SPECIAL FEATURE

Zork: A Computerized Fantasy Simulation Game

P. David Lebling
Marc S. Blank
Timoth A. Anderson
MIT Laboratory for Computer Science

Is magic real? Do swords glow if the enemy is nearby? In the demonic world of Zork, a simulated universe entices the player into a new form of problem solving.

[Side panel]

```
Welcome to Zork.
This version created December 8.
West of House.
You are in an open fieldwest of a big white house with a boarded front door.
There is a small mailbox here.
>GO NORTH
North of House
You are facing the north side of a white house. There is no door here, and all
the windows are barred.
>EAST
Behind House
You are behind the white house. In one corner of the house there is a small
window which is slightly ajar.
>OPEN THE WINDOW
With great effort, you open the window far enough to allow entry.
>GO IN
Kitchen
You are in the kitchen of the white house. A table seems to hve been used
recently for the preparation of food. A passage leads to the west, and a dark
stair- case can be seen leading upward. To the east is a small window which is
open.
On the table is an elongated brown sack, smelling of hot peppers.
A bottle is sitting on the table.
The glass contains:
  A quantity of water.
```

Beyond this nondescript kitchen, above and below the surface of the earth, lie scores of rooms, some containing traps, some containing puzzles. Hundreds of objects are scattered throughout this maze, some valuuble treasures, some magical tools. The little white house in the forest clearing is the entrance to Zork, a game developed by the authors. Zork is one example of a new type of game: the Computerized Fantasy Simulation game.

In this type of game, the player interacts conversationally with an omniscient "Master of the Dungeon," who rules on each proposed action and relates the consequences. If the player says "Go north," he may move north, or the dungeon master may say "There is no way to go in that direction." If the player says "Open the window," the dungeon master may respond "The window is locked." The results depend on the design of the game, its architecture and furnishings, so to speak: in one game picking a sword might be fatal, in another it might confer magical powers. The design and implementation of such games is as much an exercise in creative writing as in programming.

The interest in playing Zork (or any other CFS game) is twofold. First, the object of the game is usually to collect treasure, and this may be done only by solving problems; in the above example, the player would garner 10 points by being clever enough to open the window and enter the house. (Zork itself has more than two dozen distinct problems to solve, some presented in several stages.) Second, a great deal of the enjoyment of such games is derived by probing their responses in a sort of informal Turing test: "I wonder what it will say if I do this?" The players and designers delight in clever (or unexpected) responses to otherwise useless actions.

Overview: simulating the universe

The heart of any CFS game is its ability to mimic omniscience. By this we mean that the game should simulate the real world sufficiently well so that the player is able to spend most of his time solving the problems rather than solving the program. If, for example, the vocabulary is too small, the player must always wonder if his problem is only that he hasn't yet found the right word to use. Similarly, it is annoying for a possible solution to a problem to be ignored by the game. In other words, the program must simulate the real world.

Obviously, no small computer program can encompass the entire universe. What it can do, however, is simulate enough of the universe to appear to be more intelligent than it really is. This is a successful strategy only because CFS games are goal-directed. As a consequence, most players try to do only a small subset of the things they might choose to do with an object if they really had one in their possession.

Zork "simulates the universe" in an environment containing 191 different "rooms" (places to be) and 211 "objects." The vocabulary includes 908 words, of which 71 are distinct "actions" it handles. By con- trast, a person's conversational vocabulary is a factor of two (or more) larger. How then does a limited program make a creditable showing in the conversational interaction that characterizes Zork?

The technique Zork uses for simulating the universe is that of universal methods modified for particular situations. For example, when a player tries to take some object, he expects to end up carrying it. There are, as in the real world, exceptions- some objects are "nailed down," and one's carrying capacity is limited. These restrictions are included in the general TAKE function. However, the designer might want a special action in addition to, or instead of, the general TAKE: a knife might give off a blinding light when taken; an attempt to take anything in a temple might be fatal. These exceptions would not appear in the general TAKE function, but in functions associated with the knife and the temple.

The details of this method of exceptions will be taken up later. The effect of it is that "the expected thing" usually happens when the player tries to (for example) take something. If the object he is trying to take is not special, and the situation is not special, then "it works," and he gets the object. In Zork, there are quite a few of these basic verbs. They include "take," "drop," "throw," "attack," "burn," "break," and others. These basic verbs are set up to do reasonable things to everv object the player will encounter in the game. In many cases, objects have properties indicating that a certain verb is meaningful when applied to them (weapons have a "weapon" property, for example, that is checked by the verb "attack"). Applying a verb to an object lacking the necessary property often results in a sarcastic retort ("Attacking a troll with a newspaper is foolhardy."), but the point is that it does something meaningful, something the player might have expected.

Another way in which the game tries to be real is by the judicious use of assumptions in the dungeon master's command parser. Suppose the player says "Attack." Assuming that he has a weapon and there is an enemy to attack, this should work, and it does. Assumptions are implemented by the existence of verb frames (stereotypes) and memory in the parser. In the example, the parser picks up the verb frames for the verb "attack." They indicate that "Attack 'villain' with 'weapon'" is the expected form of the phrase. Now, "villain" means another denizen of the dungeon, so the parser looks for one that is presently accessible, a "villain" in the same room as the player. Similarly, the player must have a "weapon" in his possession. Assuming only one "villain" is in the room and the player has only one "weapon," they are placed in the empty slots of the frame and the fight is on.

Suppose that there is only one villain available, the troll, but the player has two weapons: a knife and sword. In that case, the dungeon master can't decide for him which to use, so it gives up, saying "Attack troll with what?" However, it remembers the last input, as augmented by the defaults ("Attack troll"). Thus, if the user replies "With sword," or even "Sword," it is merged with the leftover input and again the fight is on. This memory can last for several turns: for example, "Attack": "Attack troll with what?"; "With knife"; "Which knife?"; "Rusty knife"; and so on.

Data structure and control structure

The underlying structure of Zork consists of the data base (known as "the dungeon") and the control structure. The data base is a richly inlerconnected pointer structure joining instances of four major data types: "rooms," locations in the dungeon; "objects," things that may be referenced; "actions," verbs and their frame structures: and "actors," agents of action. Each instance of these data types may contain a function which is the specializing element of that instance. The control structure of Zork is, at one level, a specification of which of these functions is allowed to process an input sentence and in what order.

In the simplest sense, Zork runs in a loop in which three different operations are performed: command input and parsing, command execution, and background processing. (Figure 1 is a flowchart of the Zork program.)

The command input phase of the loop is relatively straightfotward. It is intended to let the user type in his command, edit it if he needs to, and terminate it with a carriage return.

[Panel]

Figure 1. Zork flowchart

The purpose of the Zork parser is to reduce the user's input specification (command) to a small structure containing an "action" and up to two "objects" where necessary.

The parser begins by verifying that all the words in the input sentence are in its vocabulary. Then, it must determine which action and objects, if any, were specified. For an object to be referenced in a sentence, it must be "available" - that is, it must be in the player's possession, in the room the player is in, or within an object that is itself available. Objects not meeting these criteria may still be referenced if they are "global objects," which are of two types: those that are always available (such as parts of the player's body), and those that are available in a restricted set of rooms (such as a forest or a house). Adjectives supplied with the sentence are used to disambiguate objects of the same generic type (such as knives and buttons) but are otherwise ignored. If an object remains ambiguous, the parser asks which of the ambiguous objects was meant (for example, "Which button should I push?").

Next is syntax checking, whereby the correct "action" is used for any verb. Syntax checking makes use of any supplied prepositions, differentiating between, for example, "look at" and "look under," which imply different actions. Finally, having determined the appropriate syntax for a given sentence, the parser ensures that all required objects for a given action were specified. The parser may, for example, decide that the correct syntax for the sentence "Give X" is "Give X to Y." An attempt is then made to supply an appropriate "Y" to complete the sentence. This is made possible by the definitions of the sctions themselves, which include the attributes of the objects to be acted upon. In the present example, the action for "Give" defines the indirect object ("Y") to be another denizen of the dungeon; the parser attempts to comply by seeing if one is available. If so, the indirect object is found, and the parse is successful. If not, the player is asked to supply the indirect object himself. ("Give X to whom?") Once this phase is completed, the parse is finished and the parser's output is returned.

The adjectives and prepositions that were in the user's input are used only to determine the "action" and the "objects," and are not part of the parser's output. In addition, all articles are ignored, though users may add them to increase the clarity (to themselves) of what they input. For example, an input of "Knock down the thief with the rusty knife" reduces to something like

```
[<action STRIKE> <object THIEF> <object RUSTY-KNIFE>]
```

If, however, the input were "Knock on the thief," the parser would reduce that to

```
[<action RAP> <object THIEF>]
```

recognizing that the "action" to be peformed depends, for the word "knock," on the syntax of the input sentence: "knock down" turns into "strike," while "knock on" turns into "rap."

[Panel]

Zork internals

The following are some examples of Zork internals. Comments (as in the MDL language) are strings followed by a semicolon. Thus ;"I am a comment."

```
;"The definition of the 'verb' READ:"

<ADD-ACTION "READ"
        "Read"
        [(,READBIT REACH AOBJS ROBJS TRY)
            ;"restrictions on characteristics and location of objects for
defaulting -
            filling in an unadorned'READ' command. The object must be readable
and accessible."
        ["READ" READER] DRIVER]
            ;"READER is the function, and the form 'READ object' is preferred
(the 'driver')"
        [(,READBIT REACH AOBJS ROBJS TRY) "WITH" OBJ["READ" READER]]
            ;specification for 'READ obj1 WITH obj'"
        [(,READBIT REACH AOBJS ROBJS TRY) "THROU" OBJ["READ" READER]]
        "specification for 'READ obj1 THROUGH obj2">

;"Synonyms for READ:"

<VSYNONYM "READ" "SKIM">

;"READER is the general reading function:"

<DEFINE READER ()
    <COND (<NOT <LIT?, HERE>> ;"There must be light to read."
        <TELL "It is imposible to read in the dark.">)
        (<AND <NOT <EMPTY? <PRSI>>>
            ;"<PRSI> is the indirect object. If there is one, the player is
trying
            to ue something as a magnifying glass."
            <NOT <TRNN &let;PRSI>, TRANSBIT>>
            ;"If so, it better be transparent!">
        <TELL "How does one look through a " 1<ODESC2 <PRSI>> "?">
            ;"It failed the test, so print sarcasm."
        (<NOT <TRNN <PRSO>, READBIT>>
            ;"The direct object should be readable."
        <TELL "How can I read a " 1<ODESC2 <PRSI>> "?">
            ;"It's not."
        (<OBJECT ACTION>
            ;"Now the object read gets a chance.")
        (ELSE ;"It didn't handle it, so print text."
        <TELL <OREAD <PRSO>> LONGTELL1>)>>

;"An object: A stack of Zorkmid bills (Zorkmids are the currence of Zork"
```

```
<OBJECT ["BILLS" "STACK" "PILE"]
            ;"The object's name and synonyms."
        ["NEAT" "200" "ZORKM"]
            ;"Adjectives which refer to the object."
        "stack of zorkmid bills"
        <+ ,OVISION, READBIT, TAKEBIT, BURNBIT>
            ;"Properties of the object: it's visible, readable, takeable,
flammable"
        BILLS-OBJECT
        () ;"The contents of the object (always empty for this object)"
        [ODESC1
        "200 neatly stacked zorkmid bills are here."
            ;"The long description."
        ODESCO
        "On the floor sit 200 neatly stacked zorkmid bills."
            ;The initial long description (when first seen by the player)"
        OSIZE 10 ;"The object's weight."
        OTVAL 15 ;"The value of the object: points for finding it and saving it."
        OFVAL 10
        OREAD ,ZORKMID-FACE
            ;"What to print when the object is read."]
            ;"The elements of an object with tokens naming them are rare (few
objects are
            ;"readable and thus need OREAD slots). The other slots are common to
all objects.">

;"The object function for the Zorkmid bills. It is there mostly to make a few
sarcastic comments."

<DEFINE BILLS-OBJECT()
        <SETG BANK-SOLVE!-FLAG T>
        <COND (<VERB? "BURN">
            <TELL "Nothing like having money to burn!">
            <>) ;"Prints sarcasm but doesn't handle the command (accomplished by
returning
            the false object <>). This allows BURN to also deal with it."
            (<VERB? "EAT">
            <TELL "Talk about eating rich foods!>
            ;"Doesn't allow EAT to run (by returning a non-false.")>>

;"A room: the vault in which the Zorkmids are found"

<ROOM "BKVAU" ;"The internal name of the room."
        "This is the Vault of the Bank of Zork, in which there are no doors."
        "Vault"
        ,NULEXIT ;"There are no exits from this room."
        (<GET-OBJ "BILLS">)
            ;"The bills are initially here."
        <>
        <+ ,RSACREDBIT ,RLANDBIT>
            ;"The room may not be entered by the thief, and is a land room."
        [RGLOBAL <+ ,WALL-ESWBIT ,WALL-NBIT>]>
            ;"The walls of the room are/may be referenced."
```

Once parsing has been completed, processing of the command is started. The functional element (if any) of each of the objects in the parsed input may be invoked. Additionally, some objects not specifically referenced, but which define the situation, are part of the processing sequence. The order in which these functions are invoked is determined by a strategy of allowing the exceptions an opportunity to hindle the input before performing the action associated with the most general case. The processing order is as follows:

(1)

The actor performing the command, if any. This allows, for example, a robot with a limited range of actions.

(2)

The vehicle the actor is in, if any. This allows the vehicle to restrict movement. For example, inside a freely drifting balloon the normal movement commands (such as "Run north") might be meaningless or even fatal.

(3)

The verb, or "action.
(a) The indirect object of the sentence, if any.
(b) The direct object of the sentence, if any.

(4)

The vehicle again, if any. The vehicle is called a second time to enable it to act based on changes in the state resulting from the action.

(5)

The room the player is in.

Each of these functions is invoked in turn and given the option of handling the command. If it decides to handle the command, processing terminates at that point, and the remaining functions are not invoked. Otherwise, the sequence continues. Note that a function may do something (such as print out a message) without completely handling the input. The invocation of an object's function is under the control of the verb; it may, after suitable checks, determine that the player's request is not reasonable ("Your load is too heavy. You will have to leave something behind.") This limits flexibility slightly, but it has the advantage that it localizes the tests for a reasonable state.

Presumably, one of the functions will handle the command a print an appropriate response. Should that not happen, the response "Nothing happens" is printed by default. However, care has been taken to ensure that most input commands produce some reasonable response. Indeed, much of the enjoyment of the game is in being allowed to try ridiculous things, and then the surprise of having the game understand them.

The functions described so far are invoked in direct response to what the user typed. Background processes, or "demons," are invoked after each input, regardless of its nature. They allow the program to do things independently of the player's actions.

Currently, there are four demons. The first is the "fighting" demon. The residents of the dungeon are frequently hostile; this demon allows them to assault the player unprovoked, and to keep fighting him even if he ignores them.

Next is the driving process behind the "thief," described as a "seedy looking gentleman carrying a large bag." The thief's purpose is to make life difficult for the player by absconding with treasures or other randomly selected objects. In many ways he acts like another, rather hostile and powerful, player in the dungeon.

The third demon is used to warn the player of the presence of hostile forces by causing his sword (if he has it) to glow if there are enemies nearby. It looks at the player's vicinity and prints an appropriate message if the "state of alert" changes; since the thief moves on his own, it is not sufficient to look for hostiles when the player moves.

Last is the "clock" demon. It is the mechanism by which the concept of future time is introduced into the game; arbitrary events can be scheduled for arbitrary future times. For example, the lamp can burn out after being on for some number of moves, and wounds inflicted in a fight will eventually heal. Out of consideration for poor typists, the clock does not tick after unparsable input.

[Panel]

Examples from Zork

A demonstration of vehicles, specifically the magic boat (various ridiculous things are attempted).

```
>LOOK
Dam Base
You are at the base of Flood Control Dam #3, which looms above you and to the
north. The River Frigid is flowing by here. Across the river are the White
Cliffs, which seem to form a giant wall stretching from north to south along the
east shore of the river as it winds its way downstream.
There is an inflated boat here.
The magic boat contains:
  A tan label
>BOARD THE MAGIC BOAT
You are now in the magic boat.
>NORTH
You can't go there in a magic boat
  ;The room to the north is defined not to allow boats.
>READ
Read what?
>LABEL
Taken.
  !!!! FROBOZZ MAGIC BOAT COMPANY !!!!

Hello, Sailor!

Instructions for use:

 To get into boat, say 'Board'
 To leave boat, say 'Disembark'

  To get into a body of water, say 'Launch'
  To get to shore, say 'Land'

>LAUNCH
Frigid River
You are on the River Frigid in the vicinity of the Dam. The river flows quietly
here. There is a landing on the west shore. You are in the magic boat.
>DOWN
Frigid River
The River turns a corner here making it impossible to see the Dam. The White
Cliffs loom on the east bank and large rocks prevent landing on the west. You are
in the magic boat.
>UP
You cannot go upstream due to strong currents.
>PUT BOAT IN BOAT
How can you do that?
>DISEMBARK
You realize, just in time, that disembarking here would probably be fatal.
>DOWN
Frigid River
The river is running faster here and the sound ahead appears to be that of
rushing water. On the west shore is a sandy beach. A small area of beach can also
be seen below the Cliffs. You are in the magic boat.
There is a red buoy here (probably a warning).
[in the room]
>EAT BUOY
Taken.
I don't think that the red buoy would agree with you.
  ;The verb function has done all the processing.
```

Playing with fire, attempting to burn the painting (in the end, the careless adventurer gets his just deserts):

```
>INVENTORY
You are carrying:
A pair of candles
A matchbook
A torch
A painting
A lamp
>LIGHT CANDLES WITH TORCH
You realize, just in time, that the candles are already lighted.
```

```
>EXTINGUISH CANDLES
The flame is extinguished.
>LIGHT CANDLES WITH TORCH
The heat from the torch is so intense that the candles are vaporized.
  ;The 'candles' function has handled being lit with the torch.
>LIGHT MATCH
One of the matches starts to burn.
>BURN PAINTING WITH MATCH
The painting catches fire.
Unfortunately, you were holding it at the time.
```

Demonstration of the parser, and a number of its features, including the remembering of previous inputs and the use of ALL and ALL EXCEPT:

```
>WEST
Living Room
You are In the living room. There is a door to the east, a wooden door with
strange gothic lettering to the west, which appears to be nailed shut, and a
large oriental rug in the center of the room.
There is a trophy case here.
A battery-powered brass lantern is on the trophy case. On hooks above the
mantelpiece hangs an elvish sword of great antiquity.
>TAKE ALL BUT PAPER AND RUG
sword:
  Taken.
lamp:
  Taken.
>READ
Read what?
>THE DOOR
The engravings translate to 'This space intentionlly left blank'
>LOOK UNDER RUG
Underneath the rug is a closed trap door.
>MOVE THE RUG
With a great effort, the rug is moved to one side of the room.
With the rug moved, the dusty cover of a closed trap-door appears.
>OPEN
Open what?
>DOOR
Which door should I open?
>THE TRAP DOOR
The door reluctantly opens to reveal a rickety staircase descending into
darkness.
>GO DOWN
It is pitch black. You are llkely to be eaten by a grue.
Your sword is glowing with a faint blue glow.
>WHAT'S A GRUE?
The grue is a sinister, lurking presence in the dark places of the earth. its
favorite diet is adventurers, but its insatiable appetite is tempered by its fear
of light. No grue has ever been seen by the light of day, and few have survived
its fearsome jaws to tell the tale.
Your sword is gloing with a faint blue glow.
>TURN ON LAMP
The lamp is now on.
Cellar
You are in a dark and damp cellar with a narrow passageway leading east and a
crawlway to the south. On the west is the bottom of a steep metal ramp which is
unclimbable.
The trap door crashes shut, and you hear someone barring it.
  ;The room function closes the trap door to bar escape.
```

The robber, demonstrating his sadistic streak:

```
You are in a tiny cave with entrances west and north, and a dark, forbidding
staircase leading down.
There is a suspicious-looking individual holding a bag, leaning against one wall.
He is armed with a vicious-looking stiletto.
>INVENTORY
```

```
You are carrying:
A lamp
A sword
A platinum bar
A jade figurine
The butt of his stiletto cracks you on the skull, and you stagger back.
  ;The fighting demon just ran.
The other occupant just left, still carrying his large bag. You may not have
noticed that he robbed you blind first.
  ;The thief demon just ran.
>INVENTORY
You are carrying:
A lamp
>SIGH
You'll have to speak up if you expect me to hear you!
```

A not atypical run-in and battle with the troll (unfortunately, this one results in the death of the adventurer):

```
>EAST
The Troll Room
You are in a small room with passages off in all directions.
Bloodstains and deep scratches (perhaps made by an axe) mar the walls.
A nasty-looking troll, brandishing a bloody axe, blocks all passages out of the
room.
Your sword has begun to glow very brightly.
>KILL TROLL WITH SWORD
A mighty blow, but it misses the troll by a mile.
The axe gets you right in the side. Ouch!
>SWING BOTTLE AT TROLL
Trying to attack a troll with a glass bottle is suicidal.
The axe crashes against the rock, throwing sparks!
>JUMP OVER TROLL
The troll is too big to jump over.
You stagger back under a hail of axe strokes.
>TIE UP TROLL
With what?
>ROPE
The troll struggles and you cannot tie him up.
The troll hits you with a glancing blow, and you are momentarily stunned.
>DIAGNOSE
You have a light wound, which will be cured after 29 moves.
You can be killed by one more light wound.
The troll swings his axe, but it misses.
  ;The troll, no gentleman, keeps fighting while you examine your wound.
>SCREAM
Aaaarrrrrrrrggggggggggggghhhhhhhhhhhhhh!
The axe crashes against the rock, throwing sparks!
>WAKE UP THE TROLL
The troll isn't sleeping.
The axe sweeps past as you jump aside.
>JUMP
Wheeeeeeeeee!!!!!
The troll's axe stroke cleaves you from the nave to the chops.
It appears that that last blow was too much for you. I'm afraid you are dead.
```

A demonstration of the Clock Demon, with the Gnome of Zurich appearing a specified number of moves following entrapment in the Small Room:

```
Small Room
You are in a small, bare room with no distinguishing features. There are no exits
from this room.
>NORTH
There is a wall there.
>EAST
There is a wall there.
>SOUTH
There is a wall there.
```

```
>WEST
There is a wall there.
>UP
There is no way up.
>DOWN
There is no way down.
>FOO
Well FOO, BAR and BLETCH to you too!
>CLIMB UP WALLS
Climbing the walls is of no avail.
An epicene Gnome of Zurich wearing a three-piece suit and carryino a safety-
deposit box materializes in the room. 'You seem to have forgotten to deposit your
valuables,' he says, tapping the lid of the box impatiently. 'We don't usually
allow customers to use the boxes here, but we can make this ONE exception, I
suppose...' He looks askance at you over his wire-rimmed bifocals.
>HELLO GNOME!
The gnome appears increasingly impatient.
>INVENTORY
You are carrying:
A lamp
A clove of garlic
A knife
>GIVE GNOME THE GARLIC
'I wouldn't put THAT in a safety deposit box,' remarks the gnome with disdain,
tossing it over his shoulder, where it disappears with an understated 'pop.'
>STAB GNOME WITH KNIFE
The gnome says 'Well, I never...' and disappears with a snap of his fingers,
leaving you alone.
>LOOK
Small Room
You are in a small, bare room with no distinguishing features. There are no exits
from this room.
```

The history of Zork

The existence of Zork is a direct consequence of the existence of two excellent games: Dungeons and Dragons, a fantasy simulation game (not computer based) invented by Dave Arneson and Gary Gygax, and Adventure, a computerized fantasy simulation game originally written by Wil Crowther and later extensively expanded by Don Woods.

Adventure itself was inspired by D&D (as it is familiarly known), in particular a D&D variation then being played out at Bolt-Beranek and Newman, a Cambridge, Massachusetts, computer firm. It eventually was released to the public, and it became one of the most popular computer games in recent memory.

One laboratory that acquired a copy of Adventure was MIT's Laboratory for Computer Science, with which the designers of Zork (the authors and Bruce K. Daniels) were all then affiliated. In the process of "solving" Adventure, however, the game's deficiencies and the competitive spirit that often animated computer researchers kindled the desire of the authors to write a successor game.

Our natural choice of language was MDL, which is one of the languages in use at LCS. MDL recommended itself for other reasons, however. It is a descendent of LISP and is functionally extensible. It also permits user-defined data types, which is important in a game of "rooms," "objects," "verbs," and "actors." Finally, MDL makes it easy to imbed implicit functional invocations in data structures to tailor the game as described above. The initial version of the game was designed and implemented in about two weeks.

The first version of Zork appeared in June 1977. Interestingly enough, it was never "announced" or "installed" for use, and the name was chosen because it was a widely used nonsense word, like "foobar."

The original version of the game was much smaller both geographically and in its capabilities. Various new sections have prompted corresponding expansions in the amount of the universe simulated. For example, the need to navigate a newly added river prompted the invention of vehicles (specifically, the boat). Similarly, the addition of a robot prompted the invention of other actors than the player himself: beings that could affect their surroundings, and so on. Fighting was added to provide a little more randomness in a fairly deterministic game.

The future of computer fantasy simulation games

Zork itself has nearly reached the practical limit of size imposed by MDL and the PDP-lO's address space. Thus the game is unlikely to expand (much?) further. However, the substrate of the game (the data types, parser, and basic verbs) is sufficiently independent that it would not be too difficult to use it as the basis for a CFS language.

There are several ways in which future computerized fantasy simulation games could evolve. The most obvious is just to write new puzzles in the same substrate as the old games. Some of the additions to Zork were exactly this, in that they required little or no expansion of the simulation universe. A sufficiently imaginative person or persons could probably do this indefinitely.

Another sirnilar direction would be to change the milieu of the game. Zork, Adventure, and Haunt (the CFS games known to the authors) all flow back to D&D and the literary tradition of fantasy exemplified by J. R. R. Tolkien, Robert E. Howard, and Fritz Leiber. There are, however, other milieus; science fiction is one that comes to mind quickly, but there are undoubtedly others.

A slightly different approach to the future would be to expand the simulation universe portrayed in the game. For example, in Zork the concept of "wearing something" is absent: with it there could be magic rings, helmets, boots, etc. Additionally, the player's body itself might be added. For example, a player could be wounded in his sword arm, reducing his fighting effectiveness, or in his leg, reducing his ability to travel.

The preceding are essentially trivial expansions to the game. A more interesting one might be the introduction of magic spells. To give some idea of the kinds of problems new concepts introduce to the game, consider this brief summary of problems that would have to be faced: If magic exists, how do players learn spells? How are they invoked? Do they come in different strengths? If so, how does a player qualify for a stronger version of a spell than he has? What will spells be used for (are they like the magic words in Adventure, for example)? How does a player retain his magic abilities over several sessions of a game?

As can be seen, what at first seems to be a fairly straightforward addition to a game that already has magical elements raises many questions. One of the lessons learned from Zork, in fact, is one that should be well known to all in the computing field: "There is no such thing as a small change!"

A still more ambitious direction for future CFS games is that of multiple-player games. The simplest possible such game introduces major problems, even ignorng the mechanism used to accomplish communication or sharing. For examlple, there are impressive problems related to the various aspects of simultanity and synchonization. How do players communicate with each other? How do they co-ordinate actions, such as attacking some enemy in concert?

Putting aside implementation problems, a multiple-player game would need to have (we believe) fundamentally different types of problems to be interesting. If the game were cooperative (as are most D&D scenarios) then problems requiring several players' aid in solving them would need to be devised. If the game were competitive and like the current Zork, the first player to acquire the (only) correct tool for a job would have an enormous advantage, to give just one example. Other issues are raised by the statistic that the average player takes weeks and many distinct sessions to finish the game; what happens to him during the time he is not playing and others are?

We believe there is a great future for this type of game, both for the players and for the implementers and designers of more complex, more sophisticated, and - in short - more real simulation games.

Zork distribution

Zork is available from two sources. The most up-to-date version is available from P. David Lebling, Room 205, 545 Technology Square, Cambridge, MA 01239. This version is compatible with the ITs, Tenex, and Tops-20 operating systems for the Digital Equipment Corp. PDP-10. To obtain this version, you must enclose a magnetic tape and return postage. Another version, translated from MDL into Fortran, is available through DECUS, (the Digital Equipment Computer Users Society), One Iron Way, Marlboro, MA 01752. This version is compatible with PDP-11 and VAX operating systems.

Bibliography

S. W. Galley and Greg Pfister, *MDL Primer and Manual*, MIT Laboratory for Computer Science, 1977.

P. David Lebling, *The MDS Programming Environment*, MIT laboratory for Computer Science, 1979.

Gary Gygax and Dave Arneson, "Dungeons and Dragons," TSR Hobbies, Inc., Lake Geneva, Wisc.

P. David Lebling is a staff member of the MIT Laboratory for Computer Science, where he works on data-intensive planning systems and computer messaging systems. He was previously involved in Morse code transcription and understanding system. He holds SB and SM degrees in political science from MIT.

Marc S. Blank is a medical student at the Albert Einstein College of Medicine; he expects to graduate in June. He is employed from time to time as a consultant at the MIT Laboratory for Computer Science, where he works on MDL data bases and system maintenance. A graduate of MIT with a BS in biology, he is a member of Phi Beta Kappa and Phi Lambda Upsilon.

Timothy A. Anderson is a member of the research staff at Computer Corporation of America, Cambridge, Massachusetts, where he works on the distributed data base system SDD-1. As a graduate student at MIT Laboratory for Computer Science, Anderson was involved in research on Morse code understanding. He is a member of ACM, Sigma Xi, Tau Beta Pi, and Eta Kappa Nu, and holds SB and SM degrees in computer science from MIT.
