琉球大学学術リポジトリ

非直線的、非/超言語的コミュニケーションの可能性について

メタデータ	言語: 出版者: 琉球大学教育学部 公開日: 2007-07-18 キーワード (Ja): キーワード (En): 作成者: Taira, Katsuaki, 平良, 勝明 メールアドレス: 所属:
URL	http://hdl.handle.net/20.500.12000/1042

On Non-linear Extra-lingual Communication and Its Possibility

Katsuaki TAIRA

Many people have wondered if they can somehow achieve the cultural and linguistic nirvana of communicating with the people living beyond their own cultural ken. They dreamed of a space in which they can freely partake of other people's ideas and understand others without any trammels. But despite many attempts to realize such a heavenly sphere (heavenly in terms of communication, that is) the realization of unbounded understanding of other parties have not been materialized. Why is that the case? Simply put, the achievement of such a communicative nirvana entail enormous effort and superhuman legerdemain, as it were. Then how about all the technical wizardry man has mastered up to this juncture in human history? Is it not enough to translate subtle nuances into different verbal and communicative mode? How about audio and visual tapes to facilitate conceptual parleys between two parties located in disparate cultural zones? They are in a way revolutionary in that they allow us to store visual and audio imprints that otherwise vanish as soon as the relevant audio/visual expressions terminate with the dislocation of the living body from which they originally initiate. However, the traditional means of storing data such as the magnetic kind just indicated, seemed to have indicated the direction in which the vast potential of untrammeled communications can be conducted. Because they can function as an auxiliary means to convey ideas regardless of the presence of the initiator/generator of that which constitute the content of the vehicles, the storage medium named here have legitimately been treated as a harbinger of what the future communication holds for man. But unfortunately the limitations of the conventional media were all too evident from the start. They do hold information and they do allow those who reach for information to reproduce what is stored in them. But they are incapable of responding to individual users in a manner that is deemed to enhance the experience of straddling the cultural and linguistic hedges separating two disparate spheres. That is, the traditional means of communication (to be more accurate, mediated communication) was more or less one directional and did not simulate the reality based interactive nature of communication. No wonder, because those media under the traditional setup did not allow programming. Neither have they been conceived to allow user interactions. This attempt that will be delineated here in my

essay attempts to break away from the rigidity imposed by such traditional approaches to "speak" with different cultures. Whether or not the proposed attempt succeeds depends on the underlying principles and the execution that in a way concretizes all the nebulous conceptions sprung out of those general principles.

Part of this reality/high-tech oriented approach stems from the old idea appropriately expressed by the German thinker Johann Gottfried Herder. He maintained that thinking is essentially identical with speaking and thus differs from language to language and culture to culture. In his words, the "human sprite thinks with words," and he asks, "What is thinking?" To which he answers decisively, "Inward language," because, he deduces, talking is just merely thinking aloud."[1] Although Herder's remark may not be so scientific after all, his remark seems to hold some grain of truth. If the inner thought and the outer expression of are somehow inexorably related not only in the bathetic sense that the latter is after all the expression of the former, which is a tautological huis clos, but also how one construes the outside world in his inner psyche and how he reconstructs it in the most uniquely human medium, language, are innately linked. But I the two are linked, be it in the subtlest manner possible or outright self-evidently, are the elements that influence the two so that a person thinks in the way he thinks? Is there a way to arrive at the one hierarchy up in the chain of human ratiocination, or, if you do not mind my going too general, etiology of human mind and consciousness? If the nebula of human thoughts can be retracted to one originating point, does it suggest anything about the way humans think and communicate in its full diversity? Perhaps yes. But the ladder from the multiple manifestations of human thought/language to that hypothetical unifying point is infinitely long. That means we have to come up with some intermediate steps to solve the mystery of language in its culturally inflected and deflected instantiations. Just when the possible originating point is posited, an insight offered by another thinker Wlhelm von Humboldt raises the barrier to keep us from arriving at the next notch in hierarchy on a way to the ultimate origin of consciousness. In the following excerpt he reiterates the reconstructive and encapsulating nature of language in the same vein as Herder.

> [E]ach language...contains a characteristic worldview. As individual sound mediates between object and person, so the whole of language mediates between human beings and the internal and external nature that affects them.... The same as which enables him [man] to spin language out of himself enables him to spin

[1] The series of quotations come from Johann Gottfried von Herder, 1877-1913, *Samtliche Werke*, edited by Bernard Suphan.

himself into language, and each language draws a circle around the people to whom it adheres which it is possible for the individual to escape only by stepping into a different one.[2]

As was hinted by Herder, Humboldt emphasizes the mediational function of language that both connects and interprets the external phenomenon and the inner psyche. Because language is assigned such a crucial operational status, it ineluctably shades the way one reconstructs his own world. That is, the interpretational angle one assumes is inseparably attached to the linguistic means one willy-nilly chooses to perceive the outside world. That is why, as Humbolt states, one needs to detach himself from his own world, which is language-delineated, in order to fully understands the other who resides in other heterogeneous linguistic realm. The whole process sounds simple enough, but if you consider the implications of leaving your own preferred language and the realm associated with it, the seemingly innocent process is bound to confound you with its enormity. That is because if one sheds his linguistic self, it entails the loss or crumbling of the system one has been living in. What is this system I am referring to here? That is the whole corpus of cultural values one has taken for granted and cherished. Does it sound far-fetched? Not really. Because of the inseverability of the language and culture one is automatically assumed to discard or break apart from the mainstream of culture in order to enter into the realm demarcated by other linguistic practice.

The perception of reality though language is also mentioned by Edward Sapir. More or less in the same vein he recapitulates both the auxiliary and reconstructive nature of language. He states that we are essentially "at the mercy of the particular language which ahs become he medium of expression for [our] society." In other words, the so-called "reality" is nothing but the reconstituted hypothesis that is ineluctably mediated by the language one was forced to adopt. If reality is language specific, then, there is no absolute phenomenon that is untinged by our reconstructive propensity. Once again in Sapir's words, the real word "is to a large extent built up on the language habits of the group." There is no exception from the all-pervasive power of language. Everything that resides in one society exists because of and thanks to this intangible yet perception-bending language. That is why it is deemed so difficult to jump from one relative perception to another. If the linguistic field is so absolute, one becomes understandably despondent over the prospects of forever enclosed in a value system from which there is no escape. Read the following to make the despair complete.

---

[2] The excerpt is from *Wlhelm von Humboldts Werke*, edited by Albert Leitzmann, volume 7, page 60.

> No two languages are ever sufficiently similar to be considered as representing the same social reality. The worlds in which different societies live are distinct worlds, not merely the same world with different labels attached.[3]

But is a linguistic system such an insular, enclosed one? In that case even a modicum of communication between two different groups become impossible. That is rather depressing. Granted that language is a program or mechanism through which one defines the eternal and internal landscapes and without which no interpretation of any kind is impossible. Does that fact exclude the possibility of the middle ground where people meet and exchange ideas to partake of each others ideas? In search of that possibility, then, I now turn to Noam Chomsky what is most relevant here is his idea of language as mere form. It is, according to him, nothing but a generator of containers in which to put associated ideas. That is, language provides labels to which all the concepts that float in society are linked, or more logically, language functions in such a manner that all the concepts that are defined in one society are assigned labels that are appropriate to them semantically. It may sound rather tautological but Chomsky's attempt is essentially to find the originating point to define the relationship between the concept and the label that is somewhat analogous to the traditional Western dichotomy of Idea and Form. Look at the following excerpt.

> Language and thought are awakened in the mind, and follow a largely predetermined course, much like other biological properties.... Human knowledge and understanding in these areas... is not derived by induction. Rather, it grows in the mind, on the basis of our biological nature, triggered by appropriate experience, and in a limited way shaped by experience that settles options left open by the innate structure of the mind.[4]

Although Comsky's observation rings a little too radical in itself,[5] it offsets the

---

[3] The preceding quotes are from *Selected Writings of Edward Sapir in Language, Culture and Personality,* edited by David Mandelbaum. The "labels" here of course refer to the linguistic holder to which each reality-bound concept and phenomenon is assigned.

[4] The excerpt is from *Reflections* on language by Chomsky. The passage here cited tends to give a rather simplistic view Chomsky expresses. But in juxtaposition with the opposing views I have been citing, a radical expression of this kind seems appropriate as it offsets the force that has been accumulating on the other end of the argument.

[5] As Anna Wierzbicka notes in her *Semantics, culture, and Cognition,* Chomsky tends to sound a little too "fanciful" with his universalizing view of language. She adds that Chomsky's idea of the "innate and culture-independent character of most concepts" rings too extreme and casts doubts on Chomsky's assertion that "there is no clear alternative to the assumption that acquisition of vocabulary is guided by a rich and invariant conceptual system, which is prior to any experience" ("Language in a Psychological Setting" from *Sophia Linguistica*). For Wierzbicka's view, see *Semantics,*

extremist positions the preceding writers took on the possibilities of intercultural/lingual dialogue in the world. My intention is somewhat in between. What if there is a neat middle ground where two ideals exemplified between the two parties can be spliced together and essential ideological/conceptual communication is made feasible? Even positing such a possibility seems exciting enough. But here I would like move out of the controversial issue on the essential nature of language and leap into an hypothetical sphere, which is also hypothesized as existing in a virtual realm hemmed in by the upper limits of imagination. It is true that there is apparently no connecting line between the sphere now hypothesized and the other turbulent one with conflicting and contradicting ideas. But the argument I am going to make from now on is intended to shed light on the nature of language and the acquisition and multiple implications culture attaches to it. In other words, I would like to suggest other approaches to the reality-impacted issue of operation, absorption and acquisition of language that are deemed to be the essential ingredients in comprehending the heterogeneous other—the center of disputes among the linguists since time immemorial.

Let us go into the nitty-gritty and look at some of the realistic approaches that are hinted at here. The one specific approach I propose takes advantage of the best technology the early twenty-first century offers. Needless to say, there are plenty of means to accomplish the close communication of the kind suggested between heterogeneous cultures or linguistic groups. Here, I utilize a program developed by a well-known software company called Macromedia. The program I have in mind is Director. Although the language incorporated into the program is not necessarily as generic as C or C++, it allows programmers plenty of freedom in the way one manipulates the contents that appear on the stage.[6] The language, named—rather facetiously I would say—Lingo, obviously is a derivative of C or, perhaps nowadays more like C++, gives programmers a great deal of control over the contents that are at hand. It enables one to turn off and on sounds, concatenate text strings, do list operations animate objects and much, much more. As long as one is conversant with the language, that is, one can easily present contents in almost exactly the way one envisions. That seems like a more than ample praise for a language, which is still owned by one particular company. But seeing how it is evolving and what broad range

---

Culture, and Cognition, pp. 5-6.

[6] The stage is the part where all the contents, be it animation, graphics, or text, all appear. In other words, the stage is an interface where all the interactions between the programmer/contents and the user take place.

of control it grants anyone adept with it, the proprietary issue may legitimately be relegated to the extraneous sphere for the moment. Well then, without further delay, let me present the general concepts on which the next generation of intercultural communication can be facilitated and developed. (Do not confuse the general concepts with the kind that simply adumbrates nebulous ideas in cocoons of super-abstruse terminologies just to hide the paucity of ideas. Mine will be completely opposite. I intend to present my ideas in the most concrete manner possible to allow the reader to grasp the foundation of communication I am proposing here. Concrete, that is, in the most technical sense. I intend to employ the proprietary computer language Lingo rather extensively. If you get lost in the process, the blame lies entirely in me. Although I will try my best to convey the structure in which the optimal interlining of heterogeneous cultures can be accomplished, there is absolutely no guarantee that my approach is the easiest or the most direct possible to address the audience. Be that as it may, seeing that I have already wasted quite a space by now, let me just begin where I deem is the best.

I propose an indirect approach to convey cultural and linguistic contents to the target group. In that context I would like you to look at the following general scheme. It may look like jargon for the moment. But please be patient. The marvelous world of multimedia will open up before you know it.

```
on beginSprite me
  pLinkSound=getAt(["righto1","righto2","righto4"], random(3))
  pLinks = [me.spriteNum]
  pDrag=FALSE
end
```

The initial handler[7] initializes the sound list named pLinkSound and gets a random value out of the three values, which happen to refer to the sounds contained in the cast library of the movie. In other words, the one line script does two things at once to make the randomly chosen sound available for later use. The pLinks variable is used to regulate an operation that is essential to the success of the game I have in mind. In other words, if pLinks is true then certain action will not be permitted while if the other

[7] The handler is a term used to denote the script line that is preceded by *on*. As the term suggests, handlers intercept whatever message is generated by the program and triggers action only when the relevant message is received. They are like fisherman's nets that capture only what is targeted and not others.

is the case the same action is guaranteed to proceed. The right item pLinks is evaluated to is the integer or the number of the channel the script is attached to. Therefore, the variable holds the integer value for each script that is discreetly associated with the channels that are relevant to the successful deployment of the game. That is not the end of the story, however. The brackets that enclose the me.spriteNum indicate that the integer value is put in a list for later manipulation of the channel numbers in conjunction with the pDrag flag that is immediately evaluated to FALSE at the beginning of the script. The temporal precedence is indicated by the beginSprite handler that includes the three-line script, which turns out just a part of a more lengthy one. That is almost the entirety of the initial stage of the programming for this very complicated operation. However, there is a pre-preliminary stage for this script. Lingo, which is as I already referred to, is the native language of the multimedia authoring tool named Director, allows evaluation of variables while the program is sitting still, as it were. In other words, the program does not have be running in order for all the variables to be initialized. Needless to say, that alternative is definitely a good choice when it is more convenient as when assigning of a value to each channel variable gets tedious, or making the program more interactive with the users as the former seemingly responds to the feedback from the latter. Now, for the pre-assigned parameter values. How are those values assigned? What is their syntax? The questions of this kind should abound at this stage. But Director indeed has an ingenious interface that allows programmers to change parameters merely at a click of a button. Even for those who are not quite versed with the native language can set those variables as long as the comment attached to those variables are apt. Since the parametric setup window pops up when the appropriate sprite is clicked, all the programmer has to do is input appropriate values in the relevant slots. But the parametric windows do not self-generate, needless to say. Even they have to be programmed to appear in the manner specified by the programmer. That means there has to be a proto-initiator programmatic schema before anything comes to take shape. Even the simplest maneuver has to be planned and structured in order for it to become a congruous part of the whole presentation. Back to the pre-assigning procedure. The following is the syntax used to initialize property variables for this particular script.

```
on getPropertyDescriptionList me
  list = [:]
    -- how near this piece needs to be to another to link it
```

```
  addProp list, #pCloseness,
    [#comment: "How close does it need to be to other pieces to lock",
     #format: #integer,
     #default: 4]

  -- sound to play when a link is made
  addProp list, #pLinkSound,
    [#comment: "Link Sound",
     #format: #string,
     #default: ""]
    -- frame to jump to when game is over
  addProp list, #pGameOverFrame,
    [#comment: "Game Over Frame",
     #format: #marker,
     #default: #next]
  return list
end
```

The handler getPropertyDescriptionList receives a message[8] from the application when the program is not running (or rather, when the movie is not running, to be more accurate) and assigns the parametric values to the variables declared on top and return those values to the program itself for use later on in the script. This is quite an ingenious way to initialize variables. So much so that I am compelled to expatiate on the mechanism a little further. When a programmer considers the timing at which to fill each variable with appropriate values, be it string, integer, marker or what not, he is often forced to choose either localize or globalize and/or pre-assign or post-assign (those pre or post are relative to the start of the movie) all the parametric values to each variable. That is when getPropertyDescriptionList handler comes in handy. Since it enables programmers to assign values even before the movie starts, all the parameters set in this manner are guaranteed to be made retrievable, thus allowing the relevant script to deploy according to the design of the programmer. The initial property that is

---

[8] The "message" may sound rather odd to those who are unfamiliar with Lingo. It is defined as the signal that is sent out from the program itself for interception by the relevant scripts. When the message is trapped by those scripts it triggers whatever action that is designed to occur by the programmer. Most likely, the initial message brings about a chain of events that contribute to the total effect of the presentation or movie.

to be evaluated is the distance between any two objects that are to be matched in this game. As mentioned in the comment section of the script it is a value that determines the coordinate distance between two objects that are on the stage. Why do you have to set this value? Or more accurately, why do you have to adjust this value rather than making it a fixed distance within which nay two objects drawn together would be conjoined? The answer to that lies in the fact that too miniscule a value would render the game extremely unplayable by putting the level of accuracy way beyond average players capability. On the other hand, if the proximity value is too large, the player would have the illusion of magically driving any two objects into their predetermined slots, thus destroying the sense accomplishment that comes from self-generated paring off of pieces on the stage. Be that as it may, the property value here called pCloseness needs to be parameterized in order for the game to be enjoyable. The next parameter pLinkSound is set to hold the string value that refers to the sound that plays when the matching pieces are conjoined. The type the parameter accepts is determined by the format section of the property list that is enclosed in the getPropertyDescriptionList declaration statements. Because the program cannot know on its own the type of data that are changed hands between variables (or any aliases that hold certain values), the type has to b specified manually, as it were, by the person who manipulates the program. In this case, as I already hinted at, the type is declared by the second item on the list #format and the type is specified as string. What is noteworthy here is that in the case of strings, the parameter can be set to an open string, or that which essentially holds no name if such function as member.name is called. In this instance, empty string holds great advantage over, say, any specific sound name the parameter is essentially designed for. In fact, if a group of sounds are to be parameterized, there is even a more direct and easier way. That is, using #sound rather than #string. But unlike the string the sound cannot take an empty name as it is evaluated to the parameter (in this case pLinkSound). That poses a problem if you do not wish to set any particular sound to pLinkSound or when there is no sound to assign at all. If one is definitely the case, you simply skip the process and leave any interactive sound out of the game. But if one of the two cases are possible and either case is as likely as the other, then the type sound becomes very inconvenient. But on the other hand, typecasting the proceeding parameter to a more comprehensive type, #marker, makes sense because pGameOverFrame needs to be evaluated to a particular marker name in order for the game to advance to the next level or come to certain level of conclusion. In this case, pGameOverFrame is set to the default #next, which will send the playback head to the next marker after the current segment.

That is the preliminary stage of the game in order for it to fully deploy to its maximum designed potential. In its conceived entirety, the game is intended to process any user input or interaction through the script placed in the frame that overviews the whole landscape. It controls when to allow pairing off of the pieces, when to and when not to play any relevant sound etc. Without the all-seeing eye of the frame script, nothing happens. From a different perspective, the frame script is like an engine that drives everything forward. All peripheral activities are somewhat contingent upon the commands that flow from the heart o the frame script. In other words, the frame script supervises and pulls together the entire game. It is the great synchronizer of the game universe and the willing participant in it. Without which no interface between the two is possible. Not only that but no inception of the dialogic engagement will materialize. Well, so far I have been explaining the preliminary stage of the whole setup for the said dialogic interactions to take place. Now, let me proceed and expound on the other scripts that are supposed to make the proposed scheme to work. After the pre-game variables are initiated, the program needs to allow the player to physically engage with the game. Therein comes the mouse-triggered message receptors. That might sound rather arcane but what they do is to intercept the messages or signals that are triggered mouse movement over the hotspots. Once the mouse is within the target areas or entering there any button action has been initiated, the following scripts receive and interpret the incoming message and pass appropriate arguments to the ensuing scripts. In other words, the following scripts function as veritable fulcrums without which no non-linear interactive communication is possible. As I already summarily described, the following the on mouseEnter handler intercepts any message that is triggered by the mouse pointer entering a target area. Once that message is intercepted the program allows the script to run and the first conditional check the intercepted message has to go through is the pDrag flag check.

```
on mouseEnter me
  if pDrag then
    exit
  end if
  -- open hand cursor
  sprite(me.spriteNum).cursor = 260
end

on mouseLeave me
  if pDrag then
```

```
    exit
  end if
  -- reset cursor
  sprite(me.spriteNum).cursor = -1
end

-- start a drag
on mouseDown me
  pDrag = TRUE
  -- closed hand cursor
  sprite(me.spriteNum).cursor = 290
  -- get click offset
  pOffset = the clickLoc - sprite(me.spriteNum).loc
  -- use locZ property to move this piece and all links to front
  moveToFrontZ(me)
end

on mouseUp me
  -- if not being dragged, then ignore
  if not pDrag then exit

  -- open hand cursor
  sprite(me.spriteNum).cursor = 260

  pDrag = FALSE
  -- set all linked pieces to current position
  -- check each piece to see if any can be linked to them
  repeat with s in pLinks
    newLoc = restrictLoc(me,the mouseLoc) - pOffset
    sprite(s).loc = newLoc
    sendSprite(s,#checkForNewLinks)
  end repeat

  -- move all sprites back to normal locZ
  moveToNormalZ(me)
end
```

```
-- send mouseUpOutsides to mouseUp
on mouseUpOutside me
  mouseUp(me)
end
```

As I already touched upon, it is necessary to allow only one action to take place at the same time by rejecting any other pairing action at other place on the computer monitor. That makes sense, because the name of the game we are developing here is to match pieces and make one whole picture. (That is, in the mundanest word, a jigsaw puzzle.) As you may well have noticed by now, the series of mouse handlers are interrelated. They are all somehow conditioned upon the pDrag property. The first instance below forces the script to leave the current line if the pDrag property evaluates to 1 or true; otherwise, the script allows the mouse pointer type to change. In the second instance, if the same property evaluates to the same Boolean value then it behaves in the similar manner to the first handler segment and skip the final line, which change the mouse pointer type to the default arrow. The on mouseDown handler, on the other hand, forces the property to evaluate to 1, thus announcing to the whole scripts that any lines that are contingent on the flag property has to heed first then act. Of course, the any other script here referred to includes the preceding handlers that happened to be in the same script.[9] The on mouseDown handler also measures the distance between the mouse pointer location (the mouseLoc) and the relative coordinate location of the hotspot the mouse pointer is over. The line preceded by the mouse pointer manipulation line does exactly that. It measures the mouseLoc offset and assigns that value to pOffset. The next thing the script does is to call the customized function by moveToFrontZ(me), which happened to be located within the same script. Why does the function signify that the function refereed to resides within the same script? Because it takes as its argument *me*. The me indicates that the function called is part of the same script as it signifies the very same instantiation of the script, which it is a part. Therefore, even though a plural number of instantiations of the same script exists in the presentation, the function called is ensured to reside within the same script. Then what does the function do? The answer lies in the function thus referred to. If you look at the

[9] It may sound odd to you that the lines that are incorporated in the same script can be located outside the same script. But once the same script is instantiated and assigned different referential numbers the handlers and the scripts that are enclosed within them come to reside outside, in a sense. When that occurs there arises a need to communicate with other scripts through various means such as sendSprite().

funtion it reads as follows.

```
-- move all sprites in pLinks to be on top
on moveToFrontZ me
  repeat with s in pLinks
    sprite(s).locZ = 1001
  end repeat
end
```

That is, the mouseDown move brings the piece so clicked to the topmost sprite or channel so that while the piece is being dragged, the player can see where it is most likely to fit.[10] The dynamo of the whole script is the following preceded by the exitFrame handler.

```
  on exitFrame me
  if pDrag then
  -- drag piece
  newLoc = restrictLoc(me,the mouseLoc) - pOffset
  sprite(me.spriteNum).loc = newLoc
  -- drag links as well
  repeat with s in pLinks
  sprite(s).loc = newLoc
  end repeat
  end if
  end
```

Since the handler intercepts the message every time the movie passes through the said frame, which theoretically is one fifteenth of a second at the current setting, the lines sandwiched within the exitFrame handler can generate commands that can be executed virtually all the time. Needless to say, constant execution of commands could be inconvenient depending o the type of message sent from the exitFrame handler. That is why exceptional clause has to be set up according to the needs of the programmer. In this case, the exception is expressed by the conditional if. The exceptional clause allows the message to go through only if the property pDrag evaluates to Boolean TRUE. Or, more prosaically put, if any of the pieces are being dragged, the offset value

---

[10] Needless to say, the channel number the given sprite is evaluated to when the function is called is programmed to be 1001. The number guarantees that the piece in question comes on top simply because the channel number is the last channel used in this presentation. There is no abstruse logic behind it.

represented by the distance (assigned to the variable newLoc) is calculated and on the heel of that the loop command preceded by the repeat with is executed. That is simple and clear. But what about the function that seems to interrupt the smooth flow of the script? What does it indicate? Is it some kind of abstruse method that needs to be appreciated without understanding on the part of the programmer? No such mystical legerdemain is possible in the world of computing, of course. It is a function that calls the handler by the same name to execute the operations predefined by the programmer. Therefore, in order for the line in the if condition to properly function, the handler so referred to needs to work in the reciprocal a manner as originally intended. If the destination handler fails in its objective, then the whole script fails and the overall presentation conceived to interact with the player ceases to exist. That is the toughest reality of the scripting world. But before we hand down judgment on the integrity of the destination handler, let us go further down the script (that is, sequentially) and analyze how the referenced handler is designed to function in the whole context. It turns out that the handler is utilized to constrict the movement of the pieces.

```
on restrictLoc me, loc
  if loc.locH < 10 then loc.locH = 10
  if loc.locH > (the stage).rect.width-10 then loc.locH = (the stage).rect.width-10
  if loc.locV < 10 then loc.locV = 10
  if loc.locV > (the stage).rect.height-10 then loc.locV = (the stage).rect.height-10
  return loc
end
```

The first line of the segment is horizontal (here, represented by H) stricture. What the if clause is checking is the horizontal coordinate of the piece being dragged. If the piece, or more strictly speaking the mouse point, is less than 10 on the stage,[11] then the horizontal value of the loc is forced to 10, thus ensuring that the piece remain on he stage. The second line does the same on the opposite horizontal end of the stage. If the piece tries to move out of the stage, the conditional line forces the piece to move back to the location defined as 10 minus the stage size. Because the x or the horizontal value increases as the point shifts towards right on the stage, the value expressed as the width of the stage minus 10 is located on the right end of the stage as the player faces the game interface. What is noteworthy, although what is manifested here is so

[11] The stage is defined as the visible area on the monitor as the player engages in the interactive game activated by the scripts. It is usually a rectangle on which most of the triggers and actions are prearranged. In a nutshell, it is the user interface through which the players are introduced to the integrated world of entertainment and education, for which the presentation of the kind I am surveying should strive.

commonsensical in the world of scripting that it may on the contrary be hardly noteworthy, is that in the computing world nothing is simply linear. Although sequential positioning of lines may mean the order in which the scripts are to be executed, when it comes to handlers and functions and methods they call and are called depending on the flow of message that is trapped in the scripts that just tracing the sequential order of the lines oftentimes becomes meaningless. The case in point is the two methods that are linked here in this script. While the first one intercepts the incoming message it functions in such a manner that it channels the message in the way that are checked by the conditional clauses that are incorporated into the script. When he conditional clauses decides the message to be sent down onto one handler rather than other that also decides the directionality in which the message flows. As I mentioned already, in this script the linkage between the function caller within the exitFrame handlers and the restrictLoc method forces the message to jump umpteenth lines in order to fulfill the scripting designing that are manifested here. But that is not all. Once the message is sent down to the intended lines, it is again checked by the conditional lines to determine its relative horizontal and vertical position on the stage. And as soon as the relative position is determined (that is, the relative position of the piece being dragged) the initial message is once again sent back to where it was sent down from. But not without any transformation, as it were. When it reaches the initial position it is immediately incorporated into a calculation that eventually evaluates the relative position of the mouse pointer within the piece being dragged. In other words, the positional values are returned so that the flow of the message allows the program to come up with the value necessary to move the piece to the intended location. It is the charm of interactivity accomplished by scripting. No event has to be just linear. It can be entangled with uncountable other events that are triggered by still innumerable contingencies that are dependent on prearranged and predesigned rules. In that sense, every action and reaction can be thought as arbitrary. But at the same time, they are all within the design of the ultimate enforcer of rules. Once again, if I may run the risk of being extremely redundant, the spherical nature of computer generated game, in this case intended to be cultural encounter of the third kind (am I being facetious?), is the very epitome of the synchronicity of all the elements of cultural possibilities manifesting themselves. The non-linearity of computer-based presentation not only gives the participant in it the opportunity to experience the cultural episteme in its totality but also allows him the rare insight into the variegated epistemic nuances in their pseudo-synchronicity. Bet that as it may, when the conditions are met, the new loc will be assigned to each sprite (or, the piece in the

appropriate channel). That is the outcome when looked at from a cold mathematical perspective.

The next function that is needed in this script is that which is preceded by checkForNewLinks. As the name indicates, the handler decides whether the needed linkage has been made when the user drops the piece on any given spots. The way it works is as follows. When the method is called it every single sprite channel that is available in this movie is looked at by the loop. The first loop has the range of gFirstSprite to gLastSprite, which are variables that have already been defined as integers.

```
on checkForNewLinks me
  repeat with s = gFirstSprite to gLastSprite
    -- make sure the piece isn't already linked
    if getOne(pLinks,s) then next repeat
    -- see if the piece is close enough to be linked
    if closeEnough(me,s) then
      -- play sound
      if pLinkSound <> "" then puppetSound pLinkSound
      -- add to list of links
      addLink(me,sprite(s).pLinks)
      -- set all old and new linked pieces to the right location
      repeat with ss in pLinks
        sprite(ss).loc = sprite(me.spriteNum).loc
      end repeat
      -- check to see if puzzle is done
      checkDoneGame(me)
      -- need look no further
      exit
    end if
  end repeat
end
```

While the handler loops between those two channel numbers it does several things simultaneously in order to come up with the right linkage that in this case happens to be one continuous picture. First, the handler checks whether the sprite number is already included in the list of linked numbers. That function is expressed as getOne(pLinks, s). If the preceding check yields Boolean TRUE, then the proceeding lines are not necessary. The script runs again from the repeat line once again in that

case. If the result of the check is FALSE, or the linked sprite number list does not hold the current sprite number then the script runs further down and custom function closeEnough() is called.

```
on closeEnough me, s
  -- close enough horizontally
  if abs(sprite(me.spriteNum).locH - sprite(s).locH) < pCloseness then
    -- close enough vertically
    if abs(sprite(me.spriteNum).locV - sprite(s).locV) < pCloseness then
      -- see if the sprite rectangles are close enough
      if intersect(sprite(s).rect,sprite(me.spriteNum).rect+
          rect(-pCloseness-1,-pCloseness-1,pCloseness+1,pCloseness+1))
          <> rect(0,0,0,0) then
        -- close enough
        return TRUE
      end if
    end if
  end if
  -- not close enough
  return FALSE
end
```

Once again, as the name indicates, the called method checks whether the released piece is close enough to its supposed target. How the designated function accomplishes it is first by checking the horizontal distance between the two pieces in question. The distance has to be the absolute distance because no matter where on the stage the two pieces are located, what is needed is the positive integer value that will be compared to the number that is assigned to pCloseness. Conveniently enough, in case you became curious where in the world that value has been assigned to that local property pCloseness, the program comes equipped with preassigning mechanism preceded by on getPropertyDescriptionList while the program is still in the editing or compiling stage. Therefore, no matter how hard you look at the script there is no concrete number that is assigned to that property. At least not visibly. All that is taken care of through the interface that is generated by the combination of the script and the internal mechanism that is already embedded in the program. To make things simpler, the pCloseness value is utilized to check both the vertical and horizontal distance between the pieces. That is reflected in the values arrived at by subtracting the current locH and locV values

from the target sprite locH and locV values. Once those conditions are chedk, the script next determines the rectangular positions of the two pieces in question. That is because even if the two pieces are close enough, they need to be close enough in certain manners. Unallowable proximity has to be checked and rejected as faulty. Overlapping is a case in point. In order to come up with the right positionality of the two pieces, they have to be rectangularized, as it were, and the differentials between the two are to be measured by the line "if intersect(sprite(s).rect,sprite(me.spriteNum).rect+ rect(-pCloseness-1,-pCloseness-1,pCloseness+1,pCloseness+1)) <> rect(0,0,0,0) then." If the resultant rect evaluates to (0,0,0,0), then the current handler returns FALSE; otherwise TRUE or close enough. The latter response triggers the alignment sequence, in which the relevant pieces are brought to line up with each other side by side as the original pictorial member dictates.

Once the Boolean check results in 1 then the next action invoked in the calling handler is the sound feedback. But before the script executes that command, it determines whether the required sound actually exits or available. Look at the if statement that comes after the closeEnough() function. It makes sure that pLinkSound does not evaluate to an empty string "". The pLinkSound property happens to be another predefined property that the programmer allows the editor of the scripts to adjust through the aforementioned interface. In this case, the type or format the variable takes is set to be strings. Not any string but a string name of a sound clip to enhance the pleasure of interacting with the presentation. What is very convenient, however, is that the string variable can accept an empty string. That means the script gives editors of the presentation a choice whether they want to put an actual sound in the variable or not. Needless to say, when they decided to assign a sound to the pLinkSound variable, they must assign an exiting sound. Otherwise, the program complains and the whole presentation at that point comes to a grinding halt. Either the editor assigns a sound or not, the programs continues to run and calls a function named addLink().

```
on addLink me, list
  newlist = []
  -- add all the links for this sprite
  repeat with s in pLinks
    add newlist, s
  end repeat
  -- add all the links for another sprite
  repeat with s in list
```

```
    add newlist, s
  end repeat
  -- set all sprites involved to have the same list
  repeat with s in newlist
    sprite(s).pLinks = newlist
  end repeat
end
```

What it does is to add paired pieces into a list named newlist to further organize the matched pairs in the game. Once the addLink() function is run, the control is returned to the calling handler where all the relative location of the relevant pieces are positioned accordingly. As soon as that operation is over the handler checks if the entire game is finished by invoking the checkDoneGame function. When the handler is called, it sets upon calculating the total number of channels used in the game. That may sound redundant, considering the ongoing game that has been played out on the monitor by now quite some time. But what gives script-based interactive games so much resilience is the generalizations such scheme enables the programmers to make as they translate their concepts onto the monitor. That is, no matter how many channels the game occupies it is equipped with the latitude with which to handle all sorts of contingencies. It will not freeze or hang up simply because the number of sprites has increased from ten to twenty, for instance. Without the generalization that is instantiated by the run-time calculation of channel numbers, programmers would be hard put to modify each and every single line of their scripts contingent upon the smallest parameter changes. The first line of the function checkDoneGame() obviates that potentially very unproductive quagmire. Now, what it executes is a simple subtraction. The two terms are predefined global variables that already hold respective values. As the names indicate the first term on the right expression represents the last sprite that is used in the game. The second term indicates the first sprite. Subtracting further from the difference between the two gives the number of links made between paired pieces.

That is the general flow of the messages trapped by the handlers contained within the script. But what is purportedly the main generator of interactions is the exitFrame handler I already referred to. Because the message intercepted by the exitFrame handler is constantly generated when the movie is playing, the handler is manipulated to generate further messages that are needed to render the whole program truly reactive. If a certain trigger by the player is to elicit a group of reactions from the program, the editor can reasonably set up interceptive handlers that are capable of

providing fun and meaningful feedback to the player through the mechanism whose dynamo, as it were, is located right at the local where all the actions are initially initiated. But that does not mean the feedback should be simplistic. On the contrary. Although the centralized mechanism could be very concise and compact in its design, ramifications of the interceptive mechanism are limitless. All it takes is the designer/editor's ingenuity to stretch the interactivity to its limit. The ultimate stricture that is placed on the structuring is the instructive potential the program holds for the player. Without it no program has its raison d'etre. If a person who designs a computer-based interactive material to achieve some pedagogic end, no amount of entertaining value can justify the saccharine content that has no ultimate educational value. Needless to say, however, that should not cause any potential programmer to shun fun learning environment, in which players can immerse themselves in an infinitely Hollywood-like setup, as long as the end result is the increased knowledge that has accumulated every so imperceptibly to the players engaged in the program. Thus, the issue is how to tread the fine line between the two extremes. Whether one should provide educationally rich content that directly aims to find the receptive niche in the players' mind (in that case, *players* should be replaced with learners since the educational intent is so strongly put forward to the exclusion of the entertaining element in the program) or one should develop more entertaining content that aims to please the audience more than anything else depends on the ultimate design the programmers started with as they embarked on the perilous journey of developing an interactive presentation. But as I already mentioned, when it comes to creating the truly educational program that is both instructive and entertaining, the two extremes need to be weighed against each other. What are truly useful and what are ultimately more appealing and what are more user-friendly and more organic (if I may use such value-dependent and relative terms rather liberally) are some of the criteria to which programmers have to subject themselves before they decide to take one strategy above another. But what should count most in all the situations would be the amount of pleasure and instruction the designed program provides the players. That is not all. The two categories have to be satisfied in such a way that the players enter the virtual world and derive from that world what they ultimately seek in the least deliberate way possible. That is, the process has to be seamless. Without that seamless transition between the two requisite categories no computer-based learning material can find its justification. As long as one expects to find some intellectual reward from immersing oneself in the virtual world, the underlying premise needs to be firmly established in order for any such program to come alive in the considered circumstance, which is most

likely one on one with each playing both active and receptive roles simultaneously. Before going further into the teleological and utilitarian roles of computer-based games, let us look at other scripts that constitute the integral part of the current game I have been presenting.

The next script is a frame script. The one that is placed within the current frame so that all messages that arise from the objects in action within that frame reach and interact with the script. There is nothing magical about the process. Even within the previous script, the handler preceded by extiFrame did exactly that. The difference between the two is mainly that while the previous one intercepted any messages emitted while the playback head moved along the timeline, the current script only responds to the message that originates and is intercepted within the current frame. The distinction seems trivial but as I proceed to explain the functionality of the current one, the difference between the two will become sufficiently evident that the integrated nature of each and every single one of the scripts that constitute the whole game will be all the more appreciated. The first part I analyze should look partly familiar by now. At least the declarations that globalize two variables. As the two line comment indicates the two terms represent the first sprite and the last sprite numbers used in this game. In other words, they signify the range of channels that need to be checked every time match inquiry is requested through the user interaction. As a keen observer might have noticed by now, the similar variables are declared as properties in the couple lines further down. They are local variables used to determined the channel range occupied the puzzle pieces. As it happens the local properties are set in the getPropertyDescriptionList handler before the game actually starts. Therefore, the global variables are ready to be utilizes once the game actually starts. What is the advantage of declaring the same values twice, initially as local and ultimately global? There may be pros and cons fro the operation. But one of the advantages would be to allow the editor to remain on the level of the interface without prying into the innards of the game every time he needs to change the channel number range used by puzzle pieces. Another advantage would be to initialize the global variables without failure as the initialization takes place through the interface. It guarantees that at least the two integral components of the game are available right from the start. Still another advantage of defining the beginning and the ending sprite numbers is to pass a vital piece of information to the program so that it does not have to go through all the possible channels that could be used in the game. That could certainly be done, of course. But if anyone who is intent on polishing his interactive presentation must cut back on the time spent on generating the game. The less time it

takes to prepare the setup, the better and more satisfactory and more natural the whole experience is likely to be.

As has been amply expounded, the initial handler (getPropertyDescriptionList) allows the editor of the presentation to initialize the two variables before the program runs.

```
-- these two globals allow the sprite range
-- to be passed to all the sprites for use
global gFirstSprite, gLastSprite

-- these properties are just used to get
-- the sprite range
property pFirstSprite, pLastSprite

-- the only two parameters are to set the sprite range
on getPropertyDescriptionList me
  list = [:]

  addProp list, #pFirstSprite,
    [#comment: "First Puzzle Piece Sprite",
     #format: #integer,
     #default: 1]

  addProp list, #pLastSprite,
    [#comment: "Last Puzzle Piece Sprite",
     #format: #integer,
     #default: 1]

  return list
end
```

By default they are set to channel 1. But if the used channels occupy a range of sprites they cannot be identical. That means the default values (integer in this case) need to be set regardless of the true value used later when the editor actually set them in accordance with his design. Once the channel numbers are fixed, they are evaluated to the global variables in the beginSprite handler, which occurs first (not the very first but in quite the early stage of the presentation when it runs) when the designed game runs.

```
-- when the frame begins, set up the sprite range globals
-- and randomize the pieces
on beginSprite me
  gFirstSprite = pFirstSprite
  gLastSprite = pLastSprite
  randomizePieces(me)
end
```

That is the equation (or, rather equation lines) within the same handler. Since the game has to be different every time the player plays it, the next function calls randomizePieces( ) handler. When called, it does exactly that. But I think I need explain the procedure by which the program accomplishes that end. The ultimate objective, of course, is to scatter the pieces all around the stage and no two pieces overlap with each other. That is what the two initial comments indicate. Now the first line within the said handler sets the numTries variable to 0. It is a preparation fro the loop check that is to follow immediately after. The loop section both determines the available sprite numbers and fixes their coordinate location on the stage. The process is fairy complex because the handler needs to take care of a situation in which the handler fails to find an assignable sprite and when found to determined if the assigned channel is in fact outside the boundary set by the programmer (, which is most likely outside the visible area of the stage) and if the piece is found to reside within the boundary then if the piece overlaps another. In other words, the location assignment continues until all the exceptional conditions are cleared. That explains the noticeable lapse of time before the actual game is ready for play. But once again, count the number of loops that are embedded within the randomizePieces handler. Quite a few, or rather more accurately, quite deep. At a glance, that may seem just another layer of conditional traps that are thrown in for the sake of accomplishing a certain objective. That may be so. But what impresses me most is the versatility with which the program handles most variegated circumstances and the apparent seamless execution with which the program allows the players to completely immerse themselves in the world of multimedia presentation in spite of the complex procedures that in fact underlie their structural organization. If one is to take a cue from that subtle and masterly handling of contingencies, then it is not too much to expect a communication that is much more than a mere one dimensional conveyance of nuances[12] that seems to

[12] Characterizing conventional manner of conveying information, of any kind and content, as one-dimensional may be unfair, I admit. But what I would like to

have been the characteristic of the conventional dissemination of ideals, be it cultural, informational or of any kind whatsoever, by some use of computer based programs exemplified by the kind I have been demonstrating. But before I forget I list the handler in question below. No amount of abstract ratiocination would be enough to give the real feel of how the program executes without the nitty-gritty lines that actually run when the presentation is in progress.

```
-- move the pieces around the Stage until all are completely on the Stage
-- and none are overlapping
on randomizePieces me
  numTries = 0
  repeat with i = gFirstSprite to gLastSprite
    repeat while TRUE
      -- see if it may be locked up because of lack of sprite
      numTries = numTries + 1
      if numTries > 1000 then
        i = gFirstSprite
        numTries = 0
      end if
      -- pick a random location
      x = random((the stage).rect.width)
      y = random((the stage).rect.height)
      sprite(i).loc = point(x,y)
      -- see if the sprite is hanging over the edge
      if sprite(i).rect.left < 0 then next repeat
      if sprite(i).rect.right > (the stage).rect.width then next repeat
      if sprite(i).rect.top < 0 then next repeat
      if sprite(i).rect.bottom > (the stage).rect.height then next repeat
      -- see if the sprite is overlapping another
```

---

emphasize here is the versatility and multidimensionality of script-based approach here demonstrated. If one function is so flexible to meet so many exceptional conditions, then a group of them, also fully demonstrated by now in my essay, should be enough to meet the challenges that might have been overwhelming in the days when all information and communication had to rely on hard-copy based and mostly linear media. If a group of them could meet the challenge that might have been formidable just a decade ago, then think what a group of those groups of functions could accomplish. Am I just blabbering that the age of information revolution coincided with the dawn of new human intellection—the way people think, store knowledge and communicate with each other?

```
        touchingOtherPiece = FALSE
        repeat with j = gFirstSprite to i-1
          if j = i then next repeat
          if sprite j intersects i then
            touchingOtherPiece = TRUE
            exit repeat
          end if
        end repeat
        if touchingOtherPiece then next repeat
        -- this piece is set, go on to next
        exit repeat
      end repeat
    end repeat
  end
```

The last bit of handler that is needed to keep the playback head in the target frame is the following function.

```
  on exitFrame
    go to the frame
  end
```

It keeps the playback head looping in the same frame unless escape mechanism is triggered. The end result of the exitFframe handler is that all the scripts that reside within the target frame respond when the relevant message is generated. That simplifies the script structure because all the relevant messages are guaranteed to be intercepted by the related handlers without any exception. Since the existence of the sprites within any given frame becomes an absolute condition for the proper execution of the scripts that are attached to those sprites, once the playback head exits those frames messages that are supposed to be trapped by those scripts are either wasted or released to wrong scripts or handlers, causing the program to object. From another perspective, the looping mechanism is a great way to create a space in which all kinds of interactivities to take place. Because the predefined structure resides within the very space where two parties of programmer, represented as the interface equipped with inner correspondent mechanism, and the player who faces the mechanism, everything that is fed into such dual organizational setup would be met with an intelligent response that is, anthropomorphically speaking, endowed with three-dimensional insight. I know my cogitation is becoming a little too abstruse but the way scripts such as these function promises revelation of facets that have been surfacing and submerging

alternately throughout history which is rife with possibilities to develop new modus to disseminate ideas, communicate with others and present human consciousness.[13]

As far as the main scripts are concerned, the ones I cited above satisfy all the interactive needs of the game here presented. Needless to say, there are other essential elements needed to smoothly deploy the mechanism designed for the realization of the kind of inter or trans cultural/linguistic activities (in other words, all the epistemological engagements there are in the world) adumbrated by the title. For instance, sounds are integral elements that enhance the interactively evolving experience as the user enters the virtual world generated by the program. As the term multimedia amply demonstrates, the texture and shades of epistemological nuances they aid to produce when the two parties (pardon again my anthropomorphic tendency) engage themselves in the uniquely organic activities malleably complement the intellectual absorption that is presumed to take pace as the player's mind vigorously confronts the scenes that development before its eyes. That is easier seen if you compared the same game with and without the relevant sounds. In my sample movie,[14] which is another word for the presentation that has been repeatedly used, sounds are short enough to minimize the load time as the program starts automatically. (Yes, there is a choice on how to start the program. But as a finished form, it is designed to launch by itself as the intention on the part of the player is manifested when he inserts the CD-ROM containing the program.) At most they are forty seconds long. Produced for monaural playback with the sample rate of 22.05KHz and the resolution of 8 bits, they make up 700 KB to 1000 KB at most. Because sounds play better if they are loaded before the entire presentation starts (as opposed to streaming playback as sound bytes are processed as they are streaming into the processing unit) you cannot afford to make their size too large. It is a pity that you cannot actually hear the sound clips I have prepared for this presentation. But suffice to say that I have made six sound files to enhance the pleasant ambiance in which the players can immerse themselves in seamless interactivity. They are, as I already

---

[13] The point here made seems rather grandiose. My emphasis rather might have been on the diversification and enlightenment of the ideas and everything that is associated with them, both epistemologically and semantically, through the means made possible from the development of computer-based script language. (Here, the technological advancement, most conspicuously culminated in readily available hardware, is implicit.)

[14] Unfortunately the sample movie here referred to cannot be distributed with this essay because of the constraint placed on the author. The general concept expounded upon in this essay would be easily comprehended if you were to actually view the movie. I must apologize for the inconvenience.

suggested sized between 4.5 KB and 53.7. (Actually much smaller than what I would have used in other circumstances.) They are compact because the game does not necessarily require bulky files to keep the player absorbed. Rather, what is needed is the bare minimum yet imperceptibly fun elements to organically expedite the game as the player engages in the multi-faceted epistemological encounter. The key word is diversion. Not a meaningless, absent-minded activity usually associated with whiling time away but teleologically oriented self-absorbing activity that is unconsciously (i.e. unbeknownst to the players concerned) knowledge enhancing. If all those small files are enough and sufficient to fulfill that condition it is all the better that their sizes are limited to the minimum. The difference between sound and no sound could be significant. As the two pieces are brought close to each other (or, more accurately, within the distance predefined by the script in the game) they are sucked into their relative location and at the same the link sound is played. If the player gets no feedback he may be subtly affected by the sense of unfulfilment as he is put in a state of uncertainly whether the two pieces were actually conjoined or they happened to be pulled into each other by some quirk of the mouse movement. The small blip of a sound not only eliminates that uncertainly but, if everything worked well, allows the player to fully concentrate on the activity that is in progress before his eyes with himself as the protagonist. At least one other sound is used to announce the inception of the game. That is a little embellishment to segue the player into the right diversionary mood. The rest of the sound clips are used to announce a transition within the game such as when the player completes the puzzle. They are set so that only one of them plays randomly when the game runs a cycle. The randomized strategy is to give the player the sense of progress as each cycle is ushered in by a different sound (determined of course by the randomize() function, meaning that the same sound could play consecutively). Another element that imparts a sense of freshness to the game is that the pieces disperse differently every time the new cycle starts. That should give an added incentive to the player to go forward and play on because the player is likely to have an illusion that he is confronting an entirely new version as he enters the new cycle. What does this overall setup entail? For one thing, it will generate an environment in which all the traditional linear communication seamlessly gives way to a multi-directional expansive space filled with unlimited potentialities. Because the player is tempted to reside within the said virtual space for a prolonged period of time, he is optimally disposed to find every cue to expound upon and make sense of in the shades and nuances his personal disposition and accumulated experience allows. The script-based program is in turn capable of reciprocating to and interacting with the

player in the most flexible manner possible. The upshot of the linkage between the two is an ever-evolving environment where epistemological development explodes exponentially. Another implication of the interface that is mentioned above is the possible (and mind you that it is a very strong possibility) transformation that could take place in the realm of communication in its many avatars. Not only does communication, in its most general sense, could become self-generating but also trans-lingual. I had perhaps better keep myself deliberately vague as to what I exactly mean by the latter. But what I have in mind is the capability the script-based programs have of transcending mere verbal communication without appealing to the verbal significations per se. As the game here presented demonstrates, by utilizing various elements (multimedia, that is) it pans out in such a manner that the extra-lingual message and ideas are sent out to be intercepted by the other interactive party (so in that sense it is not an other in the normal use of the word) to be modified and processed into a signifying part that constitutes the evolving and meaningful whole. In a way, when the two parties that constitute the whole mechanism heretofore expounded upon are integrally intertwined and dovetailed coherently then they translate into almost organic constituents of the very flexible medium capable of imparting anything that is (humanly) conceivable. If we consider that great possibility emerging from such script-based mechanism, then is it too much to speculate a space in which extra- or even trans- lingual communication unconsciously taking place not only between the conscious subject and the program itself but among all those who are involved in the fluent mechanism connected by some kind of network? Some might object that it might be possible theoretically but is it in reality? Only time could tell. But seeing the pace and the outcome of the current technological development the communication of the kind outlined above seems quite plausible and will easily be a common thing in a matter of months or years.

Books Consulted

Chomsky, A. Noam. *Reflections on Language.* New York: Pantheon, 1975.

Epsetein, Bruce A. *Director in a Nutshell.* Sebastopol, California: O'Reilly & Associates, 1999.

Epsetein, Bruce A. *Lingo in a Nutshell.* Sebastopol, California: O'Reilly & Associates, 1998.

Herder, Johann Gottfried von. 1877-1913. *Samtliche Werke.* Ed. Bernard Suphan.

Berlin: Wiedemann.

Humboldt, Carl Wilhem von. 1903-1936. *Wlhelm von Humboldts Werke.* Ed. Albert Leitzmann. Berlin: B. Behr.

Oshige, Yoshiyuki. *Lingo Super Manual.* Tokyo: Ohmsha, 1999.

Prata, Stephen. *C++ Primer Plus.* Indianapolis, Indiana: Sams Publishing, 2000.

Rosenzweig, Gary. *Advanced Lingo for Games.* Hayden Books, 2000.

Sapir, Edward. *Selected Writings of Edward Sapir in Language, Culture and Personality.* Ed. David Mandelbaum. Berkeley: University of California Press, 1949.

Wierzbicka, Anna. *Semantics, Culture, and Cognition.* Oxford: Oxford University Press, 1992.

非直線的、非/超言語的コミュニケーションの可能性について

この論文ではスクリプトを基盤にしたゲームがいかに非直線的、非/超言語的コミュニケーションの可能性を示してくれるかを見ていく。その具体的なメカニズムの深層を細かく分析するため、ゲームのファンクションを可能にするそれぞれのスクリプトを機能ごとに調べ、ゲーム全体がその成立要素（マルチメディア）を介在させていかに非直線的、非/超言語的コミュニケーションの可能性を示唆してくれるかを論じてみた。