Non-determinism in nowadays computing and IT education

J. Henno*, H. Jaakkola** and J. Mäkelä***
* Tallinn University of Technology, Estonia
** Tampere University of Technology, Pori
*** University of Lapland, Rovaniemi, Finland
jaak@cc.ttu.ee

Abstract – Once-upon-a time computers and computations they delivered were considered ultimately deterministic. But currently we encounter random events, i.e. non-determinism in many areas of computation practice: non-determinism introduced by network latency, inherently non-deterministic computations with ‘big data’ and ‘deep learning’ where the results are probability distributions with errors, which also are probability distributions and both depend on initial selection of samples etc. For IT education one of the most disturbing sources of non-determinism and non-repeatability of previous examples comes from massive use of libraries and API-s (Application Programming Interface), which has made common the ‘top-down’ style of programming and negligence to practical issues – finiteness of computer memory and speed. Significance of the classical source of knowledge - printed hard-cover books - is diminishing, since by the time books are out of print there are already new versions of programs, new protocols, new technologies and libraries, and these new versions often do not work with old ones. The most relevant source of information has become Internet. But Internet is full of useless sources, since ‘Internet never forgets’ – together with sources describing latest program versions, libraries, technologies etc. there are still around tens of publications which use some by now already outdated program versions, libraries, technologies. Students, who are eager to perform well in the next recruiting interview are spending many evenings trying to swim in this swamp of non-deterministic mess, where most of presented examples are not repeatable. And the situation is becoming worse, since many of authors e.g. YouTube videos do not want to teach, but to earn using Google AdSense.

Keywords – computation; determinism; networks; deep learning; big data; Internet

I. INTRODUCTION

“Something automatic happened. Moodle is down” was the message from our Moodle manager; Windows 10 search does not find files in your computer – Microsoft proposes six sets of instructions [1] for you to correct the problem (involving editing the registry – always considered strongly no-no action for casual computer users); computer started, but for next half an hour screen shows only spinning dots – Microsoft consoles [2]: “It won’t happen often”, “it generally means that the updates are finalizing” – but this particular computer is not connected to network so updates are impossible; computer was connected, but Firefox refused for several minutes to update any tab; security cameras which are safe only when they are disconnected [3]. Everyone can add to this list his/her own experiences of “automatic happened”.

In spite of all our achievements in IT technology such things happen and in order to decrease this “just happened” and to explain to our students how to decrease these happenings we have to understand, why these things happen.

The belief in determinism and repeatability of computations is one of cornerstones for most people dealing with computers. The repeatability principle is the basis of all science. Everyone can repeat the classical experiments of Newton or Galilei which became the base of classical laws of physics. This repeatability principle is also the cornerstone for our students – in order to believe in truths what we deliver those truths should be based on some repeatable evidence, students should be able to test them out themselves, to repeat the experiments, re-create programs and get the same results.

Unfortunately this principle of unconditional repeatability has started vacillate.

The first indications un-repeatability were voiced already in the beginning of the last decade in ‘soft’ fields - various subfields of medical, life and behavioral sciences, in fields, where all new discoveries are based on experiments and theoretical inferences could be stated only from these experiments. It turned out, that there is a “replication and reproducibility crisis” - many publications were based on experiments and data which nobody could repeat [4]. Poll organized in 2016 by the journal Nature show, that more than half of scientists considered this as a real crisis in science [5].

A similar “replication and reproducibility crisis”, but for different reasons has appeared in programming or more generally, software production.

II. ESSENCE OF PROGRAMMING

In its very essence, programming is the art and ability of foreseeing future. We write something (the code), consisting of series of ‘small’ actions – move this value or those words to that variable, add or multiply this to that, repeat these actions until some condition becomes true (or vice versa) and we have to foresee, what actually happens, when all these small piece-vice actions are automatically executed, what will be the result – it should be exactly what we wanted it to be (computers are in principle deterministic!). But when our programs grow, become longer and longer (we want to do more and more!), we can not foresee everything – our
memory is limited. Thus our foreseeing ability weakens more and more and the result which appears after execution of our well-planned actions appears as non-foreseen, i.e. looks like nondeterminism.

It is sometimes estimated, that a chess grandmaster could foresee, i.e. to calculate the situation in the game for twenty moves ahead. For computers to achieve this is needed rapid processing power, e.g. the chess program X3D Fritz which was the first computer program to play on par with the world chess champion Garry Kasparov in 2003 run on four Intel Pentium 4 Xeon CPUs (Central Processing Unit – computer’s processor) at 2.8 GHz – best PC processors at the time (today smartphones can run similar chess programs). Besides of processing speed computer chess programs have also other (for tasks from practice, e.g. chess – practically unlimited) resource – memory. For the chess programs which finally defeated the human chess master where from the very start of their development used databases of chess endgames [6]. The result – winning the human world champion – was based on these databases and a “brute force” calculation capacity – it run on custom VLSI chips to execute the alpha-beta search algorithm in parallel.

The chess program won the future with ‘brute force’, i.e. with more memory and quick processors. But in everyday programming practice programmers are constantly dealing with future and want to have for this formal concepts. Theoretical concept future together with closely related concepts promise, delay and deferred future were formalized already in the second half of seventies of the last century [7], [8]. Since the concepts future, promise etc. are very important in asynchronous web programming, these concepts are already introduced in several programming languages e.g. in the rather novel languages Dart [https://dart.dev/], Kotlin [https://kotlinlang.org/], but also in ‘old hand’ C++ [9], [10].

Formalism can only reduce programming errors, not to eliminate them. It is rather easy to err even in seemingly simple situation, as shows an example from Dart [11]). The following code contains an error – can you find it?

```dart
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() {
  return Future.delayed(Duration(seconds: 4), () => 'Large Latte');
}

void main() {
  print(createOrderMessage());
}
```

Non-determinism (and also non-repeatability of program results) comes from several sources.

Computers are finite devices, they represent e.g. real numbers (there are infinitely many of them!) using approximations, i.e. with some error; in mathematical operations these errors may accumulate, thus even the most basic laws of arithmetic’s, associativity and commutativity of addition

\[ a + (b + c) = (a + b) + c, \quad a + b = b + a \]

may not hold. For instance, summing up a series of 10000 randomly generated real numbers first in forward and then – in backwards direction will usually produce different results:

\[ \text{Difference: 4.192202140984591e-13} \]

Non-determinism in calculations appears constantly in calculations where the GPU (Graphic Processor Unit) is used [12], [13], e.g. in many Machine Learning areas. Most of Machine Learning algorithms are still local, i.e. designed and executed on one computer. But since machine learning algorithms employ massive computations, besides utilizing GPU is becoming popular also distributing these calculations over computer network and cloud [14], and communication on computer network inherently includes also non-determinism.

III. RISE OF UNCERTAINTY AND NONDETERMINISM

The world is changing – rapidly; this has been repeated also rapidly and is one of few (the only?) things which is not changing, i.e. non-deterministic.

With the first stand-alone, i.e. not network-connected computers their determinism was nearly absolute. Computers are deterministic devices, if the produce random results then they are severely broken. Von Neumann, ‘Father of the Information Theory’ commented on this: “Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin.”

Randomness appeared when computers were connected and appeared computer networks. If a computers communicate over network and one makes a request to another, then it is impossible to say when the answer arrives – this depends on all the other computers/servers on message’s path. ‘Classical’ programming languages expect determinism – they want to get everything in expected time. In order to handle network’s communication uncertainty and nondeterminism were in programming languages for functions which implemented network queries added a ‘callback’ argument – a function, which was executed when query returned the answer (in previously unknown time) And answer may not arrive at all – browser shows the response “404 – Page not found”, thus the queries begin with try and end with catch(error). Web programming is asynchronous, programs which communicate with other computers over computer network should always be ready for arbitrary delays. This creates randomness, which is a natural part of communication in networked world.

Computers were invented for calculations and at that time the idea of calculation was that this is done in a single
computer using finite amount of data what is presented as an input at the start of the program. The first successful programming language for students, the BASIC (Beginners' All-purpose Symbolic Instruction Code), created for teaching in the Dartmouth College in 1964 even included program data at the end of program’s text on separate lines beginning with “DATA …”. Such a computation is totally closed – once it is started (with fixed input), nothing will disturb it and it even finishes or does not, i.e. goes into endless loop (Alain Turing understood this and more-or-less simultaneously with Austrian logician Kurt Gödel). But there are not so many such closed-form problems, i.e. problems, where data is presented locally at-once, at start of a program, especially taking into account computer’s strongest feature – re-use of programs. The president of the mighty IBM (International Business Machines corporation - at the time of introduction of this name these ‘business machines’ were punched card tabulators) instinctively understood the situation when in 1943 he stated: "I think there is a world market for maybe five computers". And he was not alone, e.g. the grandson of the famous naturalist Sir Charles Darwin, at the time head of the Britain's National Physical Laboratory, where computers in 1946 were introduced considered one computer to suffice to solve all the problems that are demanded of it from the whole country [15].

And it seems that Watson, Darwin and several other prominent researchers of the time were right – the well-isolated problems with fixed amount of input data are already mostly solved and programs to solve these problems are already mostly written.

The programs for isolated problems with fixed data are in principle all similar to the classical programming example “Hello, World!”; but there is little need for such “Hello World”-type programs any more (although many IT teachers in universities do not believe). Nowadays computers are not used for closed, finite sets of data – they are used for communication and big, open-ended sets of data. We use them for sending e-mail, watching TV or live ‘reality’ shows and computers are connected into large, complex networks, where messages are moved as small fixed-size packets. On networks messages ‘hop’ from server to server, their next destination is calculated ‘on the fly’. Nobody can know beforehand the exact route they move or tame it takes, thus nobody can also not know the exact time of packet arrival or the rate of ‘lost’ packets – generally 1-2.5% is considered quite acceptable [16].

Packets are nearly independent, since the main Internet protocols: HTTP (Hypertext Transfer Protocol), UDP (User Datagram Protocol), DNS (Domain Name System) were designed stateless - each information packet travels on its own and does not know anything about any other packet. Each communication in Stateless Protocol is discrete and unrelated to those that precedes or follow. Stateless Protocols greatly simplify the design of Server, but increase traffic – client has to make a new request for every item. Modern web-pages contain hundreds of components, e.g. the newspaper “The New York Times” page for 25.02.2020 contains more than 40 separate items.

The ‘step-wise’ protocols were created in the ‘innocence’ era of the Internet. The current Internet, where nearly half of traffic is created by software bots – autonomous applications (apps) that execute tasks (scripts) over the Internet on connected computers. The all-expanding world of bots includes both good ones (e.g. Google crawlers, who constantly traverse Internet and store data for Google search, tools to improve programmers productivity [17] or Workshop [18] – a tool for freelancers which scans the internet for freelance job offers, chatbots, appearing on many web sites – everyone can easily build one [19] etc.) and bad ones, who seek to carry out data theft, send spam, distributed denial-of-service (DDoS) attacks, account takeovers etc. In 2018 bad bots created 20,4% of all the Internet traffic [20].

One of the most notable good bot achievements was the in the Google’s London-based AI company DeepMind developed Go-playing bot [21], which in 2016-2017 defeated top professional Go players. In chess it is estimated to be ca 10^41 different positions [22]. Go is on several orders more difficult, in Go is estimated to be approximately 2.081681994 * 10^170 legal positions [23]. Thus the Go-playing program had to use two neural networks and random search to prune the tree of future positions [24].

IV. MIMICKING HUMANS

Even more complex is the next great task (game) for computers – understanding natural language. Adult native English language speakers know 20,000–35,000 words [25] (Shakespeare – 35000 [26]), the number of possible sentences is infinite. Natural Language Processing (NLP) is done using neural networks and GPU-s and involve many random elements. But NLP programs are already on the verge of understanding natural language. Programs from the top 5-6 teams are progressing rapidly [27] and English language models allow already with very little experience in NLP to build a rather naturally behaving questions answering bot for website [28].

NLP systems have already proved their practical value: the Canadian NLP-based health monitoring platform BlueDot [29] scanned news reports and airline ticketing to predict the spread of diseases send first warning about the outbreak of coronavirus already on December 31, more than a week before the World Health Organization notified the public of virus outbreak in China [30].

Humans have tried to create systems which understand their own, natural/human languages for a long time and most approaches have been ‘top-down’, i.e. word’s meaning was defined using some hierarchical system of concepts (‘thing’ – ‘living thing’– ‘animal’– ‘mammal’– ‘human’– ‘John’). All such systems are subjective and do not agree with each other. NLP introduced a different, ‘bottom-up’ approach – word meaning is defined by word context, i.e. other words which occur often together with the word. When sufficiently large text corpus is analyzed for co-occurrences appear concepts – groups of words with similar meaning [31], e.g. processing English-language presentations from Mipre.CE 2017-2019 (25037 unique words) using the Tensorflow Python libraries.
[32] revealed many closely-packed groups of words, i.e. concepts.

V. INTERNET IS WAKING

With AI-supported bots, Internet is often already resembling a living system, where your mobile phone may itself connect to your fridge in order to report to you what you should buy in order to entertain quests whose mobile just informed your mobile about their arrival etc. This is very different from ‘classical’ computing and requires massive layers of software to make it work and, most important – to keep it safe.

The first generation of computer safety tools – virus scanners - were based on scanning application’s code for certain software patterns - this is like deciding about a visitor by looking at his/her suite. Nowadays safety software is more and more based on inspecting actor’s behavior – how does it differ from normal ‘human’ behavior? If a bad bot is constantly ‘knocking your door’, i.e. sending requests hoping to guess your password or simply cause your computer to fail (DDOS), then the number of such requests should be limited; if some application in your computer regularly emerges at midnight and tries to connect to somewhere then this looks like a bot (Windows can also wake computer up and start an update); if many customers want simultaneously order from a web shop, then also the number of requests should be limited (the web throttling [33]). Web applications (e.g. web shops) interface designers apply numerous safety guidelines [34], which are all the time becoming more stringent; e.g. the U.S. government Federal Information Processing Standard (FIPS) Publication 140-2 level 4 requires “the ability to be tamper-active, erasing the contents of the device if it detects various forms of environmental attack” [35] – the attacked computer should kill itself.

VI. REUSE – API-s

With growing complexity developers have to repeat good, already tested designs. Good, working designs are implemented as software libraries. Nowadays nobody starts to develop a (web) application with command-by-command style – all developers use ‘library-by-library’ style instead.

Libraries and their API-s (Application Programming Interface, the input-output functions which libraries reveal outside) have become the new programming language. There are thousands of libraries and API-s: the Python package index [36] lists over 200 thousand projects, the Javascripting index (“The definitive source of the best JavaScript libraries, frameworks, and plugins” [37]) lists over 1500 JavaScript libraries, PHP has over 200 thousand modules, Apache Maven project repository [38] lists for Java over 300 thousand packages and the node.js [39] – over 1.2 million packages.

The term API which originally was used only for interfaces of packages has nowadays much wider meaning, since there are many ‘things’, which also have programming interface – mobile phones, door locks and doorbells, smoke-detectors, fridges etc. Thus the number of API-s is accelerating. The web communication is mainly based on two web service API-s: the REST and the SOAP [40]. But besides these there many types of open-source API-s: for music, travel, beer and wine, machine learning, biometrix etc. [41], [42].

Public API-s allow to use programs, i.e. functionality developed elsewhere over the web: explore and download NASA imaginary [43], add to you mobile phone a button for downloading stuff from Amazon [44] etc. And use of API-s has made prevailing the top-down style of application development.
VII. THE TOP-DOWN DEVELOPMENT STYLE

The classical programming style is ‘bottom-up’ – programmer writes code line-by-line and his productivity is (if needed) measured in loc (lines of code) units. This is the still the prevalent style in most software production textbooks and courses.

But those ‘Hello World’ programming snippets are usually all local, to be executed in local computer and they show the message ‘Hello World’ also on local computer screen. To print something to local computers screen – it was quicker to say the message directly using your own human voice. In our all-connected world based on computer networks the program should show the message in response to request from another computer on network. Such a program becomes far more complex and in our security-laden software creation environment its creation begins not from bottom, i.e. by writing programs code, but from top and involves several steps: initializing programming project, creating classes etc. For instance in Java language, using the Spring framework these steps would be (at least) following:

1. Starting the Spring Initializr; this enables including all dependencies (other libraries) needed for the application, here the Spring Web dependency. Spring does also a lot of project setup, for this user has to give values to project metadata variables Group and Artifact and to select the build tool: either Maven or Gradle; if user selects Maven, Spring creates the POM (Project Object Model) description xml-file (44 lines of xml); for Gradle will appear a bit shorter build description XML-file.

2. Next should be created the web service, which handles required interactions:

- first the GET request for greeting; the request body contains values of the request id and message parameters, optionally also user name parameter in the query string; the GET request should return a 200 OK response with JSON in the body that represents the greeting

- the corresponding JSON file in response is created by a resource representation class file Greeting.java (14 lines of Java code).

3. To handle HTTP requests in Spring’s framework for building RESTful web services is needed a Resource Controller, which is created with 11 lines of Java code.

The (more-or-less) understandable are two lines in file Greeting.java:

```java
    this.id = id;
    this.content = content;
```

Although even here is difficult to understand numbering – when meeting a friend on street, we usually do not start the conversation in style:

“First: Hello John! Second: how do you do?”

All other more than 50 lines are web machinery, which programmer should use, but often does not understand, thus tries to build using the copy-paste programming method. Naturally, this is very error-prone method, thus greatly increases randomness in our digital word.

For students this takes lot of time to understand and many do not. The main thing many have learned is that there should be libraries, thus one student (experience of the first author) started a simple 2D browser (i.e. JavaScript) ‘Flappy Bird’ game [47] with linking 75 node.js classes. He was not able to use several on-line tutorials with code [48], since these tutorials do not use external libraries (not needed for such a simple task); he tried to use all ‘bells and whistles’, i.e. these classes and then composed totally senseless lines of code:

```
collides(us, them)
```

From here it follows, that in every frame the bird would collide with itself and thus could not fly.

VIII. COMPUTERS ARE FINITE

Top-down mindset usually does not support understanding of computer memory – such students often create programs, which ‘eat memory’ – they create new objects (use malloc or new) at run time; for new objects is needed also new memory, thus the memory used by program may start growing and finally the whole computer halts [49]; even if compiler manages to avoid crash with garbage collection, program still becomes slow, since new memory is obtained by request from Operating System (e.g. Windows).

With the top-down pattern of thinking students often do not understand that computer, computer’s memory, network bandwidth etc. are all finite resources - they use them as if they were endless. Fortunately there are several tools which help to investigate low-level issues which may cause troubles and computer’s unexpected behavior.

Many users drag all icons of their programs to desktop. This makes it very convenient to start these programs, but this also may make computer start (very) slow, since e.g. Windows loads all these icons at startup reading them from the file desktop.ini.

Somewhat similar attitude is prevailing among software producers. In order to make loading of their programs quick (every program wants to be quick), they make OS (Operating System) to load all these programs at computer start-up (booting) into computer’s RAM (Random Access, i.e. runtime) memory, using for this every possible way, e.g. all possible registry settings. Microsoft utility Autoruns [50] reveals, that in an average Windows computer are on startup loaded hundreds of programs (besides Windows own modules). And many programs are so clever, that after un-enabling their entries in Autoruns after a while they re-appear, e.g. even if Microsoft Outlook or Skype are not installed, in Autoruns still appear 6 entries for Outlook (e.g. Outlook Social Connector, Access Outlook Data Collection Add-In), 5 – for Skype; 4 - for Google etc.
The unnecessarily loaded on startup programs all use computer memory and make booting slower, since most of them want on startup to connect to their servers for updates. Looking with the TCPEye [51] at the list of current network connections shows, that computer is actively keeping connection with tens of remote servers – this is the pause many computers exhibit at startup.

All these programs do something in computer; the Process Monitor [52] reveals, how many (often unnecessary) processes are started in your computer (a simplified information gives also Windows Task Manager: press Ctrl+Alt+Del). My computer was just started, but Process Monitor reveals, that Windows Explorer has already four times sent queries to a quite good free tool for creating CDs, DVDs, and Blu-ray disks CDBurnerXP [53], although the program has not been used.

All browsers (Firefox, Chrome, Edge etc – currently there are web 116 browsers [54]) boast with their speed. Browser is not a very big and complex program (e.g. Firefox – 555KB, Chrome – 1672 KB, Opera – 1493 KB) and they all have to use the same ‘pipe’, i.e. network connection, thus the main source of their speed (for opening a page) comes from cache – they save everything in order to open it on next time very quickly from the local hard disk cache, e.g. in this computer Firefox’s cache is 1.2 GB.

Such strange entries in Autoruns, TCPEye, ProcessMonitor allow to discover viruses and bots, e.g. the site BleepingComputer [55] maintains a list of nearly 27 thousand automatically starting programs, divided to several categories: good, bad, unknown. Cleaning cache (or at least deleting browser’s history – easy to do from browser’s menu) deletes also all cookies, thus e.g. if an on-line newspaper first shows some pages for free and then announces “You have used all your free limit – start paying!” usually deleting the history re-enables free pages.

IX. AGE OF DATA

The world economy is changing. Humanity is overusing world’s natural resources. We use ca 1.7 times more than nature is capable to re-produce – already by August 1st we have already consumed the resources nature can regenerate in a year [56], and the speed of our consumption of natural resources is increasing To satisfy our all-increasing needs and World’s increasing population we have to invent something to replace natural resources, first of all – oil. The age of oil is replaced with the age of data, BIG DATA.

Data is not eatable, we can’t drive our cars with data, however big. Data, BIG DATA allows us to reduce waste and losses in our complex world-wide economics, streamline production, distribution and consumption. And the effects are already present – according to the United Nations Digital Economy Report 2019 [57], already in 2016 the worldwide digital sphere produced 15.5 percent of global GDP; by 2025 it is expected to grow to 24.3 percent of global GDP [58].

Just like oil, data also need processing. Processing of digital data happens in our digital devices – computers, mobile phones etc., their networks and many different communications and interactions which occur in these networks – computer-computer, human-human, human-computer-human. And waste and losses occur in many different steps of this process. Reducing or eliminating even small amount of losses, nondeterminism for message
receivers in communication would greatly increase efficiency of the whole mankind.

X. SOFTWARE error

With complexity and constant updates increases possibility of errors. Updates are often created by programmers, who did not create the original program, do not know all the previous enhancements and corrections, especially security improvements. The results are often laughable, but they seriously affect mindset of computer users, who somehow have to deal with these errors or very difficult to understand messages.

Microsoft Word is probably the most essential tool for many users. With growing security standards its complexity grows. The “Trust Center” of Microsoft Word lists in “Advanced Options” (at least) 115 options, what user should be able to decide, whether to use or not and the “Trust Center” shows 17 file formats with different security options, some obviously already very old. “Word 2 and earlier” – Word 2 appeared more than 35 years ago. For most (casual) users, all this is just a digital noise.

With growing requirements for security have increased also browser’s errors in page delivery. Clicking in a search result in Google search may produce many responses:

There has been a critical error on your website.
Learn more about debugging in WordPress.

What is ‘Wordpress’ and why should I learn its debugging – this the task for site owners!

The page you are trying to view cannot be shown because the authenticity of the received data could not be verified. Please contact the website owners to inform them of this problem.

If I can’t open the site, then how should I ‘contact the owners – please provide a phone number or an e-mail address !

The errors may seem insignificant, but for a person who is starting to learn programming they may be overwhelming.

XI. CONCLUSIONS

Complexity of software is constantly growing. In current software production overwhelming is the ‘top-down’ approach. Managers want everything to be done quicker and cheaper, software projects are presented as a hierarchical system of tasks and their implementation based on massive use of libraries and IDE-s. Low-level technological aspects – peculiarities of processes on computer networks, finiteness of all computer and network resources also grow. Although they often get less attention, these low-level technical details are important and not understanding them creates often “automatic happened”, thus should be essential part of software production education.

The amount of practical knowledge what programmers currently should know seems to be already comparable with the most knowledge and practice-intense profession – medical. But the organization of education is different: medical students have more than 60 specialties and after graduation have to pass internship before they get a full doctor’s license. It may be good to introduce something similar also to programming education. Some steps in this direction have been already taken, e.g. the Canadian e-commerce platform Shopify is offering the “Dev Degree”: a 4-year, work-integrated learning program that combines 4,500+ hours of hands-on developer experience with 4,000+ hours of academic experience and produces accredited Computer Science degree from either Carleton University or York University [59].


Number of Possible Go Games. https://senseis.xmp.net/?NumberOfPossibleGoGames

Neural Networks and Go. https://senseis.xmp.net/?NeuralNetworksAndGo


SuperGlue Leaderboard. https://super.gluebenchmark.com/leaderboard


Pypi. https://pypi.org/

.mavenn.apache.org/

https://www.npmjs.com/


8+ Biometrics APIs At Your Fingertips. https://nordicapis.com/8-biometrics-apis-at-your-fingertips/

https://rapidapi.com/blog/types-of-apis/

https://api.nasa.gov/

https://aws.amazon.com/api-gateway/

https://spring.io/


https://flappybird.io/

.w3schools. HTML game example. https://www.w3schools.com/graphics/game_intro.asp

.pc.net. https://pc.net/glossary/definition/memoryleak


https://cdburnerxp.se/

Browser List. https://www.browserlist.com/


Pypi. https://pypi.org/

.mavenn.apache.org/

https://www.npmjs.com/


8+ Biometrics APIs At Your Fingertips. https://nordicapis.com/8-biometrics-apis-at-your-fingertips/

https://rapidapi.com/blog/types-of-apis/

https://api.nasa.gov/

https://aws.amazon.com/api-gateway/

https://spring.io/


https://flappybird.io/

.w3schools. HTML game example. https://www.w3schools.com/graphics/game_intro.asp

.pc.net. https://pc.net/glossary/definition/memoryleak


https://cdburnerxp.se/

Browser List. https://www.browserlist.com/


Pypi. https://pypi.org/

.mavenn.apache.org/

https://www.npmjs.com/


8+ Biometrics APIs At Your Fingertips. https://nordicapis.com/8-biometrics-apis-at-your-fingertips/

https://rapidapi.com/blog/types-of-apis/

https://api.nasa.gov/

https://aws.amazon.com/api-gateway/

https://spring.io/


https://flappybird.io/

.w3schools. HTML game example. https://www.w3schools.com/graphics/game_intro.asp

.pc.net. https://pc.net/glossary/definition/memoryleak


https://cdburnerxp.se/

Browser List. https://www.browserlist.com/


Pypi. https://pypi.org/

.mavenn.apache.org/

https://www.npmjs.com/


8+ Biometrics APIs At Your Fingertips. https://nordicapis.com/8-biometrics-apis-at-your-fingertips/

https://rapidapi.com/blog/types-of-apis/

https://api.nasa.gov/

https://aws.amazon.com/api-gateway/

https://spring.io/


https://flappybird.io/

.w3schools. HTML game example. https://www.w3schools.com/graphics/game_intro.asp

.pc.net. https://pc.net/glossary/definition/memoryleak


https://cdburnerxp.se/

Browser List. https://www.browserlist.com/


Pypi. https://pypi.org/

.mavenn.apache.org/

https://www.npmjs.com/


8+ Biometrics APIs At Your Fingertips. https://nordicapis.com/8-biometrics-apis-at-your-fingertips/

https://rapidapi.com/blog/types-of-apis/

https://api.nasa.gov/

https://aws.amazon.com/api-gateway/

https://spring.io/


https://flappybird.io/

.w3schools. HTML game example. https://www.w3schools.com/graphics/game_intro.asp

.pc.net. https://pc.net/glossary/definition/memoryleak


https://cdburnerxp.se/

Browser List. https://www.browserlist.com/


Dev Degree. https://devdegree.ca/