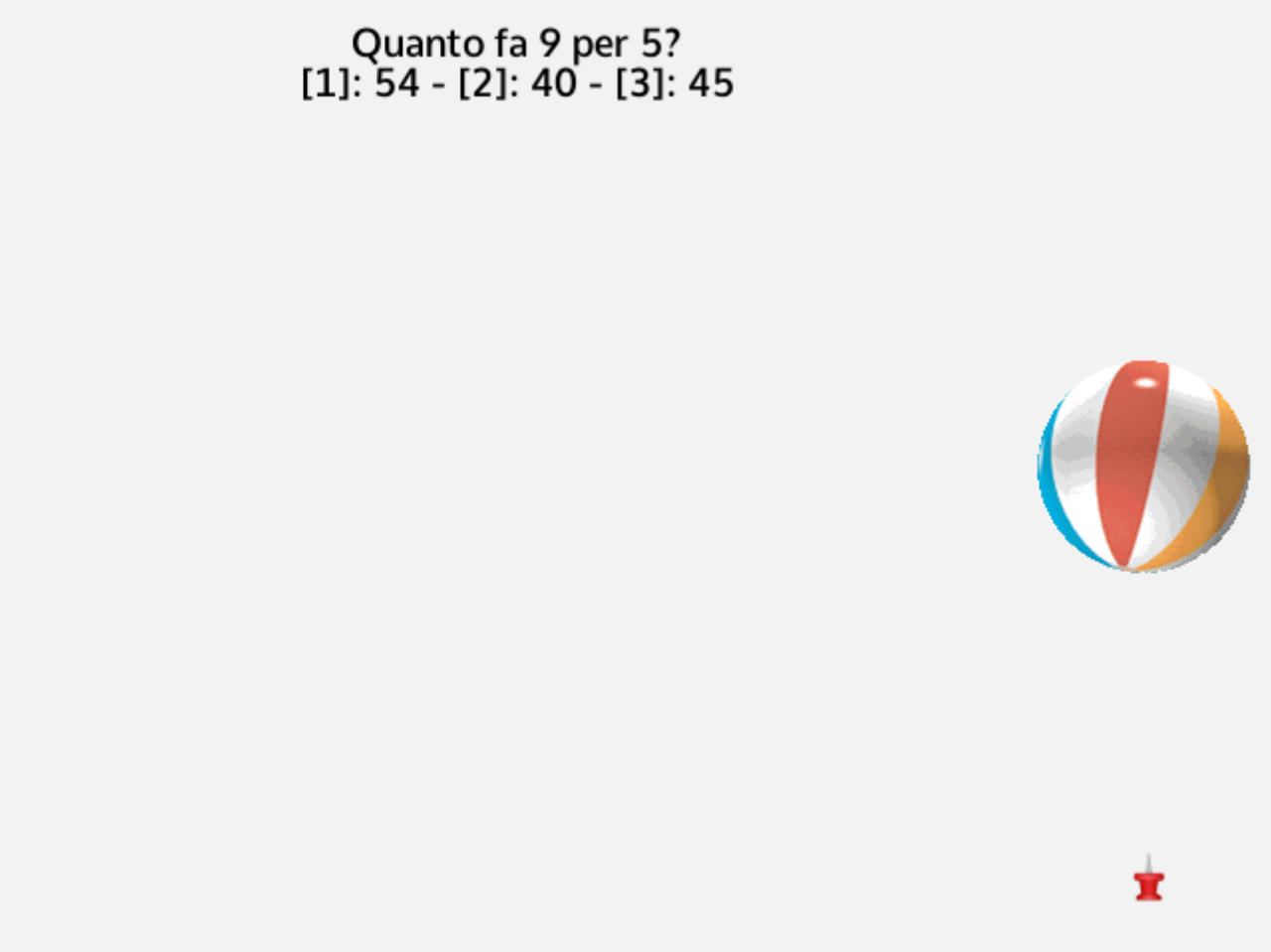
Question ball



Si potrebbe definire questo tutorial una simulazione di “Lascia o raddoppia”, per chi si ricorda la trasmissione di Mike Bongiorno (del secolo scorso…). Il funzionamento è semplice, ma programmarlo serve per fare pratica con i concetti base di un videogioco: muovere oggetti sullo schermo, reagire alle azioni del giocatore, rispettare i tempi del gioco.
Il programma fa una domanda al giocatore, e accetta una risposta entro il tempo che ci vuole alla palla per cadere sopra la puntina. Se il giocatore sceglie la risposta giusta il gioco continua con un’altra domanda; altrimenti la palla esplode e il programma si ferma rendendo conto del numero di risposte corrette.

Per funzionare, il gioco richiede che sia presente la libreria PyGame; dato che non è compresa nella dotazione standard del linguaggio, occorre scaricarla dal sito (https://www.pygame.org) e installarla seguendo le indicazioni relative al proprio sistema operativo. Per risparmiare tempo, sarebbe preferibile che fosse già presente al momento di iniziare questa attività.

1. La prima versione del gioco

Prima di tutto occorre importare i moduli che ci serviranno per la realizzazione.

Per la parte grafica importiamo tutto pygame, così le chiamate alla libreria e altri componenti saranno esplicite.
Per la generazione di numeri (pseudo)casuali invece ci serviremo di quattro funzioni messe a disposizione dal modulo random:
- `seed()`: inizializza la generazione dei numeri impostando come primo numero il valore in millisecondi dell'orologio interno
- `randint()/randrange()`: le due funzioni sono quasi identiche e generano ad ogni chiamata un numero casuale in un intervallo prestabilito
- `shuffle()`: questa funzione restituisce la lista passata in argomento con gli elementi scambiati in modo casuale

```
import pygame
from random import seed,randint,randrange,shuffle
```

Cominciamo a definire l'oggetto che servirà come indicatore del tempo in esaurimento.

```
class Ball:

    def __init__(self,filename):
        self.x = 10
        self.y = 0
        self.bitmap = pygame.image.load(filename)
        self.bitmap.set_colorkey((0,0,0))
```

Il costruttore della classe, chiamato ad ogni creazione di un nuovo oggetto, imposta la posizione della palla in alto sullo schermo (attributi x e y, corrispondenti all'ascissa partendo da sx e dall'ordinata partendo dall'alto), carica l'immagine e le imposta il colore di sfondo per la trasparenza.

```
    def set_position(self, xpos, ypos):
        self.x = xpos
        self.y = ypos

    def render(self,screen):
        screen.blit(self.bitmap,(self.x, self.y))

    def reset(self):
        self.y = 0
```

Il metodo `set_position()` imposta nuove coordinate all'oggetto, realizzando in pratica il movimento fotogramma per fotogramma, il metodo `render()` è invocato per "stampare" l'immagine della palla sullo schermo del gioco (anche se la palla sarà effettivamente disegnata con il metodo `display()` dell'oggetto predefinito `pygame.display`) e infine il metodo `reset()` si limita ad azzerare il tempo impostando la posizione della palla di nuovo nel punto più alto.

Passiamo alla classe che gestisce la domanda vera e propria.

```
class Question:
```

```
····def __init__(self):
········self.question = ""
········self.answer = 0
········self.choice = [0,0,0]
········self.newQuest()

····def newQuest(self):
········# generazione della nuova domanda (moltiplicazione)
········facts = [randint(2,10),randint(2,10)]
········while (facts[0] * facts[1]) == self.answer:
············facts = [randint(2,10),randint(2,10)]
········self.question = "Quanto fa {} per {}?".format(facts[0],facts[1])
········swapper = [randrange(-1,2,2),randrange(-1,2,2)]
········self.answer = facts[0] * facts[1]
········self.choice = [self.answer
············,self.answer+(facts[0]*swapper[0])
············,self.answer+(facts[1]*swapper[1])]
········shuffle(self.choice)
```

Il costruttore inizializza (per il momento a vuoti o azzerati) una serie di attributi: la domanda che sarà posta, la lista delle risposte possibili e il riferimento alla risposta esatta, cioè la sua posizione nella lista. Quindi, per avere una domanda già pronta, richiama il metodo `newQuest()`.
Quest'ultimo usa le funzioni di generazione dei numeri pseudocasuali per creare una coppia di numeri compresi tra 2 e 10. La domanda chiederà il risultato della moltiplicazione di questi due numeri. La scelta va fatta fra 3 opzioni, una è la risposta esatta, le altre 2 sono sbagliate e sono generate a partire dai medesimi due fattori con formule che cambiano in modo casuale ciascuno dei due per avere valori sbagliati (ma verosimili per chi sbaglia i conti...). Ci pensa poi la chiamata a `shuffle()` a scambiare in modo casuale le posizioni dell risposte, in modo che la risposta giusta non rimanga necessariamente nella prima posizione.
Qui emerge la differenza tra le due funzioni `randint()` e `randrange()` accennata in precedenza. La funzione `randint()` genera un numero a caso intero da un intervallo tra due numeri passati come argomenti; `randrange()` fa quasi lo stesso, ma richiede un terzo argomento come valore di separazione tra due possibili numeri generati dall'intervallo. Nel nostro caso si chiede un numero tra -1 e 2 (escluso, come sempre negli intervalli definiti in python), separato di 2 o un multiplo di 2 da ogni altro possibile valore. La scelta si riduce a -1 e 1 come possibili valori di ritorno.
Occorre notare una cosa sull'uso di `randrange()` in questo metodo: la funzione non prevede necessariamente numeri interi come valori di ritorno: se avessimo impostato 0.5 come valore di separazione, la scelta dei possibili valori di ritorno si sarebbe estesa a (-1, -0.5, 0, 0.5, 1, 1.5). Con 2 come separatore e l'intervallo (-1,2) siamo sicuri invece che:
1. i valori saranno interi
2. non sarà possibile ottenere 0.

```
····def getQuestText(self):
········if self.question == "":
············self.newQuest()
········return self.question

····def getOptions(self):
········if self.question == "":
············self.newQuest()
········return "[1]: {} - [2]: {} - [3]:
```

```
→ {}".format(self.choice[0],self.choice[1],self.choice[2])

    def getChoiceVal(self,btn):
        if btn in range(1,4):
            return choice[(btn-1)]
        else:
            return False

    def getAnswer(self):
        return self.answer
```

I metodi con prefisso get... servono a leggere dall'oggetto il testo della domanda (`getQuestText()`), il testo delle possibili risposte associate al tasto corrispondente di scelta (`getOptions()`), il valore effettivo di una risposta dato il suo tasto di selezione (`getChoiceVal()`), il valore della risposta esatta (`getAnswer()`).
Da notare che le posizioni delle risposte sulla lista mantenuta dall'oggetto corrispondono "quasi" ai tasti di scelta; va ricordato come sempre che python conta le posizioni sulle liste e tutto il resto cominciando sempre da 0, ma i tasti scelti sono 1, 2, 3 (per comodità). Così la prima risposta (tasto 1) è associata alla posizione 0, la seconda risposta alla posizione 1 e la terza alla posizione 3.

```
    def check(self,ans):
        if ans not in range(1,4):
            return False
        if self.choice[(ans-1)] == self.answer:
            return True
        else:
            return False
```

Infine, il metodo `check()` verifica se la posizione passata in argomento sia la risposta giusta o no.

OK, adesso cominciamo a scrivere la funzione principale.

```
def main(args):

    # inizializzazione num. pseuodcasuali
    seed()

    pygame.init()
    screen = pygame.display.set_mode((131,480))
    pygame.key.set_repeat(1,1)
    pygame.display.set_caption("question ball")
    backdrop = pygame.image.load('gfx/backdrop_slim.png')
    player = Ball("gfx/intro_ball.png")
```

Primo passo per un programma che deve generare numeri casuali è l'inizializzazione della generazione degli stessi: nelle note al termine c'è una breve spiegazione relativa al motivo per cui occorre farlo. Dopodiché il programma inizializza le strutture di della libreria PyGame, imposta dimensione e titolo della finestra che conterrà il nostro contatore del tempo (detto anche "generatore d'ansia"...), quindi carica le immagini per palla e sfondo.

```
    qst = Question()
    print(qst.getQuestText())
    print(qst.getOptions())
```

Le tre istruzioni generano la domanda da porre al giocatore, che viene scritta sul terminale assieme alle tre risposte possibili

```
    responses = 0
    quit = 0
    delay = 50 # passo std tra fotogrammi
```

Queste 3 variabili sono molto importanti per il funzionamento del programma: la prima tiene conto delle risposte esatte date in sequenza, la seconda decide il comportamento del programma in base al tempo che scorre e alla risposta data e l'ultima "dà il ritmo" allo scorrere del tempo a disposizione, perché decide ogni quanti millisecondi la palla deve fare un passo verso il basso.

```
    while quit == 0:
        screen.blit(backdrop, (0, 0))

        if player.y <= 319:
            for ourevent in pygame.event.get():
                if ourevent.type == pygame.QUIT or (ourevent.type ==
→ pygame.KEYDOWN and ourevent.key == pygame.K_q):
                    quit = 1
```

Da questo momento comincia il ciclo di Pygame che comanda l'animazione in attesa della risposta. Il ciclo si concluderà quando quit diventerà diversa da 0.
Dato che ad ogni passo va ridisegnato tutto lo schermo, la prima cosa è ridisegnare lo sfondo; poi un controllo sulla posizione della palla ci dirà se il tempo è scaduto o il giocatore può ancora rispondere.
Se c'è ancora tempo, il programma controlla se ci sono stati eventi. Se è stato premuto il tasto 'q', oppure è stata chiusa la finestra contatempo, il programma imposterà la chiusura immediata impostando quit a 1. Altrimenti si prosegue.

```
                if ourevent.type == pygame.KEYDOWN:
                    ans_list = {
                                pygame.K_1: 1
                                ,pygame.K_2: 2
                                ,pygame.K_3: 3
                    }
                    if ourevent.key in ans_list:
                                delay = 2000 # attesa per messaggi a schermo
                                if qst.check(ans_list[ourevent.key]) == True:
                                            # indovinato - ferma tempo
                                            print("Indovinato")
                                            responses += 1
                                            quit = 2 # in realtà serve segnalibro per innescare
→ azzeramento tempo e nuova domanda
                                else:
                                            # errore
```

```
                            print("Errore - la risposta era
→ {}".format(qst.getAnswer()))
                            quit = 1
```

Ancora, si controllano eventi sui tasti. Viene costruito un dizionario particolare, in cui le chiavi sono i riferimenti ai tasti premuti e i valori sono i corrispondenti riferimenti numerici alle risposte. Il fatto di usare riferimenti ai tasti non deve stupire più di tanto, in quanto non sono comunque oggetti.
Grazie a questo particolare dizionario possiamo condensare la logica di risposta del programma a poche righe: se l'evento rientra nelle chiavi usate dal dizionario (quindi è un tasto accettato come input) si allunga il tempo di attesa per il prossimo passo, così da mostrare la risposta in 2 secondi (questo farà comodo più avanti nella modifica del programma). Se la risposta scelta è quella giusta, si scrive il messaggio positivo, si incrementa il numero di risposte esatte e si imposta la varibile quit a 2: il programma uscirà dal ciclo, ma non col valore previsto per arrestarsi.
Se viceversa la risposta è sbagliata, sarà visualizzato il messaggio di errore e il ciclo sarà concluso per chiudere il programma.

```
            player.y +=2
        else:
            print("Tempo scaduto!")
            quit = 1
```

Può darsi che il giocatore non abbia ancora premuto un tasto in questo passo (del resto ogni passo normale del ciclo dura 50 millisecondi - mai visto qualcuno capace di leggere la domanda, scegliere la risposta e premere il tasto corrispondente entro il primo passo del ciclo...), la palla viene abbassata di due pixel.
Su questo gruppo di righe c'è anche l'altro ramo della condizione che valuta se il tempo era scaduto, scritta in precedenza. Le due istruzioni che semplicemente annunciano a schermo la fine del tempo e impostano la variabile quit per l'uscita si attivano quando la palla è arrivata alla fine della sua caduta, sulla puntina.

```
        player.render(screen)
        pygame.display.update()
        pygame.time.delay(delay)
        if delay != 50:
            delay = 50
```

Queste istruzioni concludono il lavoro del passo: realizzano il nuovo fotogramma con l'aggiornamento della posizione della palla, lo stampano a schermo e aspettano il tempo deciso dalla variabile delay. Se questa fosse diversa da 50 (è stato premuto un pulsante, quindi occorre aspettare che l'utente legga l'esito della sua scelta), viene riportata al suo valore previsto all'inizio del ciclo.

```
        if quit == 2:
            # msg congratulazioni, nuovo quesito e azzeramento tempo
            qst.newQuest()
            print(qst.getQuestText())
            print(qst.getOptions())
            player.y = 0
```

```
            quit = 0
```

Ma c'è ancora una cosa da fare: se quit è uguale a 2, vuol dire che il giocatore ha indovinato, e ha diritto a un'altra domanda. Quindi sono richiamate tutte le istruzioni ce riportano il programma alle condizioni di partenza: generazione di una nuova domanda e delle possibili risposte, visualizzazione a schermo di tutto, palla riportata in cima per azzerare il timer, variabile quit azzerata per ripetere il ciclo.

```
    if responses > 0:
        if responses > 1:
            lastchar = "e"
        else:
            lastchar = "a"
        print("hai risposto esattamente a {}
→ domand{}".format(responses,lastchar))
        pygame.time.delay(3000)
    return 0
```

Che succede dopo che il programma è uscito dal ciclo? Siamo in fase di chiusura, ma se il giocatore è stato capace di rispondere ad almeno una domanda è giusto dargli un riconoscimento. Così si esamina il contenuto della variabile `responses`: se è maggiore di uno sarà scritto un messaggio, in accordo al numero - così il messaggio non litigherà con la lingua italiana.

```
if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
```

Le ultime istruzioni sono fuori dalla funzione principale, e servono semplicemente a richiamarla come prima operazione, facendo in modo da chiudere il programma alla sua uscita.

2. Una variante: messaggi in finestra

Il programma è già usabile così, ma ha un problema: ha bisogno di essere lanciato da riga di comando per poter scrivere messaggi a schermo. Sarebbe meglio poter visualizzare i messaggi direttamente nella finestra dove è contenuta la palla in caduta.
Per far questo aumenteremo la dimensione della finestra, aggiornando lo sfondo e la posizione della palla, e introdurremo delle modifiche per scrivere i nostri messaggi in grafica.

Le modifiche sono mostrate sul codice fin qui ottenuto, quindi conviene salvarne una copia a parte, ad esempio col nome `questionball_testo.py` e salvare le modifiche in `questionball_grafica.py`

Iniziamo dalla classe Ball.

```
class Ball:

    def __init__(self,filename):
```

```
        self.x = 520
        self.y = 0
    ....
```

L'unica modifica realmente necessaria, come già detto, è quella di spostare la palla più a destra, nella finestra che diventerà più grande.
Sulla classe `Question` le modifiche saranno più corpose, perché la classe si occupa anche dei messaggi a schermo

```
class Question:

    def __init__(self,screen):
        self.question = ""
        self.answer = 0
        self.choice = [0,0,0]
        self.font = pygame.font.Font('gfx/Oxygen-Sans-Bold.ttf',20)
        self.screen = screen
        self.newQuest()

    ....
```

Il metodo `init__()` ha un argomento che prima non c'era: si tratta di un oggetto screen che farà da riferimento alla finestra dove mostrare i messaggi. Questo oggetto è un attributo della classe così come lo è il font scelto per mostrare i messaggi a schermo.

```
    ....

    def printMsg(self,msg,posx,posy):
        if msg == "" or posx < 0 or posy < 0:
            return False
        msg_text = self.font.render(msg,True,(255,255,255))
        msg_rect = msg_text.get_rect()
        msg_rect.centerx = posx
        msg_rect.y = posy
        self.screen.blit(msg_text,msg_rect)
        pygame.display.update(msg_rect)

    ....
```

La classe prevede il nuovo metodo printMsg(), che come dice il suo nome si occupa della scrittura di un messaggio sullo schermo alle coordinate indicate. Oltre al messaggio vero e proprio, prevede anche l'indicazione delle coordinate alle quali scriverlo. Attenzione, la coordinata x è riferita al centro del messaggio: questo ci aiuterà a posizionare in modo ordinato tutte le scritte.

La funzione principale è quella che prevede i maggiori cambiamenti, legati al nuovo modo di mostrare i messaggi.

```
def main(args):
```

```
    # inizializzazione num. pseuodcasuali
    seed()

    pygame.init()
    screen = pygame.display.set_mode((640,480))
    pygame.key.set_repeat(1,1)
    pygame.display.set_caption("question ball")
    backdrop = pygame.image.load('gfx/backdrop.png')
    player = Ball("gfx/intro_ball.png")

    qst = Question(screen)

    responses = 0
    quit = 0
    delay = 50 # passo std tra fotogrammi
```

Cambia innanzitutto la dimensione della finestra, che prima conteneva solo la palla e la puntina e adesso contiene anche tutti i messaggi. Cambia anche l'immagine di sfondo, ma soprattutto spariscono le istruzioni che scrivono sulla linea di comando i messaggi, perché non servono più

```
    while quit == 0:
        screen.blit(backdrop, (0, 0))
        msg = qst.getQuestText()
        qst.printMsg(msg,260,5)
        msg = qst.getOptions()
        qst.printMsg(msg,260,25)

        if player.y <= 319:

....
```

A differenza della versione precedente, i messaggi relativi a domanda e lista risposte vanno scritti all'interno del ciclo principale, perché ad ogni passo la finestra viene completamente ridisegnata e nella vecchia posizione le scritte apparirebbero solo nel primo fotogramma.

```
....

                        if qst.check(ans_list[ourevent.key]) == True:
                            # indovinato - ferma tempo
                            qst.printMsg("Indovinato",260,55)
                            responses += 1
                            quit = 2
                        else:
                            # errore
                            msg = "Errore - la risposta era
→ {}".format(qst.getAnswer())
                            qst.printMsg(msg,260,55)
                            quit = 1

....
```

Anche in questo caso si sostituiscono i messaggi di errore sulla linea di comando con chiamate al

metodo di stampa messaggi sullo schermo del gioco.

```
....

        else:
            player.bang("gfx/bang.png");
            msg = "Tempo scaduto!"
            qst.printMsg(msg,260,55)
            quit = 1

....

        if quit == 2:
            # msg congratulazioni, nuovo quesito e azzeramento tempo
            qst.newQuest()
            msg = qst.getQuestText()
            qst.printMsg(msg,260,5)
            player.y = 0
            quit = 0

    if responses > 0:
        if responses > 1:
            lastchar = "e"
        else:
            lastchar = "a"
        msg = "hai risposto esattamente a {}
→ domand{}".format(responses,lastchar)
        qst.printMsg(msg,260,105)
        pygame.time.delay(3000)
    return 0
```

Idem come sopra. Il resto del codice non ha bisogno di modifiche.

3. Note finali sul codice

(e proposte di cambiamento)

Il codice presentato è di per sé autosufficiente, tuttavia, una volta scritto potrebbe venir voglia di migliorarlo. Un possibile punto di partenza è rappresentato dal metodo `newQuest()` della classe `Question`.

Se si presta attenzione si scopre che l'intera classe è costruita per essere indipendente da questo metodo, pur essendo proprio lui la parte di codice più importante del progetto: con `newQuestion()` si possono generare la domanda e l'insieme delle tre risposte da proporre al giocatore. Ed è tutto quello che serve alla classe.

Se sostituissimo il metodo con un altro codice che, semplicemente, imposta la domanda e le risposte seguendo un qualsiasi altro criterio (ad esempio, impostando la domanda "Di che colore era il cavallo bianco di Garibaldi" e le tre risposte "bianco","rosso","verde") la classe, e conseguentemente il programma, continuerebbero a funzionare indisturbati.
Allora un buon esercizio potrebbe essere cambiare il tipo di domanda, ad esempio impostando una diversa operazione matematica e un diverso modo di ricavare le risposte false. O addirittura

cambiare genere, costruendo una lista di brevi parole e proponendone gli anagrammi (suggerimento: un anagramma facile da ottenere è il riordino alfabetico delle lettere che compongono la parola), chiedendo al giocatore di individuare la parola giusta da una lista di tre. In questo caso le parole false potrebbero essere costruite a partire dal cambio di una lettera nella risposta giusta.

Appendice. Un piccolo approfondimento sui numeri casuali

La generazione dei numeri da parte del computer non è proprio casuale: la funzione usata di solito applica ad ogni chiamata un'operazione matematica al risultato della chiamata precedente. Se l'operazione è ben fatta (di solito moltiplicazioni o divisioni tra numeri sul cui risultato si opera una funzione chiamata congruenza matematica per ottenere un valore in un intervallo definito), la sequenza dei valori ottenuti è indistinguibile da una sequenza realmente casuale. Il problema è che se si parte sempre dallo stesso numero, la sequenza ottenuta è sempre la stessa (la matematica non è un'opinione...). Ecco perché di solito si imposta un seme (seed in inglese) per inizializzare ogni volta la sequenza a un valore diverso. Da quando sui computer esiste un orologio interno si usa il numero di millisecondi trascorsi da un dato evento, come il tempo trascorso dall'accensione del computer o dalla mezzanotte.

A volte però le cose diventano serie, quando serve una chiave numerica *veramente* casuale. Esempi di questo tipo sono le comunicazioni riservate tra banche, o i messaggi segreti per motivi di privacy o (contro)spionaggio. In questo caso serve un qualche meccanismo più raffinato dei nostri algoritmi. Da qualche tempo i processori hanno al loro interno dei circuiti pensati per generare numeri con una casualità molto più alta di quella raggiungibile con gli algoritmi più comuni; ma anche questi potrebbero essere forzati da metodi matematici applicati su potenti computer, anche se ci vorrebbero decenni di calcoli già per i metodi di cifratura più semplici.

L’ultima frontiera