



**invent**

Imagine...Explore...&...Learn

# invent

Design of a programming language for children

By  
Pranav Mistry

Guide  
Prof. Ravi Poovaiah

# Approval sheet

The Visual Communication Project - 3 entitled '**invent - design of a programming language for children**' by **Pranav Mistry**, 03625008 is approved as partial fulfillment of the requirements of the Masters of Design Degree in Visual Communication.

Project guide

---

Chair person

---

External examiner

---

Internal examiner

---

# Acknowledgements

I would like to thank Prof. Ravi Poovaiah for guiding me with his abundant zeal throughout the project. I heartily thank Mr. Srinu Koppulu, MD, Microsoft India R&D for his kind support. A thank to Prof. Anirudha Joshi, Prof. Shilpa Ranade and Dr. Ajanta Sen Poovaiah for giving me their valuable insights on my project. I would also love to thank my friends Gajendra and Amisha for their constant help throughout the project work. I am also in debt of my friend Pushpam for his wonderful illustration for my story 'The School Train'.

At the last & the most, I thank all the children of the world for giving me such a wonderful opportunity to invent 'invent'.

**Pranav Mistry**

# Contents

<b>Abstract</b> .....	01	
<b>1. Introduction</b> .....	02	
. The beginning		
. Chapters		
<b>2. The School Train – one more story of Swami and friends</b> .....	06	
. ‘Malgudi days’ and I		
. The School Train		
<b>3. From Pavlov to Piaget &amp; from Bruner to Papert</b> .....	14	
. An overview		
. Constructivism and constructionism		
. Some experiments		
. I think ...		
<b>4. Why a programming language?</b> .....	21	
. A medium to think		
. The language to communicate		
. Learning problem solving		
. Learning how to learn		
<b>5. Children &amp; programming</b> .....	25	
. What is programming?		
. Programming languages		
. Children and programming languages		
<b>6. Ideation</b> .....	40	
. Interacting with children		
. Inferences		
. Invent is ....		
. Create, Animate and Instruct		
. ‘Everything is an object’		
. A drawing paper		
. Interaction design and system design		
. Refinements		
<b>7. ‘invent’</b> .....	50	
. What is ‘invent’?		
. The design		
. Create		
. Animate		
. Instruct		
. The in and out of ‘invent’		
. The water carrying train of Malgudi		
<b>References</b> .....	72	

# Abstract

A conceptual gap exists between the representations that people use in their minds when thinking about a problem and the representations that computers will accept when they are programmed. For most people learning programming, this gap is as wide as the Grand Canyon. Besides this, it is observed that one learns better by exploring. 'invent' is a research project under the guidance of Prof. Ravi Poovaiah initiated with the vision to help provide a medium to do the same. The goal of the project 'invent' is to design a programming language for children. The key ideas are to use representations in the computer that are analogous to the real world objects being represented or letting children create them as per their imagination and to allow those representations to be directly manipulated in the process of programming. The child can create objects, enliven their objects and use those objects to create their world creating challenges and solutions to those challenges. 'To know the world, one must construct it' and 'invent' is the medium to do it for children to learn themselves by exploring their ideas. The overall design of 'invent' provides a mean 'to create what a child wants', 'to decide how it will look, behave or act' and 'to instruct it what to do and when' in an intuitive way. In other words, 'invent' is a medium to think and explore. It is an environment that allows children to explore their imagination.

'invent' is an attempt to make programming more like thinking or rather say to have a medium to think. In brief, **invent = 'Imagine...Explore...&...Learn'**

# Introduction

## The beginning

I was in the 2nd standard. I don't remember exactly how it was looking at that time, but I have some memories that there was something that I used to connect to my Television set and had great fun at that time.

That something belonged to my father's friend who was a video shooter in my hometown Palanpur and used to shoot marriage videos. In those days he used to use handwritten colorful texts on paper to provide starting screen to those marriage videos like '*Manish weds Sangita*'. He put those paper posters in front of his camera to provide information like who marries whom, where and when. This new something was to replace this old method and to provide so called 'computerized' video shooting in which that paper screens were replaced by colorful graphics generated with that something.

I remember that my father used to play with it often. I was also keen to learn it and used to observe what they both (my father and that device) are doing. I was so happy that day when my father gave me a chance to play with it. Though there were a lot of instructions given by him along with, I was ready to obey all to use it. Father gave me a book and told me to copy something written in it as same as in the book. I did it.

Today when I think about those days I feel excited that I did programming when I was in 2nd. Yes, that something was nothing but the first computer I had used, **commodore C64** and those alien characters I typed was **BASIC**, a programming language.

After this first introduction, I met a new form of computing again in my 5th standard in school. We had an optional course 'computer fundamentals' in my school those days. I met those alien characters of BASIC again here in a much more instructional way. I learnt BASIC programming good enough. My interest and friendship with computers from early age urged me to have my graduation also in computer engineering. From BASIC to COBOL and from LISP to C++, I learnt so many of them. Today, I am very happy that I am here with my own, '**invent**'.

*'All understanding begins with our not accepting the world as it appears.'*

*- Alan C. Kay*

# Introduction

It began with my interest in programming and working with children. During this journey of designing a programming language for children and reading about thoughts of some great people on the same I came across some very motivating words of *Alan Kay*. Here I would love to refer to the excerpt from the Foreword written by *Alan Kay* for the book 'Watch what I do' which is a collection of presented work at the workshop on Programming by Demonstration that was held at Apple Computer in March, 1992. Alen Cypher has compiled the work.

"I don't know who first made the parallel between programming a computer and using a tool, but it was certainly implicit in Jack Licklider's thoughts about "man-machine symbiosis" as he set up the ARPA IPTO research projects in the early sixties. In 1962, Ivan Sutherland's Sketchpad became the exemplar to this day for what interactive computing should be like, including having the end-user be able to reshape the tool.

The idea that programming should be set up so it could be metaphorically like writing is harder to track down, but you could see it in Cliff Shaw's JOSS from the same early period. Besides being the first real "end-user" language, and the first attempt at a really "user-friendly" interface, it included a special terminal design adapted from a high-quality IBM electric typewriter that printed in two colors in lower and upper case on drilled fanfold 8\*11 paper so that the output was a direct extension of one's notebook.

The vague term "computer literacy" also surfaced in the sixties, and in its strongest sense reflected a belief that the computer was going to be more like the book than a swiss army knife. Being able to "read" and "write" in it would be as universally necessary as reading and writing became after Gutenberg. The Dynabook idea was a prime focus during this time as the kind of thing computers were going to turn into, forced by engineering possibility and sociological necessity.

# Introduction

.....continue from previous page

The analogy to reading was the easiest to see. If “reading” is the skill to be able to understand and use messages represented as gestures in a medium whose conventions are in close agreement between writer and reader, then the equivalent of reading on a computer would require the invention of a user-interface language that could universally frame the works of many thousands of authors whose interests would range far beyond those of the interface designers. “Writing” on the other hand requires the end-users to somehow construct the same kinds of things that they had been reading, a much more difficult skill.

Wally Feurzig and Seymour Papert had a different notion about the place of computer “reading” and “writing”: that like the reading and writing of books, it wasn’t just about getting and conveying information, but the very act of learning and doing them expands one’s horizons and adds new ways of thinking about the world. In other words, programming could be good for people, and thus some effort should be put into designing systems that would have pedagogical benefit for both children and adults.  
.....”

-Alan Kay

[Excerpt from, Kay, Alan. Foreword, Watch What I Do: Programming by Demonstration. Edited by Cypher, Alen. The MIT press. 1993.]

*Alan Kay’s words very well depict the picture of programming language design vision from historical as well as from the aspect of having a need of a learning medium for children. Some of the analogies like ‘reading’-‘writing’ are wonderful thoughts. Using different concepts like ‘Programming by demonstration’ or ‘Rule-based language’ there are so many efforts to have end-user programming languages in the history of computing. Though, intentions and visions behind these efforts have made those designs unique in themselves.*

*‘invent’ is my kind attempt to design a programming language for children. **Imagine...explore...&...learn.** With this vision I would love to put forward my ideas about ‘invent’ in the rest of the chapters.*

# Introduction

## Chapters

Through out the text of the report, I will go in a logical way. This chapter '**introduction**' gives a basic idea of 'what is?' Some of the thoughts of *Alan Kay* motivated me a lot in my thinking and work, too. The chapter is in real sense 'introduction' to what I want to convey next.

In the next chapter, I start with a story **The School Train**. Through the story I want to generate a visualization in the mind about for whom the 'invent' is for. At this stage let me clear again that 'invent' is a programming language for children I designed. I think stories are wonderful tool to explore and explain concepts. Characters of *Swami, Rajam* and friends and their stories can help well providing good scenario for my project.

In the chapter '**From Pavlov to Piaget & from Bruner to Papert**', theories and viewpoints of *Pavlov, Piaget, Papert, Bruner*, and others about how children think and learn are discussed in brief along with my personal ideas on the same. Some experiments I conducted are also mentioned in brief at the end.

I want to design a programming language. But, why? In the chapter, '**Why a programming language?**' I have tried to answer the same. This answer follows the logical base of discussion and thinking of the preceding chapters.

The generic understanding about programming languages and children are described the '**Children & programming**'. A brief about others' attempts to design a programming language for children are also mentioned.

'**Ideation**' talks about the core design process I went through along with the main ideas which make 'invent' what I dreamt for. It also discusses the process and interaction design issues of 'invent'.

Chapter '**invent**' speaks about what after all invent is and what the final design is. At last I have discussed some thoughts about what 'invent' can do with the help of my Malgudi freinds.

**Let's invent.**

# 'The School Train' – a one more story of Swami and friends

## 'Malgudi Days' and I

After a long, for I seen *Swami* first time in TV serial '**Malgudi Days**', recently I could manage to meet him again by reading the story book, '**Swami and his friends**'. '*Swami and Friends*' is one of the first novels written by *R.K. Narayan*, an English novelist from India. The novel is set in pre-independence days in India, in a fictional town - *Malgudi*, which has almost become a real place in India today, due to the wide recognition and popularity of Narayan's many novels.

'*Swami and Friends*' is the story of a 9-year-old boy, growing up during this particular time, his innocence, wonder, mischief and growing pains. He is a student at *Albert Mission School*, a school established by the British which gives importance to Christianity, English literature and education. His life is dramatically changed when *Rajam* joins the school and he and *Rajam* become friends. The central theme of the novel is growing up of young *Swami*. He is a spontaneous, impulsive, mischievous and yet a very innocent child.

I found in all these stories my own experiences and environs of childhood, adolescence and adulthood. *Malgudi* was my own *Palanpur* of primary school days. And *Swami* was myself-innocent ("Rajam, can you lend me a policeman?"), afraid of bullies, fearful of unseen forces and patronizing to boys smaller than I was. Everything I read in *R. K. Narayan*'s stories is familiar and dear to me; I have seen them all around me as I grew up. *Swami*'s grandmother was my own grandmother, indeed everyone's grandmother in India's middle-class households and every town is *Malgudi*. Though, *Narayan* described the age of pre-independence, one can see *Swami* in oneself or a kid today, that same innocence, mischief, wonder and curiosity. His character is a child in the fullest sense of the world.

The stories and amazing descriptions of events and environs urged me to use the characters *Swami*, *Rajam* and friends to explain my thoughts about 'invent'. More so over I think stories are wonderful tool to explore and explain concepts. In my case I think, *Swami* and *Rajam* are good personas for the scenario I am thinking about. Let's have one more story of *Swami* and his friends.

# 'The School Train' – a one more story of Swami and friends

## 'The School Train'

By Pranav Mistry

The bell hasn't rung for days in this summer. It looks all dead around the Albert Mission School of Malgudi. Even play grounds are all empty. Swami and his friends too got bored playing cricket in this summer vacation. At train tracks near Malgudi railway station they are sitting and waiting for the train to come. This time they have got two coins to put on the track to flatten them. Swami is interested in something else also other than this coin game. He loves trains. He dreams to be a train driver, and the most fascinating activity for him is to whistle the train. Rajam has a different perspective to trains, He is always curious to get how it works. Why those two wheels are connected in train engine? What that flagman does? Why?

"Rajam, its coming!" Swami is excited. All are curious, but they know what is going to happen. Last time the coin flattened, was about 9 days before. "Nothing will happen." Someone said. Swami replied, "I have said the coin, 'you need to be flattened, dear coin', he will." Rajam has no interest in this talk. Finally train went. They also went with their un-flattened coins and flattened faces.



# 'The School Train' – a one more story of Swami and friends



"This vacation is boring."

"Let's play Cricket."

"It is also boring now."

Playing almost for days in this summer Malgudi Cricket Club members are not in mood to have it. Anyway they play. The team is divided in two. This is the ground in almost front of Albert Mission School. Captain Rajam is quite good at cricket. A ball hit by him goes far away. Far away in the Albert Mission school.

"Yes, it went there in school, Swami, go get it."

"I.....?" Swami doesn't want to go there in the school as no one is there. He is frightened.

"Swami, you are the fielder, go, get it."

With his usual babbling and fear in mind he goes there. It is all alone there. No one is there. "There is no ball here." from the gate only he shouts. He is feared of going in the school campus to see there.

"It went inside, go in there."

With an unknown fear, He goes in the school campus. 'No one is here. It looks like that for years no one turned up there. Who told you to hit so hard?' He keeps babbling to himself.

"Where is the ball?", he keeps talking to himself. In search of ball He is almost in the back of the school. His sound didn't reach to friends,

"It is here.", some one said from his back.

"Oh, it is here!", Swami sees ball. He gets it and about to run back to ground. It was tree from where the

# 'The School Train' – a one more story of Swami and friends

sound "It is here" came, frightens him fully. He has no courage to see back to look. Swami runs at his best.

"Rajam, Rajam, Rajjjjjamm...."

"What?"

He finds him self on ground.

"What happened Swami? Why you are looking feared?"

"There,..."

"There what?" Rajam asked.

"Tree talks"

"What? You got mad."

"No, I swear, I heard."

"Let's go there"

"No, Rajam, please, not me."

Even after his hundreds crying and requests Rajam fetched him with all his friends there to school back. All are there at back of the school. Swami is still in his dream. "There is nothing Swami." Rajam said. Some of swami's friends start ringing the school bell. Days after they are back here in the school. Some are imitating head master at open stage. It seems that all again got life back. Swami is now not frightened as friends are all around with him. They played there. After having some fun they returned to town as soon as the sun returned. "They are dying, Can we give them water?" Rajam proposed.

"Water to whom? Those trees? How can we? Where is water?" Swami is full of question. "Swami, without water they are all dying. We should give them water. Our school water tank must have water. Let's do it."

"It is already dark Rajam."

"I know. I mean tomorrow."

The very next day morning at ground Rajam is there with some papers in hand. Swami is there besides him. He is smiling as they got something like a treasure and about to announce.

"My dear MCC friends, we are going to do something for these trees at back of our school. I and Swaminathan have come up with a plan for it." The head Rajam announced. MCC is Malgudi Cricket Club.

"We will make a canal."

"A canal?" almost all other than Swami and Rajam asked altogether.

"Yes, a canal carrying water from our school tank to all these trees at the back of our school. They are dying without water. This is how it will be." With saying this Rajam stared showing all the drawings of his canal proposals to friends that how water will come from main tank to all the trees of school.

# 'The School Train' – a one more story of Swami and friends

"And this is like that...." Swami added.

"What is that?"

"That is the main tank from where water will come." Rajam explained all in detail. All are smiling as same as Swami was as they are excited about this new activity. The day went with wonderful fun and a lot of work under the leadership of Rajam. They dig a canal. They needed to return home as sun returned too. It was very early morning and surprisingly all were again there at the back of the school and very excited to see their water irrigation system working. Swami opened the tap of main tank. Water started flowing through the canal. They have dig out the path for water. And this path covers all tree of school. All are happy seeing the water reaching trees one by one.

But, it stopped.

"Swami, why did you turn it off?" Rajam shouted.

...

"I did nothing! Water stopped coming out of tank." Actually, school tank got empty. Only 4-5 trees got some water. Rather than saying anything to each other, they left for railway station with their two unlucky coins. All are thinking in the way to station.

"We can get water from the hand pump near the tracks."

"Can we get a long pipe?"

"It is too long."

"We can fill water in buckets and fetch them to trees."

"But there are so many trees and we can't carry that much water that far."

the talk went on. Rajam has his mind somewhere else.

It is the time for train to come. Swami puts the same two coins there on track. They are waiting for the train. But all are sad. No one is saying anything.

"Let's make a train!" Suddenly Rajam says.

"A train?" this time all asked and swami was the loudest.

"Yes, a train, a train carrying water to the trees."

"how?" ... "what?" ... "but...." .... "Wonderful!!!!!"

The track is full of loud cheering even before the train comes. Without waiting for the train to come, Rajam, Swami and friends are at Rajam's place in no time to work on details of their discovery, 'the water carrying train'.

## 'The School Train' – a one more story of Swami and friends



I will describe you later how they come up with their train design that day at Rajam's place but let me describe that they made it. Yes, a water carrying train. They manage to get two small hand-lorries to fill with water and are going to use the dig out canal of earlier experiment with slight modifications in it as the track for one of the lorries, I mean the trains. Yes, there will be two trains. One train will carry water from hand pump to school and the other train will carry this water to trees at the back of the school. While trees are having water from a train the other train can have water from the pump. Such great ideas will run the train of MCC. The plan is ready.

From the very next morning all are working again with double zeal. They repaired an incomplete, unused rail path somewhat directing it towards the school. For that they managed some rail Patris near the track they used to have the coin game. Almost 3 days of great efforts ends today and they are ready to have green flag to their train system. In these days they have not played cricket nor the coin game. They were busy in their 'mission train'. Our train driver Swami is excited to have his first ride. And ....

# 'The School Train' – a one more story of Swami and friends

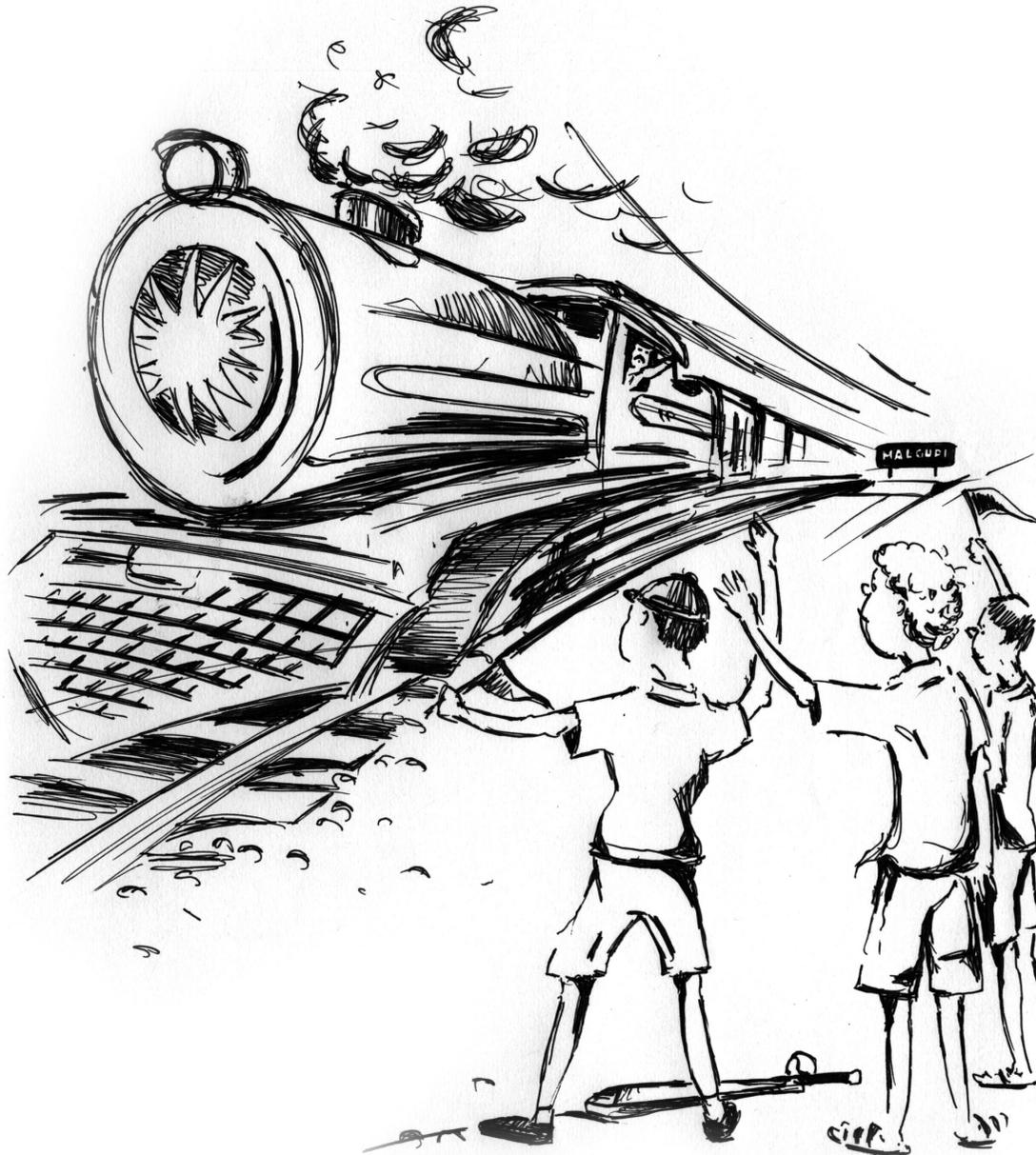


here it is. First train is full of water. And this time there is no mistake they did. They won. If you will ask “what you want? Swami.” He will answer “a whistle, with which I can cheer. Hurrah, Rajam.... he, he”

There is all live there. Trees are not talking. They are singing though. MCC proved their win. Rajam is quite happy and so same all too. The next day they come with exact calculations of water quantity and all and a schedule for their train system.

In some 2-3 days MCC could manage to expand the train routes a lot more than only school campus and started watering trees around the school, too. It is now almost no days 'Albert Mission Jail' (Swami says it so.) will start again.

## 'The School Train' – a one more story of Swami and friends



The bell rung after days today. All are quite surprised in school today seeing this creation of MCC. Swami is frightened again today thinking that he and team will be punished a lot for all this dig outs and manipulations they did. All were somewhat the same. After the prayer the head-master shouted “now say me, who did this?” Innocent Swami, Rajam and friends are standing with their hands in front to have their punishment. In school Swami closes it eyes average twice a day.

“Boys, I am very sorry that I forgot about these trees. And I thank you and congratulate you all for such a wonderful thing you did. Well-done you all” It was almost first time Swami is got praising form master. MCC cheered once again and other school friends joined that cheering.

Malgudi Construction Club is sitting besides tracks today again. Swami still doesn't go to school alone. He remembers that “It is here.” dream.

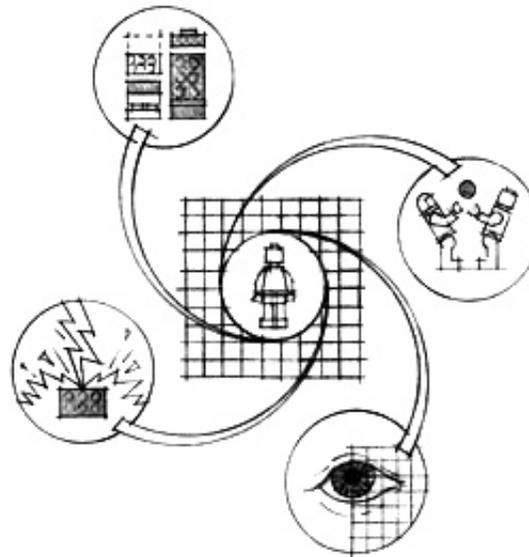
“Rajam, believe me, the tree talked that day when I went to get the ball back”.

“Swami....., put the coins fast, train is there!”

# From Pavlov to Piaget & from Bruner to Papert

Theories about how children think and learn have been put forward and debated by philosophers, educators and psychologists for centuries.

Here, I will be describing and discussing views as well as theories about learning and thinking in children that have been formulated and explored over past years. I will try in this chapter to give a brief overview of theories and thoughts by different schools of thinking, from *Pavlov* to *Papert*. The core idea of constructivism and constructionism are discussed after that. Some of the experiments conducted with children are also described in brief. At the end I will put forward some thoughts which lead me to this wonderful project.



images courtesy of *Legoseriousplay*

**invent** - imagine...explore...&...learn

## An overview

He was *Ivan Pavlov* who came up with the simple concept of **classical conditioning** which has led to an overwhelmingly successful multi-level approach to investigate into the mechanisms of learning. Today research on classical conditioning has increased to a complexity level that is hardly comprehensible but to a few experts in the various fields this science has spawned.

In the beginning of the 20th century, *Jean Piaget* conducted a program of naturalistic research that has profoundly affected our understanding of child development. This general theoretical framework of *Piaget* is called '**genetic epistemology**', (Piaget 1969) because he was primarily interested in how knowledge developed in human organisms. *Piaget* had a background in both Biology and Philosophy and concepts from both these disciplines influences his theories and research of child development.

# From Pavlov to Piaget & from Bruner to Papert

The theory of 'genetic epistemology' is basically a concept about cognitive structure. **Cognitive structures** (i.e. Schemas) are patterns of physical or mental action that underlie specific acts of intelligence and correspond to stages of child development. There are four primary cognitive structures (i.e., development stages) according to Piaget: sensorimotor, preoperational, concrete operational and formal operational. In the **sensorimotor stage** (0-2 years), intelligence takes the form of motor actions. Intelligence in the **preoperational period** (3-7 years) is intuitive in nature. The cognitive structure during the **concrete operational stage** (7/8-11 years) is logical but depends upon concrete referents. In the final stage of **formal operations** (12-15 years), thinking involves abstractions.

Cognitive structures change through the processes of **adaptation**: assimilation and accommodation.

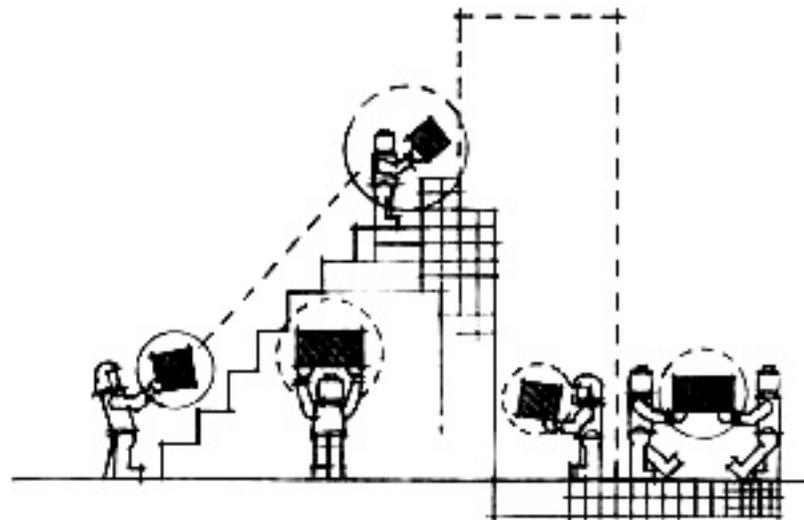
**Assimilation** involves the interpretation of events in terms of existing cognitive structure whereas **accommodation**

refers to changing the cognitive structure to make sense of the environment. Cognitive development consists of a constant effort to adapt to the environment in terms of assimilation and accommodation. In this sense, *Piaget's* theory is similar in nature to other **constructivist** perspectives of learning (e.g., *Bruner, Vygotsky*).

While the stages of cognitive development identified by *Piaget* are associated with characteristic age spans, they vary for every individual. Furthermore, each stage has many detailed structural forms. For example, the concrete operational period has more than forty distinct structures covering classification and relations, spatial relationships, time, movement, chance, number, conservation and measurement. Similar detailed analysis of intellectual functions is provided by theories of intelligence such as *Guilford, Gardner, and Sternberg*.

*Vygotsky's* theory suggests that **social interaction** plays a fundamental role in the development of cognition. (*Vygotsky 1962*) *Vygotsky* states: "Every function in the child's cultural development appears twice: first, on the social level, and later, on the individual level; first, between people (interpsychological) and then inside the child (intrapsychological). This applies equally to voluntary attention, to logical memory, and to the formation of concepts. All the higher functions originate as actual relationships between individuals."

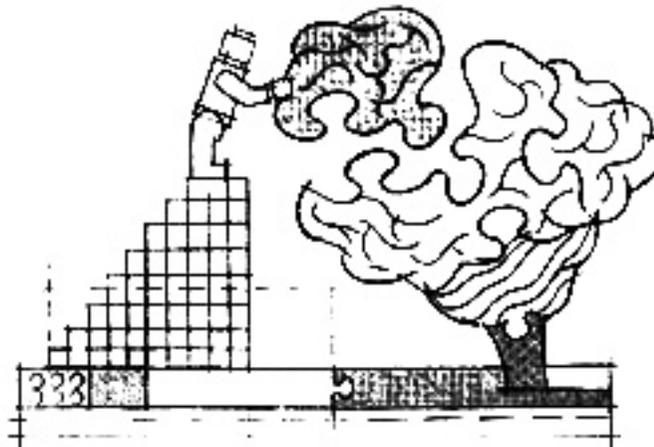
A major theme in the theoretical framework of *J. Bruner* is that learning is an active process in which learners construct new ideas or concepts based upon their current/past knowledge. (*Bruner 1990*) The learner selects and transforms information, constructs hypotheses, and makes decisions, relying on a cognitive structure to do so. Cognitive structure (i.e., schema, mental models) provides meaning and organization to experiences and allows the individual to 'go beyond the information given'.



# From Pavlov to Piaget & from Bruner to Papert

A mathematician and one of the pioneers of AI at MIT, *Dr. Seymour Papert* is internationally recognized as the seminal thinker about ways in which computers can change learning. *Dr. Papert* pursued mathematical research at Cambridge University from 1954-1958. He then worked with *Jean Piaget* at the University of Geneva from 1958-1963. It was this collaboration that led him to consider of understanding how children learn and think.

*Papert* proposed '**constructionism**' as a theory of learning and education. He proposes that Constructionism is based on two different senses of 'construction.' It is grounded in the idea that people learn by actively constructing new knowledge, rather than having information poured into their heads. (Papert 1980) Moreover, constructionism asserts that people learn with particular effectiveness when they are engaged in constructing personally meaningful artifacts (such as computer programs, animations, or robots). To show his ideas to world *Papert* and colleagues at MIT media lab came up with a wonderful turtle of LOGO. Still today it is a very powerful language and is taught as a part of school curriculum. So many flavors of LOGO in different languages are available.



## Constructivism and constructionism

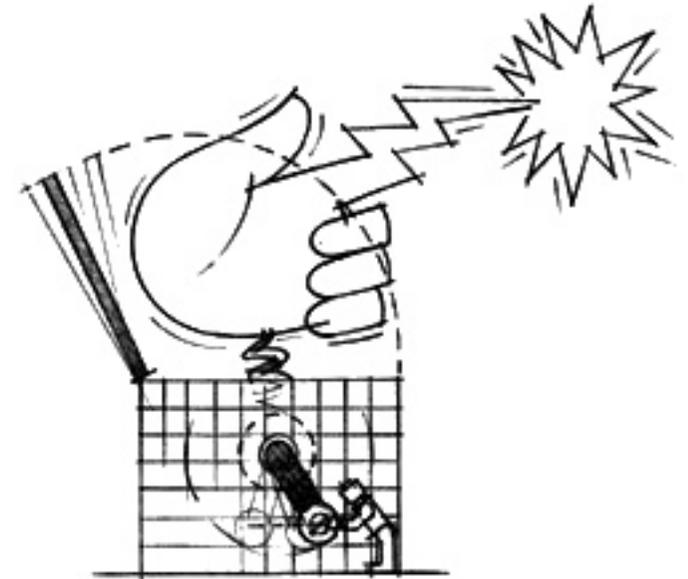
While exploring about constructivism one can trace back to the eighteenth century and the work of the philosopher *Giambattista Vico* who emphasized that humans can understand only what they have themselves constructed. A great many philosophers and educationalists have worked with these ideas, but the first major contemporaries to develop a clear idea of what constructivism consists in were *Jean Piaget* and *John Dewey*, to name but a few. *Vico* proposed **Constructivism as a philosophy of learning**. Constructivism takes an interdisciplinary perspective, inasmuch as it draws upon a diversity of psychological, sociological, philosophical, and critical educational theories.

In the constructivist paradigm, the accent is on the learner rather than the teacher. It is the learner who interacts with his or her environment and thus gains an understanding of its features and characteristics. The learner constructs his own conceptualizations and finds his own solutions to problems, mastering autonomy and independence. According to constructivism, learning is the result of individual mental construction, whereby the learner learns by dint of matching new against given information and establishing meaningful connections. In constructivist thinking, learning is inescapably affected by the context and the beliefs and attitudes of the learner. Here, learners are given more latitude in becoming effective problem solvers, identifying and evaluating problems, as well as deciphering ways in which to transfer their learning to these problems.

## From Pavlov to Piaget & from Bruner to Papert

If a child is able to perform in a problem solving situation, a meaningful learning should then occur because he has constructed an interpretation of how things work using pre-existing structures. This is the theory behind Constructivism. By creating a personal interpretation of external ideas and experiences, constructivism allows children the ability to understand how ideas can relate to each other and pre-existing knowledge.

While talking about constructivism, it is very important to discuss the theories of *John Dewey*, *Jean Piaget*, and *Jerome Bruner* that have certainly influenced our stance toward the nature of learning and teaching. As *Dewey* proposed, knowledge emerges only from situations in which learners have to draw them out of meaningful experiences. Further, these situations have to be embedded in a social context, such as a classroom, where children can take part in manipulating materials and, thus, forming a community of learners who construct their knowledge together. Children cannot learn by means of rote memorization; they can only learn by 'directed living,' whereby concrete activities are combined with theory. The obvious implication of *Dewey's* theory is that children must be engaged in meaningful activities that induce them to apply the concepts they are trying to learn.



*Piaget's* constructivism is premised on his view of the psychological development of children. Within his theory, the basis of learning is discovery: 'To understand is to discover, or reconstruct by rediscovery and such conditions must be complied with if in the future individuals are to be formed who are capable of production and creativity and not simply repetition'. According to *Piaget*, children go through stages in which they accept ideas they may later discard as wrong. Understanding, therefore, is built up step by step through active participation and involvement.

According to *Bruner*, learning is a social process, whereby children construct new concepts based on current knowledge. The children selects information, constructs hypotheses, and makes decisions, with the aim of integrating new experiences into his existing mental constructs. It is cognitive structures that provide meaning and organization to experiences and allow learners to

## From Pavlov to Piaget & from Bruner to Papert

go beyond the boundaries of the information given. For him, learner independence, learning through encouraging children to discover new principles of their own, lies at the heart of **effective learning**.

For *Hein*, constructivism, although it appears radical on an everyday level, 'is a position which has been frequently adopted ever since people began to ponder epistemology'. According to him, we have to recognize that knowledge is not "out there," independent of the knower, but knowledge is what we construct for ourselves as we learn. Besides, we have to concede that learning is not tantamount to understanding the "true" nature of things, nor is it (as Plato suggested) akin to remembering perfect ideas, 'but rather a personal and social construction of meaning out of the bewildering array of sensations which have no order or structure besides the explanations...which we fabricate for them'.

Besides providing opportunities for independent thinking, constructivism allows children to take responsibility for their own learning, by framing questions and then analyzing them. Reaching beyond simple factual information, learners are induced to establish connections between ideas and thus to predict, justify, and defend their ideas.

*'Learning is contextual: we do not learn isolated facts and theories in some abstract ethereal land of the mind separate from the rest of our lives: we learn in relationship to what else we know, what we believe, our prejudices and our fears. On reflection, it becomes clear that this point is actually a corollary of the idea that learning is active and social. We cannot divorce our learning from our lives'*

**- Hein.**

# From Pavlov to Piaget & from Bruner to Papert

## Some experiments

The best way to help understand the words of this great philosophers and psychologists was to observe and interact with children. From my various projects on designing for children as well as my interest to interact and play with them have given me some unique perspectives. To validate these perspectives and intuition, I gone through some experiments with children of different ages.

It was a compass box. I have thought they would at most come up with imagine it as a car or a building block. No, it was not only a compass box nor only a car, it was one's cricket bat, it was a space-shuttle(one young astronomer suggested), it was a harmonium, and a fridge, was a laptop(when opened), was Mickey's house and was a mobile phone, too. And so many other things, that was. I had given children a compass box with a figure of Mickey Mouse on it and asked them to think what that compass box can be other than the compass box.

I provided them with some random pictures of objects. A cycle, a monkey, a key, a car, a factory, a crow, star, trees, and some clouds. Interestingly, they came up with some wonderful imaginative stories that I and you are not able to imagine. Not only that the stories were full relations and events that 'when the cloud touched the car, it became red. Monkey was there on his cycle. And so, on...'. The children were actually moving the pictures to show how the objects interact in their stories.

Some other interesting experiments were to create worlds. Children were asked to do role playing in groups. They themselves came up with managing individual roles and what the one will do. It was a post office, and I still can't believe that they were that clever to explain what a post office is. One other group role played a hotel. Families and couples were there to have food. Counter man was billing and waiter was there to serve and take orders.

The core idea behind these and some other experiments was to have a better understanding about children's thinking and imaginations. To understand hows and whats about children, these experiments were the best medium, I think.

# From Pavlov to Piaget & from Bruner to Papert

## I think ...

Thus, the interaction, observation and a lot more experiments in the same line refined some of my thoughts and I could come up with some wonderful inferences like,

Children love activities like storytelling, role-playing and World creation

They 'Do, Relate, Perform'.

We can't imagine that 'what they can imagine'.

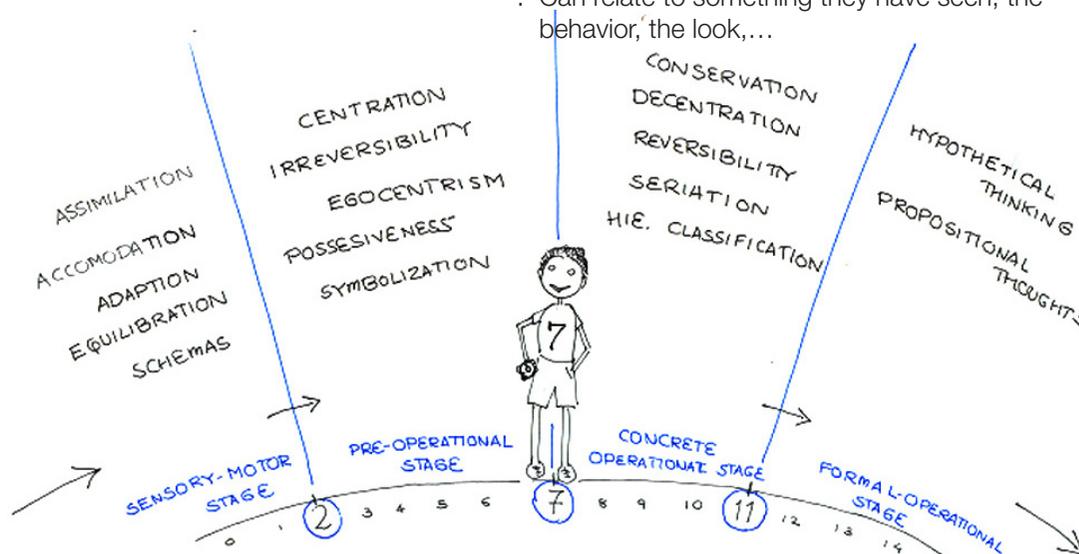
They are 'Ready to learn new things'.

This is that

- . Children can find use for things other than it is.
- . They can imagine something as something else.

This is like that

- . Can relate to something they have seen, the behavior, the look,...



**invent** - imagine...explore...&...learn

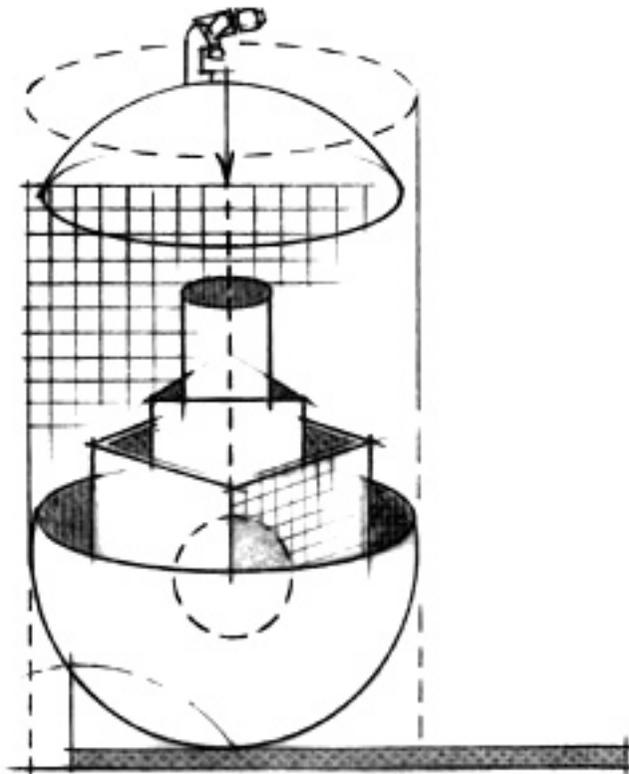
The **constructivist approach** to learning is based on a combination of a subset of research within cognitive psychology and a subset of research within social psychology. The basic premise is that an individual learner must actively "build" knowledge and skills and that information exists within these built constructs rather than in the external environment. However, all advocates of constructivism also agree that it is the individual's processing of stimuli from the environment and the resulting cognitive structures, that produce adaptive behavior, rather than the stimuli themselves.

As *Piaget* suggested, the cognitive structure during the **concrete operational stage** (7/8-11 years) is logical but depends upon concrete referents. In this stage children are learning by exploring things, events, and actions around. They learn concepts by validating by exploring their conceptual models about something. Even the theories behind the constructivist approach say that the child's learning is done in a hands-on approach. The children learn by doing, and not by being told what will happen. They are left to make their own inferences, discoveries and conclusions and imaginations.

With this base of thoughts, theories, interactions and inferences, I moved ahead in search of something that I dream for. 'invent'. That is a programming language for children.

# Why a programming language?

Inferences, insights, an inner urge and intuition, all together led me to the wonderful project '**invent**'. I think the reasons having '**a programming language for children**' are quite intuitive to me. I put them in words to explain in brief.



## A medium to think

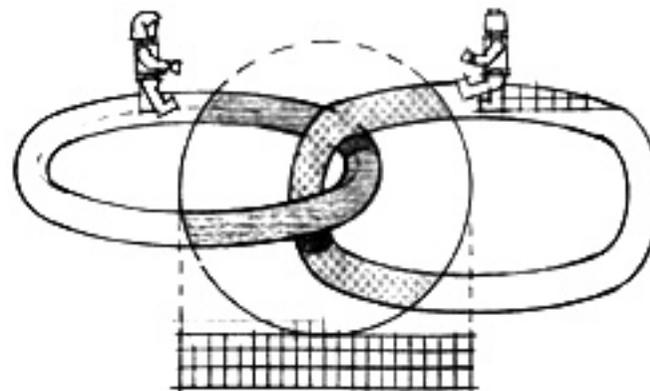
A pre-operational stage child needs something to touch when he counts. He counts things by touching them one by one with his parallel process of speaking it. Later he learns to use his fingers to do the same and also silently. With those fingers he learns and applies concepts of summation and subtraction. Having the medium of fingers or things to touch and count helps comprehend the concepts much better. A medium to think and explore one's imagination will definitely be a great help to learning and understanding the world.

As I mentioned earlier there is no question that a human with a goal wants to have the sub-goals ready made and at hand. One shouldn't have to learn about internal combustion engines, or even just hand cranking it, in order to drive an automobile. And agents that can be told goals and can go off and solve them have been valuable and sought after for as long as humanity has endured. On the other hand, it takes a very special value system for everyone to be able to exist as learning creatures, indeed as humans at all, in the presence of an environment that does all for them. The key idea is to have something to explore one's thoughts, ideas, and imagination; may be a medium to think.

# Why a programming language?

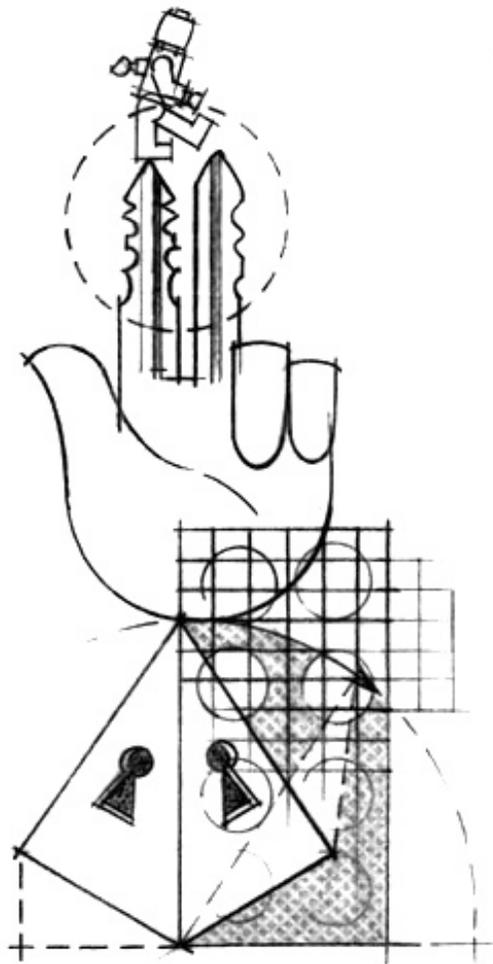
## The language to communicate

At some point it is necessary to understand something about thermodynamics and waiting until then to try to learn it doesn't work. Nature's rule is "use it or lose it". Most social systems that have incorporated intelligent slaves or amanuenses have "lost it". In fact the most never gained it to lose. In an ideal imaginative world in which we can make just about anything we desire, and almost everything we do can be replaced with vicarious experience, we have to 'decide to do' the activities that make us into actualized humans. We have to decide to exercise, to not eat too much fat and sugar, to learn, to read, to explore, to experiment, to make, to love, to think. In short, to exist. (Alan 1982)



The analogy to reading-writing by *Alan Kay* was the easiest to see. As he says 'reading' is the skill to be able to understand and use messages represented as gestures in a medium whose conventions are in close agreement between writer and reader, then the equivalent of reading on a computer would require the invention of a user-interface language that could universally frame the works of many thousands of authors whose interests would range far beyond those of the interface designers. 'Writing' on the other hand requires the end-users to somehow construct the same kinds of things that they had been reading. 21st century humans that don't understand the hows and whys of their technologies are not in a position to make judgments and shape futures.

# Why a programming language?

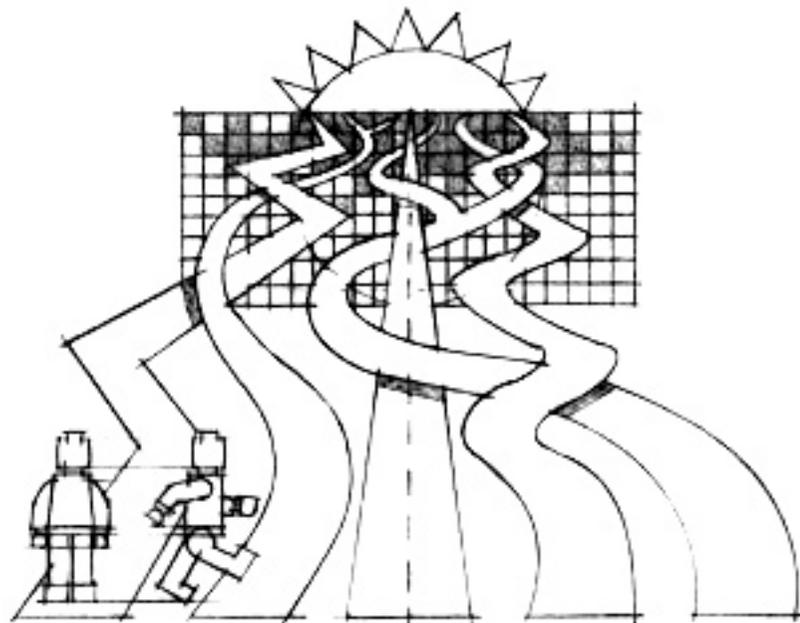


## Learning problem solving

Traditional cultures haven't set up schools. Children are set up by nature to learn the world around them by watching adult activity and playing imitation games. Most of the important things concerning what it means to be a member of a traditional culture and how to make a living in it are out in the open enough for children to learn through imitative play.

One of the special insights was that a main task of early education was to reshape the ordinary common sense that every child picks up into the 'uncommon sense' that is needed as the foundation for many modern ideas. Much of today's motivation for the education of children is vocational, to prepare children for the job market. More important is the need to raise problem solvers to join the 'big conversation' with enough background, thinking skills, and points of view to participate fully in the development. But there are larger considerations than job and problem solving. These have to do with actualizing the human possibilities that are as yet unrealized in children. With this in mind, we can argue not just for learning to read and write, but for fluent learning, and to be able to read literature and think and write about ideas. This gets to some of the older reasons for education. Deep experiences with deep ideas help grow deep people.

# Why a programming language?



## Learning how to learn

Learning how to exist needs not only learning how to repair one's car or how an aeroplane works. A wonderful aspect to learn is 'how to learn'. By this, I mean that one should be able not only to apply his learning about something to some other problems but also be able to learn from his experiences about the approaches as well how to have such approaches. It is like that. 'I know that if I will think about it like this I may get some solution, why can't I do this?' At overall one should be an ever learning and thinking creature. That is the life.

These are some core ideas to justify the need of such a programming language for children. I am not saying that this is the only way to do it. There might be other things. I found it interesting to have such a medium to think and explore. That is 'invent'.

# Children & programming

In simple words, programming is 'designing and writing a computer program'. A computer cannot understand the words spoken by us, and this warrants the need of a language, which can prompt the machine to carry the functions we desire. The pursuit of developing the perfect computer language, which started from the simple punched cards and algorithms, has taken a whole new dimension today.

The present day computer languages are immensely capable in making the computer perform varied tasks in fields like art, engineering, medical science etc. A complete programming language includes components like, the computational model, the syntax and semantics of programs, and the other pragmatic considerations that shape the language.

A lot about children and learning as well the vision behind the project are discussed in the earlier chapters. Here I try to provide a little background of programming and programming languages.

## What is programming?

Programming is the activity that involves a person writing instructions in a language closely related to the English language, in order to make a computer perform useful operations. These instructions collectively form a program, which is like a "recipe" for a computer, which it follows in order to perform useful operations. This program is then compiled by a compiler, into a form that the computer understands. This compiled program can then be run on a computer. The programmer must decide what the program needs to do, develop the logic of how to do it, and write instructions for the computer in a programming language that the computer can translate into its own language and execute.

Learning how to program a computer is very different from learning how to use a computer. The purpose of writing a computer program is to have the computer solve a specific problem for you, in a specific and repeatable way. As such, the programmer must understand the problem in all of its ramifications and then develop a logical solution to the problem that can be understood by the computer.



# Children & programming

## Programming languages

[ Taken from Ferguson, Andrew. **The History of Computer Programming Languages** [online]. Available from: [http://www.princeton.edu/~ferguson/adw/programming\\_languages.shtml](http://www.princeton.edu/~ferguson/adw/programming_languages.shtml). accessed on 21st February, 2005 ]

In the beginning, *Charles Babbage's* difference engine could only be made to execute tasks by changing the gears which executed the calculations. Thus, the earliest form of a computer language was physical motion. Eventually, physical motion was replaced by electrical signals when the US Government built the ENIAC in 1942. It followed many of the same principles of *Babbage's* engine and hence, could only be “programmed” by presetting switches and rewiring the entire system for each new “program” or calculation. This process proved to be very tedious.

In 1945, *John Von Neumann* was working at the Institute for Advanced Study. He developed two important concepts that directly affected the path of computer programming languages. The first was known as “**shared-program technique**”. This technique stated that the actual computer hardware should be simple and not need to be hand-wired for each program. Instead, complex instructions should be used to control the simple hardware, allowing it to be reprogrammed much faster.

The second concept was also extremely important to the development of programming languages. Von Neumann called it “**conditional control transfer**”. This idea gave rise to the notion of subroutines, or small blocks of code that could be jumped to in any order, instead of a single set of chronologically ordered steps for the computer to take. The second part of the idea stated that computer code should be able to branch based on logical statements such as IF (expression) THEN, and looped such as with a FOR statement. “Conditional control transfer” gave rise to the idea of “libraries,” which are blocks of code that can be reused over and over.

In 1949, a few years after *Von Neumann's* work, the language Short Code appeared. It was the first computer language for electronic devices and it required the programmer to change its statements into 0's and 1's by hand. Still, it was the first step towards the complex languages of today. In 1951, *Grace Hopper* wrote the first compiler, **A-0**. A compiler is a program that turns the language's statements into 0's and 1's for the computer to understand. This led to faster programming, as the programmer no longer had to do the work by hand.

# Children & programming

In 1957, the first of the major languages appeared in the form of **FORTAN**. Its name stands for FORMula TRANslating system. The language was designed at IBM for scientific computing. The components were very simple, and provided the programmer with low-level access to the computers innards. Today, this language would be considered restrictive as it only included IF, DO, and GOTO statements, but at the time, these commands were a big step forward. The basic types of data in use today got their start in FORTRAN, these included logical variables (TRUE or FALSE), and integer, real, and double-precision numbers.

Though FORTAN was good at handling numbers, it was not so good at handling input and output, which mattered most to business computing. Business computing started to take off in 1959, and because of this, COBOL was developed. It was designed from the ground up as the language for businessmen. Its only data types were numbers and strings of text. It also allowed for these to be grouped into arrays and records, so that data could be tracked and organized better. It is interesting to note that a COBOL program is built in a way similar to an essay, with four or five major sections that build into an elegant whole. COBOL statements also have a very English-like grammar, making it quite easy to learn. All of these features were designed to make it easier for the average business to learn and adopt it.

In 1958, *John McCarthy* of MIT created the LISt Processing (or **LISP**) language. It was designed for Artificial Intelligence (AI) research. Because it was designed for such a highly specialized field, its syntax has rarely been seen before or since. The most obvious difference between this language and other languages is that the basic and only type of data is the list, denoted by a sequence of items enclosed by parentheses. LISP programs themselves are written as a set of lists, so that LISP has the unique ability to modify itself, and hence grow on its own. The LISP syntax was known as “Cambridge Polish,” as it was very different from standard Boolean logic:

- x V y** - Cambridge Polish, what was used to describe the LISP program
- OR(x,y)** - parenthesized prefix notation, what was used in the LISP program
- x OR y** - standard Boolean logic

LISP remains in use today because its highly specialized and abstract nature.

The **Algol** language was created by a committee for scientific use in 1958. It's major contribution is being the root of the tree that has led to such languages as Pascal, C, C++, and Java. It was also the first language with a formal grammar, known as Backus-Naar Form or BNF. Though Algol implemented some novel concepts, such as recursive calling of functions, the next version of the language, Algol 68, became bloated and difficult to use. This lead to the adoption of smaller and more compact languages, such as Pascal.

# Children & programming

**Pascal** was begun in 1968 by *Niklaus Wirth*. Its development was mainly out of necessity for a good teaching tool. In the beginning, the language designers had no hopes for it to enjoy widespread adoption. Instead, they concentrated on developing good tools for teaching such as a debugger and editing system and support for common early microprocessor machines which were in use in teaching institutions.

Pascal was designed in a very orderly approach; it combined many of the best features of the languages in use at the time, COBOL, FORTRAN, and ALGOL. While doing so, many of the irregularities and oddball statements of these languages were cleaned up, which helped it gain users. The combination of features, input/output and solid mathematical features, made it a highly successful language. Pascal also improved the “pointer” data type, a very powerful feature of any language that implements it. It also added a CASE statement that allowed instructions to branch like a tree manner. Pascal also helped the development of dynamic variables, which could be created while a program was being run, through the NEW and DISPOSE commands. However, Pascal did not implement dynamic arrays, or groups of variables, which proved to be needed and led to its downfall. Wirth later created a successor to Pascal, Modula-2, but by the time it appeared, C was gaining popularity and users at a rapid pace.

**C** was developed in 1972 by *Dennis Ritchie* while working at Bell Labs in New Jersey. The transition in usage from the first major languages to the major languages of today occurred with the transition between Pascal and C. Its direct ancestors are B and BCPL, but its similarities to Pascal are quite obvious. All of the features of Pascal, including the new ones such as the CASE statement are available in C. C uses pointers extensively and was built to be fast and powerful at the expense of being hard to read. But because it fixed most of the mistakes Pascal had, it won over former-Pascal users quite rapidly. Ritchie developed C for the new Unix system being created at the same time. Because of this, C and Unix go hand in hand. Unix gives C such advanced features as dynamic variables, multitasking, interrupt handling, forking, and strong, low-level, input-output. Because of this, C is very commonly used to program operating systems such as Unix, Windows, the MacOS, and Linux.

In the late 1970's and early 1980's, a new programming method was being developed. It was known as Object Oriented Programming, or OOP. Objects are pieces of data that can be packaged and manipulated by the programmer. *Bjarne Stroustrup* liked this method and developed extensions to C known as “C With Classes.” This set of extensions developed into the full-featured language **C++**, which was released in 1983. C++ was designed to organize the raw power of C using OOP, but maintain the speed of C and be able to run on many different types of computers. C++ is most often used in simulations, such as games. C++ provides an elegant way to track and manipulate hundreds of instances of people in elevators, or armies filled with different types of soldiers. It is the language of choice in today's Computer Science courses.

# Children & programming

In the early 1990's, interactive TV was the technology of the future. Sun Microsystems decided that interactive TV needed a special, portable (can run on many types of machines), language. This language eventually became **Java**. In 1994, the Java project team changed their focus to the web, which was becoming "the cool thing" after interactive TV failed. The next year, Netscape licensed Java for use in their internet browser, Navigator. At this point, Java became the language of the future and several companies announced applications which would be written in Java, none of which came into use.

1957 FORTRAN  
1958 ALGOL  
1960 LISP  
1960 COBOL  
1962 APL  
1962 SIMULA  
1964 BASIC  
1964 PL/I  
1966 ISWIM  
1970 Prolog  
1972 C  
1975 Pascal  
1975 Scheme  
1977 OPS5  
1978 CSP  
1978 FP  
1980 dBASE II  
1983 Smalltalk-80  
1983 Ada  
1983 Parlog  
1984 Standard ML  
1986 C++  
1986 CLP(R)  
1986 Eiffel  
1988 CLOS  
1988 Mathematica  
1988 Oberon  
1989 HTML

Microsoft has extended BASIC in its **Visual Basic (VB)** product. The heart of VB is the form, or blank window on which you drag and drop components such as menus, pictures, and slider bars. These items are known as "widgets." Widgets have properties (such as its color) and events (such as clicks and double-clicks) and are central to building any user interface today in any language. VB is most often used today to create quick and simple interfaces to other Microsoft products such as Excel and Access without needing a lot of code, though it is possible to create full applications with it.

Programming languages have been under development for years and will remain so for many years to come. They got their start with a list of steps to wire a computer to perform a task. These steps eventually found their way into software and began to acquire newer and better features. The first major languages were characterized by the simple fact that they were intended for one purpose and one purpose only, while the languages of today are differentiated by the way they are programmed in, as they can be used for almost any purpose. And perhaps the languages of tomorrow will be more natural with the invention of quantum and biological computers.

There are different types of computer languages developed at different point of time, for different needs. A generic classification of language types is as follows: Logical Programming, Functional Programming, Imperative Programming, Concurrent Programming and Object orientated Programming. We can also have a genetic classification of programming languages, like which language derived from which and when.

A lot more classification schemas are there. Interestingly, I could not manage to have a classification from the user's perspective as there is none such existing.

# Children & programming

## Children and programming languages

Extensive study of attempts to have a programming language for children or end-users has provided me a wonderful picture. Though the most interesting finding of this study was that, the final aims behind these efforts to have a programming language by various people were different. It was quite a wonderful experience studying these designs like KIDSIM, BASIC, LOGO, SmallTalk, Squeak, JUDO, JOSS, ... there are lots of. Here I cite some of them which I found inspiring and interesting.

BASIC  
JOSS  
LOGO  
KIDSIM  
Squeak and Smalltalk  
HyperCard  
MicroWorlds

```
0010 PRINT 'HELLO THERE'  
0020 FOR X=1 TO 5  
0030 PRINT X  
0040 NEXT X  
0050 END  
RUN  
HELLO THERE  
1  
2  
3  
4
```

### BASIC

I love BASIC. The one is the first language which introduced me to the wonderful world of computing in my 2nd standard. Many of us know this language of 10 PRINT "HELLO", GOTO 50, ....

BASIC is often taught as a first programming language even today. It is developed in 1964 by *John Kemeny* and *Thomas Kurtz*. Though, BASIC is a limited language and was designed for non-computer science people it has provided a wonderful platform to its users as well as designers of other programming languages. Statements are chiefly run sequentially, but program control can change based on IF..THEN, and GOSUB statements which execute a certain block of code and then return to the original point in the program's flow.

# Children & programming

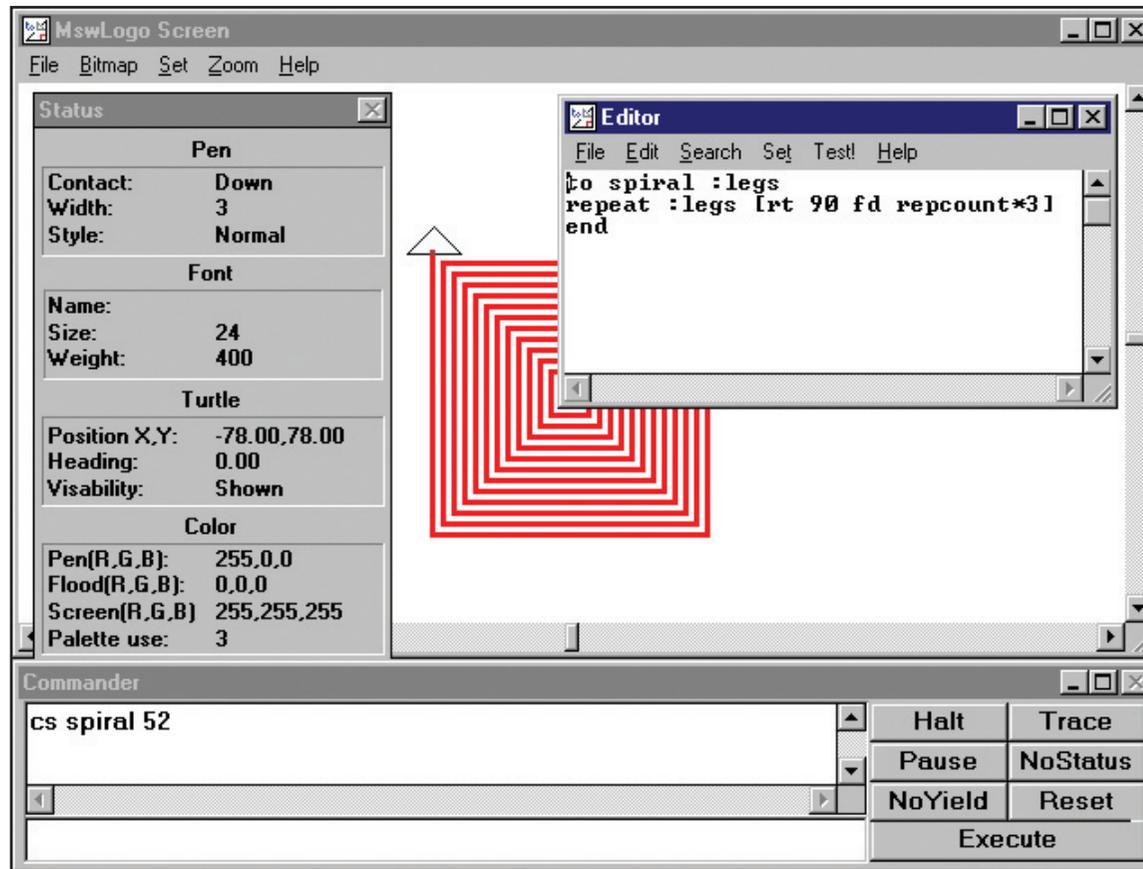
## **JOSS**

JOSS (an acronym for JOHNNIAC Open Shop System), was one of the very first interactive, time sharing programming languages. JOSS I, developed by *J. Clifford Shaw* at RAND was first implemented, in beta form, on the JOHNNIAC computer in May 1963. The full implementation was deployed in January 1964, supporting 5 terminals and the final version, JOSS In, supporting 10 terminals, was deployed in January 1965. JOSS was written in a symbolic assembly language called EasyFox (E and F in the US military's phonetic alphabet of that time). EasyFox was also developed by Cliff Shaw. JOSS was dubbed "The Helpful Assistant" and is renowned for its conversational user interface. Originally green/black typewriter ribbons were used in its terminals with green being used for user input and black for the computer's response. Any command that was not understood elicited the response "EH?". A large number (20 or more) of variants of JOSS were developed and implemented on a variety of platforms, including one for the IBM System/360 which was still in use up to the mid 1970s. In overall, JOSS was an invention before its time.

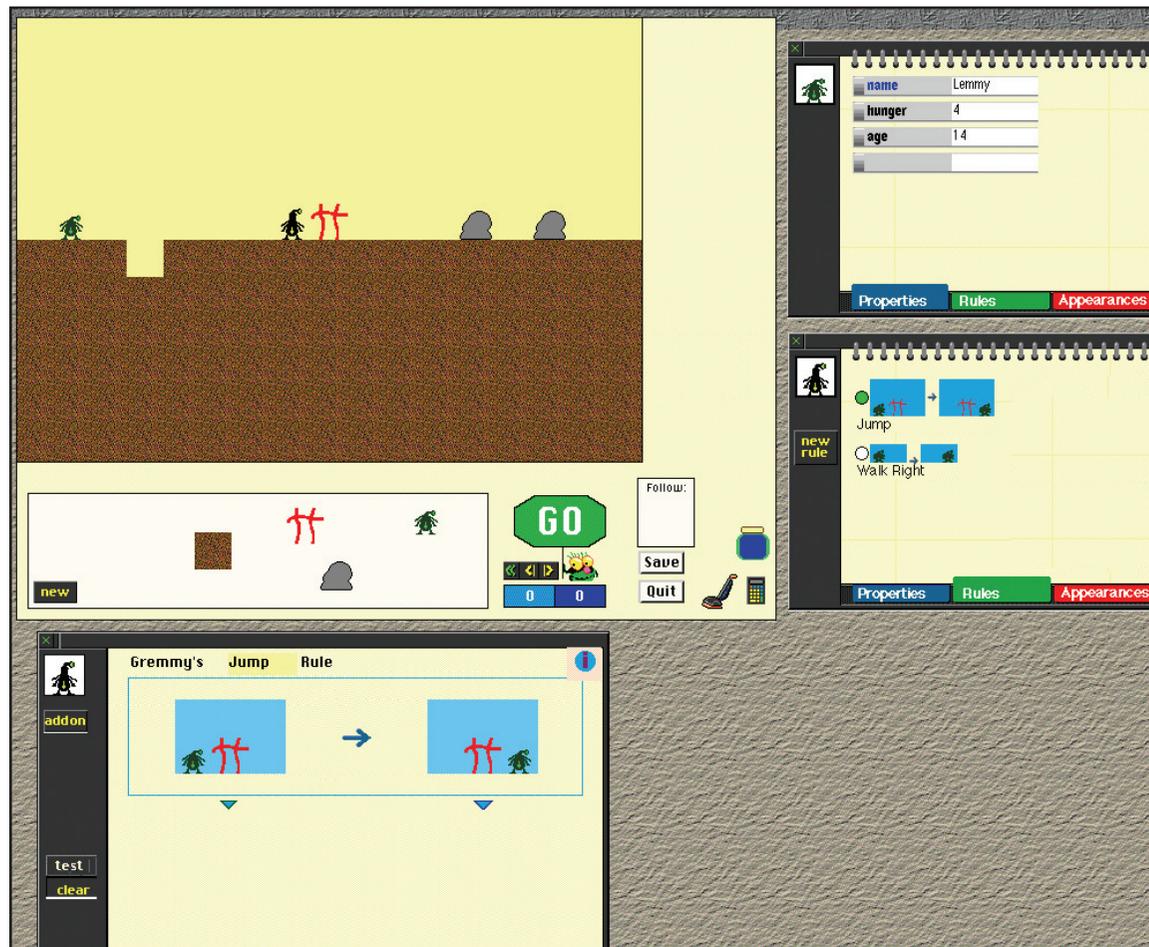
# Children & programming

## LOGO

The Logo programming language is an adaptation by *Wally Feurzeig* and *Seymour Papert* of the Lisp programming language that is easier to read. One could say that Logo is Lisp without the parentheses. Today, it is known principally for its “turtle graphics”, but it also has significant list handling facilities, file handling and I/O facilities. Logo can be used to teach most computer science concepts, as Brian Harvey does in his “Computer Science Logo Style” trilogy. It can also be used to prepare “microworlds” for students to investigate. There are over 130 implementations of Logo, each of which has its own strengths. A popular cross-platform implementation is UCBLLogo. MSWLogo, its freeware Windows derivative, is commonly used in schools in the United Kingdom. Comenius Logo is available in Dutch, German, Czech etc. and is worth considering. A modern derivative of Logo is a variation that allows thousands of “turtles”, each moving independently. There are two popular implementations: MIT StarLogo and NetLogo. These derivatives allow for the exploration of emergent phenomena and come with many experiments in social studies, biology, physics, and many other sciences. Although the focus is on the interactions of a large number of independent agents, these variations still capture the original flavor of Logo. The idea is that a turtle with a pen strapped to it can be instructed to do simple things like move forward 100 spaces or turn around. From these building blocks you can build more complex shapes like squares, triangles, circles--using these to draw houses or sail boats. The turtle moves with commands that are relative to its own position, “LEFT 90” meant rotate left by 90 degrees. A child could understand (and predict and reason about) the turtle’s motion by imagining what they would do if they were the turtle. Papert called this “body syntonic” reasoning.



# Children & programming

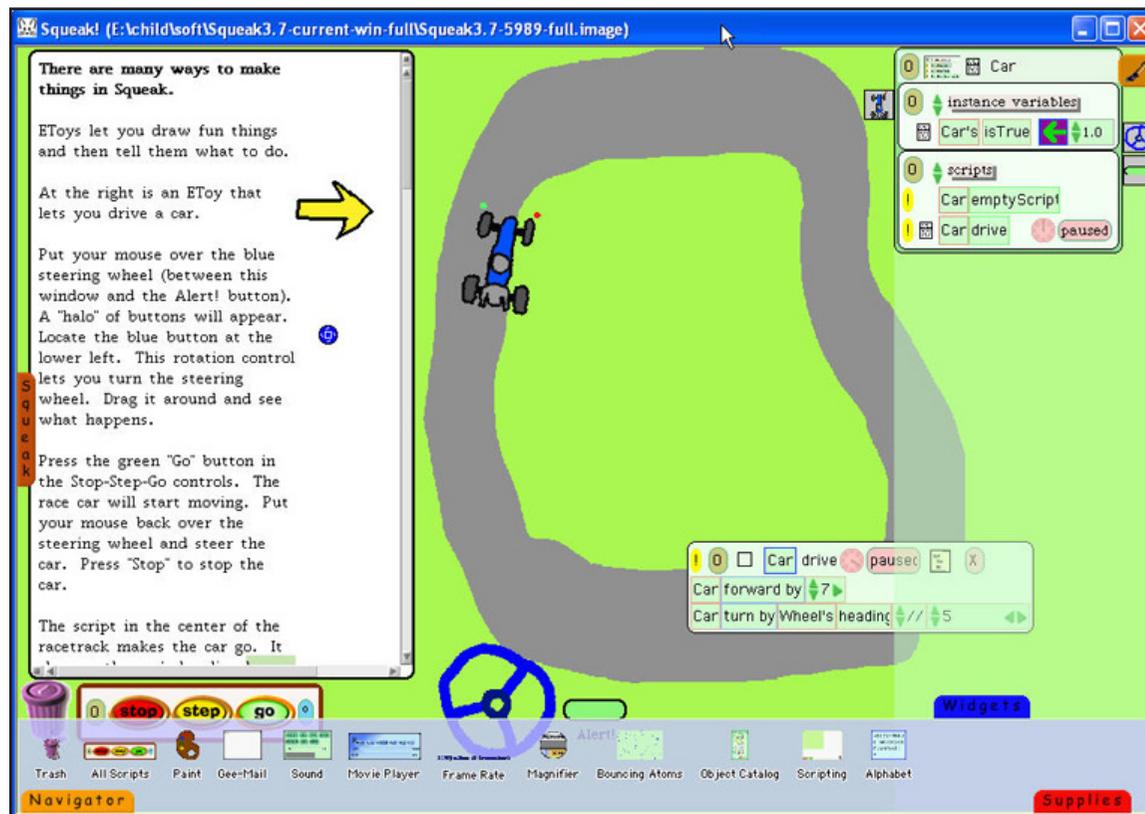


## KIDSIM

KidSim is an environment that allows children to create their own simulations. KidSim enables children to specify how characters are to behave and interact by providing a special graphical interface for specifying the spatial relations between characters. In addition, KidSim provides a graphical interface for creating, testing and manipulating properties, which maintain information about the character. Children can create animation and sound to make their simulations more exciting. Children are taught to use the program with a demonstration technique in which the system is set on “record mode.” Then the child goes through the program while the computer records so that users can concentrate on the game alone and not on the programming of their game.

KidSim has been changed and renamed Cocoa. Children have the option of putting their games that they create through Cocoa up on the internet.

# Children & programming



## Squeak

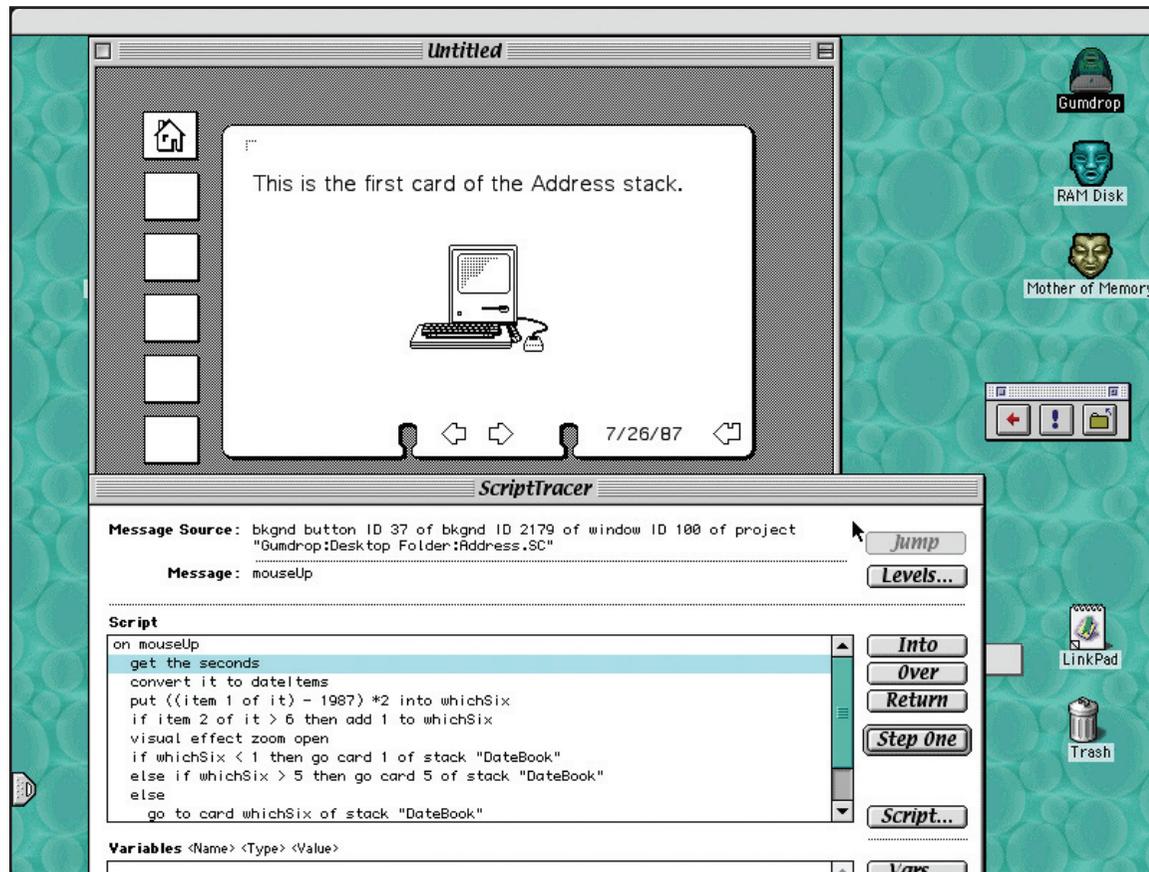
Squeak is a free open source implementation of the Smalltalk programming language. It is available on many platforms and the programs produced on one platform run bit-identical on the other platforms. It comes with an implementation of Morphic, Self's graphical, direct object-manipulation framework.

Squeak incorporates many of the elements *Alan Kay* proposed in the Dynabook concept, which he formulated in the 1960s. In overall, Squeak is a wonderful environment for children.

## Smalltalk

Smalltalk is a dynamically typed object oriented programming language designed at Xerox PARC by *Alan Kay*, *Dan Ingalls*, *Ted Kaehler*, *Adele Goldberg*, and others during the 1970s. The language was generally released as Smalltalk-80 and has been widely used since. Smalltalk's big ideas include: "Everything is an object." Strings, integers, booleans, class definitions, blocks of code, stack frames, memory are all represented as objects. Execution consists of sending messages between objects. Any message can be sent to any object; the receiver object determines whether this message is appropriate and what to do to process it. Everything is available for modification. If you want to change the IDE, you can do it, in a running system, without stopping to recompile and restart. If you want a new control construct in the language, you can add it. In some implementations, you can change even the syntax of the language.

# Children & programming

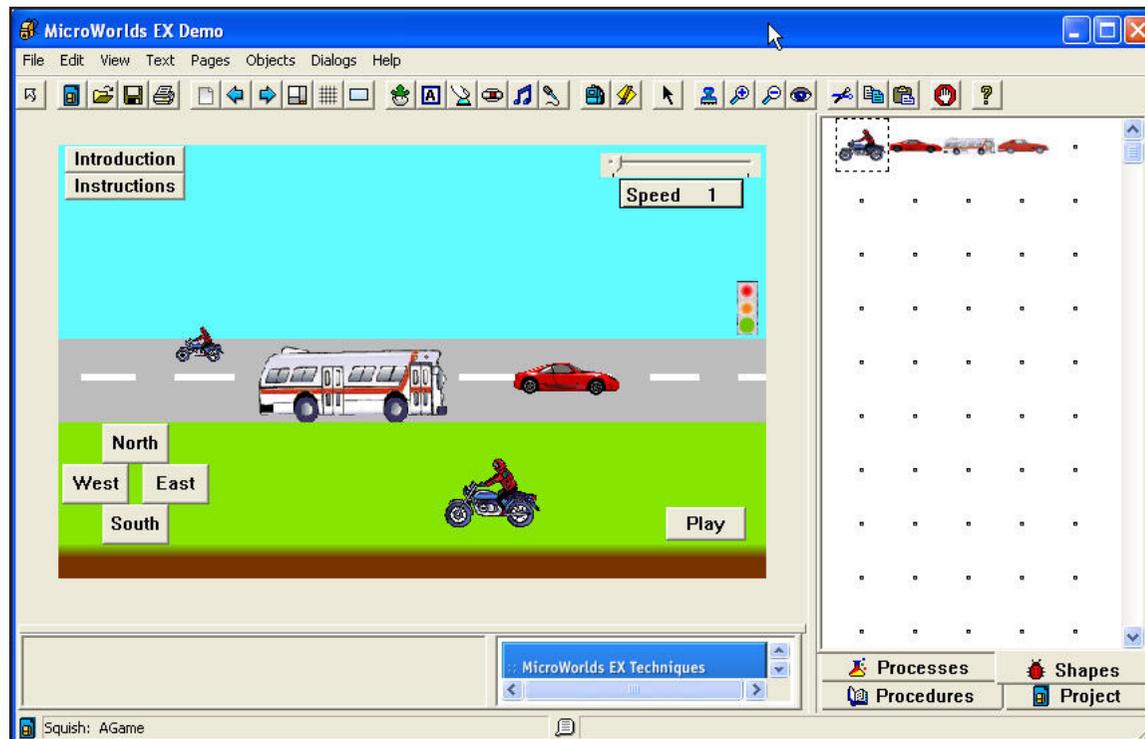


## HyperCard

HyperCard is an application program and a simple programming environment produced by *Apple Computer* which runs only in Mac OS versions 9 or earlier. It most closely resembles a database application in concept, in that it stores information, but unlike traditional database systems HyperCard is graphical, very flexible and trivially easy to modify. In addition HyperCard includes HyperTalk, a powerful and easy to use programming language to manipulate data and the user interface. HyperCard users often used it as a programming system as opposed to a database. HyperCard, a hypertext programming environment for the Macintosh introduced by Apple in 1987 was finally withdrawn from sale in March, 2004.

The HyperCard model consists of cards, and collections of cards, called stacks. You can connect the cards in various ways, and leaf through them the way you would with a set of Rolodex cards. In addition to data, each card can contain graphics and buttons that trigger other events, such as sound or video. Each object in a HyperCard system; stack, card, text field, button, or background can have a script associated with it. A script is a set of instructions that specify what actions should take place when a user selects an object with the mouse or when some other event occurs.

# Children & programming



## MicroWorlds

MicroWorlds is an environment in which students explore and test their ideas in this idea exploration and project creation environment. MicroWorlds EX is can be used to create: science simulations, mathematical explorations or interactive multimedia stories. In fact, MicroWorlds is a enhanced flavor of LOGO programming language. In one sense it is a LOGO with capabilities of GUI, Music and a ready-made library of characters and objects to use in simulations.

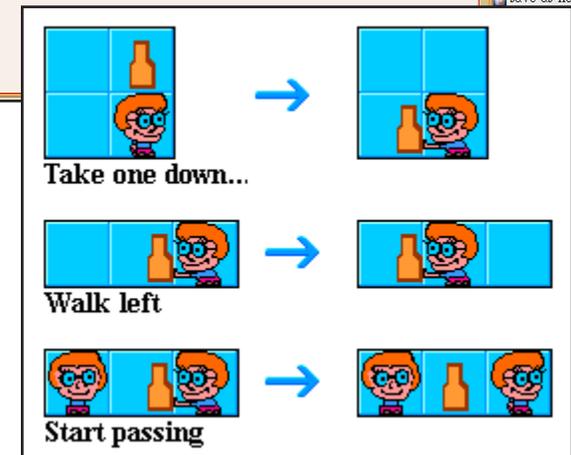
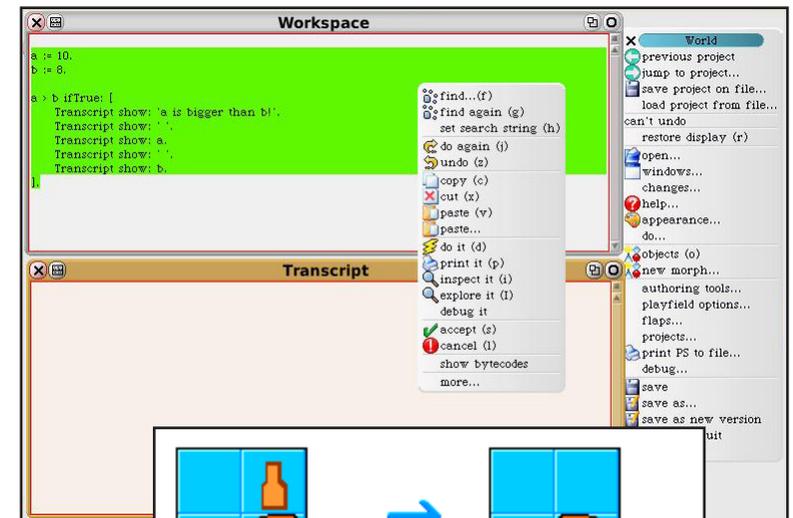
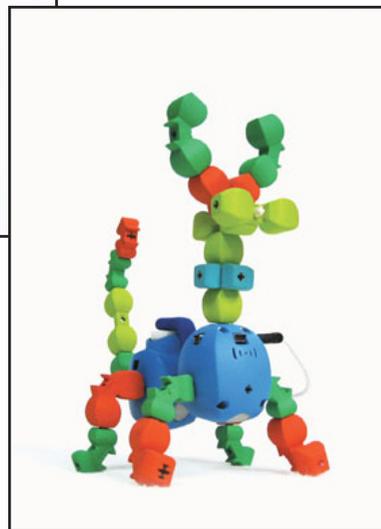
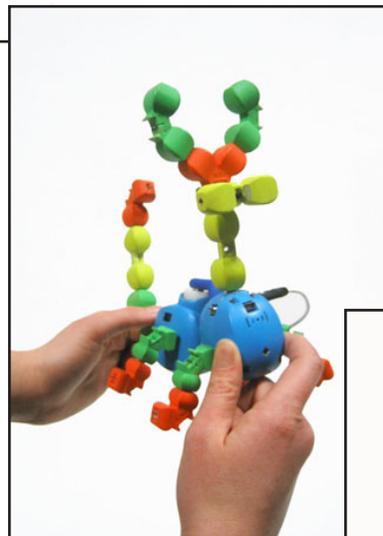
There are a lot more other nice tools for children. Just to name a few, there are KIDSIM, Stagecast Creator, Klik and Play... and the list goes on. Also, there are some nice research projects to design a programming language for children. 'invent' is my kind effort to add a one more name in the list.

# Children & programming



Other than these milestones mentioned there are lot more other interactive environments played important role in my studies in understanding about programming languages, children and their interaction with them. Some of them like KIDPAD are creativity or storytelling tools. There are lots of variations to the languages like LOGO are there available in different languages as well as with some changes in that. Lots of simulation tools for children are also there for children.

# Children & programming



Not only these environments but also some of the very powerful concepts like **programming by example**, **programming by demonstration** used in these languages and environment provided me a powerful base to validate my thoughts and come up with 'invent' with much powerful interaction and vision. '**Programming by doing**' concept of TOPOBO or '**rule making**' of Stagecast Creator and '**undo**' icon of Microworlds to **on-place menu** of Squeak are some of the really wonderful designs.

# Ideation

*'I can't understand why people are frightened of new ideas. I'm frightened of the old ones.'*

*- John Cage*

I do programming from my 2nd standard. Domain knowledge in the field of programming as well as my liking to work with children led me to this wonderful project. Thus, experience, inferences, analysis, evaluations and most importantly my intuition are there behind 'invent'.

**What it will be?**

**How the child will do it?**

**How it will help?**

The unclear idea about 'invent' started with such questions and led me to ideation. I will try to describe it in brief in the chapter.

I started with studying and analyzing ...

- . Languages (programming)
- . Learning theories
- . Designs
- . Interactions
- ...

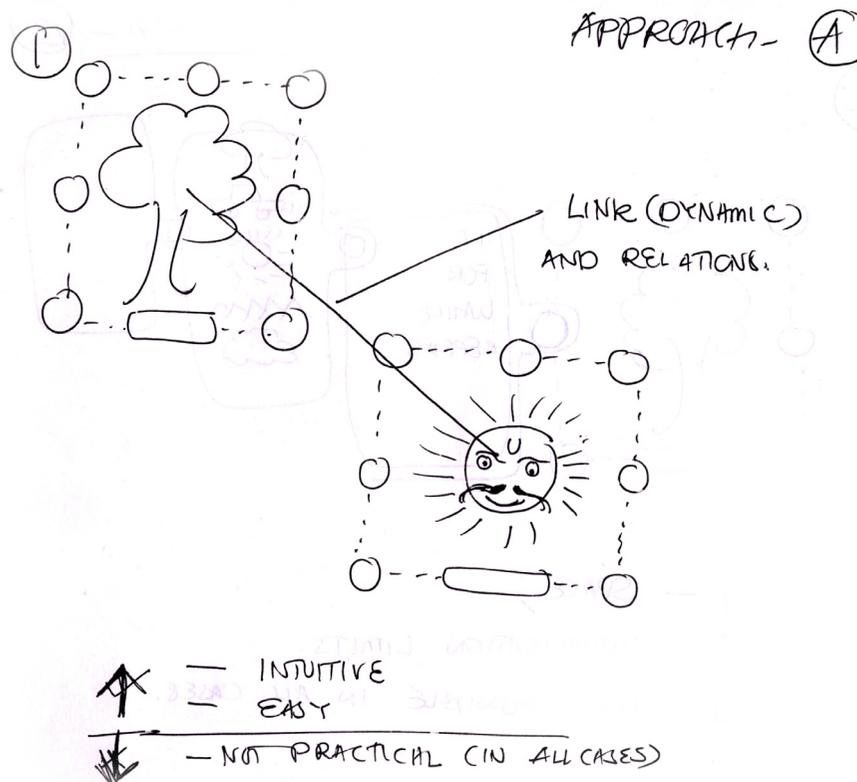
# Ideation



From different programming languages to learning theories and from child psychology to interaction design paradigms gave me a wonderful vision towards actualizing my dream. I love to work with children and I am very lucky that I got good opportunities to do the same with my projects, hobbies and interests as well. Here are some of wonderful opportunities of my interaction with children.

- . **MARBO**- an MSR research project to design a communication device for children
- . 'Ghost in the machine'- special project
- . Teaching computer to kids at **ACT computer center, Palanpur**
- . Teaching **table-tennis** in vacation to children
- . 5th std. **Computer class** at *VidyaMandir, Palanpur*
- . Interacting with children at **New Era School, Mumbai** and **Kendriya Vidyalaya, IIT Bombay**
- . Working under *Prof. Ravi Poovaiah* on [www.designforchildren.com](http://www.designforchildren.com)

# Ideation



Major of these interactions, analyses and intuitions are captured here through inferences. 'How children think and learn?' 'Why they love story-telling, role playing,...?' 'What will help them learn? not only maths and English but also to solve problem, to create something or to invent' Answers to these questions are what motivated to invent 'invent'.

## Inferences

- . Children love activities like storytelling, role-playing and World creation
- . They 'Do, Relate, Perform'.
- . We can't imagine that 'what they can imagine'.
- . They are 'Ready to learn new things'.
- . This is that
  - Children can find use for things other than it is.
  - They can imagine something as something else.
  - Ex. A pen as a rocket and play flying gave with it
- . This is like that
  - Can relate to something they have seen the behavior, the look,...
  - Ex. Can imagine a tree crying or a train engine cheering

All these led me to the equation

## Imagine...explore...&...learn

'invent' is

Learning by imaging

Learning by doing

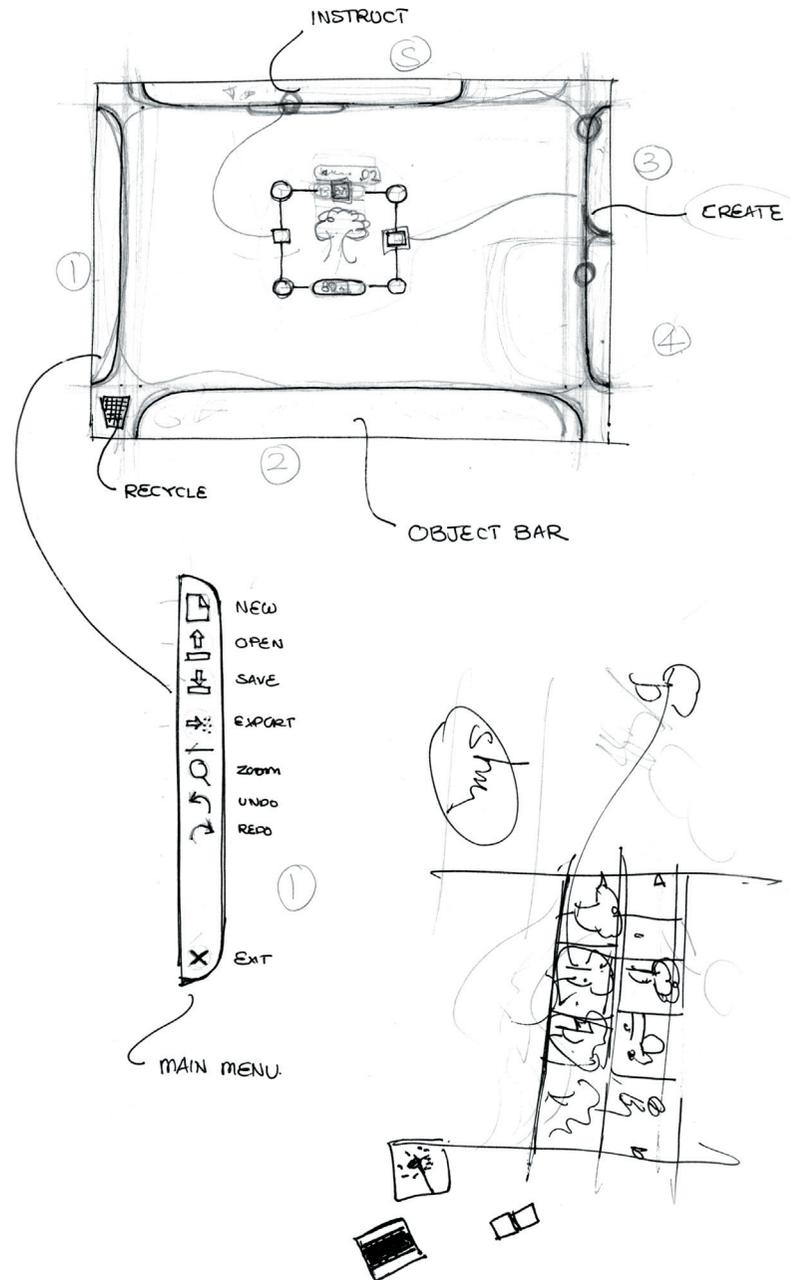
Learning by exploring

A medium to create strategies, challenges, problems, solutions

In brief,

**A medium to think and explore**

# Ideation



To actualize the dream of 'imagine...explore...&...learn' of invent I stated down what is that can provide such a medium. At this I have taken a unique perspective to dealing with question. In spite of answering what should be there I started with what they want.

From a child's perspective, Invent is something with which, One

- . Can create anything.
  - . Can decide how it will look, behave or act.
  - . Can relate it to world and other things.
  - . Can instruct or order.
  - .....
  - . Can create worlds, challenges or can tell stories.
- & thus,
- . Can explore one's imagination.

To do the same, to provide them that, I need to design a medium with which, One

- . Can create an object by drawing, coloring, selecting, ...
- . Can edit, duplicate ,modify,.. objects.
- . Can provide properties, behaviors to those objects.
- . Can relate one object to other objects of world
- . Can program(instruct) objects and events
- ...

& finally the brainstorming can help come up with three steps to 'invent'

1. Create
2. Animate
3. Instruct

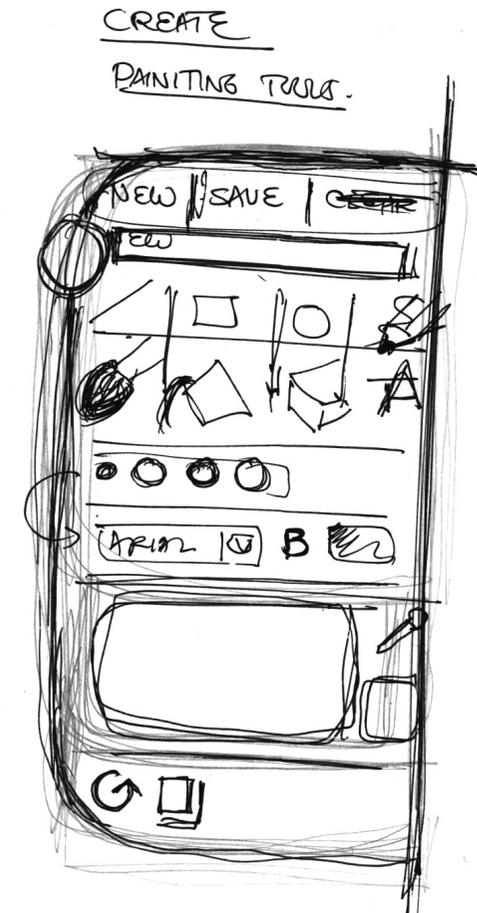
# Ideation

## Create

The idea is to give freedom to children to create whatever they want. A tree, a car, an aeroplane or even his dad or school teacher. From Pockemon to pencil the child should be able make it. One can create objects by drawing them. One can use colors. One can also be able move, rotate or scale them. By placing all this objects he should be able to create the worlds of his imagination. There will be no bound of color, look or any other for creating anything, giving any name to it. This I named as 'create'.

## Animate

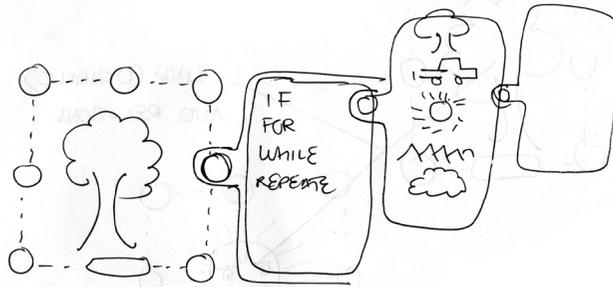
By 'animate' I mean giving life to the world the child has created. One can give life to the objects created in the world, can make them move, run, jump, fly, ....., can also turn into something else or can. Under animate one other important wonderful idea is to provide states to objects (swimming, flying, small, in school, .....) and with the use of 'create' to decide how these states will be. With 'animate' child will show object how to jump, how to run, how to hit and how to smile. Overall, animate is the magical stick with which child should be able to liven the world.



# Ideation

APPROACH - (B)

(2)



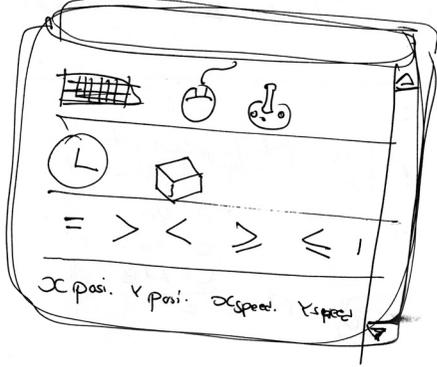
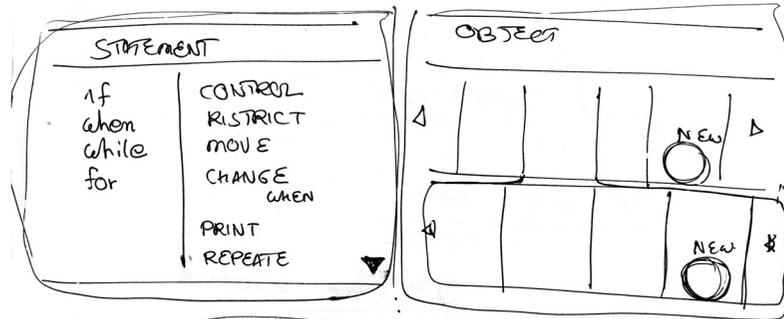
- ↓ - SPACE,
- ↓ - VISUALISATION LIMITS.
- ↓ - NOT POSSIBLE IN ALL CASES.

IF  IS ON ,

THEN 'S COLOR IS  GREEN

IF  IS IN ,

THEN MAKE 

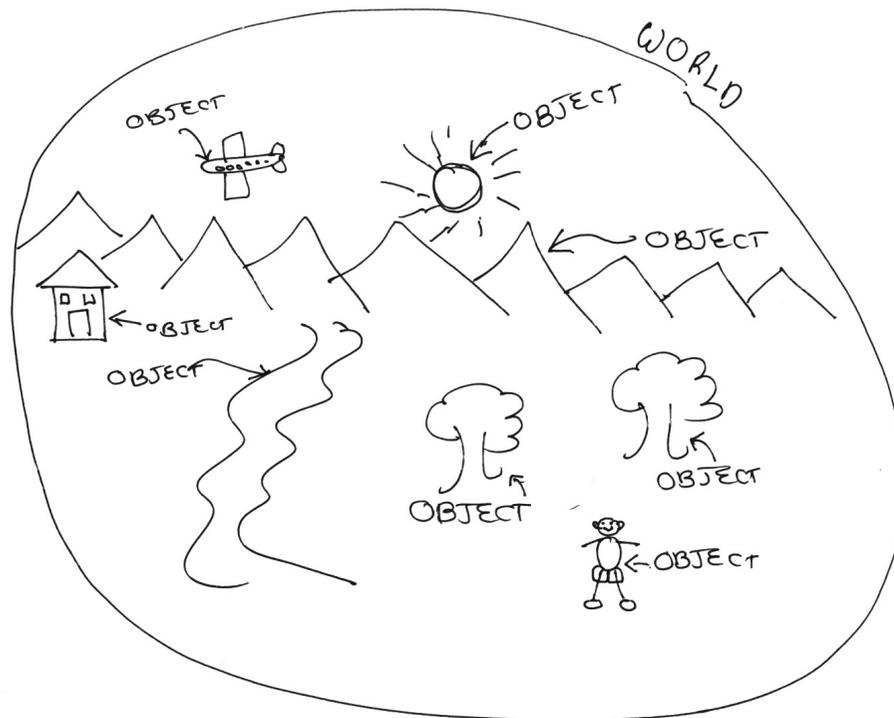


## Instruct

Now when he has worlds and live objects in his world a child need to instruct them what to do , when. When this happens .. Do this. If .... then .... Do this three times When clouds are there in the sky, do rain. When I say 'jump', you jump

The wonderful 'instruct' will give the child the freedom to instruct whatever they want to instruct. There are some wonderful school of thoughts like **Instruct by showing, Instruct by selecting objects and instruction, Instruct by changing properties and behaviors.** I tried to provide the child to instruct the way he like. In, instruct I came up a 'visual language' too for these orders and instructions to objects. Now, The child will control when the boy in his world will swim and how. The child will decide that whether he want to move car in which direction in his town and when he want to horn his car.

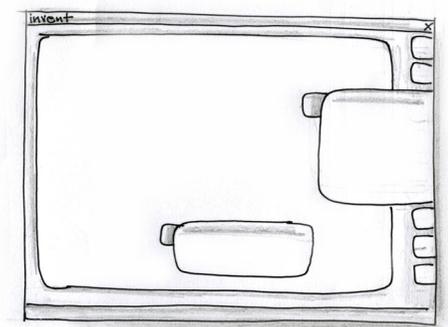
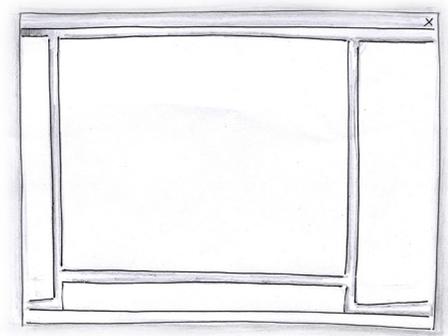
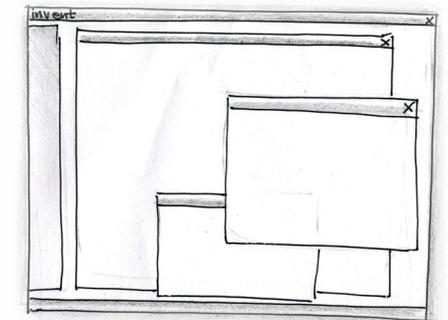
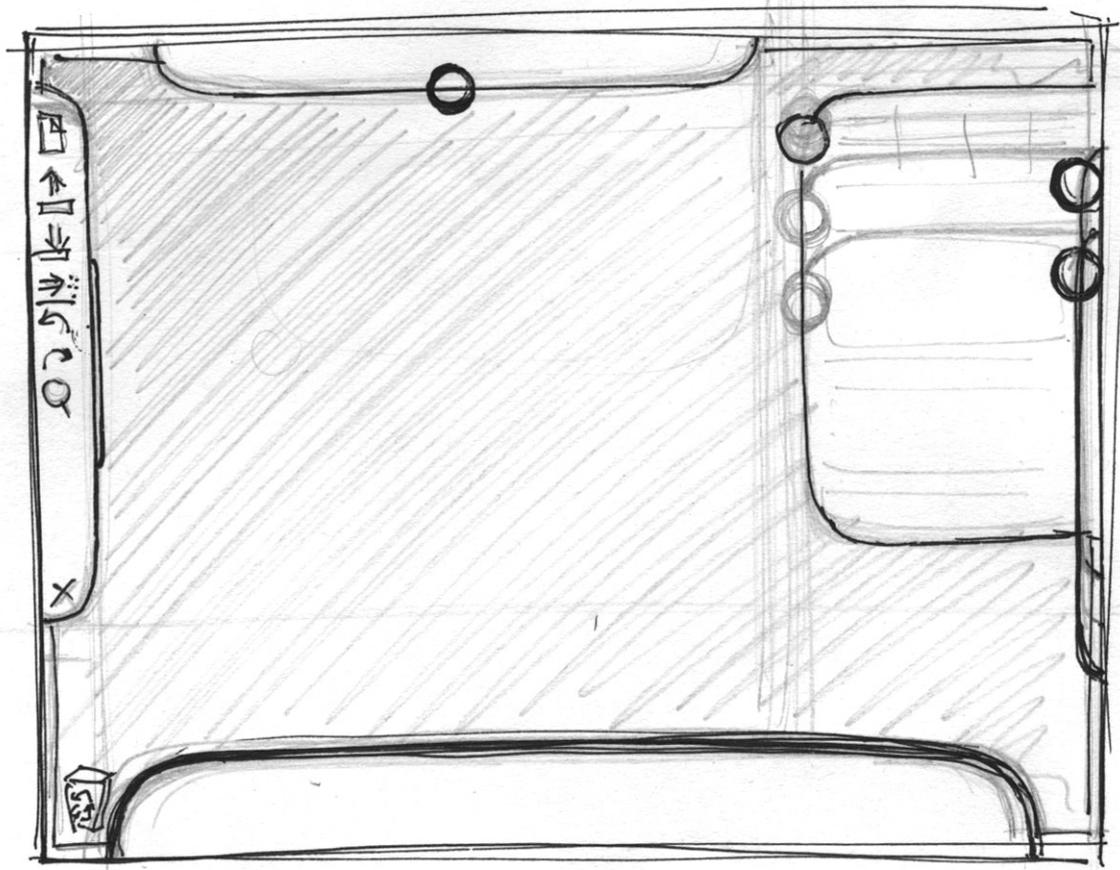
# Ideation



## 'Everything is an object'

One can create, liven and instruct any object in 'invent'. A tree, a car, a rocket, a school bell anything he imagines, he can create, liven and instruct what to do and how. Even he can have worlds of his imagination and can share or reuse those objects. The generic structure of 'invent' says 'Everything is an object'.

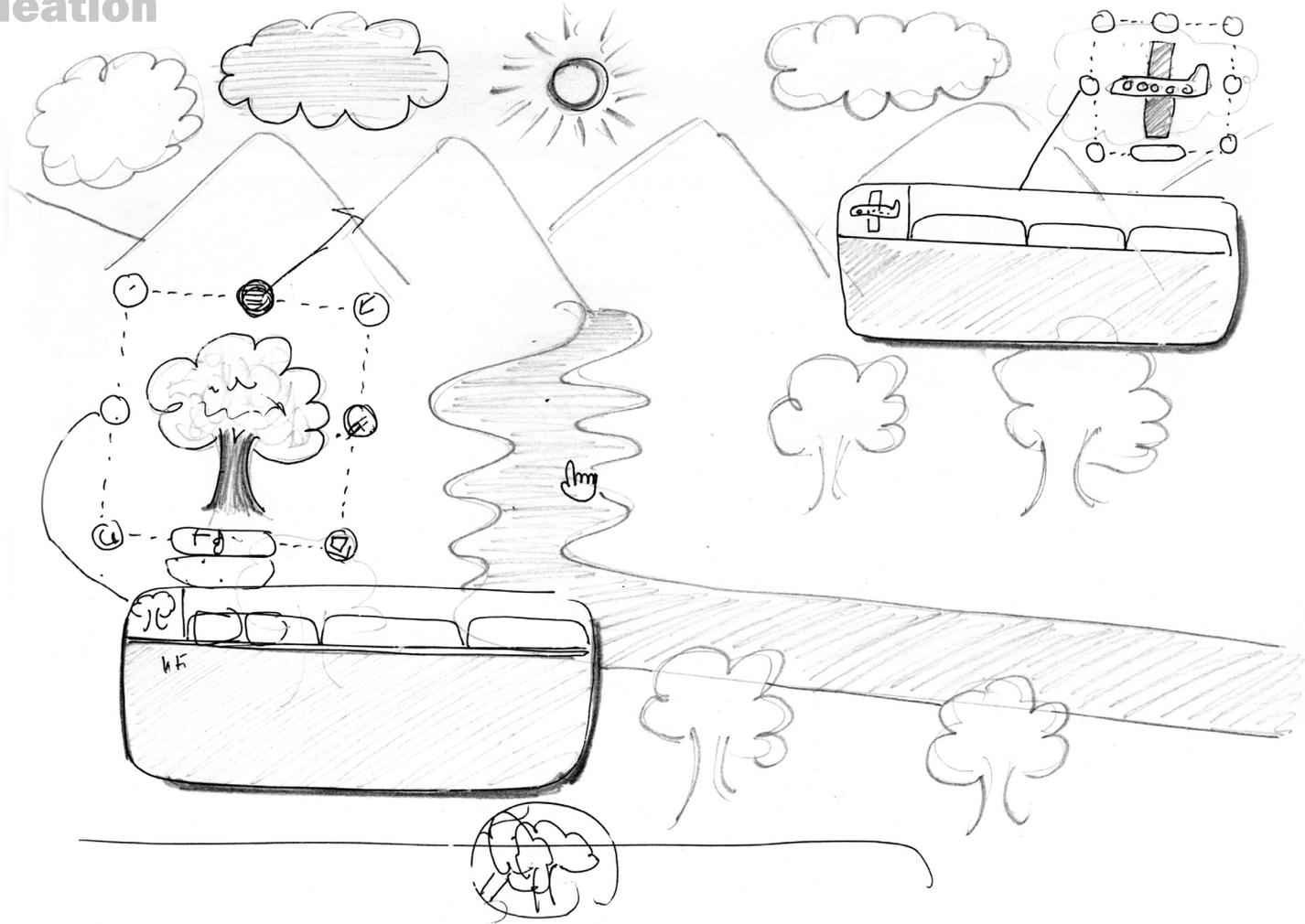
# Ideation



## A drawing paper

With the conceptual architecture and structural design ready in mind I started working on layouts and interaction design issues of 'invent'. In 'invent' I thought a paper as a metaphor for the interaction space of the child. A drawing paper, where the child will create his world of imagination. For providing him the medium to create, animate and instruct I came up with different designs regarding layouts, menus, interaction methods, ...

# Ideation



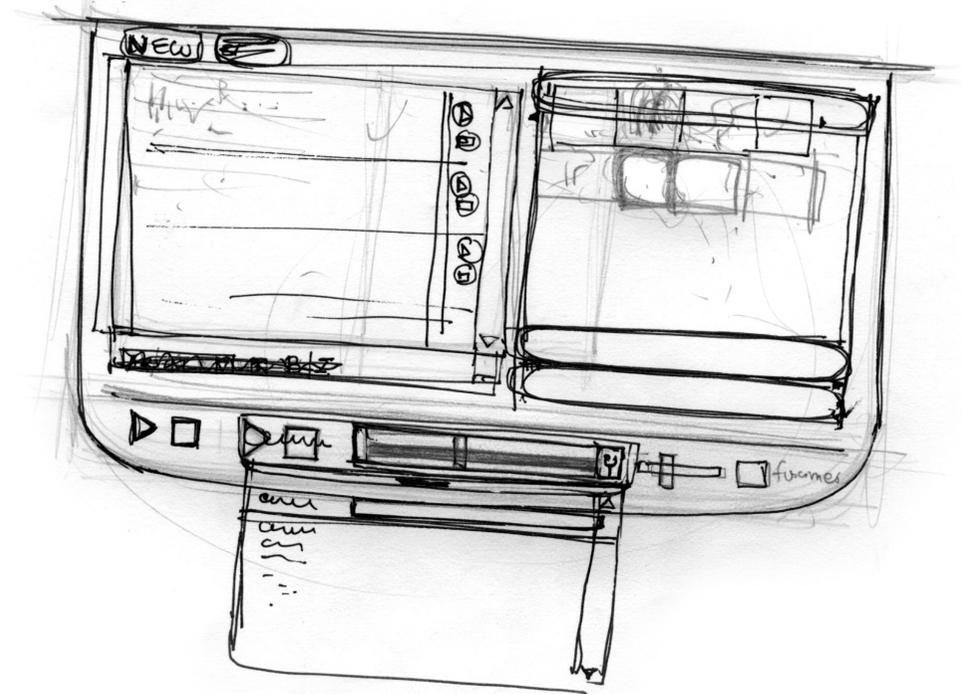
## Interaction design and system design concepts

In parallel, I was also brainstorming about how all this will happen. I studied and analyzed a lot of concepts about programming language design as well as interaction design for that. The major outcome of this effort was the internal system design of the programming language 'invent' as

well the detail interaction design of 'invent'. Some of the concepts used can be listed as

- . Prototype based O.O. programming
- . Intuitive interactions
- . A visual syntax language
- . Visual programming environment
- . Staying in the flow (creative unfolding of something)
- . Programming by demonstration
- ...

# Ideation



## Refinements

During the stage of ideation I also come up with some refinements (mainly enhancement) to my design. I added **sound** as a major feature to 'invent'. In the later version of concept prototype of invent I added a feature to have world **backgrounds** from any drawings, painting or photograph the child have in addition to create it by drawing. The one more enhancement was to have a **object library** where the child can save and reuse any object he has created. Even one can share objects with other children as well as can be able to select any image file as an object, can modify the object as he want.

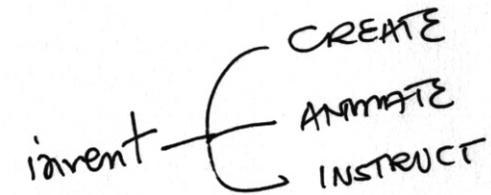
# 'invent'

## What is 'invent'?

Here is 'invent'.

'invent' is a medium to think and explore. 'invent' is an environment that allows children to explore their imagination. 'invent' is the design of a programming language for children.

'invent' enables children to create whatever they want to create or imagine. It lets them specify how those objects are to look, behave and interact by providing a special graphical interface to do all that. Not only that, 'invent' also provides a way to instruct those objects what to do, when and how. 'invent' is a design of a children's computer program that allows them to create their own worlds in which they can explore their imaginations and by the way they learn concepts. Thus, it provides a medium for learning by doing, learning by exploring.



As the Italian poet *Cesare Pavese* says: '**To know the world, one must construct it**'. And 'invent' is the medium to do it for children to know about the world.

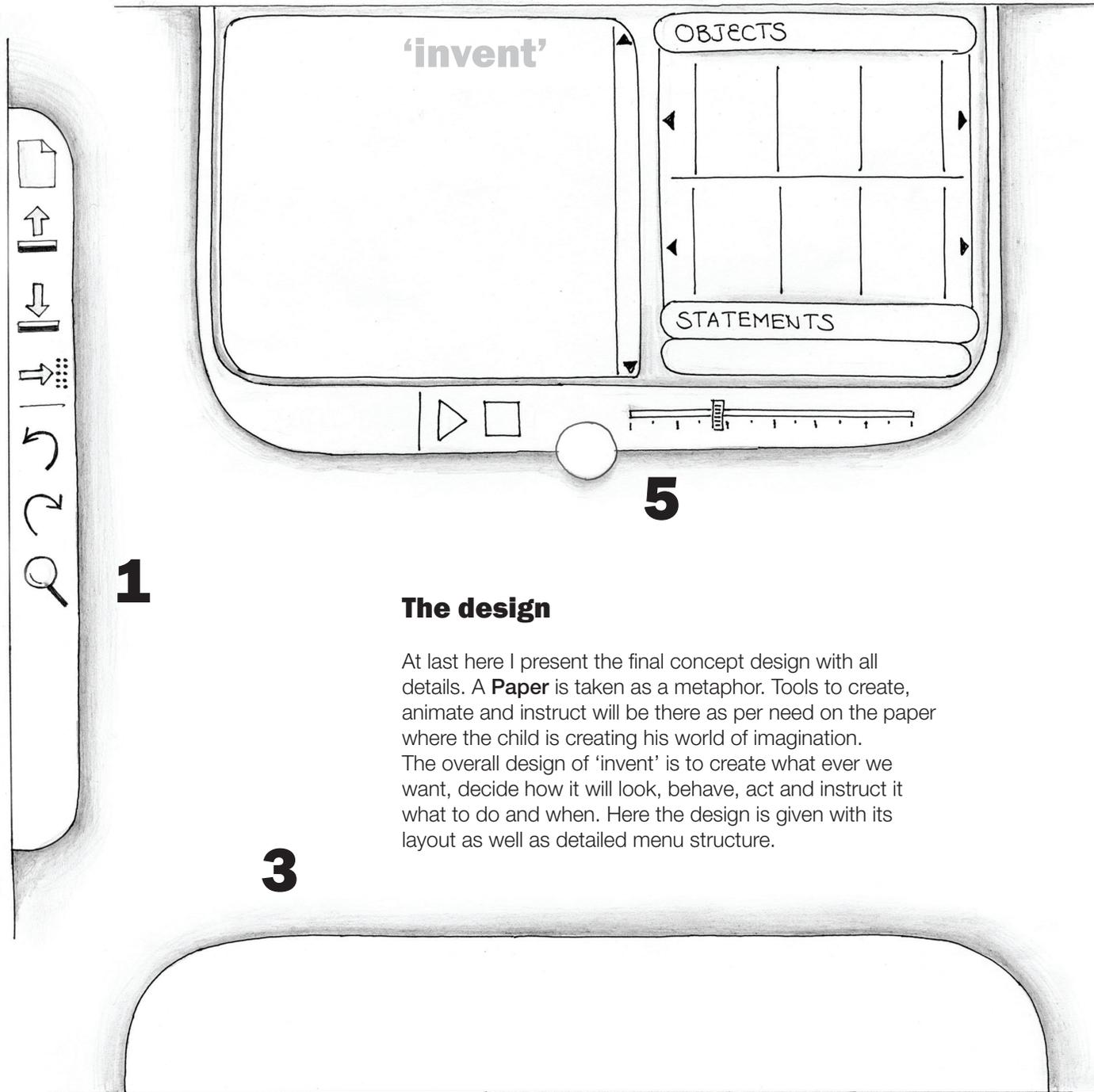
In brief,

**Invent = Create + Animate + Instruct**

**Create:** Create objects. Create world.

**Animate:** Give life to the objects and thus liven the world.

**Instruct:** Instruct/program objects what to do. (when, where, how, ...)



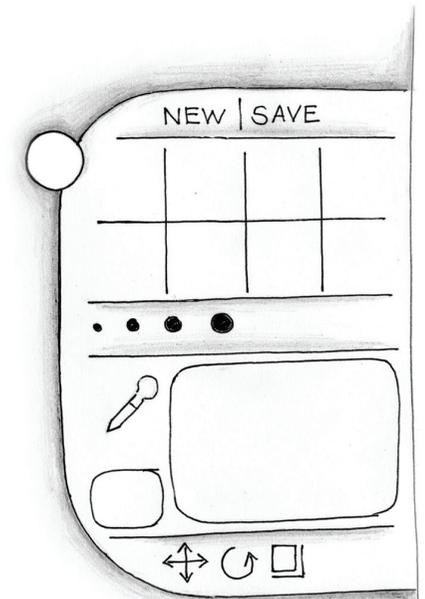
1

### The design

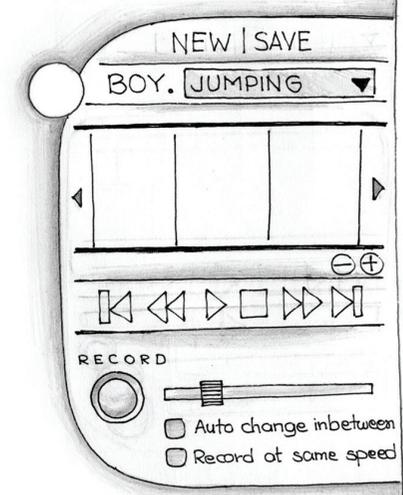
At last here I present the final concept design with all details. A **Paper** is taken as a metaphor. Tools to create, animate and instruct will be there as per need on the paper where the child is creating his world of imagination. The overall design of 'invent' is to create what ever we want, decide how it will look, behave, act and instruct it what to do and when. Here the design is given with its layout as well as detailed menu structure.

3

5

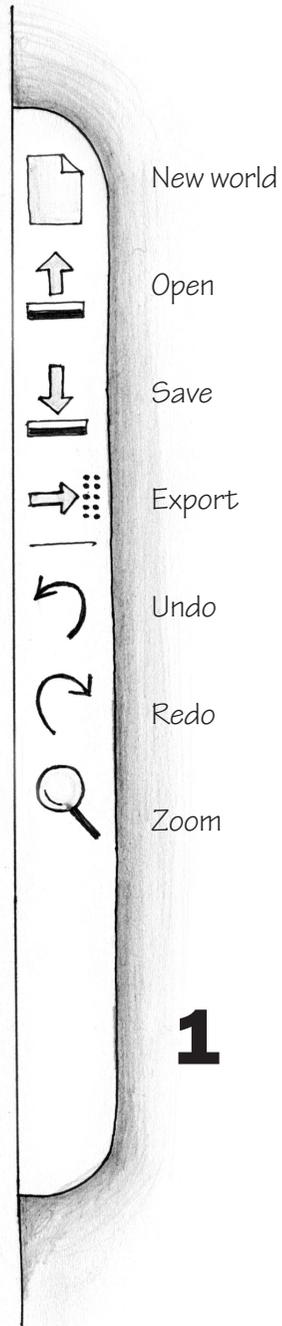


2



4

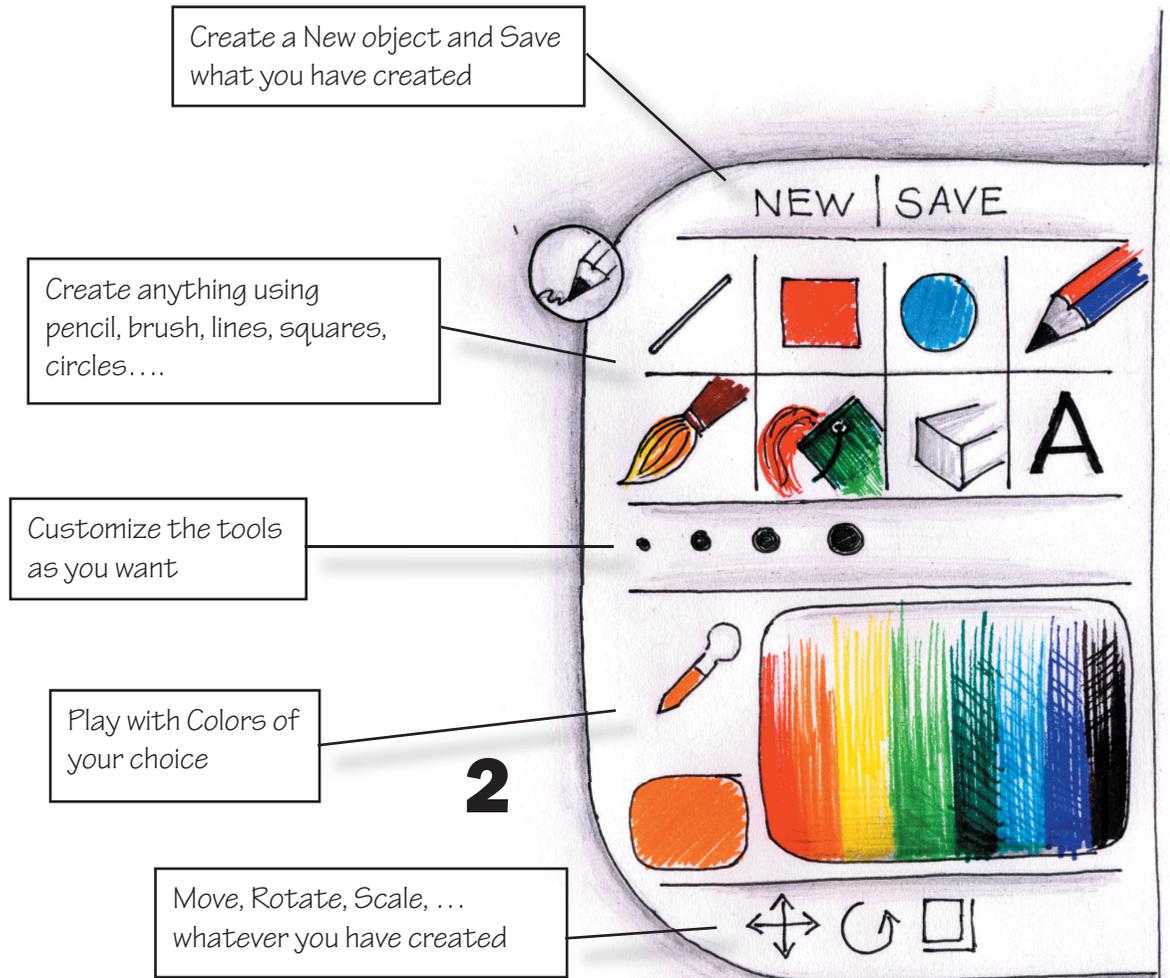
# 'invent'



## Let's have a new world

One can create a new world. One can open, save and export the world. Some basic operations like undo, redo as well as zooming are also provided in the menu.

# 'invent'



## Create

Create menu lets the child draw anything with tools like pencil, brush, line, square, and circle. The child can play with colors as well can use move, rotate and scale tools in creating tree, car, book, boy, river ..... anything. The customize space below the basic tools also let one customize tools. Create is a context menu. It will be there somewhat overlapping the world area when in use. When not in use it can be minimized.

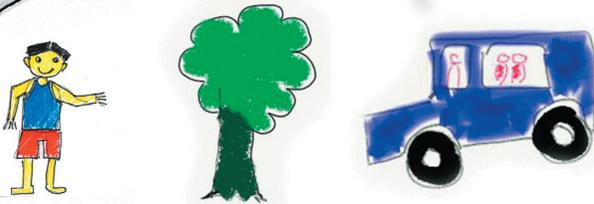
# 'invent'

## Objects

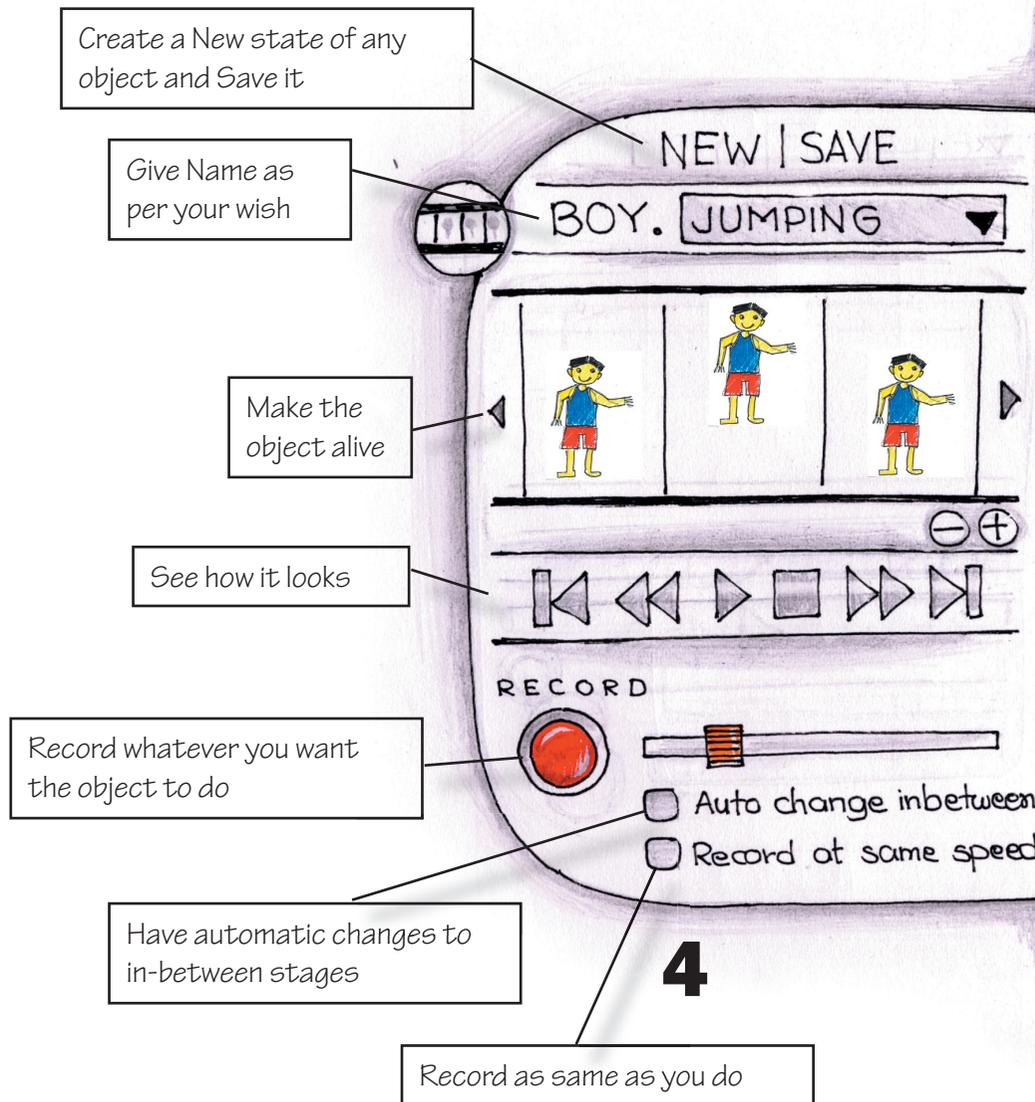
**Everything is an object.** Object bar at the bottom provides one with objects he created. The tree, car, boy and river all are there to use and place to the world. One can have any number of copies of these objects. Later one can also edit specific instance of the object and save in object bar as a separate object. One can remove the object just by dragging them to trash bin at the left of the object bar. One can minimize the object bar as per wish.

Save and reuse any object you have created. create your world with the objects

3



# 'invent'



## Animate

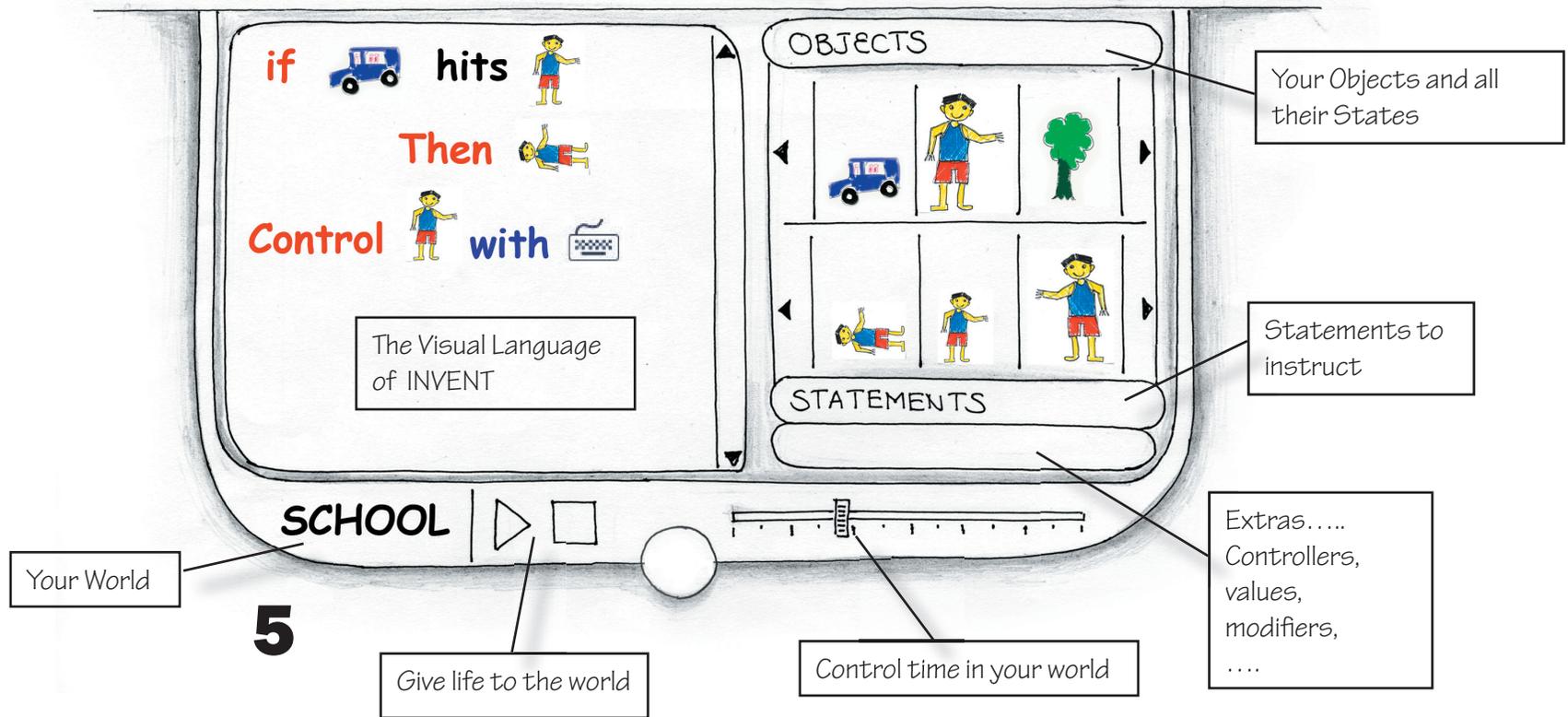
'invent' is based on several concepts like '**Programming by Demonstration**' and '**Direct Manipulation techniques**'. (Cypher 1993), These techniques allow users to create behavioral rules by demonstrating what the object is to do in a specific situation. 'Animate' provides one with the environment to liven the world one has created.

There are basically 3 methods 'animate' provides. One can give different states to the object created. Like, one can give a boy(an object) different looks at different conditions. 'In school', 'in red t-shirt', 'small', 'happy', 'fallen down' and so on. By the other two method one can even have these states enlivened. Like the boy can be jumping, or nodding his hand or can running. One will be able to use these states of objects as per requirements when he is instructing them.

The two core methods one can do all these are '**frame method**' and '**record method**'. In frame method one creates frames(looks) of the object by changing color, size, orientation, or by other drawing tools. Sequencing these frames and playing them will give the desired state.

In record method one will be able to record what ever he want the object to do later. One will do it and tell object that, see, what I teach you is called jumping. Object will be able to do the same later when needed by playing that state. Thus, animate liven the world one has created by enliven the objects.

# 'invent'



## Instruct

The most important of all in invent is 'instruct'. When now one have all his objects, a world created with these objects ready, even also he has liven the world by methods in 'animate', the most important thing is to say those objects what to do and when. 'Instruct' is that.

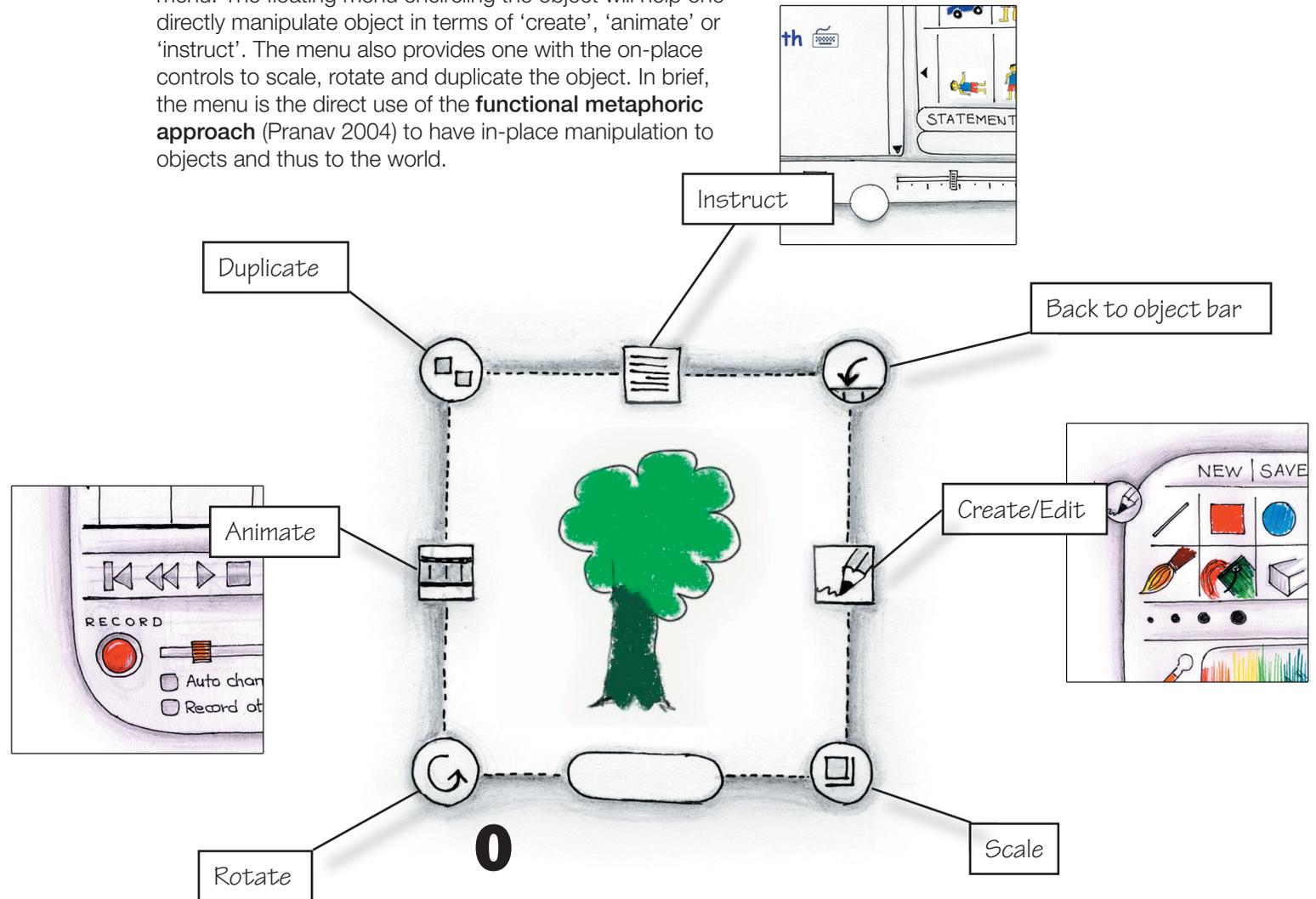
Instruct has a **visual syntax language**. One can instruct any object anything to do by just selecting and dragging objects, their states programming language structures and

handlers or supporting elements. One just needs to drag and drop controllers to have instruction to the objects. By very intuitive interaction methods, 'instruct' provides one with a great control to one's world. In 'instruct' one can very easily decide that he wanted the car to be controlled by the arrow keys on his keyboard. He will also instruct the car not to go out of road he has created. One will blow horn or say the boy that when car hits you, you fall down; He will instruct the clouds when to rain or the sun when to come. It seems somewhat like a dream. Yes, but this is 'invent' is all about. It will let one explore one's imagination.

# 'invent'

## 'The' menu

One more wonderful feature of 'invent' is the object context menu. The floating menu encircling the object will help one directly manipulate object in terms of 'create', 'animate' or 'instruct'. The menu also provides one with the on-place controls to scale, rotate and duplicate the object. In brief, the menu is the direct use of the **functional metaphoric approach** (Pranav 2004) to have in-place manipulation to objects and thus to the world.

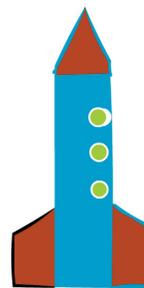
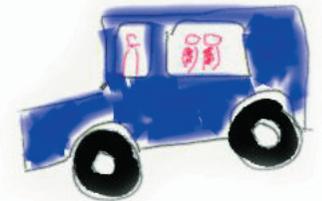
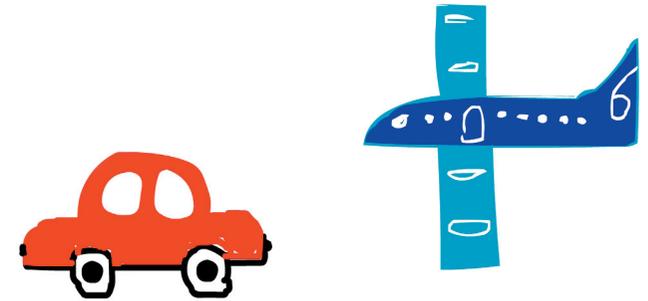


# 'invent'

## In and out of 'invent'

Think 'invent' as a drawing paper for a child. A child can draw on that paper. Not only that, he can rearrange those objects to create a world, and as well can enliven those objects and the world by 'animate'. 'invent' also let a child to instruct the objects of his world what to do and when. The Poet Cesare Pavese has said 'To know the world one must construct it.' This is my dream to provide children with such a medium where they learn themselves by exploring their ideas and understanding about the world. 'invent' is an effort to have a medium for children to explore their imagination.

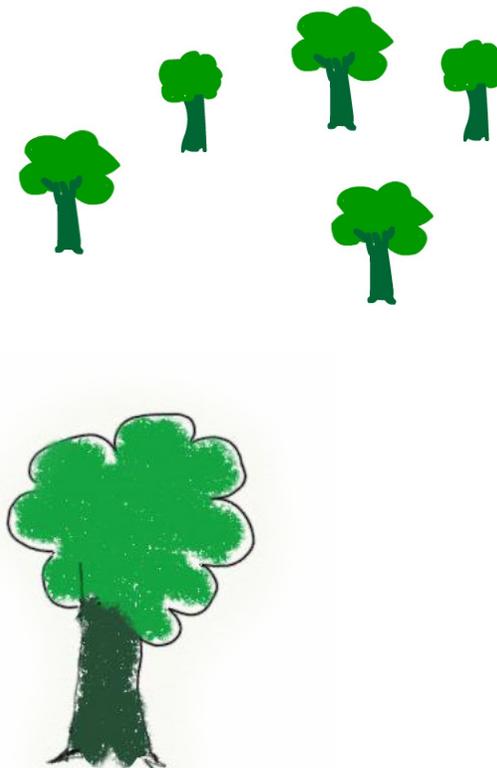
Some of the features of invent which make it wonderful is mentioned here briefly.



### a. Everything is an object

'invent' propose a child to create any object by method of drawing it, same as he draws with pencil and colors on a paper. A child can draw a tree, a car, Pokemon, mountains, clouds, ... Everything, one draws is treated as an object in 'invent'. With the tools like pencil, brush and geometric shapes like line, circle, square a child can draw any desired object. He can color his objects with color tools. Besides these tools 'invent' provides move, rotate and scale tool. A child can move, rotate or scale the whole object as well parts of it with them.

# 'invent'



## b. Every object is unique

The only thing common to objects in 'invent' is 'every object is unique'. At architectural level any modified copy of an object is treated also as a unique object. For example if an instance of a tree is made, that instance will act as a separate object now and any modification to that tree will be to that tree only.

Now, if a child wants to have that object also in the object bar to make more such trees than he can put it there to the object bar as well. The other interesting feature 'invent' provides is the controlled inheritance. When one wants to have some changes to be made to all the copies of an object he can do it by modifying the object copy at the object bar. For example if after having all his trees in place in a world, if a child want to add fruits on all tree now, he doesn't need to do it individually for each tree. He can select the tree at the object bar and draw mangoes on his tree. All the trees in his world will have those mangoes. Now if he doesn't want those mangoes in one of the trees then he can select the particular tree and take the mangoes off also.

# 'invent'

## c. Three steps 'to invent'

Create, animate and instruct. Invent provides three major operations.

In simple words,

### **Create:**

Create objects. Create world.

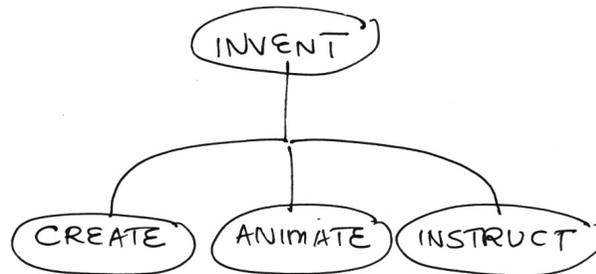
### **Animate:**

Give life to the objects and thus enliven the world.

### **Instruct:**

Instruct/program objects what to do. (when, where, how, ....)

The interesting feature of 'invent' having these three(create, animate, instruct) is that they stand in themselves as a separate tool, too.



### 1. A drawing pad

In 'invent' the tool 'create' has all the features to be used as a drawing application. A child can draw, color and even rearrange his created objects in a drawing.

### 2. A simulation tool

With 'animate' a child can enliven the world he has created. A child can select any object he has created with the 'create' or he can use the objects he has saved in his object library and animate them to enliven it. He can make a boy jump, swim, run, smile and can also make a train run on a track or make it whistle.

### 3. A programming environment

Now if the child wants his objects to be instructed like 'what to do when and how', 'instruct' helps him in doing the same. He can instruct the boy when to smile or can control the jumping boy with his joystick or keyboard. He can instruct the train when to stop and whistle. By 'instruct' a child can instruct, relate or direct the objects of his world he has created with create and animate.

Thus, invent provides a level of fullness as a tool at all three stages. This helps a child even when he is learning or becoming familiar to 'invent' as a medium.

# ‘invent’

## **d. Object library**

Created objects along with their states can be saved for the future use. A child can reuse these saved objects in any new world he creates. A child can use the tree, he has made in some other world created with invent some days back and saved in object library, in the new world he is creating.

## **e. ‘Do it’**

A one more wonderful feature of ‘invent’ is its ‘Do it’ concept. From animating an object to instructing it invent helps a child to do things very intuitively. Interactions in ‘invent’ are of ‘do it’ nature. For example if a child wants a ball to jump he can show the ball how to jump by making it jump the way he likes and can thus teach a ball ‘how to jump’. Later he can ask the ball to jump as same as he has shown him. Not only this, using the create, animate and instruct and their features in ‘invent’ is also very intuitive at all level from putting an object to trash bin to swap the control of a car from keyboard to joystick in visual syntax at ‘instruct’.

Other than these ‘invent’ provides an on place manipulation handlers for objects of the world. A child can rotate, scale, copy or put the object back to the object bar with this on-place menu, which appears surrounding the object when double-clicked. Other than these handlers the menu also lets the child ‘create’, ‘animate’ or ‘instruct’ the object by selecting the related operation, on place itself.

# 'invent'

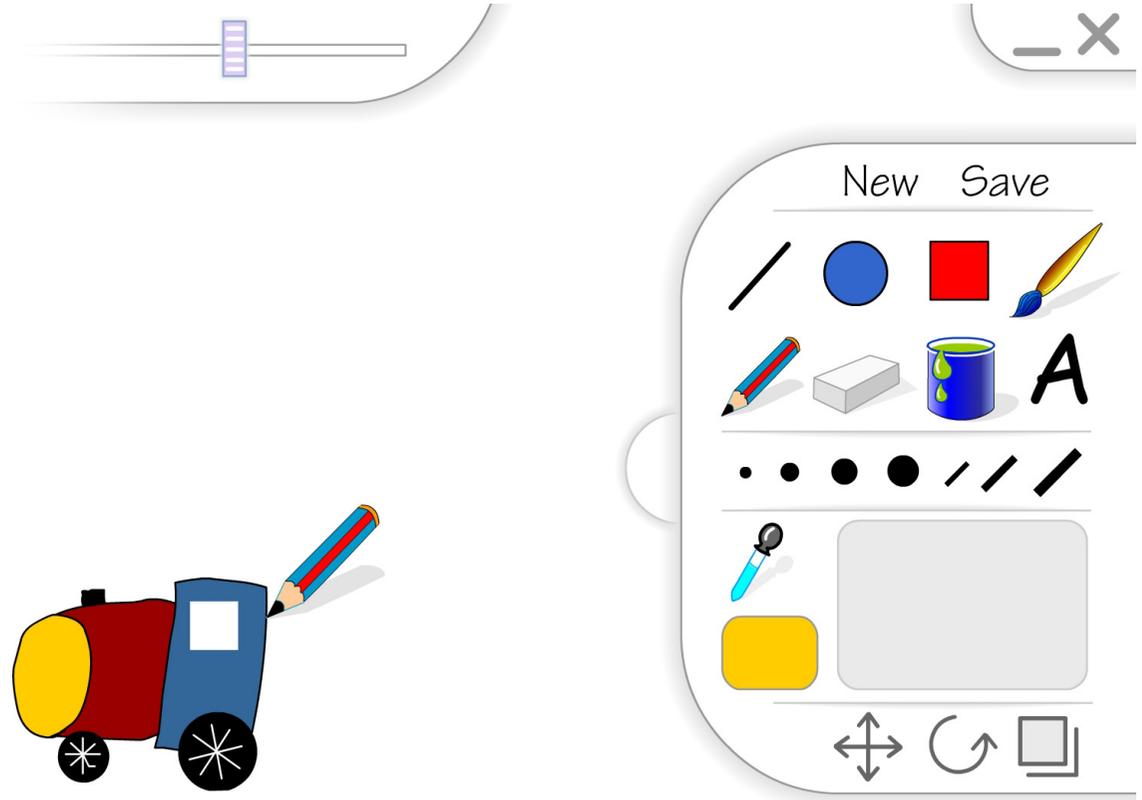


## 'invent' and the water carrying train of Malgudi

Here I would love to refer back to the story '**The school train**' to explain how 'invent' can help children to explore their ideas and imaginations. *Swami, Rajam* and friends have a thought of having a train to water those trees at the back of *Albert Mission School of Malgudi*. Yes, a train which will carry water from the tap near the railway station to the school. 'invent' is there with them to help explore their this thought.

*Rajam* has a vague thought to have a train like system to help solve the water problem. He observed the lying down old train track pieces here around. At the backyard of his home he has two lorrys to carry something. He has an idea but not so clear. *Swami* and other friends don't have a single clue. At *Rajam's* home at that evening they could do it all with invent. They could come up with the plan for the wonderful, "the school train of *Malgudi*". Not only could they explore their ideas they could understand and tackle with the problems to do the same. At last they made a game also to water the school trees with their school train.

'invent'



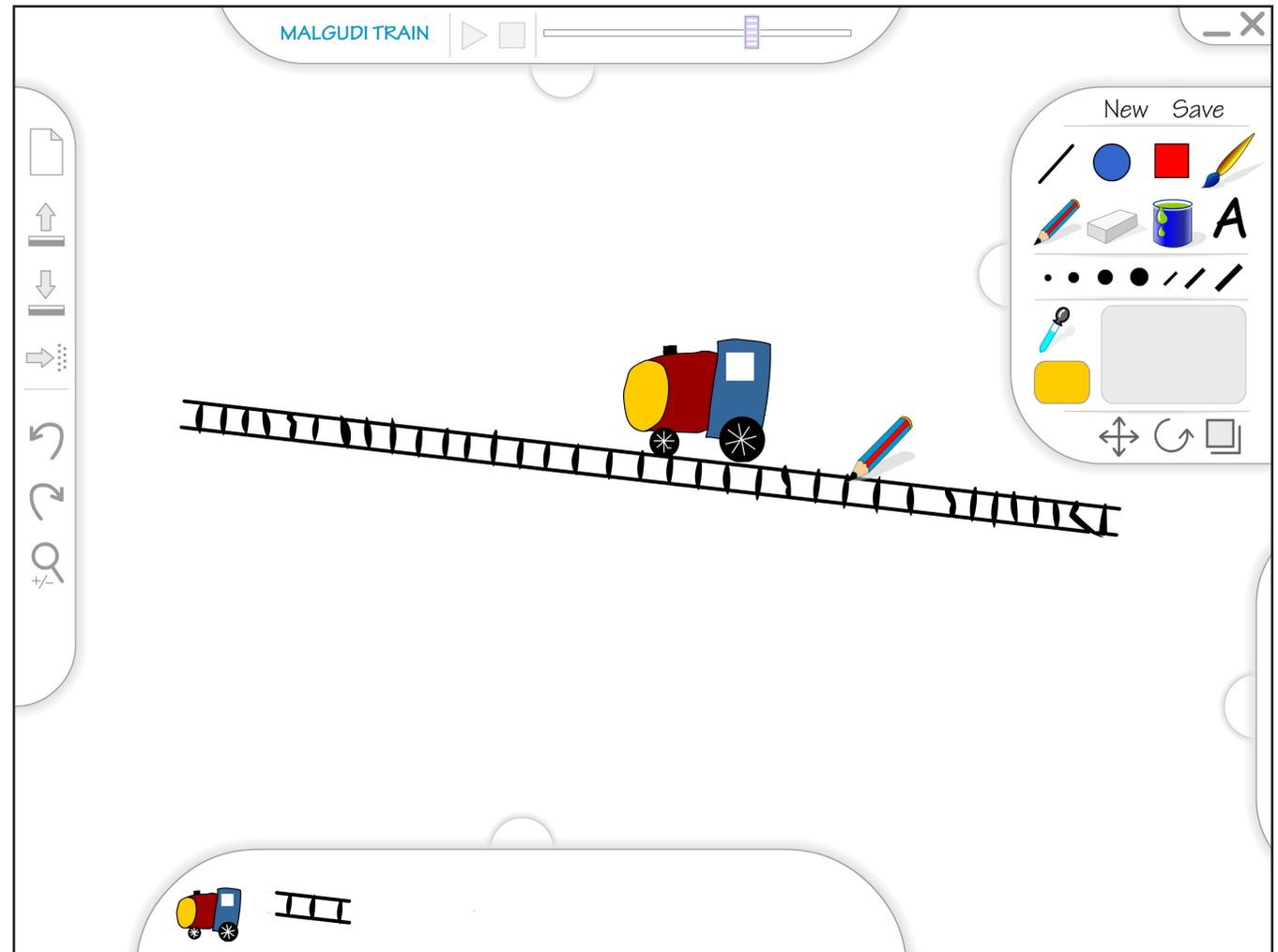
To help understand better what is 'invent' and how a child can use it, here are some examples explaining some basic interactions.

I think we can't imagine that 'what a child can imagine'. But, to understand 'invent' lets assume that a child imagines having a running train and ...

### **A running train**

A child wants to have a train. He draws it with pencil and colors and other tools in 'create'. He also wants to have that train running. By pressing record in 'animate' and dragging the train from one place to the other he makes his train running.

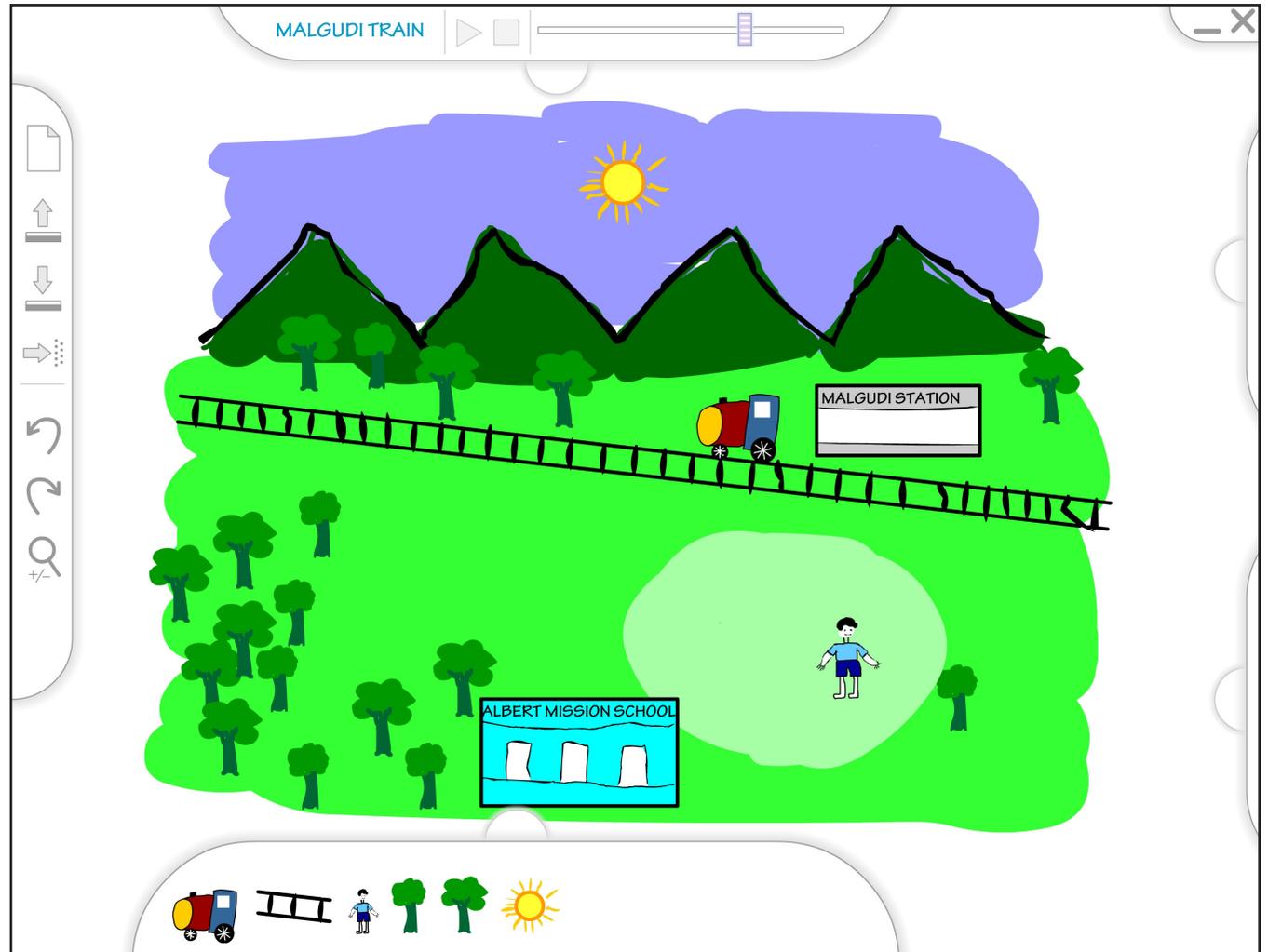
# 'invent'



## Train is on the track

Now he wants his train on track. What he does he draws track and puts the train on the track. Simple. No, but he doesn't want his train to go out of that track. He selects the train and with 'select object as a path' tool of 'animate', he very easily does the same.

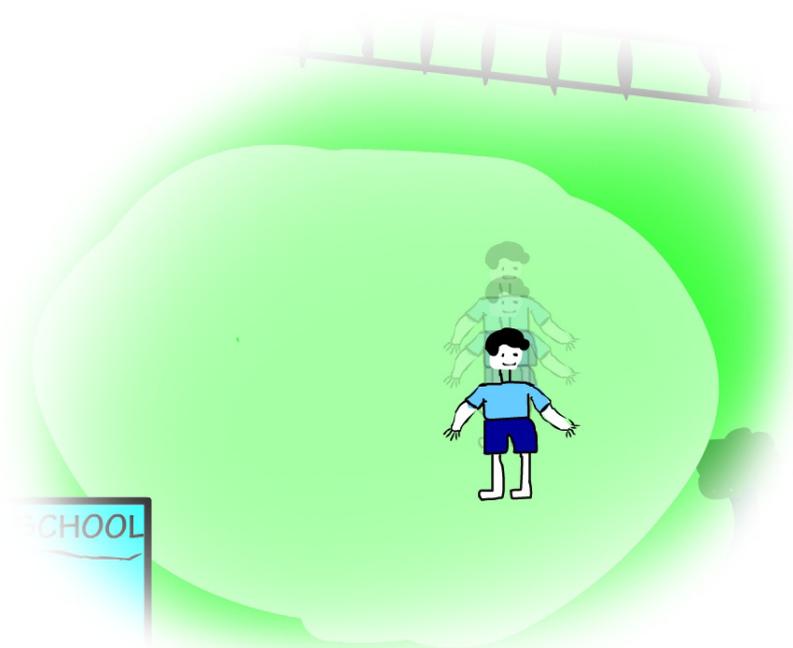
# 'invent'



**There are trees, birds, mountain, houses and me**

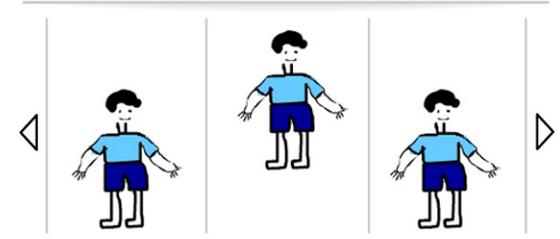
Wow, the train is running on the track, but it is looking all empty. He creates mountains, sun, trees, school, railway station, play ground and finally himself. He copies trees to have so many of them.

'invent'



New Save

BOY (JUMPING)



Record



Auto change inbetweens  
Record at same speed

### I am jumping

Now he wants to jump, too. With the 'animate' framing method he makes 2 new frames and in the second frame he makes himself somewhat up. He plays that state of jump. Hey, he is jumping too now. He also makes the sun shine, river flow and the engine of the train smoking.

# 'invent'

CONTROL  with 

Objects

Statements

Extras

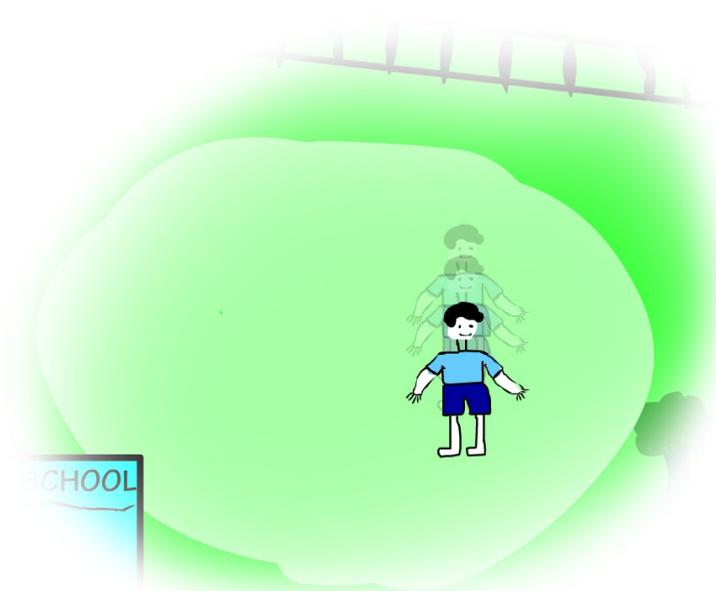
 

> | < | = | >= | <= | on | in

X pos. | Y pos. | Xspe. | Y spe

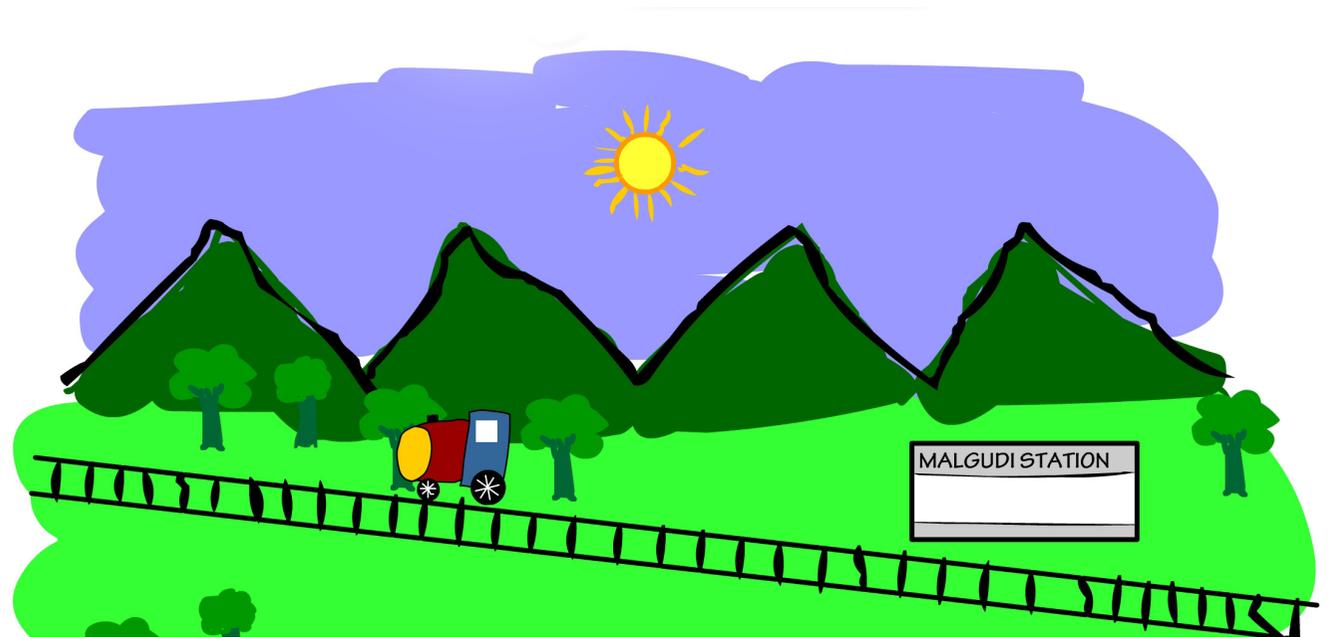
MALGUDI TRAIN   



## I jump when I want

No, I don't want to jump forever. I want to jump when I want. Ok, he opens 'instruct' and with 'CONTROL' he selects to control jumping by mouse click. Yes, now he jumps whenever he clicks mouse. Ha, ha.

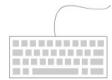
# 'invent'



## **Train stops when I say to 'stop'**

Same way he also want the train to stop whenever he say 'stop'. By methods in 'instruct' he does that also very easily. When he presses 'S' on the keyboard the train stops. 'G' says the train to GO.

# 'invent'

CONTROL  with 

RESTRICT  in 

Objects

Statements

Extras

> | < | = | >= | <= | on | in

X pos. | Y pos. | Xspe. | Yspe

MALGUDI TRAIN   

MISSION SCHOOL

## I run on the ground

I also want to run on the playground. He does that also and selects the arrow keys on keyboard to move around on the play ground. He restricts himself to the ground, too.

# 'invent'



## This is the 'water carrying train of Malgudi'

Hey no, this is the water carrying train of Malgudi. The train brings water from hand-pump near the railway station to Albert Mission School of Malgudi and provides water to the trees at the back of the school.

Hurrah. Let me whistle the train, too.

## 'invent'

This is what my work and 'invent' is all about. There are a lot more efforts in terms of understanding the behavior of a child and to understand how they think and learn. A lot more explorations have been done to come up with 'what they want' and once decided that, to solve 'how to do it?' I am interested in helping children learn to think better and deeper. I made the medium to serve as a new kind of electronic paper that can hold new ways to represent powerful ideas. In Alan Kay's words 'Readers can also become writers'.

**Let's 'imagine...explore...&...learn'.  
Let's invent.**

*'Design is not just what it looks like and feels like. Design is how it works.'*

*- Steve Jobs*

# References

- Ackerman, Edith. Piaget's Constructivism, Papert's Constructionism, What's the Difference? [online]. Available from: <http://learnng.media.mit.edu/publications.html> [Accessed 26th February 2005]
- Bergin, Thomas J. and Richard G. Gibson, eds. History of Programming Languages-II. New York: ACM Press, 1996
- Brooks, Jacqueline G. and Brooks, Martin G. In Search of Understanding: The Case for Constructivist Classrooms. Alexandria, VA: Association for Supervision and Curriculum Development, 1993
- Bruner, J. Acts of Meaning. Cambridge, MA: Harvard University Press. 1990
- Clement, D. and Gullo, D. Effects of Computer Programming on Young Children's Cognition, Journal of Educational Psychology (vol. 76, no. 6). 1984
- Cypher, Allen. Watch What I Do: Programming by Demonstration. The MIT Press. 1993
- Cypher, A. and Smith, D. KidSim: End user programming of simulations. In Proceedings of CHI'95 (Denver, Colo., May 7-11). ACM Press, New York, 1995, pp. 27-34.
- David Canfield Smith , Allen Cypher , Larry Tesler, Programming by example: novice programming comes of age, Communications of the ACM, v.43 n.3, p.75-81, March 2000
- Ferguson, Andrew. **The History of Computer Programming Languages** [online]. Available from: [http://www.princeton.edu/~ferguson/adw/programming\\_languages.shtml](http://www.princeton.edu/~ferguson/adw/programming_languages.shtml). [accessed on 21st February, 2005]
- J.F. Pane, "A Programming System for Children that is Designed for Usability," Ph.D. Thesis, Carnegie Mellon University, Computer Science Department, CMU-CS-02-127, Pittsburgh, PA, May 3, 2002.
- Kay, Alan. Computers, Networks and Education [online]. Scientific American Magazine. September 1991. Available from: <http://minnow.cc.gatech.edu/learn/9> [Accessed 4th January 2005]
- Kay, Alan. The Power of the context. Remarks upon being awarded - with Bob Taylor, Butler Lampson and Chuck Thacker - the Charles Stark Draper Prize of the National Academy of Engineering, 2004
- Kay, Alan. Background on how children learn [online]. Scientific American Magazine. Available from: [http://www.squeakland.org/school/HTML/essays/how\\_child\\_learn.html](http://www.squeakland.org/school/HTML/essays/how_child_learn.html) [Accessed 20th January 2005]
- Lammers, Susan. Programmers at work: Interviews with 19 programmers who shaped the computer industry. Tempus Books. 1989

## References

LEGO serious play [online]

Available from: <http://www.seriousplay.com> [accessed February 01, 2005]

McNeil Jr., Donald G. The Last Time You Used Algebra Was ...

New York Times, Late Edition - Final , Section 4 , Page 3 , Column 1. December 12, 2004. Available from: [http://www.unm.edu/~pre/law/articles\\_advise/algebra.htm](http://www.unm.edu/~pre/law/articles_advise/algebra.htm) [Accessed 21st March 2005]

Mistry, Pranav and Agrawal, Gajendra. Functinal metaphoric approach to be in the FLOW with software interfaces. In: IHCI2004, The First All Indian Conference on Human-Computer Interaction, Bangalore. December 2004

Noble, J., Taivalsaari, A. and Moore, I. Prototype-Based Programming: Concepts, Languages and Applications. Springer-Verlag Berlin and Heidelberg GmbH & Co. K. 1999

Papert, Seymour. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books. New York. 1980

Papert, Seymour. Child Power: Keys to the New Learning of the Digital Century [Lecture] The eleventh Colin Cherry Memorial Lecture on Communication, at the Imperial College in London. June 2, 1998

Papert, Seymour, Papert on Piaget [online].

Available from: <http://www.papert.org/articles/Papertonpiaget.html> [accessed February 07, 2005]

Papert, Seymour. Ghost in the Machine: Seymour Papert on How Computers Fundamentally Change the Way Kids Learn. Interview by Dan Schwartz. <http://www.papert.org/articles/GhostInTheMachine.html> [Accessed February 01, 2005].

Piaget, J. & Inhelder, B. The Psychology of the Child. NY: Basic Books. 1969

Piaget, J. To Understand Is To Invent. New York: The Viking Press, Inc. 1972

Rader, C., Brand, C., and Lewis, C. Degrees of comprehension: Children's mental models of a visual programming environment. In Proceedings of CHI'97 (Atlanta, Ga.). ACM Press, 1997, pp. 351-358.

Read, Jeff. SQUEAK if You Love GUI [online]. World Tech Tribune. Available from: [http://www.squeakland.org/school/HTML/essays/love\\_gui.html](http://www.squeakland.org/school/HTML/essays/love_gui.html). 2002. [Accessed 23rd March 2005]

Rogers, C.R. & Freiberg, H.J. Freedom to Learn (3rd Ed). Columbus, OH: Merrill/Macmillan. 1994

Sen, Ajanta and Poovaiah, Ravi. Into the world of the "really not real". Leveraging a child's make-belief abilities for design clues to build a cross-cultural collaborative environment on the Internet.

## References

Smith, Randall B. and Ungar, D. Programming as an Experience: The Inspiration for Self. ECOOP '95 Conference Proceedings, Aarhus, Denmark, August, 1995

Squeakland [online]

Available from: <http://www.squeakland.org> [accessed February 12, 2005]

Thanasoulas, Dimitrios. Constructivist Learning [online]. Karen's Linguistics Issues, November 2002. Available from: <http://www3.telus.net/linguisticsissues/constructivist.html>. [Accessed on 30th February, 2005]

Vygotsky, L.S. Thought and Language. Cambridge, MA: MIT Press. 1962

Weir, Sylvia. Cultivating Minds: A Logo Casebook. New York: Harper & Row, 1987

Wood, D. How children think and learn: Understanding children's worlds. Cambridge, MA: Basil Blackwell. 1988

*'The best way to predict the future is to **invent** it.'*  
- Alan C. Kay



# **invent**

Imagine...Explore...&...Learn

By  
Pranav Mistry

Guide  
Prof. Ravi Poovaiah