Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 11, 2009

P. Peterson
robotfindskitten consortium

June 2009

Standard for robotfindskitten Zen Simulation Environments
rfk-rfc-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

Abstract

This is the abstract.

Table of Contents

1. Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and Status of this protocol. Distribution of this memo is unlimited.

2. Copyright Notice

3. Introduction

robotfindskitten was originally designed and implemented by Leonard Richardson in 1997 as a manifestation of the concept ’robotfindskitten’ as suggested by Jacob Berendes. Richardson’s robotfindskitten was the winning (read: only) entry in the defunct webzine Nerth Pork’s contest of the same name.

robotfindskitten is a zen simulation wherein a user plays the part of robot whose only purpose is to find kitten in an environment. robot’s task is complicated by many objects which are not kitten (non-kitten objects or NKI) but which are indistinguishable from kitten without direct inspection. Thus, the simulation experience consists of navigating the environment, attempting to locate kitten, and finding kitten (or barring that, quitting).

robotfindskitten is simple, and the canonical text only two-dimensional interface is highly portable. For this and other reasons, robotfindskitten has been ported to a variety of different hardware and language platforms. At the time of writing, there are over 30 distinct versions of robotfindskitten with no end in sight. However, lack of a standard has led to small but significant differences in subsequent versions, each generation of which are further removed from the original.

The vast majority of simulation characteristics have remained stable since the original 1997 version and in the POSIX reference implementation, including the name, version numbering, experience, terminology, style of NKIs, notification method, etc. However, the lack of a standard means that there is no way to identify what is or is not robotfindskitten. Arguments, misunderstandings, and lengthy correspondence can arise in the process of mediating these disputes, which are sure to increase as robotfindskitten becomes more widely distributed. The ad hoc gatekeepers of robotfindskitten cannot impose their informal judgement on implementations without a standard. The resulting disorder could ultimately dilute robotfindskitten and allow for unfaithful simulation environments. We define the following standard in the interest of avoiding these and other negative outcomes.

4. A Note on Alternative Designs

This standard is based on and attempts to describe in detail robotfindskitten via the original character-based implementation and

the extant reference implementation for the POSIX environment. The character-based interface is by far the most popular and widespread. However, the consortium desires to leave room in the standard for alternative designs such as three-dimensional or raster-based interfaces which may choose to implement certain elements of robotfindskitten in a different manner.

This complicates the standard because certain design choices (for example, how to represent robot) are technically open to interpretation, but in character-based implementations there exists a unanimous preference (#) that should be respected whenever possible. As a result, this document attempts to describe the essential elements of robotfindskitten -- those elements without which a program is not truly robotfindskitten -- and the elements which are merely strongly recommended (especially for character-based interfaces) but which are not strictly required. It should be understood that strong recommendations should be violated only in special cases and when the implementor truly understands what he or she is doing.

5. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

5.1. 'robotfindskitten'

The name of the simulation MUST be spelled 'robotfindskitten'. robotfindskitten is not a phrase or a proper noun, rather it is a single word which attempts to reflect the generic universe within the simulation. Accordingly, robotfindskitten MUST NOT contain spaces, punctuation, capital letters, numbers, or any other characters. This is true even if 'robotfindskitten' is the start of a sentence and would otherwise be capitalized in English. Translation or transliteration to a foreign language MUST reflect this sensibility.

5.2. 'robot' and 'kitten'

The entities 'robot' and 'kitten' are likewise not proper nouns because they are not a particular robot or kitten but rather their archetypical forms within the simulation. robot and kitten MUST conform to the same character and translation guidelines as 'robotfindskitten'.

5.3. 'item' or 'items'

In this document and in general, the word 'item' is a typical English noun which refers to any object within the simulation that is not robot. In other words, 'item' refers to objects which may or may not be kitten. Kitten is an item, but robot is not.

5.4. Non-kitten item

A non-kitten item is any object, idea, entity, abstract concept, or any other thing within the simulation that is not robot or kitten. Accordingly, the word 'kitten' in non-kitten item MUST NOT be treated as a proper noun.

5.5. Abbreviations

robotfindskitten and non-kitten item MAY be respectively abbreviated as rfk or RFK, and nki or NKI. Even though rfk and nki may be more consistent, the consortium follows the English tradition of capitalizing abbreviations and acronyms and because of cases (such as pluralization) where rfk and nki could be confusing.

[Note: some might argue that robotfindskitten should be abbreviated as 'r', however it has historically been abbreviated rfk and the consortium supports this use.]

6. Version Number

The version number of the simulation is the software version number of the *current* implementation. The version number MUST follow this format:

major.minor.num_nki

7. Program Definition

A compliant robotfindskitten implementation includes an Instruction Screen followed by a Simulation Environment. In the RECOMMENDED character-based implementation, the Simulation Environment is interacted with through a Standard Interface. The simulation runs until robotfindskitten or an OPTIONAL input is given.

8. Instruction Screen

All robotfindskitten implementations SHOULD display an instruction screen prior to the first execution of the simulation environment. The strongly RECOMMENDED instruction text contains the following information in this order:

1. On one line: The word robotfindskitten followed by a version number. (See 'Version Number' below for more information.)

2. Author credit for this version of robotfindskitten and a reference to the original robotfindskitten. (More than one line is acceptable.)

3. Simulation instructions. (See below).

The instruction screen for the official POSIX version appears below:

```
robotfindskitten v1.7320508.406
By the illustrious Leonard Richardson (C) 1997, 2000
Written originally for the Nerth Pork robotfindskitten contest
In this game, you are robot (#). Your job is to find kitten. This task
is complicated by the existence of various things which are not kitten.
Robot must touch items to determine if they are kitten or not. The game
ends when robotfindskitten. Alternatively, you may end the game by hitting
the Esc key. See the documentation for more information.
Press any key to start.
[blank lines to fill the remaining screen space]
```

This instruction screen is RECOMMENDED rather than REQUIRED because platform constraints could motivate a different design. If an instruction screen is used, the RECOMMENDED instruction format and text SHOULD be used.

The simulation is interacted with via an Interface immediately following the Instruction Screen.

9. Standard Interface

The interface consists of the Status and the Field. There MUST NOT be any additional elements.

In the RECOMMENDED character based simulation, the interface is a text window (e.g. a text terminal) filling the available application window and divided into two sections, the Status and the Field.

9.1. Status

The Status is the messaging interface between the user and the simulation. In the RECOMMENDED simulation, the Status SHOULD consist of a reserved space of three lines visible at the top of the interface at all times.

In the RECOMMENDED Status, the first line SHALL display the version number of the simulation at all times. The second line SHALL display the non-kitten item robot has most recently touched or nothing if robot has not touched any non-kitten items yet, and the third line SHALL be a full line of underscore characters ('_', ASCII decimal 95) separating the Status from the field. The Status SHALL also communicate a success message when robotfindskitten.

9.1.1. Success Message

The success message SHOULD contain an animation of robot and kitten (where kitten is consistent in representation from the current simulation) approaching one another in five frames (robot from the left, kitten from the right). Each frame SHOULD be displayed for approximately one second. The animation takes place in the rightmost third of the Status and looks like this (where 'k' is the current kitten character):

```
FRAME       DISPLAY
1.     #      k
2.      #    k
3.       #  k
4.        #k
       [heart]
5.        #k
```

figure 1. Success Animation

The fifth frame with the 'heart' character (which is printed in the version number line) is OPTIONAL because the 'heart' character is not available on all platforms.

After the frame delay time has passed, the characters of the last frame remain in place and a message is printed in the second line (where item descriptions are normally printed). Regardless of Status implementations, the success message MUST read "You found kitten! Way to go, robot!" (without double quotes).

9.1.2. Alternate Status Implementations

Alternate Status implementations can be appropriate in certain circumstances. To date, the two main reasons are constraint issues and non-character-based implementations.

First, hardware/software constraints can require a relaxation of the simulation specification. For example, screen space constraints (such as on the PalmOS or Atari versions) have made the persistent Status too costly to justify. In these cases, it is acceptable to omit or modify the Status, for example to use a popup-style method for communicating messages to the user.

Second, alternative implementation styles often lend themselves towards a different design for the Status. For example, three-dimensional implementations tend not to be character based and so the Status described above may not be directly implementable. However, in the only three-dimensional version to date, the author chose to emulate the recommended Status format through the raster graphics engine.

In any case, the Status MUST identify the items in the simulation and display the success message upon the finding of kitten.

9.2. Field

The Field is the finite space which contains the Simulation Environment. Accordingly, robot and all items exist within the Field, and robot's movement (see below) is limited to the confines of the Field.

In the RECOMMENDED two-dimensional, character-based robotfindskitten implementation, the Field size MUST be limited by the size of the application window (that is, the field does not scroll within the window). If the application window is resized, the Field SHOULD resize to utilize the new window size.

10. Simulation Environment

The simulation environment MUST consist of robot, kitten, and a variable number of discrete non-kitten items placed in the Field. robot is navigated by a user through the Field in order to find kitten. Items, including kitten, are stationary.

10.1. robot

10.1.1. Appearance

robot's visible appearance MUST remain consistent between simulations. In the RECOMMENDED character-based interface, robot MUST be represented by a moveable '#' character (ASCII decimal 35).

10.1.2. Position

robot's starting position is chosen at random (compliant with guidelines restricting the positions of simulation items).

10.1.3. Color

If color is available, robot SHOULD be colored gray. robot MUST be distinguishable from the background color.

10.1.4. Movement

robot can be moved throughout the Field in order to 'touch' items in order to determine their identity. In the traditional character-based implementation, this is accomplished with arrow keys. However, alternative implementations MAY use different input methods.

Regardless of input method, robot MUST be able to move up, down, left, and right. It is STRONGLY RECOMMENDED that robot also be able to move diagonally. It is OPTIONAL whether robot can "speed walk", that is, walk as far in one direction as possible upon one input.

Regardless of implemented navigational directions and input method, robot MUST NOT be able to walk through items. Some implementations use a "destination selection" control scheme where the user clicks or selects the location to which they want robot to travel. Implementations which direct robot by the selection of a destination MUST be careful to ensure that the path generated does not violate item solidity.

10.1.5. Identifying Items

When robot is adjacent to an item (i.e., robot is in a space where any further movement towards the object would result in a collision), robot can identify the item by attempting to "bump" into it. For example, in the RECOMMENDED character-based design, if robot is one character to the left of a non-kitten item, any attempt to move right MUST result in identifying that item. A message identifying the item MUST be displayed in the Status as previously described.

10.1.6. Finding Kitten

When robotfindskitten, the simulation ends in success and the previously described success sequence MUST be carried out.

The simulation ends after robotfindskitten.

10.2. kitten and Non-kitten Items

Non-kitten items are the items in the simulation which are not kitten, and are described by a short text string. (See below for more guidelines on non-kitten items.)

The number of non-kitten items SHOULD be configurable before the start of a new simulation.

10.2.1. Appearance

Appearance of items (kitten and all NKIs) MUST NOT identify kitten in any way. In addition, all items (including kitten) MUST be visible, discrete, and not likely to be the same between simulations. In the RECOMMENDED character-based implementation, each item MUST be represented by a character selected pseudorandomly from the printable character set. (As seen in figure 2 below.) In any case, their appearance MUST NOT indicate in any way which item is or is not kitten.

10.2.2. Position

All items (Non-kitten and kitten) SHALL be pseudorandomly distributed throughout the Field at the beginning of a simulation. In any case, their position MUST NOT indicate in any way which item is or is not kitten.

There are two requirements for the pseudorandom distribution of non-kitten items in the field:

1. There MUST NOT be destructive collisions. For example, if the simulation is run with N items, it cannot be that two items exist in the same space and so one overwrites the other or combines with it. Each of the N items must appear within the field.

2. There MUST NOT be any programmatic mechanism to restrict the location of kitten in relation to items. In particular, this means that it may not be possible for robot to find kitten in some instances of the simulation. This is life.

Finally, if the Field is resized (e.g. due to resizing of the application window), the items (and robot) MAY OPTIONALLY be automatically moved to fill the new space in proportion to their original positions. They SHOULD NOT be re-randomized in any way.

10.2.3. Color

If color is available, all items (kitten and non-kitten) SHOULD be individually colored pseudorandomly from the available palette. All items MUST be distinguishable from the background color. In any case, their color MUST NOT indicate in any way which item is or is not kitten.

10.2.4. Movement

All items are stationary during the simulation, with the exception of the aforementioned proportional position changes after a resizing of the Field.

10.3. Alternate Simulation Environments

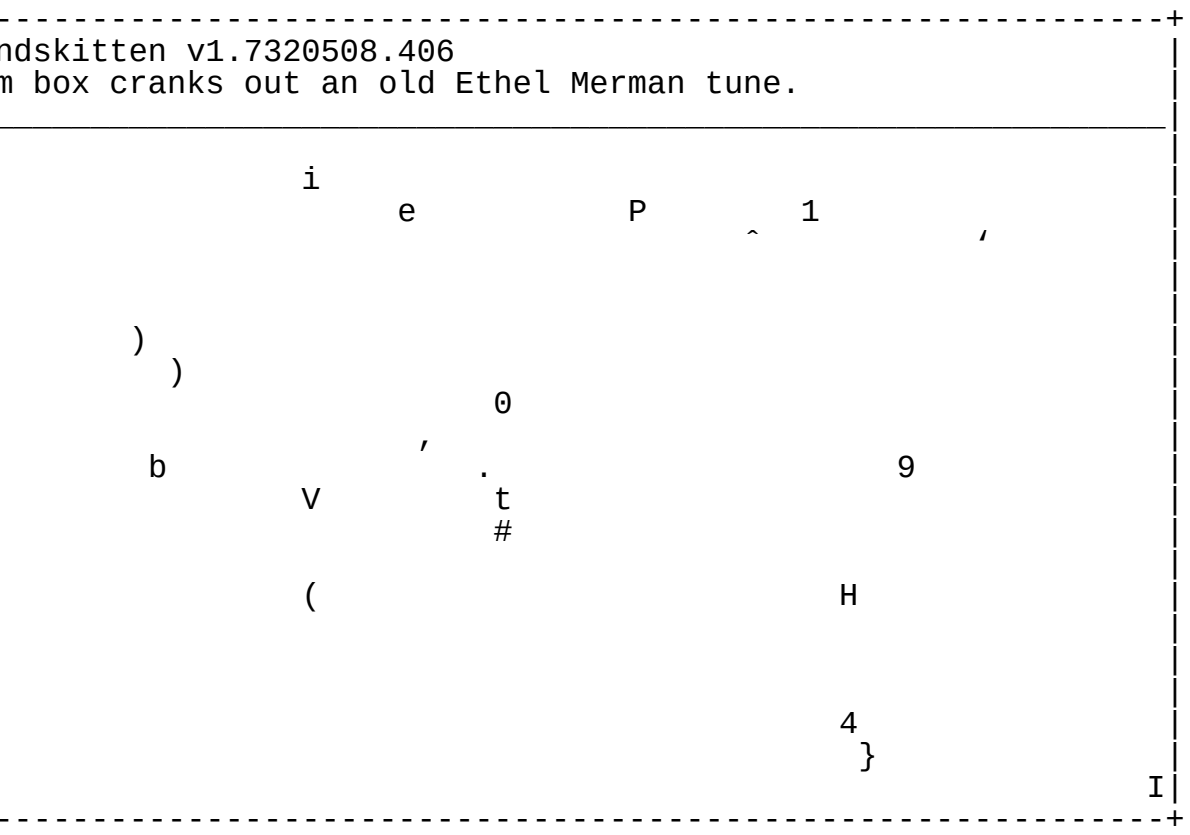
Three-dimensional versions of robotfindskitten exist with alternative (but conformant) representations. However, the canonical and RECOMMENDED robotfindskitten simulation is a two-dimensional environment using an ASCII representation. In such implementations, robot MUST be represented by a '#' character (ASCII decimal 35) and other items (kitten and non-kitten) MUST be represented by pseudorandom, pseudounique characters from the printable character set.

In three-dimensional implementations, the Field size and resize mechanics SHOULD mimic the reference implementation whenever possible.

10.4. Screenshot

The following is a black and white example of a robotfindskitten simulation in progress. Note the Status in the top three lines, followed by the Field below. robot is near the middle of the field, close to an item represented by the 't' character.

figure 2. robotfindskitten 'screenshot'



10.5. Quitting

It is RECOMMENDED that the simulation have an interrupt method whereby the user can quit the simulation if desired. If an interrupt key is implemented, it is RECOMMENDED for historical reasons that the Escape (ESC) key serve in this capacity along with any other desired interrupt sequence such as ^C, etc. Valid reasons for omitting an interrupt method include embedded, single-purpose robotfindskitten devices.

11. Compliant Non-kitten Items

11.1. Introduction

Non-kitten items can be anything that can be described or communicated in 72 characters. This includes real objects, jokes, ideas, abstract concepts, etc. In concert with unpredictability, the ability of NKI to be essentially any kind of information is central to the zen simulation experience.

Each non-kitten item MUST be described by a static string drawn from a pool of item descriptions.

11.2. Programmatic Sources

These descriptions SHOULD come from a static, external source in order to facilitate the addition and subtraction of custom non-kitten items without recompilation. Early versions of robotfindskitten used an embedded, hard-coded NKI list. This design is deprecated. There is no standard format for the NKI file, although a user-friendly format (such as plain text) is RECOMMENDED.

The individual descriptions and the collection of descriptions for any given simulation instance have additional requirements (see below).

11.3. Conformant NKIs

A core element of robotfindskitten is total unpredictability within the scope of the simulation. Earlier requirements in this standard have focused on keeping kitten's location and identity unpredictable. However, an additional requirement is that non-kitten items themselves *be* unpredictable.

11.3.1. Unpredictability

In a very real sense, it is the unpredictable and "infinite" possibilities of non-kitten items that drive the fidelity of the zen simulation. As a result, the fidelity of the simulation is inversely proportional to the predictability of the identity of kitten. As the number of high-quality, unpredictable non-kitten items increases, the chance that you have recently encountered any specific or similar items decreases (unless you always run the simulation with nearly the same number of non-kitten items as the pool size). In other words, as non-kitten items become more predictable, the less faithful (and satisfying) the simulation becomes.

Figure 3. Relationship between Predictability and Fidelity

$$\frac{1}{\text{Predictability}} = \text{Fidelity}$$

As a result, the fidelity of the simulation depends not so much on the number of non-kitten items in the field, but rather on the number of substantially unique non-kitten items. In fact, similar non-kitten items have a multiplicative and negative effect on the fidelity of the simulation and must be strictly avoided.

11.3.2. Guidelines

The following guidelines exist:

1. The identities of NKI MUST NOT be predictable or be overly similar to any other NKI. This ensures that the operator of the simulation will not regularly encounter similar NKI. This

precludes the creation of classes of similar items (e.g., Hardy Boys book titles or Tom Waits albums). As a rough guideline, any NKI should be substantially similar to less than 1% of the total NKI pool in the simulation by subject, and less than perhaps 4% of the pool by style or form (i.e., a pun, an almost-kitten, a math reference, etc.).

2. Non-kitten items SHOULD NOT include current popular culture references. Not only are they unlikely to age well, but they are, by definition, inherently more predictable by virtue of being well known. Similarly, non-kitten items SHOULD NOT include vanity references or self promotion.

3. Non-kitten items SHOULD NOT break the illusion of the simulation. While a non-kitten item can represent virtually anything and be phrased in any way, it SHOULD communicate something to the user of the simulation. In other words, NKI can be obscure, but they SHOULD NOT be nonsense. The illusion of the simulation can be broken by nonsense or by too many similar NKI because the simulation begins to feel like a meaningless in-joke.

4. It is possible that some NKI are required to be similar for the purpose of a larger goal. For example, a hypothetical NKI might be "It's John Denver's left boot." Another NKI could be "It's John Denver's right boot." These two NKI function individually but also as a single, meta-NKI. This is acceptable as long as the meta-NKI is not overly similar to other meta- or simple NKI (e.g. more NKI involving boots). Ultimately, meta-items should be developed with caution; the simulation is about finding kitten, not understanding or inventing complicated meta-items.

NOTE: Some NKI in the robotfindskitten reference implementation may break these rules (especially in regards to vanity or self-promotion). These NKI are left in the official distribution for historical reasons.

11.3.3. Subjectivity

The robotfindskitten consortium recognizes that the non-kitten item guidelines are somewhat subjective. Additionally, since new implementations and NKI lists regularly appear and NKI should be user-editable, there is no way to control the quality of NKI used in individual installations. However, the consortium reserves the right of editorial control over NKI packaged with the reference implementation of robotfindskitten or with implementations that wish to be recognized as compliant with this standard.

12. Acknowledgements

The author wishes to acknowledge the pioneers of robotfindskitten, Leonard Richardson (original author and designer) and Jacob Berendes (inspiration). Additional acknowledgements are due to the POSIX

developers and contributors of robotfindskitten implementations and various rfkiana, who are too legion to name.

13. Website

http://www.robotfindskitten.org/

14. Security Considerations

No security considerations are addressed by this memo.

15. Internationalization Considerations

No internationalization considerations are addressed other than those regarding Unicode character sets and robotfindskitten implementations.

16. IANA Considerations

This memo adds no new IANA considerations.

Author's Address

Peter A. H. Peterson
robotfindskitten consortium

Telephone: 310-924-2425
Email: pedro@robotfindskitten.org
URL: http://robotfindskitten.org/

Full Copyright Statement

Intellectual Property Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.