Atti del X Convegno Nazionale di Didattica della Fisica e della Matematica DI.FI.MA. 2021

Apprendimento laboratoriale in Matematica e Fisica in presenza e a distanza

Torino, 11-12-13 ottobre 2021 - online

2001-2021
Il convegno del ventennale

A cura di:
Raffaella Bonino
Daniela Marocchi
Marta Rinaudo
Marina Serio

UNIVERSITÀ DEGLI STUDI DI TORINO

Apprendimento laboratoriale in Matematica e Fisica in presenza e a distanza
Atti del X Convegno Nazionale di Didattica della Fisica e della Matematica, DI.FI.MA. 2021
A cura di R. Bonino, D. Marocchi, M. Rinaudo, M. Serio

Responsabile del convegno: Ornella Robutti
Responsabili scientifici: Giulia Bini, Alessio Drivet, Matteo Leone, Tommaso Marino, Daniela Marocchi, Ornella Robutti, Cristina Sabena, Ada Sargenti, Marina Serio, Germana Trinchero

Esperti Tecnici : Tiziana Armano e Filippo Cosma Liardi

Coordinamento rapporti con le scuole : Daniela Truffo (Città Metropolitana di Torino, CE.SE.DI)

Collane@unito.it
Università degli Studi di Torino
ISBN: 9788875902292

Disegno grafico: Maria Grazia Imarisio
Immagine di copertina: rielaborazione grafica di Elisa Gentile, collage di Marina Serio

DALLA COSTRUZIONE DI UN VIDEOGIOCO AGLI ALGORITMI DECISIONALI DI SCELTA

Laura Lamberti[1], Francesca Tovena[2]
[1]Liceo Scientifico Augusto Righi, Roma
[2]Dipartimento di Matematica, Università degli Studi di Roma "Tor Vergata", Roma
lamberti.laura@gmail.com

Abstract
L'articolo descrive l'attività svolta in una classe seconda di liceo scientifico durante le ore di didattica a distanza (DAD): nata come progetto conclusivo del percorso di informatica svolto durante il biennio, l'attività ha favorito la riflessione su numerosi argomenti di matematica e sulle strategie del pensiero.
Seguendo, in particolare, le indicazioni di Bright et al. (1985), la proposta didattica si basa su un gioco, che svolge il ruolo di strumento didattico per coinvolgere attivamente e appassionare i ragazzi.
Il gioco utilizzato, Mastermind, si presta a essere progettato e programmato in linguaggi informatici; esso consente di introdurre elementi di insiemistica, logica e combinatoria e ne favorisce la sperimentazione. La ricerca delle strategie vincenti permette di parlare di algoritmi di scelta decisionali.
Il gioco coinvolge due giocatori: uno dei due giocatori sceglie un codice formato da una sequenza di quattro colori scelti da un insieme di sei. Il secondo giocatore ha l'obiettivo di indovinare il codice in un numero limitato di mosse; per ciascun tentativo riceve una risposta che lo aiuta.
L'attività didattica è stata finalizzata alla progettazione e all'implementazione in Python di un videogioco basato sulle regole di Mastermind. Sono state considerate due versioni del videogioco: la prima è quella in cui il codice segreto è scelto dal pc, mentre la seconda prevede che sia il pc ad indovinare il codice segreto scelto dal giocatore. Questa ultima versione è molto più interessante dal punto di vista algoritmico e didattico; inoltre, a differenza della prima, non è così diffusa sulla rete. Tale versione viene indicata con il nome *reverse*, perché in essa i ruoli sono invertiti.
Il laboratorio proposto prevede una prima fase di gioco online, che favorisce la riflessione sulle strategie di gioco: si evidenziano le competenze di matematica necessarie, si esplicita una descrizione delle strategie di risposta utilizzate inizialmente in modo intuitivo e si analizzano le procedure che determinano la scelta dei tentativi. Questa fase è fondamentale per la comprensione dell'algoritmo risolutivo della versione *reverse*. Viene infine presentato, discusso e implementato l'algoritmo di scelta minimax proposto una decina di anni or sono da Knuth.
In classe, gli studenti hanno prima progettato il videogioco in cui il codice è scelto dal pc; la codifica in Python ha fornito spunti per introdurre molti elementi di informatica e rafforzare concetti di matematica. Poi, in un secondo momento è stata implementata la versione *reverse*.
La possibilità di lavorare ciascuno dal proprio pc ha favorito la riuscita del progetto in regime DAD, più che se si fosse svolto in presenza; seppur da remoto, non è mancata l'interazione tra studenti e la produzione del progetto è stata organizzata in gruppi di lavoro.

Parole-chiave
Mastermind, Coding, Python, Combinatorica, Algoritmi Minimax

MASTERMIND COME GIOCO MATEMATICO

Il valore pedagogico dell'utilizzo del gioco come strumento di apprendimento è ampiamente riconosciuto (Bright et al., 1985); in particolare, vari autori hanno evidenziato che i giochi a base matematica si inseriscono efficacemente in un contesto di apprendimento per tentativi ed errori e promuovono l'apprendimento disciplinare attraverso il fare e la partecipazione attiva ed empatica da parte degli studenti (Tokac et al., 2019). Inoltre, l'uso consapevole del gioco viene ritenuto uno

strumento didattico efficace anche nello stimolare competenze trasversali quali la creatività e l'attitudine alla sperimentazione, favorendo la capacità di mantenere a lungo la concentrazione.
Si ritiene, inoltre, che tale strumento possa contribuire a ridurre l'insorgere di reazioni di rifiuto e di ansia nei confronti della matematica, tramite un coinvolgimento spontaneo e volontario da parte degli studenti; nell'ambito del gioco, gli errori compiuti possono più facilmente essere discussi e corretti tra pari. La propria strategia di gioco viene, in modo spontaneo, condivisa e motivata, stimolando così le capacità di verbalizzazione e argomentazione, insieme all'uso corretto del linguaggio tecnico.
Più specificamente, il gioco Mastermind è stato sperimentato da Mitchell (1999), Strom & Barolo (2011), Fiore et al. (2018), che hanno messo in evidenza come sia possibile utilizzarlo in modo flessibile e modulato per difficoltà crescenti; inoltre, tale gioco sostiene lo sviluppo del pensiero deduttivo e strategico, incoraggiando la formulazione di ipotesi e la loro verifica. In (Lamberti & Tovena, 2021) sono stati trattati gli aspetti combinatorici e la loro applicazione ad un apprendimento disciplinare in lingua inglese, mentre, nella presente proposta, l'intento di programmare un videogioco favorisce una impostazione algoritmica e sollecita competenze informatiche.

Le regole del gioco
Il gioco Mastermind, inventato da Mordechai Meirovitz, è stato commercializzato con successo nel 1971 da Invicta Plastics come gioco da tavolo. Attualmente, sono accessibili gratuitamente versioni online.
Si gioca tra due giocatori: il *codificatore* sceglie e tiene segreto un codice, mentre il *decodificatore* cerca di indovinarlo e ha a disposizione un numero limitato di tentativi. Il decodificatore vince se indovina il codice segreto entro il numero massimo di tentativi. In caso contrario, vince il codificatore.
Il codice segreto è costituito da una sequenza ordinata di elementi. Nella versione commerciale del gioco, il codice consiste in 4 chiodini colorati, ognuno dei quali è selezionato da un insieme di 6 colori diversi. Le ripetizioni sono permesse. Il decodificatore dichiara il proprio tentativo mettendo i chiodini colorati in una fila di una tavoletta perforata. Il codificatore confronta il tentativo con il codice segreto e risponde aggiungendo dei chiodini bianchi e/o neri più piccoli, in base a regole note a entrambi:

1. il codificatore calcola inizialmente quanti chiodini del tentativo hanno colore e posizione uguale a quelli del codice; nella risposta, inserisce un chiodino nero per ognuno di questi elementi.
2. in seguito, il codificatore non prende più in considerazione le posizioni già conteggiate e confronta le parti rimanenti del codice e del tentativo; nella risposta, inserisce un chiodino bianco per ciascun un elemento del tentativo che abbia il colore giusto ma che non si trovi nel posto corretto, facendo attenzione a non contare nessun chiodino del tentativo più di una volta.
3. la posizione dei chiodini nella risposta non è correlata a quella dei chiodini colorati corrispondenti.

Nell'esempio della figura 1, il codice segreto è nella prima riga (e non è visibile al decodificatore), e il tentativo è mostrato nella seconda riga. Per facilitare la lettura, sono state inserite le iniziali dei colori. Il chiodino nero nella risposta corrisponde alla corrispondenza dei chiodini rossi nella quarta posizione; il chiodino rosso nella prima posizione del tentativo non concorre alla risposta, perché il codice segreto non contiene altri rossi oltre a quello già considerato. I due chiodini bianchi nella risposta corrispondono al chiodino verde e a quello blu nel tentativo (che hanno colore uguale a un chiodino nel codice segreto, ma la posizione errata). Il chiodino blu nel tentativo concorre solo con un chiodino bianco nella risposta, anche se nel codice segreto ci sono due blu, perché ogni chiodino può essere considerato solo una volta.

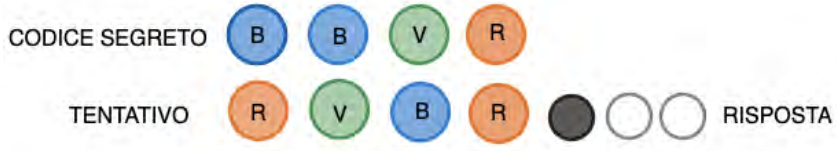

Figura 1. Esempio di codice segreto, tentativo e risposta in Mastermind.

Verso una formalizzazione matematica
L'esperienza di gioco illustra come, al di là di colpi di fortuna estemporanei, l'efficacia dei tentativi effettuati dal decodificatore dipende dalla capacità di utilizzare al meglio le risposte ottenute.

Il decodificatore conosce solo il proprio tentativo e la risposta: è da essi che deve partire e trarre indizi utili nella scelta del tentativo successivo. Riprendiamo l'esempio in Figura 1 e osserviamo che il decodificatore sa che il codice segreto è una quaterna ordinata nei sei colori che, confrontata con il suo tentativo RVBR, riceve la risposta *1 nero e 2 bianchi*; in particolare, la quaterna RRBV sicuramente non è il codice segreto, perché coincide con il tentativo RVBR in due posizioni e non una sola. Infatti, la risposta registra proprietà comuni al codice segreto e al tentativo: è dunque una proprietà della coppia non ordinata *codice-tentativo* e non cambia scambiando i ruoli del codice segreto e del tentativo.
Nell'esempio in Figura 1, il decodificatore può quindi considerare l'insieme di tutte le quaterne ordinate che ricevono la risposta *1 nero e 2 bianchi* nel confronto con il tentativo RVBR (che sono dette *compatibili con la risposta ottenuta*): tale insieme contiene sicuramente il codice segreto. Knuth (1976-1977) suggerisce di effettuare il tentativo successivo scegliendolo nell'insieme delle quaterne compatibili. La risposta al secondo tentativo permetterà di restringere ulteriormente l'insieme delle quaterne compatibili, imponendo un ulteriore vincolo di compatibilità. In questo modo, si individua in ogni caso una strategia che permette di individuare il codice segreto.
Il problema da risolvere, ora, è che la precedente strategia potrebbe richiedere più tentativi di quanto siano permessi dal gioco. Occorre quindi cambiare approccio o individuare un criterio efficace di scelta del tentativo da effettuare: nel lavoro in classe si è optato per questa seconda opzione. Seguendo nuovamente le indicazioni di Knuth (1976-1977), una misura di efficacia per uno specifico tentativo è legata alla sua capacità di ridurre il numero di quaterne compatibili. Per ogni tentativo ammissibile, non disponendo ancora della risposta, occorre quindi tenere in considerazione tutte le possibili risposte: si inizia facendo l'elenco delle risposte che in teoria si possono ricevere; poi, per ciascuna risposta, si conta il numero di quaterne che (nel confronto con il tentativo che stiamo valutando) sono compatibili.
Osserviamo che ogni tentativo individua una *partizione* dell'insieme delle quaterne, cioè lo suddivide in sottoinsiemi disgiunti, ciascuno dei quali è caratterizzato da una risposta: se uno di tali sottoinsiemi è molto grande, si rischia (nel caso sfortunato) di ridurre poco l'insieme compatibile. L'algoritmo *minimax* proposto da Knuth sceglie il tentativo che assicura, nel peggiore dei casi, il miglior risultato.
Per codificare questo algoritmo, i ragazzi:

- iniziano elencando le possibili risposte,
- per ogni fissato tentativo, contano, per ciascuna risposta, quanti sono le quaterne compatibili e tengono memoria dell'opzione peggiore
- scelgono il tentativo minimizzando l'opzione peggiore.

Per elencare le risposte, è risultato efficace rappresentarle come coppie ordinate numeriche (n_{neri}, $n_{bianchi}$), indicando con n_{neri} il numero di chiodini neri e con $n_{bianchi}$ il numero di chiodini bianchi. Con questa convenzione, le risposte possibili sono raffigurate nella Figura 2.

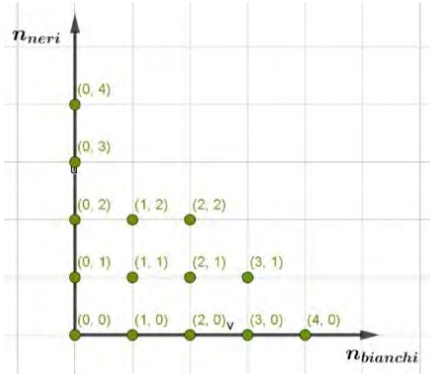

Figura 2. Rappresentazione delle possibili risposte.

ATTIVITÀ DIDATTICA: PARTE 1

L'attività è stata svolta a distanza in una classe seconda di liceo scientifico, a completamento del percorso di informatica. Dopo aver introdotto le regole del gioco di Mastermind, aver sperimentato

alcune partite e aver discusso le strutture matematiche che ne permettono la formalizzazione, si è proceduto alla progettazione del software che simulasse una partita: il linguaggio di programmazione scelto è stato Python. L'attività di coding ha realizzato due differenti versioni del gioco: nella prima (denominata dai ragazzi *Indovina Tu*) la scelta del codice segreto viene fatta dal calcolatore, mentre nella seconda (*Indovina il PC*) il calcolatore deve indovinare.

Implementazione del codice nella versione *Indovina Tu* in cui il giocatore indovina il codice

La scelta del codice segreto viene fatta in modo random dal PC. Il computer seleziona una tra le possibili disposizioni con ripetizione di quattro colori scelti da un insieme di sei utilizzando la funzione `choice` della libreria `random`.

Il software ha due compiti: scegliere il codice segreto e fornire le risposte ai tentativi inseriti dal giocatore, dopo averli confrontati con il codice scelto.

Per costruire le risposte è stata definita una funzione `confronta`, cuore del programma, che restituisce la risposta da dare al giocatore a fronte del tentativo inserito. È stata inoltre aggiunta la funzione `pulisci`, non strettamente necessaria, ma molto interessante, in quanto fornisce una indicazione sull'efficacia del tentativo effettuato.

La descrizione e la spiegazione puntuale delle due funzioni sono riportate nei paragrafi successivi.

La progettazione del codice è stata discussa in classe; dalla discussione è nata una sorta di lista di "ingredienti" necessari alla scrittura del programma:

- una struttura dati che contenga l'elenco dei colori `colours[ ]`
- una struttura dati che contenga il codice segreto `code[ ]`
- una struttura dati che contenga le disposizioni `set_disp[ ]`
- una sezione di input e una struttura dati che contenga l'input `guess[ ]`
- una funzione che scelga il codice segreto `choice` (dalla libreria `random`)
- una funzione che conti le concordanze tra il codice segreto e il tentativo `confronta`
- una struttura che contenga la coppia risposta `(n_neri, n_bianchi)`
- un contatore che conti il numero di tentativi effettuati `conta_tentativi`

Dopo una prima fase di sperimentazione è stata costruita una interfaccia grafica: la libreria utilizzata è stata `pygame`. La costruzione dell'interfaccia ha richiesto molto tempo; pur non sollecitando l'introduzione della formalizzazione matematica, è stata comunque utile. A qualsiasi livello sia richiesta, l'attività di coding risulta piuttosto complessa, e quindi non tutti gli alunni riescono ad ottenere risultati soddisfacenti. Una parte della classe che non era stata particolarmente coinvolta nella implementazione delle funzioni e nella realizzazione del codice ha partecipato comunque con entusiasmo alla progettazione e costruzione dell'interfaccia grafica.

Il cuore del codice: la funzione `confronta`

La funzione `confronta` produce la risposta corrispondente a ciascun tentativo del giocatore. La risposta, la coppia (n_{neri}, $n_{bianchi}$), è generata dal confronto tra il codice segreto memorizzato nell'array `code` e il tentativo immesso nell'array `guess`.

Gli elementi (i colori) uguali e allo stesso posto sia nel `code` che nel `guess` sono sostituiti in entrambi gli array con la stringa `"PEG!"`, viceversa, gli elementi uguali sia nel `code` che nel `guess` ma in posizioni differenti sono sostituiti, questa volta solo nel `code`, con la stringa `"PEG!"`: tali sostituzioni evitano di contare più volte, in modo errato, le corrispondenze `code`–`guess` e quindi producono la corretta coppia (n_{neri}, $n_{bianchi}$) in risposta.

Una volta terminato il confronto, vengono ripristinati i valori iniziali degli elementi dei due array `code` e `guess` e restituita la risposta. Qui di seguito è riportato il listato:

```
def confronta(guess, code):
        n_neri = 0
        n_bianchi = 0
        for i in range(4):
```

```
            if guess[i] == code[i]:
                  n_neri += 1                        #aumento i chiodini neri
                  guess[i] += "PEG!"
                  code[i] += "PEG!"
      for i in range(4):
           if guess[i] in code and guess[i] != code[i]:
                  n_bianchi += 1             #aumento i chiodini bianchi
                  code[code.index(guess[i])] += "PEG!"
      for i in range(4):
          if len(code[i]) > 1:
              code[i] = (code[i])[0]
              guess[i]=(guess[i])[0]
      return n_neri, n_bianchi
```

Uno strumento per analizzare l'efficacia del tentativo: la funzione `pulisci`

Per studiare l'efficacia di un tentativo, è necessario che il PC possa valutare il numero di disposizioni compatibili con le risposte ricevute in precedenza. Tale conteggio svolto dalla funzione `pulisci`.
Indichiamo con $D_{6,4}$ il numero delle disposizioni con ripetizione di 4 elementi estratti da 6, elencate in `set_disp`. La funzione `pulisci` chiama la funzione `confronta` per ottenere la risposta (n_{neri_cod}, $n_{bianchi\_cod}$) relativa al confronto `code–guess`. Successivamente, prende in considerazione ogni elemento `set_disp[j]`, (j = 1, 2, …, $D_{6,4}$) dell'insieme di tutte le possibili disposizioni. Chiama `confronta` per ricevere la risposta relativa alla coppia `set_disp[j]–guess`. Se tale risposta è differente da (n_{neri_cod}, $n_{bianchi\_cod}$), allora l'elemento `set_disp[j]` non può essere il codice segreto.
Le disposizioni che non forniscono la risposta (n_{neri_cod}, $n_{bianchi\_cod}$) della `code – guess` e che quindi non possono essere il codice segreto, vengono memorizzate in un array `elimino`**.**
L'ultimo compito della funzione `pulisci` è quello di ripulire l'insieme `set_disp` dagli elementi di `elimino`. Nelle fasi successive, l'analisi sarà limitata alle rimanenti disposizioni compatibili.
La differenza tra la cardinalità di `set_disp` prima e dopo l'azione della funzione `pulisci` indica l'efficacia del tentativo `guess` effettuato.

```
def pulisci(set_disp, tentativo, codice):
      elimino=[]
      n_neri_cod=confronta(tentativo,codice)[0]
      n_bianchi_cod=confronta(tentativo,codice)[1]

      if (n_neri_cod==0 and n_bianchi_cod==0):
             for j in range (0,len(set_disp)):
                    for i in range(4):
                           if (set_disp[j][i] in tentativo):
                                  if (set_disp[j] not in elimino):
                                      elimino.append(set_disp[j])
                                      break

     for j in range (0,len(set_disp)):
             for i in range(4):
                    if (confronta(set_disp[j],tentativo)[0]!= n_neri_cod \
                    or confronta(set_disp[j],tentativo)[1]!= n_bianchi_cod):
                        if (set_disp[j] not in elimino):
                             elimino.append(set_disp[j])
                             break

      for item in elimino:
             set_disp.remove(item)
      if tentativo in set_disp:
             set_disp.remove(tentativo)
      elimino=[]
      return
```

ATTIVITÀ DIDATTICA: PARTE 2

Implementazione del codice nella versione reverse *Indovina il PC* in cui il PC indovina il codice
Lo scambio dei ruoli rende il gioco sicuramente meno attraente dal punto di vista ricreativo, ma molto più interessante dal punto di vista matematico e logico. La necessità di formalizzare e codificare la strategia risolutiva costringe gli studenti a riflettere sulle procedure risolutive intuitivamente seguite. In breve tempo tutti si rendono conto che la formalizzazione matematica è la strada vincente per determinare l'algoritmo risolutivo. Come premesso, la strategia risolutiva implementata è quella indicata in (Knuth, 1976-1977) basata sull'algoritmo minimax. Il programma è stato integrato con le istruzioni dedicate al funzionamento del minimax e con la funzione `punteggio` che conteggia il numero di disposizioni compatibili con il `code` e non ancora utilizzate nei tentativi.

Funzione `punteggio`
Per valutare un fissato tentativo, la funzione `punteggio` chiama la funzione `confronta` per ricevere, per ciascuna disposizione, la risposta (n_{neri}, $n_{bianchi}$) relativa alla coppia formata dal tentativo e dalla disposizione scelta.
La funzione `punteggio` conteggia, per ogni risposta, il numero di disposizioni che generino quella risposta e registra tale numero nell'array `occurrences`.
Infine, determina e restituisce il massimo dei numeri memorizzati.

```
def punteggio (item, set_disp):
      max_occur=0
      occurrences=[0]
      for j in range (len(set_disp)):
            n_bianchi=confronta(item,set_disp[j])[0]
            n_neri=confronta(item,set_disp[j])[1]
            t=tuple(n_bianchi, n_neri)
            occurrences[t] +=1
      max_occur= max(occurrences.values())
      return max_occur
```

L'algoritmo minimax
Dato che lo scopo del giocatore è quello di vincere il gioco nel numero minore di mosse, la sua strategia sarà diretta alla costruzione di sottoinsiemi compatibili in cui scegliere i tentativi, sottoinsiemi che siano di cardinalità via via minore.
Il primo tentativo determina la partizione iniziale in classi. Per ciascuna partizione *è necessario valutare la cardinalità K del sottoinsieme più numeroso (caso peggiore)*. Poiché ogni partizione dipende dal tentativo iniziale, l'algoritmo sceglie il tentativo che produce *il minimo valore di K.*
In questo modo il numero dei tentativi da sperimentare sarà il minimo possibile, restando comunque certi che anche il codice segreto è compreso tra di loro. La soluzione è raggiunta con certezza in un numero finito di passi.
Il procedimento è iterativo. Ad ogni mossa il tentativo da scegliere deve essere paragonato con tutti gli altri possibili, in modo da scegliere quello che garantisca nuovamente che il caso peggiore sia il meno numeroso.

Implementazione dell'algoritmo minimax:
Qui di seguito è riportato un estratto delle istruzioni relative all'algoritmo minimax.
win è una variabile booleana che diventa true se si indovina il codice segreto; l'algoritmo viene iterato fino a quando non si raggiunge la soluzione del gioco.

```
while not win :
     if (conta_tentativi==1):
         tentativo=["A", "A", "B", "B"]
     else:
         set_disp0.remove(tentativo)   #elimino il tentativo che sto per utilizzare
```

```
Min_scores=1296
best_guesses = []
for j in range(len(set_disp0)):
        scores[j]=int(punteggio(set_disp0 [j],set_disp))
                                              # la punteggio trova i casi peggiori
        if (scores[j]<Min_scores):
           Min_scores=scores[j] #trova punteggio minore tra i casi peggiori
for j in range(len(set_disp0)):
    if (scores[j]==Min_scores):
             best_guesses.append(set_disp0 [j])
                   #inserisco in best_guesses i migliori tra i peggiori

tentativo = None
for item in set_disp:
    if item in best_guesses:
        tentativo = item
        break
```

Scelta del tentativo iniziale

La funzione `pulisci` è utilizzata anche per valutare il valore strategico dei tentativi da sperimentare. A tal fine, le $D_{6,4}$ disposizioni vengono suddivise in 5 tipologie: la prima è quella delle disposizioni che hanno i 4 elementi di colori differenti (come rappresentante si può considerare ABCD), la seconda quella delle disposizioni in cui è presente una sola coppia (AABC), la terza due coppie (AABB), la quarta una terna (AAAB) e infine la quinta tutti gli elementi dello stesso colore (AAAA). Per motivi di simmetria nella distribuzione delle disposizioni, gli elementi di una stessa tipologia mostrano lo stesso comportamento e producono partizioni equivalenti.
La funzione `pulisci` permette di calcolare, per ciascuna delle cinque tipologie e per ogni possibile risposta, la cardinalità dei sottoinsiemi compatibili.
Nelle tabelle 1-5 sono riportati le cardinalità di ciascun sottoinsieme della partizione formata, al variare del primo tentativo.

Tabelle 1-5. Cardinalità dei sottoinsiemi compatibili dell'insieme di tutte le $D_{6,4}$ possibili disposizioni generati a partire da un primo tentativo indicato in alto a sinistra (**ABCD, …**) in funzione delle risposte ottenute (n_{neri}, $n_{bianchi}$), con n_{neri} indice di riga e $n_{bianchi}$ indice di colonna

ABCD	**0**	**1**	**2**	**3**	**4**
0	16	152	312	136	9
1	108	252	132	8	0
2	96	48	6	0	0
3	20	0	0	0	0

AABC	**0**	**1**	**2**	**3**	**4**
0	81	276	222	44	2
1	182	230	84	4	0
2	105	40	5	0	0
3	20	0	0	0	0

AABB	**0**	**1**	**2**	**3**	**4**
0	0	256	96	16	1
1	256	208	36	0	0
2	114	32	4	0	0
3	20	0	0	0	0

AAAB	**0**	**1**	**2**	**3**	**4**
0	256	308	61	0	0
1	317	156	27	0	0
2	123	24	3	0	0
3	20	0	0	0	0

AAAA	**0**	**1**	**2**	**3**	**4**
0	625	0	0	0	0
1	500	0	0	0	0
2	150	0	0	0	0
3	20	0	0	0	0

In accordo con l'algoritmo minimax anche il tentativo iniziale deve essere scelto in modo da produrre la massima riduzione del numero di possibili disposizioni compatibili. Nella tabella seguente sono stati sintetizzati i casi peggiori per ogni tipologia di tentativo: la seconda colonna rappresenta i valori K massimi. Si nota come il sottoinsieme meno numeroso si ottiene partendo con un tentativo formato da una doppia coppia. Nella versione *Indovina il PC* il primo tentativo è quindi fissato al valore AABB.

Tabella 6. Cardinalità K dei sottoinsiemi più numerosi (caso peggiore) in funzione della tipologia del primo tentativo e delle risposte (n_{neri}, $n_{bianchi}$) relative a questi casi

Tipologia del primo tentativo	n° massimo di disposizioni compatibili	(n_{neri}, $n_{bianchi}$)
ABCD	312	(0,2)
AABC	276	(0,1)
AABB	**256**	(1,0), (0,1)
AAAB	317	(1,0)
AAAA	625	(0,0)

CONCLUSIONI

L'attività laboratoriale presentata si presta molto efficacemente a facilitare l'introduzione o l'approfondimento di argomenti di matematica quali la combinatoria e l'insiemistica e all'utilizzo di alcuni strumenti di informatica.
Mentre gli aspetti didattici relativi agli aspetti combinatorici sono stati descritti in (Lamberti L. et al, 2021), in questo articolo si sono voluti evidenziare i contenuti algoritmici mettendo in luce le funzioni e le strutture dati utilizzate per codificare la procedura risolutiva. Occorre sottolineare che il laboratorio è stato offerto ad una classe seconda di liceo scientifico, alla fine di un percorso di studio dell'informatica in Python durato due anni: tutte le istruzioni utilizzate, ad eccezione di quelle relative alla grafica, erano ben note agli studenti già prima della progettazione del codice del videogioco.
La versione *reverse Indovina il Pc* non è molto diffusa in rete e questo forse fornisce un valore aggiunto all'attività; la sua implementazione, inoltre, ha permesso di esplorare in profondità la logica necessaria e intuitivamente già utilizzata dai ragazzi per scoprire il codice segreto.
Il percorso didattico ha coinvolto i ragazzi su vari livelli: gli algoritmi, sicuramente la parte più complessa di tutto il laboratorio, sono stati appresi e compresi, mentre la progettazione dell'interfaccia grafica ha stimolato la fantasia dei ragazzi, ma anche mostrato la necessità di progettare in modo puntuale. Gli alunni hanno lavorato in gruppo, da casa e in presenza.
L'efficacia dell'attività didattica si è manifestata sotto vari aspetti: ha stimolato la partecipazione dei ragazzi, catturandone l'interesse tramite il gioco, ha rafforzato le loro abilità logiche; i ragazzi sono stati spinti a sviluppare il pensiero critico formulando ipotesi, discutendole, deducendone le conseguenze. Il lavoro in squadra ha messo all'opera le loro abilità dialettiche e argomentative. Infine, un ultimo risultato importante è stato l'aver favorito la socializzazione durante il periodo di didattica a distanza.
Riflettendo sul percorso didattico effettuato, le dodici ore dedicate non hanno permesso di apprezzare al pieno le potenzialità dell'algoritmo minimax; sarebbe stato opportuno poterlo applicare in settori diversi, per illustrarne meglio la versatilità e permettere un suo uso ancora più concreto.

RINGRAZIAMENTI

Questa ricerca è stata supportata dal Piano Nazionale Lauree Scientifiche e dal MIUR Excellence Department Project attribuito al Dipartimento di Matematica dell'Università degli Studi di Roma "Tor Vergata", [CUP E83C18000100006].

BIBLIOGRAFIA

Bright, G.W., Harvey, J. G., & Wheeler, M. M. (1985). Learning and Mathematics Games. *Journal for Research in Mathematics Games 1,* 1-189.
Fiore, T. M., Lang, A., & Perucca, A. (2018). Tactile Tools for Teaching: Implementing Knuth's Algorithm for Mastering Mastermind. *The College Mathematics Journal 49*(4), 278-286.
Gros, B. (2007). Digital Games in Education: The Design of Game-Based-Learning Environment. *Journal of Research on technology in Education 40*(1), 23-38.

Knuth, D.E. (1976-77). The computer as Mastermind. *Journal of Recreational Mathematics*, *9*(1), 1-6.
Lamberti, L. & Tovena, F. (2021). *The Mastermind Game: a Combinatorial Approach*, [Paper presentation]. International Conference Gamifying Mathematics in CLIL Contexts: Approaches And Good Practices, Córdoba, Spagna. http://crf.uniroma2.it/wp-content/uploads/2022/01/Mastermind_Spagna_finale.pdf
Mitchell, M. (1999). *Mastermind mathematics: logic, strategies, and proofs*. Key Curriculum Press.
Strom, A.R., & Barolo, S. (2011). Using the Game of Mastermind to Teach, Practice and Discuss Scientific Reasoning Skills. *PLoS Biology, 9*(1), 1-3.
Tokac, U., Novak, E., & Thompson, C. G. (2019). Effects of game-based learning on students' mathematics achievement: A meta-analysis. *Journal of Computer Assisted Learning, 35*(3), 407-420.