1):

$$P = \{p_1, p_2, \ldots, p_n\}$$

Summing up the probabilities $\forall p_i \in P$, we obtain I:

$$I = \sum_{i=1}^{n} p_i = const$$

which is constant due to Definition 1.3.

Suppose that a seeding event is requested along direction d_1. Following Definition 2.1, we know that the resulting adjusted set P will become:

$$\begin{aligned} P_1 &= \{p_1 + \epsilon, p_2 - \frac{\epsilon}{n-1}, \ldots, p_n - \frac{\epsilon}{n-1}\} \\ \epsilon &: \Re \mapsto [0, p_x] \end{aligned}$$

Now, by summing up the elements of the resulting set P_1, we obtain $I \rightsquigarrow I_1$ after the collision:

$$\begin{aligned} I_1 &= \sum_{i=1}^{n} p_i = \\ &= p_1 + \epsilon + p_2 - \frac{\epsilon}{n-1} + p_3 - \frac{\epsilon}{n-1} + \ldots + p_n - \frac{\epsilon}{n-1} \\ &= p_1 + p_2 + p_3 + \ldots + p_n + \epsilon - (n-1)\frac{\epsilon}{n-1} \\ &= p_1 + p_2 + p_3 + \ldots + p_n \end{aligned}$$

which is equal to I before the seeding event was requested. The same calculations apply by using Definition 1.5 for all directions in $d_i \in D$. The invariants I_1, I_2, ..., I_n are obtained such that:

$$I_1 \equiv I_2 \equiv \ldots \equiv I_n = \sum_{i=1}^{n} p_i = const$$

which shows that Lemma 2.1 holds regardless on what direction a collision occurs.

It can be shown symmetrically to Proof 1.1 that after a seeding event, the probabilities still add up to a constant value and that the invariant from Lemma 2.1 will hold. For seeding events along a certain direction $d_i \in P$ to influence the movement, we must show that after applying ϵ to d_i the Great Wanderer will indeed follow the direction d_i.

Lemma 2.2. *After successively applying a seeding factor ϵ to a certain probability p_x in the list of probabilities P, the Great Wanderer will select the direction d_x corresponding to the probability p_x as the next direction to travel in.*

In order to prove Lemma 2.2, we follow the same procedure for collisions from Proof 1.1 and show that by adding ϵ to p_x, eventually the Great Wanderer will choose d_x as the next traveling direction, which is shown in Proof 2.2.

Proof 2.2. *Let there be a set of directions D:*

$$D = \{d_x, d_2, \ldots, d_n\}$$

and a corresponding set of probabilities P (from Definition 1.1):

$$P = \{p_x, p_2, \ldots, p_n\}$$

The Great Wanderer should follow the direction $d_x \in D$ that corresponds to the probability $p_x \in P$. From Definition 2.1, we know that after one addition of a factor ϵ to a probability p_x, the new set of probabilities $P \rightsquigarrow P'$ will have the form:

$$\begin{aligned} P' &= \{p_x + \epsilon, p_{x+1} - \frac{\epsilon}{n-1}, \ldots, p_n - \frac{\epsilon}{n-1}\} \\ \epsilon &: \Re \mapsto [0, M] \end{aligned}$$

Following Definition 2.1, after adding ϵ, m times to the probability p_x, we have a new probability set $P \rightsquigarrow P_m$:

$$\begin{aligned} P_m &= \{p_x + m\epsilon, p_{x+1} - m\frac{\epsilon}{n-1}, \ldots, p_{x+n} - m\frac{\epsilon}{n-1}\} \\ \epsilon &: \Re \mapsto [0, M] \end{aligned}$$

We have to show that eventually the Great Wanderer will follow the direction corresponding to the first element of P_m. Thus, we apply $m \mapsto \infty$ to P_m and obtain the new set of probabilities $P_m \rightsquigarrow P_\infty$:

$$\begin{aligned} P_\infty &= \{\lim_{m\mapsto\infty}(p_x + m\epsilon), \lim_{m\mapsto\infty}(p_{x+1} - m\frac{\epsilon}{n-1}), \ldots, \\ &\quad \lim_{m\mapsto\infty}(p_{x+n-1} - m\frac{\epsilon}{n-1})\} = \\ &\quad \{\lim_{m\mapsto\infty}(p_x + m\epsilon), 0, \ldots, 0\} \end{aligned}$$

However, from Definition 1.2 we know that the upper bound for all elements of P must be M, such that we can reduce P_∞ to:

$$P_\infty = \{M, 0, .., 0\}$$

which means that the selector function σ from Definition 1.4 can only select $p_x = M$, all other probabilities being zero. Thus the travel direction d_x corresponding to the probability p_x will be selected via Definition 1.1, which is what we had to prove.

In order to get from a point in space $A(x_2, y_2, z_2)$ to a point $B(x_1, y_1, z_1)$ one would need to reach the final conditions $x_2 = x_1$, $y_2 = y_1$ and $z_2 = z_1$. Assuming that travel on a direct path between point A and point B is not possible, for example if the path is obstructed with obstacles, the probability to select a favorable direction towards point B can be performed by seeding the set of probabilities P (Definition 2.1). Although the article focuses on the engines rather than orientation or pathfinding, a sketch of a possible seeding algorithm is provided in Figure 6.

Symmetric to collisions, given Definition 2.1, the correction value ϵ could have some encoded meaning, built dynamically depending on the type of sensors that the rover is equipped with or the environment that the Great Wanderer is meant to explore.

The Great Wanderer does not directly offer a method of orientation. Instead the constant-memory engine that can be used with more elaborate pathfinding algorithms. The purpose of the experiments illustrated in Figures 8 and 9 is to show the rover's ability to adapt to the surroundings, rather than reach a given destination. The advantage of the probabilistic pathfinding compared to feedback or expert-based systems where a rover makes decisions based on a set of human-assisted inputs, is that the Great

```
let destination := B(x_1, y_1, z_1)
let D := [up, back, down, left, right]
while x_1 ≠ x_2 and y_1 ≠ y_2 and z_1 ≠ z_2
 let x_2 := current(x) and y_2 := current(y) and z_2 := current(z)
 if x_1 > x_2 seed(left)
 if x_1 < x_2 seed(right)
 if y_1 > y_2 seed(down)
 if y_1 < y_2 seed(up)
 if z_1 > z_2 seed(back)
 if z_1 < z_2 seed(front)
```

Figure 6: A generic algorithm for implementing a conditioned probabilistic movement of the Great Wanderer. When executed, all cases are fall-through such that multiple corrections can be applied one-shot.

Wanderer grants some autonomy to the rover when dealing with the finer details of the space to explore. The algorithm from Figure 6 does not bother about fine-grain course-corrections but rather seeds the list of probabilities to induce a tendency of movement towards the destination. The result is that the rover does move towards the destination but may choose its own trajectory in order to avoid obstacles.

3 Extending the Great Wanderer

It is observable that the example given in Figure 1 is generalizable and that the Great Wanderer is not restricted to only `up`, `down`, `left`, `right`, `front` and `back` states. It is possible to generalize the Great Wanderer to any number of directions on a spherical surface by extending the automata with states for as many angles of a sphere as necessary. This makes the Great Wanderer illustrated in Figure 1 a particular case of a generalized Great Wanderer with six possible senses of movement.

Lemma 3.1. *The Great Wanderer can be extended to any number of directions such that it will preserve the same invariant I for any collision or seeding event.*

Proof 3.1. *In Proof 1.1, we conclude that:*

$$I_1 \equiv I_2 \equiv \ldots \equiv I_n \;\; = \;\; \sum_{i=1}^{n} p_i = const$$

In order to show that the Great Wanderer can be extended to include any number of directions, we have to show that by extending the list of directions D and, implicitly via Definition 1.1, the list of probabilities P such that $n \rightsquigarrow n+1$ does not violate the invariant after a collision or a seeding event.

Given a finite set of directions D:

$$D \;\; = \;\; \{d_1, d_2, \ldots, d_n\}$$

and a corresponding set of probabilities P (from Definition 1.1):

$$P \;\; = \;\; \{p_1, p_2, \ldots, p_n\}$$

From Definition 1.3 we know that:

$$I \;\; = \;\; \sum_{i=1}^{n} p_i = const$$

We thus extend the sets D, respectively P by applying $n \rightsquigarrow n+1$ such that we obtain the new set of directions $D \rightsquigarrow D'$:

$$D' \;\; = \;\; \{d_1, d_2, \ldots, d_n, d_{n+1}\}$$

and a corresponding set of probabilities $P \rightsquigarrow P'$ (Definition 1.1):

$$P' \;\; = \;\; \{p_1, p_2, \ldots, p_n, p_{n+1}\}$$

Summing up the probabilities in the extended set P', we obtain I':

$$I' \;\; = \;\; \sum_{i=1}^{n+1} p_i = const$$

which via Definition 1.3 is constant, and has to be maintained for collision and seeding events.

Expanding the sum, we obtain:

$$I' \;\; = \;\; \sum_{i=1}^{n+1} p_i = p_1 + p_2 + \ldots + p_n + p_{n+1}$$

and obeserve that $p_1 + p_2 + \ldots + p_n = \sum_{i=1}^{n} p_i$ *is equal to* I*. Rewriting the equation, we have:*

$$I' \;=\; I + p_{n+1}$$

Previously we have shown in Proof 1.1 and Proof 2.1 that the invariant I *is maintained for both collisions and seeding events. We now have to show that the transformed invariant* I' *of the extended Great Wanderer is maintained for both collision and seeding events.*

After a collision along direction d_1*, via Definition 1.5, we have* $I' \rightsquigarrow I'_1$*:*

$$\begin{aligned} I'_1 &= \sum_{i=1}^{n+1} p_i = \\ &= p_1 - \epsilon + p_2 + \frac{\epsilon}{n} + p_3 + \frac{\epsilon}{n} + \ldots + p_n + \frac{\epsilon}{n} + p_{n+1} + \frac{\epsilon}{n} \\ &= p_1 + p_2 + p_3 + \ldots + p_n - \epsilon + n\frac{\epsilon}{n} + p_{n+1} \\ &= p_1 + p_2 + p_3 + \ldots + p_n + p_{n+1} \\ &= I + p_{n+1} \end{aligned}$$

The same procedure can be applied $\forall d_i \in D$ *obtaining the invariants:*

$$I'_1 \equiv I'_2 \equiv \ldots \equiv I'_{n+1} \equiv I' \;=\; I + p_{n+1}$$

which allows us to conclude that the invariant is preserved for collisions when the Great Wanderer is extended.

Symmetrically, for a seeding event along direction d_1*, via Definition 2.1, we have the new transformed invariant* $I' \mapsto I_1$*:*

$$\begin{aligned} I'_1 &= \sum_{i=1}^{n+1} p_i = \\ &= p_1 + \epsilon + p_2 - \frac{\epsilon}{n} + p_3 - \frac{\epsilon}{n} + \ldots + p_n - \frac{\epsilon}{n} + p_{n+1} - \frac{\epsilon}{n} \\ &= p_1 + p_2 + p_3 + \ldots + p_n + \epsilon - n\frac{\epsilon}{n} + p_{n+1} \\ &= p_1 + p_2 + p_3 + \ldots + p_n + p_{n+1} \\ &= I + p_{n+1} \end{aligned}$$

The same procedure applies for seeding events $\forall d_i \in D$ *obtaining the same constant* I' *such that:*

$$I'_1 \equiv I'_2 \equiv \ldots \equiv I'_{n+1} \equiv I' \;=\; I + p_{n+1}$$

This proves that when the Great Wanderer is extended with new directions, both collisions and seeding events will preserve the invariant I', which is what we had to prove.

4 Experimental Results

A possible practical application is to use the Great Wanderer to trace enclosures. Cartography (Reichenbacher, 2003) is an application for the Great Wanderer that demonstrates its usefulness because the space to explore is unknown. When there is no previous knowledge of the space to explore, it is difficult to calculate the resource requirements of a rover. In Second Life, we have taken a $4.5m \times 4.5m \times 4.5m$ enclosure, filled with an ϵ-shaped object and placed the Great Wanderer inside (Figure 7). The shape of the Great Wanderer was selected as a cube with dimensions $0.5m \times 0.5m \times 0.5m$ such that each face corresponds to a state and the related direction.

The coordinate of every collision was stored in order to render Figure 8. Every plotted point point represents a coordinate of a collision between Great Wanderer and the enclosure or the ϵ-shaped obstacle inside the enclosure.

The extremities of the collision points represent the edges of the enclosure. The corners of the enclosure (Figure 8) are more pronounced because the Great Wanderer has fewer escape routes available in case the number of escape directions are smaller than the total number of directions possible. Otherwise, the Great Wanderer will collide with a surface again (Yoshida, Maruki, & Yano, 2002).

Figure 9 illustrates a trace (dashed line) along the clusters of collision points in order to highlight the contour of the ϵ-shaped obstacle. The error for both Figure 8 and Figure 9 is $\xi = 2.5e^{-1}$ due to the fact that collisions are registered from the centre of the Great Wanderer. The VW that hosted the Great Wanderer prototype allows objects to have a minimal dimension of $.01m$ on all axes such that the errors could be reduced to $\xi_{min} = 5e^{-3}$ or even eliminated if the error is subtracted from the direction of collision.

5 Discussion

The Great Wanderer presents a rover that is able to adapt to its environment without intervention and without using increasing storage in order to track

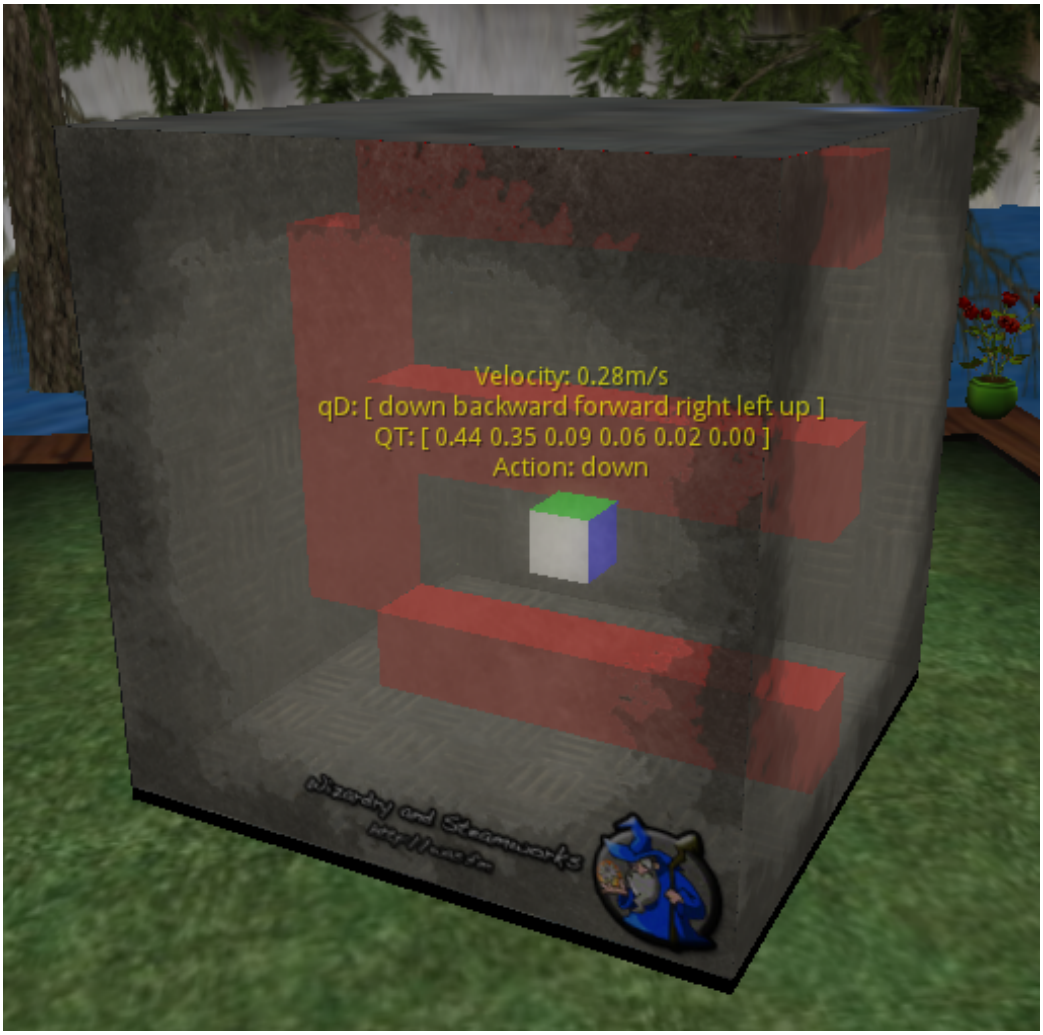

Figure 7: This screenshot shows the Great Wanderer placed within an enclosure containing an ϵ-shaped obstacle. The Great Wanderer's faces are colored corresponding to the color of the positive axes of the VW. The overhead text displays the current traveling velocity of the Great Wanderer, the list of directions `qD`, the list of probabilities QT and the selected direction of travel.

previously visited locations. The design contrasts well with other robots that require storage that is inevitably proportional to the space to explore (Bender & Slonim, 1994). As shown in the Background 1 section, the Great Wanderer could be extended to include any other axes of movement. The prototype

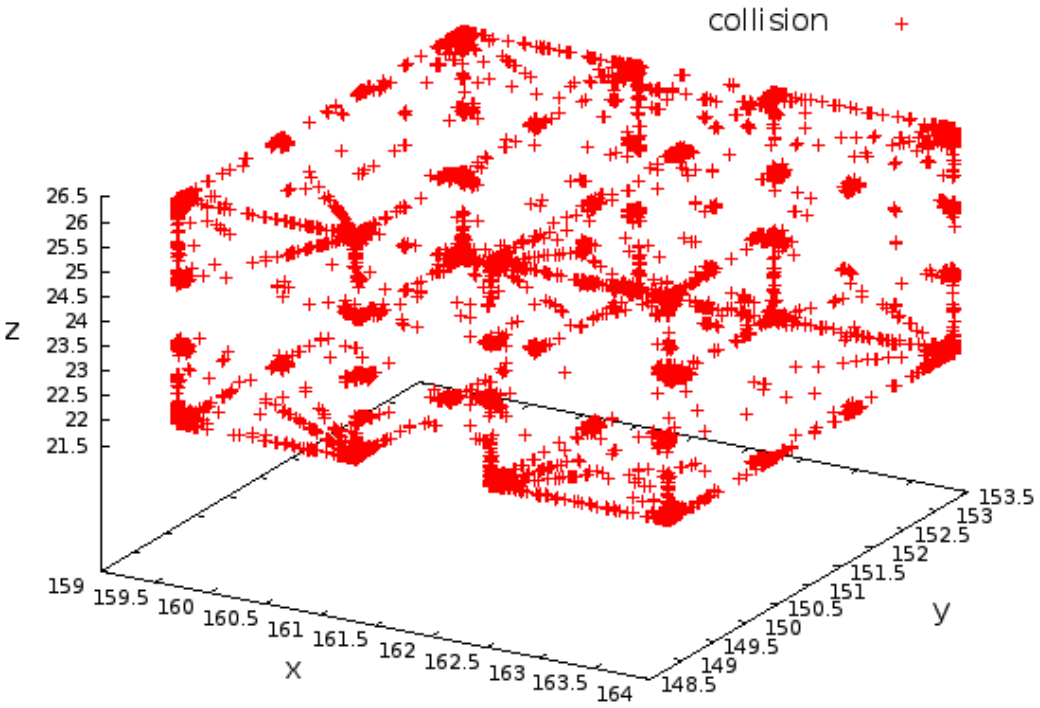

Figure 8: Plotting unique collisions between the Great Wanderer and the environment gives us an overview of the enclosure's interior. Clusters points indicate Great Wanderer's successive attempts to maneouver around difficult objects where more than one escape direction is blocked. It is perhaps interesting to observe that the collisions are uniformly distributed although the enclosure only provided two $0.5m$ apertures that allowed the Great Wanderer to pass from one side to the other.

presented in this paper uses six senses, corresponding to three axes but it is possible to extend the movement to any number of axes. In case the Great Wanderer were to be extended to include other axes its shape would be altered as well. Following the symmetry, by gradually adding faces, the shape of the Great Wanderer will converge to a sphere.

Paths are chosen based on what previous collisions occurred, out of a fixed-list of probabilities that is conditioned by the obstacles that the Great Wanderer encounters. Related research consists in establishing a map such that the rover is aware of the environment (Zender, Martínez Mozos, Jensfelt, Kruijff, & Burgard, 2008). For the Great Wanderer, that information is to be found implicitly in the probability map and is an expression of experience-driven pathfinding (Murillo, Košecká, Guerrero, & Sagüés, 2008). Compared

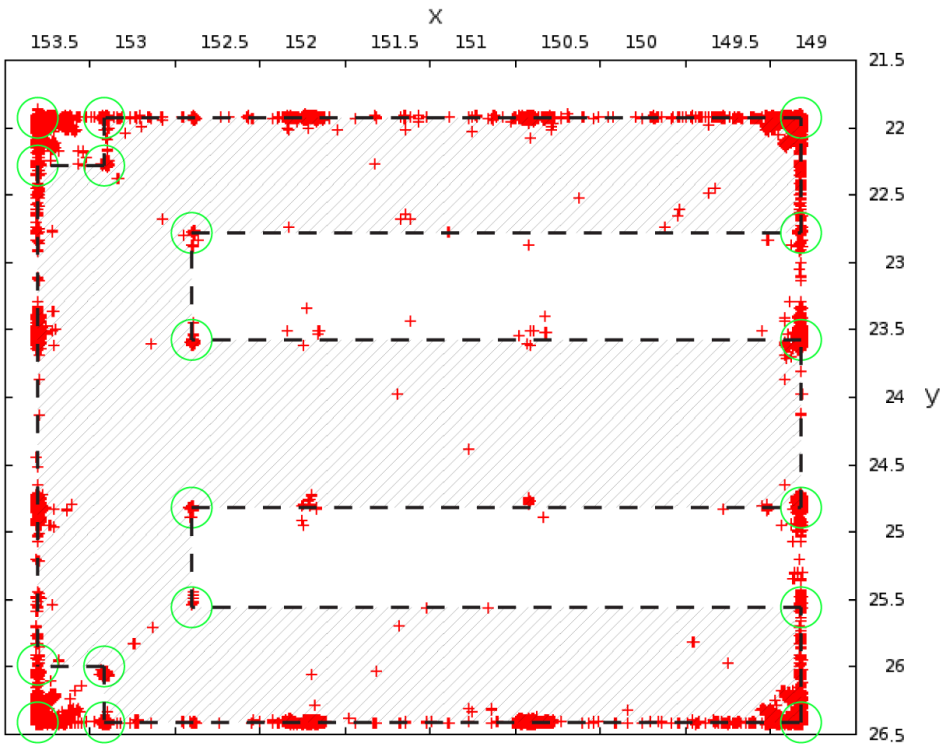

Figure 9: By projecting all the collected coordinates of collisions in the $\widehat{yOz}$ plane the ϵ-shaped obstacle can be observed due to the clustering of points (marked with green). On the left side of the figure we observe the two apertures, approximately $0.5m \times 0.5m$, corresponding to the shape of the ϵ-shaped obstacle in Firgure 7 that allowed the Great Wanderer to pass from one half of the enclosure to the other.

to a hardcoded or a transmitted path, the Great Wanderer is able to probabilistically determine its own direction based on what choice is best for a given environment.

For the scope of the experiment, coordinates were sent via HTTP request to a web-server and collected in a database. Only a single Great Wanderer was used to map the interior of the enclosing cube, however it is feasible to use multiple Great Wanderers to collaboratively map an enclosure (Yamauchi, 1998; Grabowski, Navarro-Serment, Paredis, & Khosla, 2000). Under the assumption that collisions between Great Wanderers are made distinguishable, then each data-set from each Great Wanderer would represent an addition

and refinement of the previously collected data. The concept is similar to techniques in distributed data gathering using exploration rovers (Ren & Sorensen, 2008).

6 Conclusions

Virtual Worlds have the ability to offer the immediate next step to theoretical approaches on robotics and engineering. The Great Wanderer is a physical object that is governed by the laws of motion and dynamics that the physics engine implements. The Havok engine is, to the date of this article, one of the more advanced graphics engine that is known for its advanced real-time collision and dynamics of rigid bodies. Later, the prototype was moved to OpenSimulator and ran on both the Open Dynamics Engine (Drumwright, Hsu, Koenig, & Shell, 2010) and the Bullet engine (Real-Time Physics Simulation, n.d.). The outcome of the experiment did not differ between engines although some workarounds had to be applied due to OpenSimulator's partial support of ODE and Bullet physics. Propulsion is achieved by applying a velocity vector using LSL scripting. It should be mentioned that the prototype is a physical object and that the coding does not conveniently switch between physical and non-physical movement.

The first Great Wanderer was built an year to the date of this paper and is still exploring the enclosed space that was built for the experiments on the University of New Orleans virtual campus. The source-code is publicly available (Wizardry and Steamworks, 2011) and licensed under the GNU GPLv3 (Foundation, 2007) to the Wizardry and Steamworks group.

7 Acknowledgments

The authors would like to thank Stephen L. Gasior, University of New Orleans, of the Biological Interactive Objects for Science Education (BIO-SE) group for the continuous support, advice and for supplying the necessary the land on which we built the initial Great Wanderer prototype. The authors would like to thank both Stephen L. Gasior and Lazaros Papadopoulos for their helpful comments and suggestions to this paper.

References

Bar-Shalom, Y., Li, X., & Kirubarajan, T. (2004). *Estimation with applications to tracking and navigation: Theory algorithms and software*. Wiley.

Bender, M., & Slonim, D. (1994, nov). The power of team exploration: two robots can learn unlabeled directed graphs. In *Foundations of computer science, 1994 proceedings., 35th annual symposium on* (p. 75 -85). doi: 10.1109/SFCS.1994.365703

Drumwright, E., Hsu, J., Koenig, N., & Shell, D. (2010). Extending open dynamics engine for robotics simulation. In *Proceedings of the second international conference on simulation, modeling, and programming for autonomous robots* (pp. 38–50). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=1947545.1947554`

Eschenbrenner, B., Nah, F. F.-H., & Siau, K. (2008). 3-d virtual worlds in education: Applications, benefits, issues, and opportunities. *J. Database Manag.*, *19*(4), 91-110. Retrieved from `http://dblp.uni-trier.de/db/journals/jdm/jdm19.html#EschenbrennerNS08`

Foley, M., & Hoare, C. A. R. (1971). Proof of a recursive program: Quicksort. *Comput. J.*, *14*(4), 391-395. Retrieved from `http://dblp.uni-trier.de/db/journals/cj/cj14.html#FoleyH71`

Foundation, T. F. S. (2007, June). *Gnu general public license, version 3.* `http://www.gnu.org/licenses/gpl.html`. (Last retrieved 2012-05-10)

Grabowski, R., Navarro-Serment, L., Paredis, C., & Khosla, P. (2000). Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, *8*, 293-308. Retrieved from `http://dx.doi.org/10.1023/A%3A1008933826411` doi: 10.1023/A:1008933826411

Harmon, P., & King, D. (1985). *Expert systems.* New York, NY, USA: John Wiley & Sons, Inc.

Hopcroft, J., Motwani, R., & Ullman, J. (1979). *Introduction to automata theory, languages, and computation* (Vol. 2). Addison-wesley Reading, MA.

Kanode, C., & Haddad, H. (2009, april). Software engineering challenges in game development. In *Information technology: New generations, 2009. itng '09. sixth international conference on* (p. 260 -265). doi: 10.1109/ITNG.2009.74

L., P., & E., C. (2012, April). Simulating cardiac electrical activity in open-metaverses. In *Proceedings of the 1st medical education informatics conference, thessaloniki, greece.*

Linden Lab. (2007). *Second life website.* http://www.secondlife.com.

Magori, V., & Walker, H. (1987). Ultrasonic presence sensors with wide range and high local resolution. *IEEE Trans Ultrason Ferroelectr Freq Control*, *34*(2), 202-11. Retrieved from http://www.biomedsearch.com/nih/Ultrasonic-presence-sensors-with-wide/18290110.html

Ма́рков, А. А. (1906). Распространение закон больших чисел на величину, зависящие друг от друга. *Известия Физико-математического общества при Казанском университете*, *15*, 135—156.

Murillo, A. C., Košecká, J., Guerrero, J. J., & Sagüés, C. (2008, June). Visual door detection integrating appearance and shape cues. *Robot. Auton. Syst.*, *56*(6), 512–521. Retrieved from http://dx.doi.org/10.1016/j.robot.2008.03.003 doi: 10.1016/j.robot.2008.03.003

Olivera, A., & Viera, O. (2007). Adaptive memory programming for the vehicle routing problem with multiple trips. *Computers and OR*, *34*(1), 28-47. doi: http://dx.doi.org/10.1016/j.cor.2005.02.044

Real-Time Physics Simulation. (n.d.). *Bullet physics library.* http://bulletphysics.org.

Reichenbacher, T. (2003). *Abstract adaptive methods for mobile cartography.* Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.4698

Ren, W., & Sorensen, N. (2008, April). Distributed coordination architecture for multi-robot formation control. *Robot. Auton. Syst.*, *56*(4), 324–333. Retrieved from http://dx.doi.org/10.1016/j.robot.2007.08.005 doi: 10.1016/j.robot.2007.08.005

Saul, L. K., & Jordan, M. I. (1999). Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, *37*, 75-87. Retrieved from http://dx.doi.org/10.1023/A%3A1007649326333 doi: 10.1023/A:1007649326333

Simmons, R., & Koenig, S. (1995, July). Probabilistic robot navigation in partially observable environments. In *Proceedings of the international joint conference on artificial intelligence (ijcai '95)* (p. 1080 - 1087).

Virtual Islands for Better Education. (2011). *Organization website.* http://wiki.bio-se.info.

Wizardry and Steamworks. (2011). *Great wanderer project.* http://was.fm/

secondlife:great_wanderer.

Yamauchi, B. (1998). Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on autonomous agents* (pp. 47–53). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/280765.280773 doi: 10.1145/280765.280773

Yoshida, K., Maruki, T., & Yano, H. (2002, October). A novel strategy for asteroid exploration with a surface robot. In *The second world space congress.* 34th COSPAR Scientific Assembly.

Zender, H., Martínez Mozos, O., Jensfelt, P., Kruijff, G. J. M., & Burgard, W. (2008, June). Conceptual spatial representations for indoor mobile robots. *Robot. Auton. Syst.*, *56*(6), 493–502. Retrieved from http://dx.doi.org/10.1016/j.robot.2008.03.007 doi: 10.1016/j.robot.2008.03.007

Zheng, Y., & Hemami, H. (1985). Mathematical modeling of a robot collision with its environment. *Journal of Robotic Systems*, *2*, 289-307.

A Appendix

```
list dualQuicksort(list a, list b) {

  if(llGetListLength(a) <= 1) return a+b;

  list pivot_a =
    llList2List(a, llGetListLength(a)/2, llGetListLength(a)/2);
  list pivot_b =
    llList2List(b, llGetListLength(b)/2, llGetListLength(b)/2);

  a =
    llDeleteSubList(a, llGetListLength(a)/2, llGetListLength(a)/2);
  b =
    llDeleteSubList(b, llGetListLength(b)/2, llGetListLength(b)/2);

  list less = [];
  list less_b = [];
  list more = [];
  list more_b = [];

  integer i;
  for(i=0; i<llGetListLength(a)*2; i++) {
      if(stringComparer(llList2List(a, i, i), pivot_a) == TRUE)
      {
          less += llList2List(a, i, i);
          less_b += llList2List(b, i, i);
      }
      else
      {
          more += llList2List(a, i, i);
          more_b += llList2List(b, i, i);
      }
  }
  return dualQuicksort(less, less_b) +
    (list)pivot_a + (list)pivot_b +
    dualQuicksort(more, more_b);
}
```

Figure 10: The dual-Quicksort implementation in the Linden Scripting Language (LSL) sorts the list of directions relative to their probabilities.

```
// Now, compute next hop based on the highest probability in the list
// and switch to first state, front, back, left, right, up or down.

list P = dualQuicksort(QT, qD);
list dirs = llList2ListStrided(llDeleteSubList(P, 0, 0), 0,
    llGetListLength(P)-1, 2);
list dirs_prob = llList2ListStrided(P, 0, llGetListLength(P)-1, 2);
float rnd = llFrand(1);
float cumulative = 0;
integer i=llGetListLength(dirs)-1;
do {
    cumulative += llList2Float(dirs_prob, i);
    if(cumulative >= rnd) {
        if(llList2String(dirs, i) == "front") state front;
        if(llList2String(dirs, i) == "back") state back;
        if(llList2String(dirs, i) == "left") state left;
        if(llList2String(dirs, i) == "right") state right;
        if(llList2String(dirs, i) == "up") state up;
        if(llList2String(dirs, i) == "down") state down;
    }
} while(--i>=0);
```

Figure 11: The cumulative probability algorithm uses the dual-Quicksort algorithm found in Appendix 10 in order to first sort the map in decreasing order and then determines the next direction of travel and commutes to its respective state.