A naïve lesson on Recursion

Stefano Penge, may 2022

Réné Magritte, Clairvoyance, 1936

Why

- Why one should teach recursive algorithms to students?
- Since "to explain" means "to put in relation something new with something known", which kind of previous experience should one exploit?
- Do normal people already know something about recursion?

Recursion in everyday life

- Romanesco broccoli
- Sourdough
- Language ("Mary thinks that John thinks that..")
- Pictures (Droste effects)
- Humour (well, programmers' humor: recursive acronyms like GNU)
- Reproduction

https://en.wikipedia.org/wiki/Recursion

Recursive algorithms

- The classical explanation of recursion and iteration is "Trading elegance for efficiency".
- It starts with the informal idea of something defined in terms of self
- This should be impossible or lead to an infinite loop; but there are some conditions under which the loop halts.
- The explanation continues with some examples taken from mathematics: Fibonacci numbers, factorials, … these are all concepts defined in terms of themselves
- For example, the definition of factorial is:
  - n! = (n-1)! * n ("the factorial of n equals the factorial of n less 1 times n")
- n! appears on *right* side of definition, where normally it should not be

https://it.wikipedia.org/wiki/Algoritmo_ricorsivo

Recursive algorithms

An iterative implementation:

```
Factorial (n):
{
      fact = 1;
      while (1 < = n) {
            fact = fact * n;
            n --;
       }
      return fact;
}
```

A recursive implementation:

```
RFactorial (n):
{
    if (n < = 1) return 1;
    return n * RFactorial (n-1);
}
```

Differences

- The two versions are not so different in size
- The iterative version start from the end (n) and goes backward to 1 while calculating
- The recursive version first goes back to 1 and then goes forward to n while calculating
- The fact that the recursive version maintain a local knowledge (its state) is not so evident

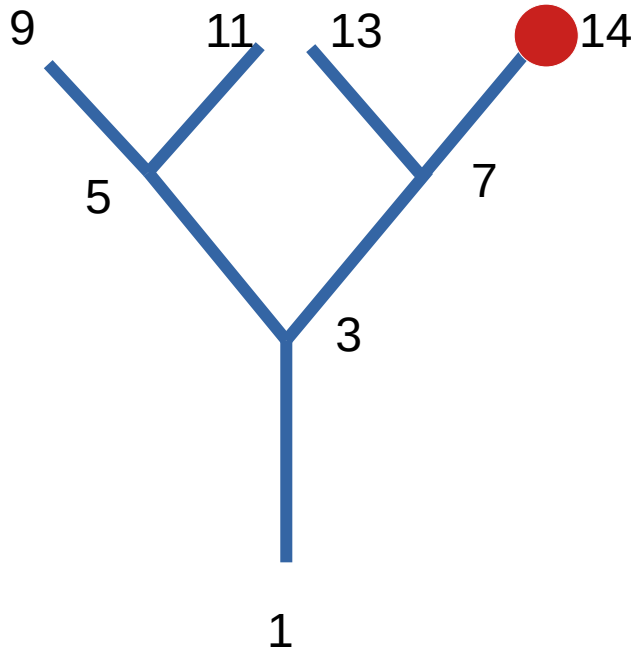
Problems

- Factorial is a complex concept, that cannot be used to explain recursion to young pupils
- What is the interest to find Fact(9)?
- In any case, this is an explanation of the implementation of a mathematics function which is already recursive in itself

Recursive problems

- Recursive problems are those in which:
  - there is ramification
  - the solution could be anywhere in the space
- Generally speaking, it seems that there are problems that are well suited for a recursive implementation (trees, labyrinth,..) and problems which are not (sequences)
- But it depends on the manner in which we represent data

The Data



A classical representation

- 4 lists of nodes (=paths)
  - 1,3,5,9
  - 1,3,5,11
  - 1,3,7,13
  - 1,3,7,14
- The function that checks for solution(s) is simply:
  - isEven(x)?

Another classical representation

- A list of couples: "parent", "son":
  - 1,3
  - 3,5
  - 3,7
  - 5,9
  - 5,11
  - 7,13
  - 7,14

The programs

- Iterative
  - for each path:
    - for each node of the path
      - check if it is the solution
    - if there aren't more path to check, fail

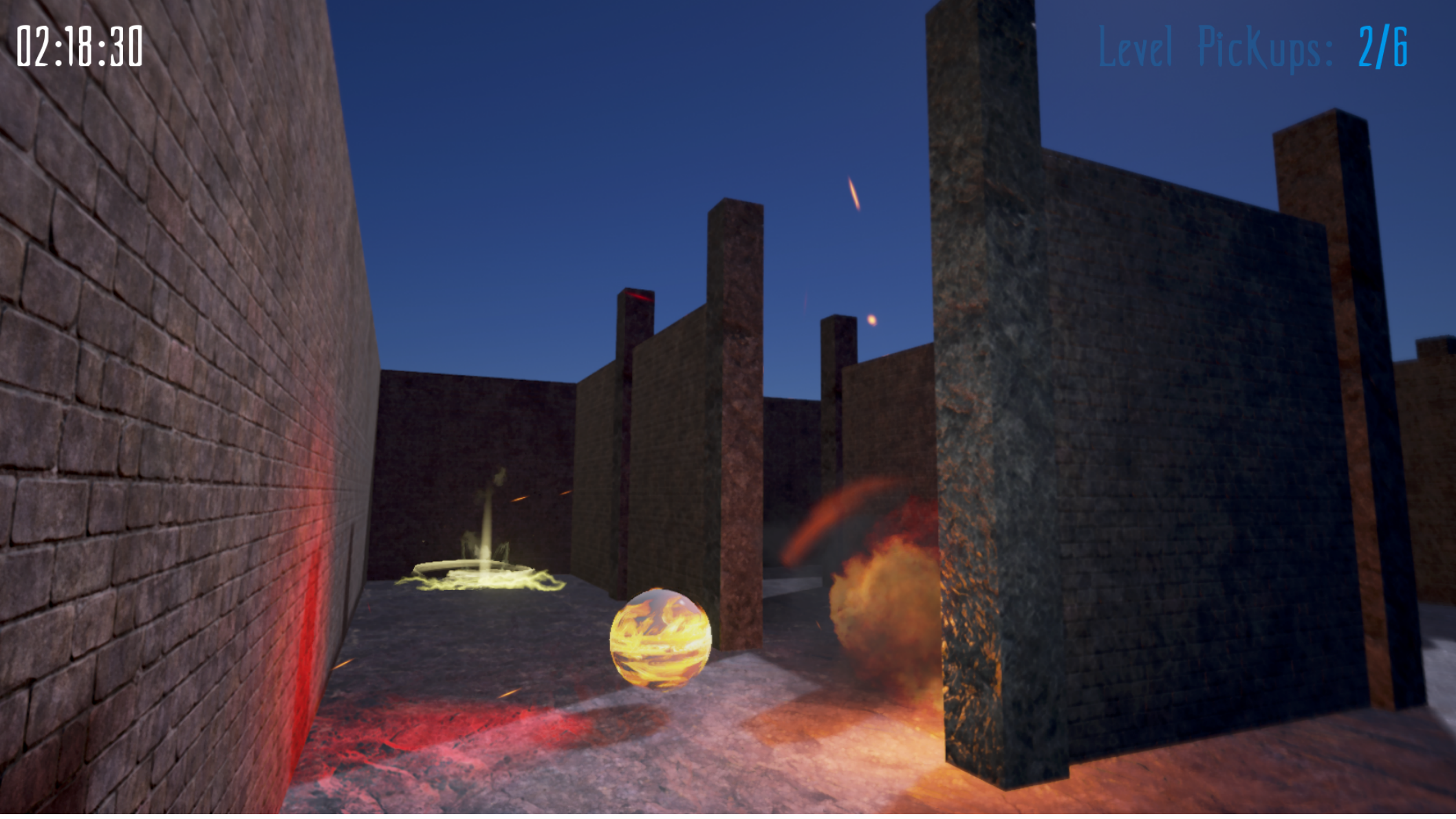
- Recursive
  - take a couple x,y
  - check if y (= son of x) is the solution
    - if yes, exit
    - if no,
      - if there are no more couples, exit
    - find another couple that has y as parent and start again

Knowledge of the world

- Iteration algorithms have a complete knowledge of the world
- Recursive algorithm *knows nothing* about the world
- From this point of view, they are a little bit more intelligent: they try to find their way *alone*
- From the point of teacher, it is important to mark the difference and explain recursion on the basis of which amount of knowledge the algorithm has
- Recursive implementations make sense when the algorithm don't have a complete knowledge of the world
- A good example could be the research activity

Research

- When doing research, we don't have a full knowledge of the world; we have hypotheses, theory, but still not knowledge
- We have only some portion of it (perhaps thanks to sensors, which are doors between real world and digital world)
- But to explain recursion to students in a real research field could be difficult...


02:18:30
Level PicKups: 2/6

Games

- When programming a game there are two types of knowledge:
  - the knowledge of the universe, which is complete and in the hands of designer who imagine that universe
  - the knowledge of the agents, which is dynamic and change during the game
- During game the agents explore the universe, i.e. they extend their knowledge towards the limit of global knowledge
- In this context, recursion is a natural option because it is a natural way of representing the partial knowledge of each agent

Matryoshka dolls

Invented in XIX century in Russia by Zvyozdochkin (craftsman) and Malyutin (painter)

The M-Game

- A Matryoshka wants to find a treasure
- We (the programmers) design the world as a dungeon, a tree, or whatever
- The Matryoshka knows how to recognize a treasure, but doesn't know where the treasure is; but *we* know
- The Matryoshka has some limits: she cannot do more than ONE step at a time
- But she can produce unlimited little Matryoshkas...

The Matryoshka program

- Matryoshka thinks:
  - "A solution is an even number"
  - "If I found an even number, I report it back to my Mother"
  - "If not, I will check how many path stems from here and I produce a Daughter to which I give the same task"

Advantages

- Every Matryoshka has a limited knowledge of the tree (a "state"), which is specific and not shared with other Matryoshka
- It is easy to show a version of the program with real wooden Matryoshkas
- The idea of a doll bearing inside another doll is familiar to humans: its name is pregnancy

Implementation

- There could be good implementations of this game using clones in Scratch or Snap! :
- https://en.scratch-wiki.info/wiki/Advanced_Clone_Usage
- https://snap.berkeley.edu/snap/help/SnapManual.pdf

  pages 70-74