

A pixel art landscape featuring a bright blue sky with large, fluffy white clouds. In the foreground, there is a green field and a blue lake with a small sandy beach. In the background, there are dark green mountains. The text "Sviluppare un videogioco retrò" is centered in the sky area in a black, pixelated font.

**Sviluppare un
videogioco retrò**

A pixel art landscape featuring a bright blue sky with large, fluffy white clouds. Below the sky is a range of dark green, forested hills. In the foreground, there is a vibrant green field. On the right side, a small body of water, possibly a lake or a bay, is visible, with a small sandy beach and a few tall, thin trees on the right edge.

**Sviluppare un
videogioco retrò**
(in meno di un mese)

CHI SIAMO?

CHI SIAMO?



IVAN

ivan.martini@studenti.unitn.it



@IvanMartini



GIULIA

giulia.peserico@studenti.unitn.it



@giuliapeserico

REQUISITI

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

REQUISITI

COSA (BISOGNA SAPERE)?

La base di un **qualsiasi paradigma procedurale** (assegnamento, selezione, cicli e vettori). Abbiamo scelto python per la semplicità, ma non è dipendente da librerie o proprietà del linguaggio. Non è necessaria una conoscenza profonda dell'argomento, per dei neofiti si potrebbe pensare a un minicorso di un paio di giorni sulle basi di python

REQUISITI

COSA (BISOGNA SAPERE)?

La base di un un **qualsiasi paradigma procedurale** (assegnamento, selezione, cicli e vettori). Abbiamo scelto python per la semplicità, ma non è dipendente da librerie o proprietà del linguaggio. Non è necessaria una conoscenza profonda dell'argomento, per dei neofiti si potrebbe pensare a un minicorso di un paio di giorni sulle basi di python

CHI?

Ragazze e ragazzi di medie, 1a e 2a superiore, meglio se un po' **nerd**

REQUISITI

COSA (BISOGNA SAPERE)?

La base di un un **qualsiasi paradigma procedurale** (assegnamento, selezione, cicli e vettori). Abbiamo scelto python per la semplicità, ma non è dipendente da librerie o proprietà del linguaggio. Non è necessaria una conoscenza profonda dell'argomento, per dei neofiti si potrebbe pensare a un minicorso di un paio di giorni sulle basi di python

CHI?

Ragazze e ragazzi di medie, 1a e 2a superiore, meglio se un po' **nerd**

QUANDO?

3 giugno 2021, 17.30-19.00 circa

REQUISITI

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

Il corso è riservato agli studenti di laurea triennale in Ingegneria Informatica.

REQUISITI

COME?

Con un pc per persona, con `python 3.9` e un qualsiasi terminale.

REQUISITI

COME?

Con un pc per persona, con `python 3.9` e un qualsiasi terminale.

COSA (FAREMO)?

Svilupperemo un videogioco **Roguelike**, come i primi videogiochi usciti più di 40 anni fa.

REQUISITI

COME?

Con un pc per persona, con **python 3.9** e un qualsiasi terminale.

COSA (FAREMO)?

Svilupperemo un videogioco **Roguelike**, come i primi videogiochi usciti più di 40 anni fa.

PERCHÈ?

Perchè è un progetto corposo e interessante, ma non richiede particolari conoscenze informatiche (oggetti, classi, ambienti grafici). Il prodotto è quindi molto articolato, ma utilizza istruzioni semplici. Per questo si evita l'effetto "**copia e incolla**", che è un sempre un rischio quando si prova qualcosa di complesso (e interessante) su un gruppo di inesperti

COS'è UN ROGUELIKE

COS'è UN ROGUELIKE

Un videogioco di ruolo

Ispirato a Dungeons and Dragons

COS'è UN ROGUELIKE

Un videogioco di ruolo

Ispirato a Dungeons and Dragons

L'interfaccia è in caratteri ASCII

L'aspetto visivo viene "sacrificato" in favore della profondità di gioco

COS'è UN ROGUELIKE

Un videogioco di ruolo

Ispirato a Dungeons and Dragons

L'interfaccia è in caratteri ASCII

L'aspetto visivo viene "sacrificato" in favore della profondità di gioco

I livelli sono generati "casualmente"

così come gli effetti degli oggetti che vengono trovati

COS'è UN ROGUELIKE

Un videogioco di ruolo

Ispirato a Dungeons and Dragons

L'interfaccia è in caratteri ASCII

L'aspetto visivo viene "sacrificato" in favore della profondità di gioco

I livelli sono generati "casualmente"

così come gli effetti degli oggetti che vengono trovati

Non esiste possibilità di salvataggio

La morte è permanente e non si può ricaricare

COS'è UN ROGUELIKE

COS'è UN ROGUELIKE



ROGUE (1980)

COS' È UN ROGUELIKE

You ascend to the status of Demigod...--More--

Level: 2 Gold: 120 Hp: 14(

[Mati 31 the Necromancer] St:25 Dx:17 Co:16 In:20 Wi:20 Ch:18 Chaotic
Astral Plane \$:348 HP:152(152) Pw:68(392) AC:-29 Xp:23/30039185 T:46311

NETHACK (1985)

COS'è UN ROGUELIKE

You ascend to the status of Demigod...--More--

DWARF FORTRESS (2006)



20 Wi:20 Ch:18 Chaotic
3/30039185 T:46311

COS'è UN ROGUELIKE

You ascend to the status of Demigod...--More--

THE BINDING OF ISAAC (2012)



INIZIAMO

SETUP

I file principali da utilizzare sono:

`main.py`

il file principale del programma

`game.py`

contiene le classi:

- Dungeon
- Player

SETUP

I file principali da utilizzare sono:

```
6
7  class Dungeon:
8      def __init__(self, rows, columns):
9          self.rows = rows
10         self.columns = columns
11         self.player = None
12
13     def blank(self):
14
15
16
17
18
19
20
21
22
23
24
25
26     def cave(self):
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49     def show(self, stdscr, fog = False):
50
51
52
53
54
55
56
57
58
59
60
61
62
63     def drawPlayer(self, stdscr):
64
65
66
67
68
69
```

SETUP

I file principali da utilizzare sono:

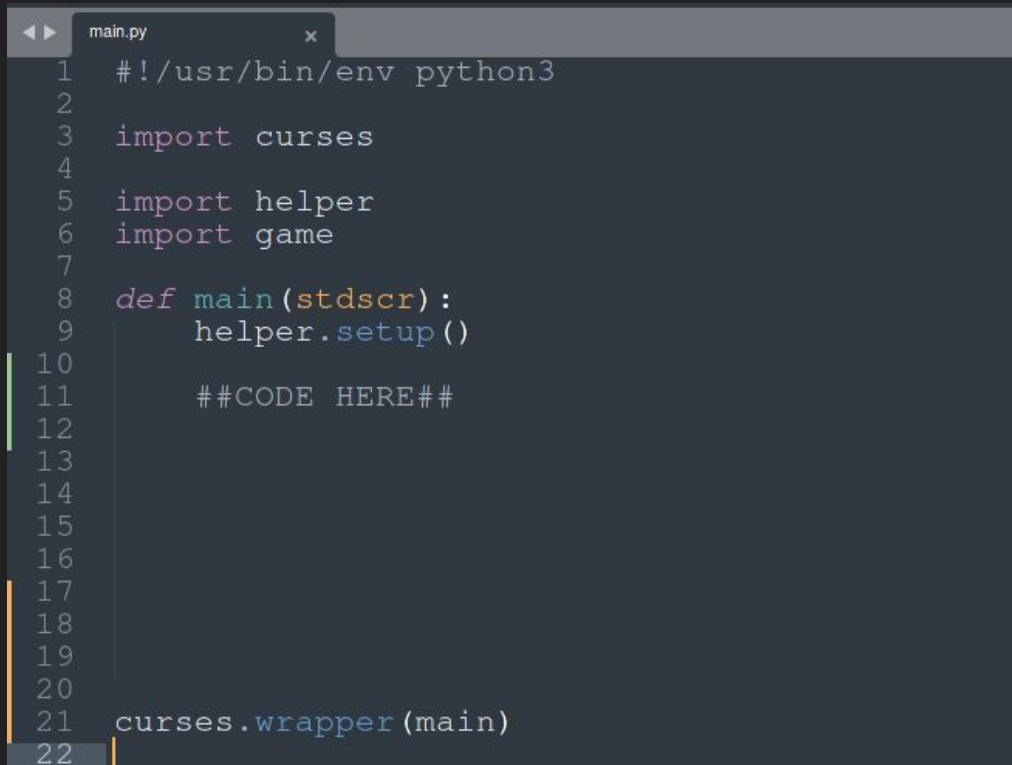
```
7 class Dungeon:
8     def __init__(self, rows, columns):
9         self.rows = rows
10        self.columns = columns
11        self.player = None
12
13    def blank(self):
14
15    def cave(self):
16
17    def show(self, stdscr, fog =
18
19    def drawPlayer(self, stdscr):
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70 class Player:
71     def __init__(self, dungeon, playerX, playerY):
72         self.X = playerX
73         self.Y = playerY
74         self.dungeon = dungeon
75
76     def getPosition(self):
77         return self.X, self.Y
78
79     def moveUp(self):
80         self.X = self.X-1
81
82     def moveDown(self):
83         self.X = self.X+1
84
85     def moveRight(self):
86         self.Y = self.Y+1
87
88     def moveLeft(self):
89         self.Y = self.Y-1
90
91     def getWalls(self, direction):
```

SETUP

Aprirete il file `main.py`, e procediamo a creare il nostro mondo

SETUP

Il vostro file dovrebbe essere così:



```
main.py x
1  #!/usr/bin/env python3
2
3  import curses
4
5  import helper
6  import game
7
8  def main(stdscr):
9      helper.setup()
10
11     ##CODE HERE##
12
13
14
15
16
17
18
19
20
21  curses.wrapper(main)
22
```

SETUP

Salviamo in due variabili le dimensioni del terminale con `screen_size()` e utilizziamole per creare una caverna, detta **Dungeon**.

```
main.py x
1  #!/usr/bin/env python3
2
3  import curses
4
5  import helper
6  import game
7
8  def main(stdscr):
9      helper.setup()
10
11      ##CODE HERE##
12
13      rows, columns = helper.screen_size()
14
15      dung = game.Dungeon(rows, columns)
16
17
18
19
20
21  curses.wrapper(main)
22
```

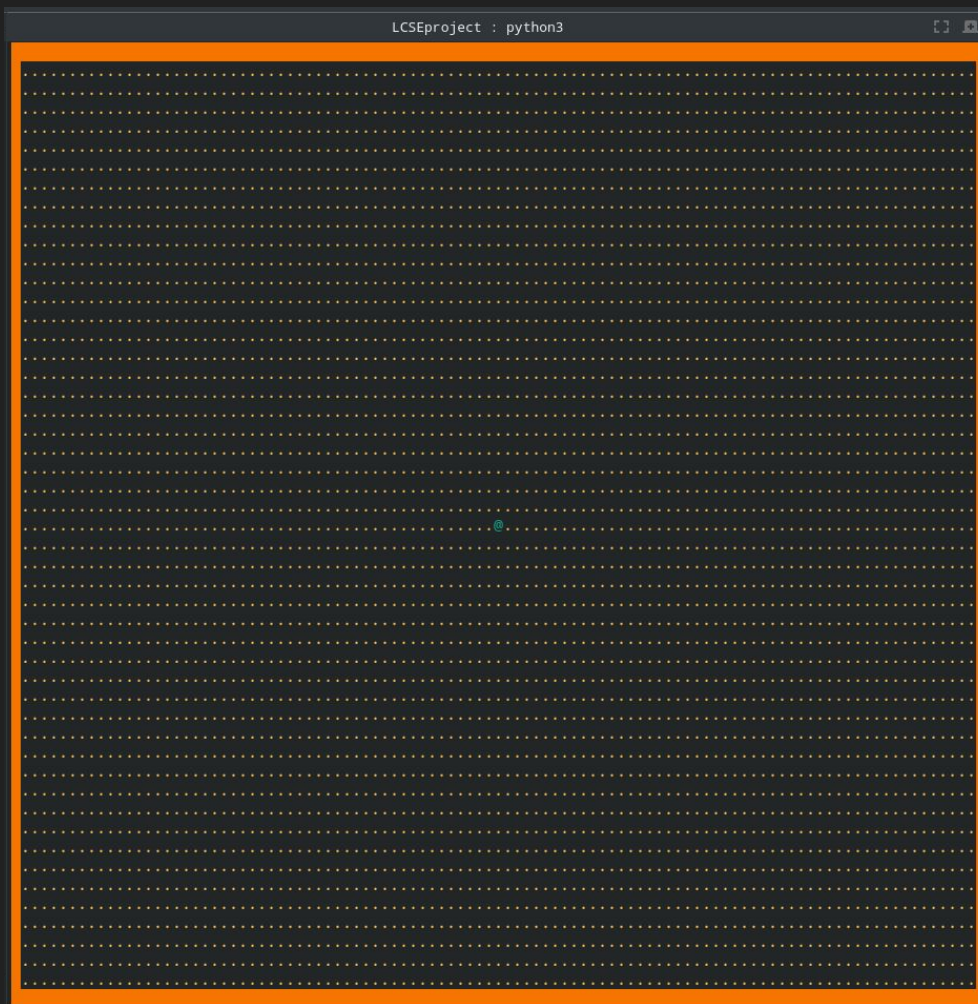
SETUP

Con la funzione **cave()** inizializziamo una caverna vuota. Mostriamola con **show()** e aspettiamo prima di terminare il processo con **getch()**.

```
main.py x
1  #!/usr/bin/env python3
2
3  import curses
4
5  import helper
6  import game
7
8  def main(stdscr):
9      helper.setup()
10
11     ##CODE HERE##
12
13     rows, columns = helper.screen_size()
14
15     dung = game.Dungeon(rows, columns)
16     dung.cave()
17     dung.show(stdscr)
18
19     stdscr.getch()
20
21 curses.wrapper(main)
22
```

SETUP

Questo è il nostro gioco, per ora.



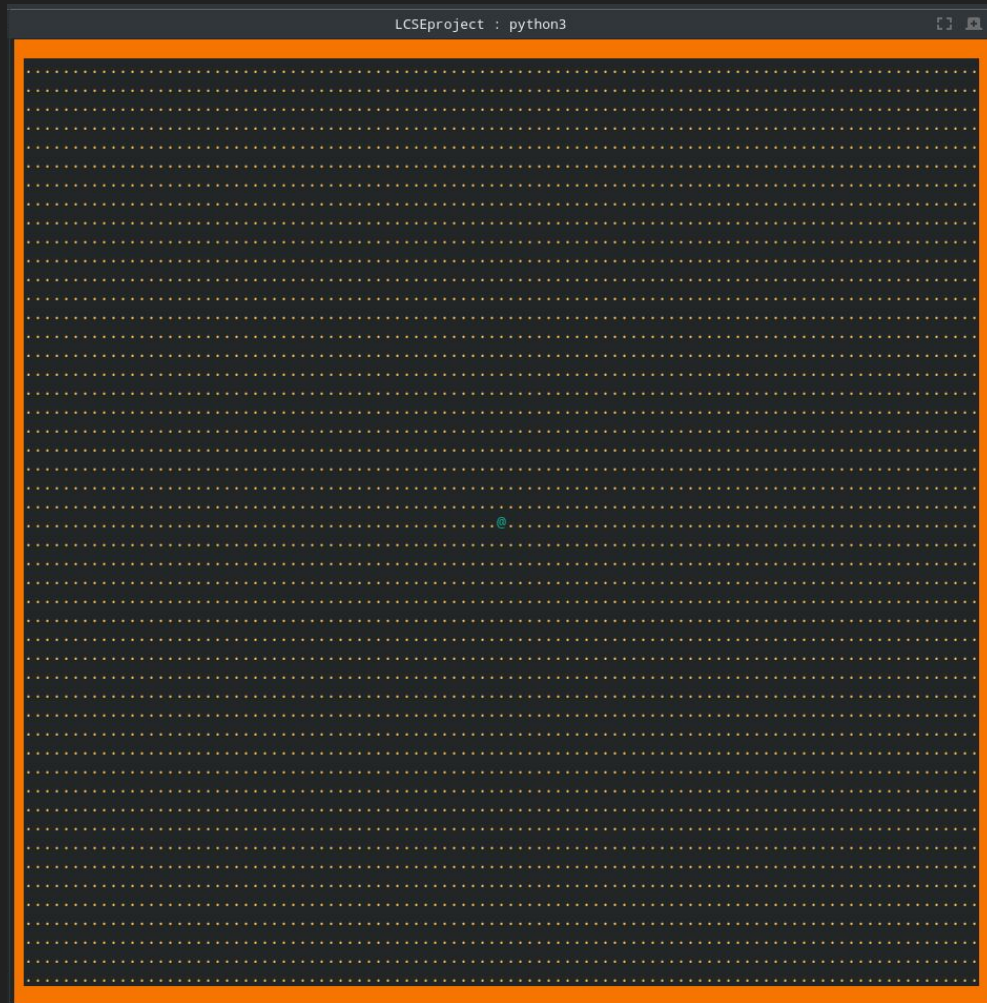
SETUP

Questo è il nostro gioco, per ora.

@ invece è il protagonista

- sono le celle vuote

sono le i muri



MOVIMENTO

La mappa è salvata come una matrice (un vettore a due dimensioni). Spostare il protagonista significa spostare la sua posizione nella matrice.

```
15     dung = game.Dungeon(rows, columns)
16     dung.blank()
17     dung.show(stdscr)
18
19     stdscr.getch()
20
21
22
23
24
25
26
27
28
29
30
31
32 curses.wrapper(main)
33
```

MOVIMENTO

Rimuoviamo il `getch()`, visto che non servirà più e inseriamolo in un **ciclo** "infinito". Adesso però **salviamo anche l'input** dell'utente

```
15     dung = game.Dungeon(rows, columns)
16     dung.blank()
17     dung.show(stdscr)
18
19
20
21     while True:
22         choice = chr(stdscr.getch())
23
24
25
26
27
28
29
30
31
32 curses.wrapper(main)
33
```

MOVIMENTO

Inseriamo la condizione di uscita dal ciclo infinito, quando l'utente preme **ZERO**. Aggiorniamo anche lo schermo a ogni iterazione.

```
15     dung = game.Dungeon(rows, columns)
16     dung.blank()
17     dung.show(stdscr)
18
19
20
21     while True:
22         choice = chr(stdscr.getch())
23
24         if choice == "0":
25             break
26
27
28
29
30         dung.show(stdscr)
31
32 curses.wrapper(main)
33
```

MOVIMENTO

Se l'utente preme la **W** spostiamo il protagonista in alto. Possiamo provare a implementare anche gli altri movimenti (**A,S,D**).

```
15     dung = game.Dungeon(rows, columns)
16     dung.blank()
17     dung.show(stdscr)
18
19
20
21     while True:
22         choice = chr(stdscr.getch())
23
24         if choice == "0":
25             break
26
27         if (choice == "W" or choice == "w"):
28             dung.player.moveUp()
29
30         dung.show(stdscr)
31
32 curses.wrapper(main)
33
```

MOVIMENTO

```
17     dung.show(stdscr)
18
19     while True:
20         choice = chr(stdscr.getch())
21
22         if choice == "0":
23             break
24
25         if choice == "W" or choice == "w":
26             dung.player.moveUp()
27
28         if choice == "S" or choice == "s":
29             dung.player.moveDown()
30
31         if choice == "D" or choice == "d":
32             dung.player.moveRight()
33
34         if choice == "A" or choice == "a":
35             dung.player.moveLeft()
36
37         dung.show(stdscr)
38
39 curses.wrapper(main)
40
```

MOVIMENTO PT. 2

Il movimento funziona bene, ma i muri non bloccano il personaggio.

MOVIMENTO PT. 2

Il movimento funziona bene, ma i muri non bloccano il personaggio. Sistemiamo il problema con un controllo prima del movimento.

```
21     while True:
22         choice = chr(stdscr.getch())
23
24         if choice == "0":
25             break
26
27         if choice == "W" or choice == "w" and not dung.player.getWalls("up"):
28             dung.player.moveUp()
29
30         if choice == "S" or choice == "s":
31             dung.player.moveDown()
32
33         if choice == "D" or choice == "d":
34             dung.player.moveRight()
35
36         if choice == "A" or choice == "a":
37             dung.player.moveLeft()
38
39     dung.show(stdscr)
```

MOVIMENTO PT. 2

Applichiamo lo stesso principio anche agli altri movimenti. Al posto di "up" serve utilizzare le altre direzioni

```
21     while True:
22         choice = chr(stdscr.getch())
23
24         if choice == "0":
25             break
26
27         if choice == "W" or choice == "w" and not dung.player.getWalls("up"):
28             dung.player.moveUp()
29
30         if choice == "S" or choice == "s" and not dung.player.getWalls("down"):
31             dung.player.moveDown()
32
33         if choice == "D" or choice == "d" and not dung.player.getWalls("right"):
34             dung.player.moveRight()
35
36         if choice == "A" or choice == "a" and not dung.player.getWalls("left"):
37             dung.player.moveLeft()
38
39     dung.show(stdscr)
```


AUTOMA CELLULARE

SETUP

I movimenti sono fatti, ma la mappa è semplicemente un quadrato.

Aprirete il file **automa.py**, e procediamo a scrivere un codice per generare una mappa più interessante

SETUP

I movimenti sono fatti, ma la mappa è semplicemente un quadrato.
Aprite il file **automa.py**, e procediamo a scrivere un codice per generare una mappa più interessante

un **automa cellulare** è un modello che cerca di spiegare come le cellule interagiscono tra di loro, su una **griglia**.

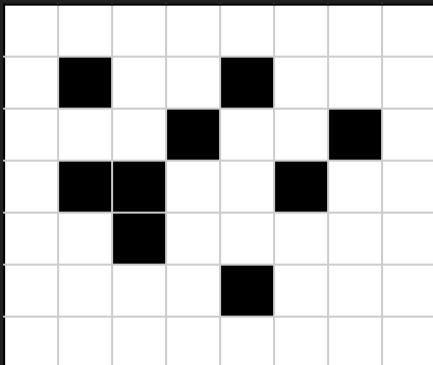
In particolare, si conta il numero di vicini **accesi** o **spenti** e si prendono delle decisioni per la prossima generazione.

SETUP

I movimenti sono fatti, ma la mappa è semplicemente un quadrato. Aprite il file **automa.py**, e procediamo a scrivere un codice per generare una mappa più interessante

un **automa cellulare** è un modello che cerca di spiegare come le cellule interagiscono tra di loro, su una **griglia**.

In particolare, si conta il numero di vicini **accesi** o **spenti** e si prendono delle decisioni per la prossima generazione.

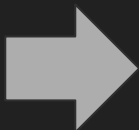
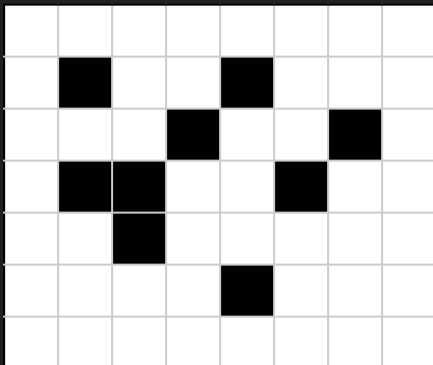


SETUP

I movimenti sono fatti, ma la mappa è semplicemente un quadrato.
Aprire il file **automa.py**, e procediamo a scrivere un codice per generare una mappa più interessante

un **automa cellulare** è un modello che cerca di spiegare come le cellule interagiscono tra di loro, su una **griglia**.

In particolare, si conta il numero di vicini **accesi** o **spenti** e si prendono delle decisioni per la prossima generazione.



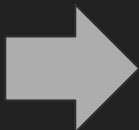
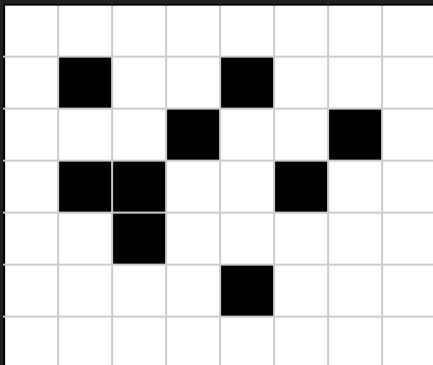
1	1	1	1	1	1	0	0
1	0	2	2	1	2	2	1
2	3	4	2	3	3	1	1
1	2	3	3	2	1	2	1
1	3	2	3	2	2	1	0
0	1	1	2	0	1	0	0
0	0	0	1	1	1	0	0

SETUP

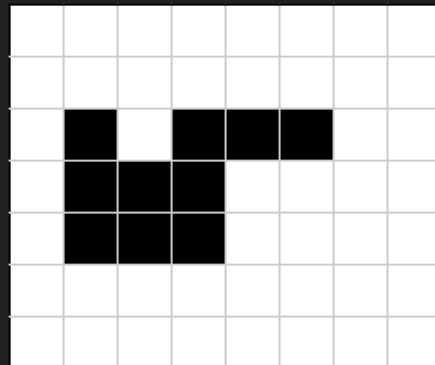
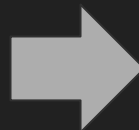
I movimenti sono fatti, ma la mappa è semplicemente un quadrato.
Aprire il file **automa.py**, e procediamo a scrivere un codice per generare una mappa più interessante

un **automa cellulare** è un modello che cerca di spiegare come le cellule interagiscono tra di loro, su una **griglia**.

In particolare, si conta il numero di vicini **accesi** o **spenti** e si prendono delle decisioni per la prossima generazione.



1	1	1	1	1	1	0	0
1	0	2	2	1	2	2	1
2	3	4	2	3	3	1	1
1	2	3	3	2	1	2	1
1	3	2	3	2	2	1	0
0	1	1	2	0	1	0	0
0	0	0	1	1	1	0	0



SETUP — REGOLE

Le **regole** definiscono se una **cellula attiva** sopravvivrà alla prossima generazione o meno.

Definiscono inoltre se una **cellula spenta** diventerà attiva nella prossima generazione o meno.

SETUP — REGOLE

Le **regole** definiscono se una **cellula attiva** sopravvivrà alla prossima generazione o meno.

Definiscono inoltre se una **cellula spenta** diventerà attiva nella prossima generazione o meno.

Per ora manterremo le regole più semplici, ovvero quelle del gioco della vita di **Conway**.

Cellula attiva

Cellula spenta

SETUP — REGOLE

Le **regole** definiscono se una **cellula attiva** sopravvivrà alla prossima generazione o meno.

Definiscono inoltre se una **cellula spenta** diventerà attiva nella prossima generazione o meno.

Per ora manterremo le regole più semplici, ovvero quelle del gioco della vita di **Conway**.

Cellula attiva

- con meno di due cellule vive adiacenti muore (isolamento)
- con 2 o 3 cellule vive adiacenti sopravvive
- con più di 3 cellule vive adiacenti muore (sovrappopolazione)

Cellula spenta

- nasce se ha esattamente 3 cellule vive adiacenti (riproduzione)
- altrimenti resta morta

AUTOMA — MAIN

Iniziamo a aprendo **automa.py** e scendiamo fino alla funzione **main**, che contiene queste istruzioni qui

```
46     helper.setup()
47     rows, columns = helper.screen_size()
48
49     automa = init(rows, columns)
50     show(stdscr, automa)
51
52     ##CODE HERE##
53
54
55
56
57
58
59
60
61
62
63
64 #MAIN
65 if (__name__ == '__main__'):
66     curses.wrapper(main)
67
```

AUTOMA — MAIN

La prima parte è molto simile al programma precedente, un **ciclo** che esca quando l' input dell'utente è uguale a **ZERO**.

```
46     helper.setup()
47     rows, columns = helper.screen_size()
48
49     automa = init(rows,columns)
50     show(stdscr,automa)
51
52     ##CODE HERE##
53
54     while True:
55         choice = chr(stdscr.getch())
56
57         if choice == "0":
58             break;
59
60
61
62
63
64 #MAIN
65 if (__name__ == '__main__'):
66     curses.wrapper(main)
67
```

AUTOMA — MAIN

L'automa è già stato inizializzato come una matrice di 0 e 1 in posizione **casuale**. Richiamiamo la funzione per evolvere, che implementeremo ora.

```
46     helper.setup()
47     rows, columns = helper.screen_size()
48
49     automa = init(rows, columns)
50     show(stdscr, automa)
51
52     ##CODE HERE##
53
54     while True:
55         choice = chr(stdscr.getch())
56
57         if choice == "0":
58             break;
59
60         #automa = cave(automa, 1)
61         automa = evolve(automa)
62         show(stdscr, automa)
63
64     #MAIN
65     if (__name__ == '__main__'):
66         curses.wrapper(main)
67
```

AUTOMA – EVOLVE

Andiamo a completare la funzione per evolvere l'automa, che ad ora è più o meno così:

```
56 def evolve(matrix):  
57     ##CODE HERE##  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80     return matrix  
81
```

AUTOMA - EVOLVE

Per prima cosa andiamo a creare una **matrice parallela**, per evitare di fare confusione con quella iniziale. Ritorneremo quella e non **matrix**

```
56 def evolve(matrix):  
57     ##CODE HERE##  
58     replacement = np.zeros((len(matrix), len(matrix[0])), dtype=int)  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79     return replacement  
80  
81
```

AUTOMA - EVOLVE

Scorriamo la matrice cella per cella, utilizzando due **cicli**. Dato che abbiamo bisogno delle celle con 8 vicini, non contiamo quelle esterne.

```
56 def evolve(matrix):  
57     ##CODE HERE##  
58     replacement = np.zeros((len(matrix), len(matrix[0])), dtype=int)  
59  
60     for x in range(1, len(matrix)-1):  
61         for y in range(1, len(matrix[0])-1):  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79     return replacement  
80  
81
```

AUTOMA - EVOLVE

Salviamo il totale delle **8 celle vicine** (0-1). Il modo più immediato è sommarle tutte e 9 e sottrarre il valore della cella stessa.

```
56 def evolve(matrix):
57     ##CODE HERE##
58     replacement = np.zeros((len(matrix), len(matrix[0])), dtype=int)
59
60     for x in range(1, len(matrix)-1):
61         for y in range(1, len(matrix[0])-1):
62             total = -matrix[x][y]
63             for a in range(x-1, x+2):
64                 for b in range(y-1, y+2):
65                     total += matrix[a][b]
66
67
68
69
70
71
72
73
74
75
76
77
78
79     return replacement
80
81
```


AUTOMA - EVOLVE

Implementiamo le regole per le **cellule attive**: meno di 2 - muore; 2 o 3 - sopravvive; più di 3 - muore:

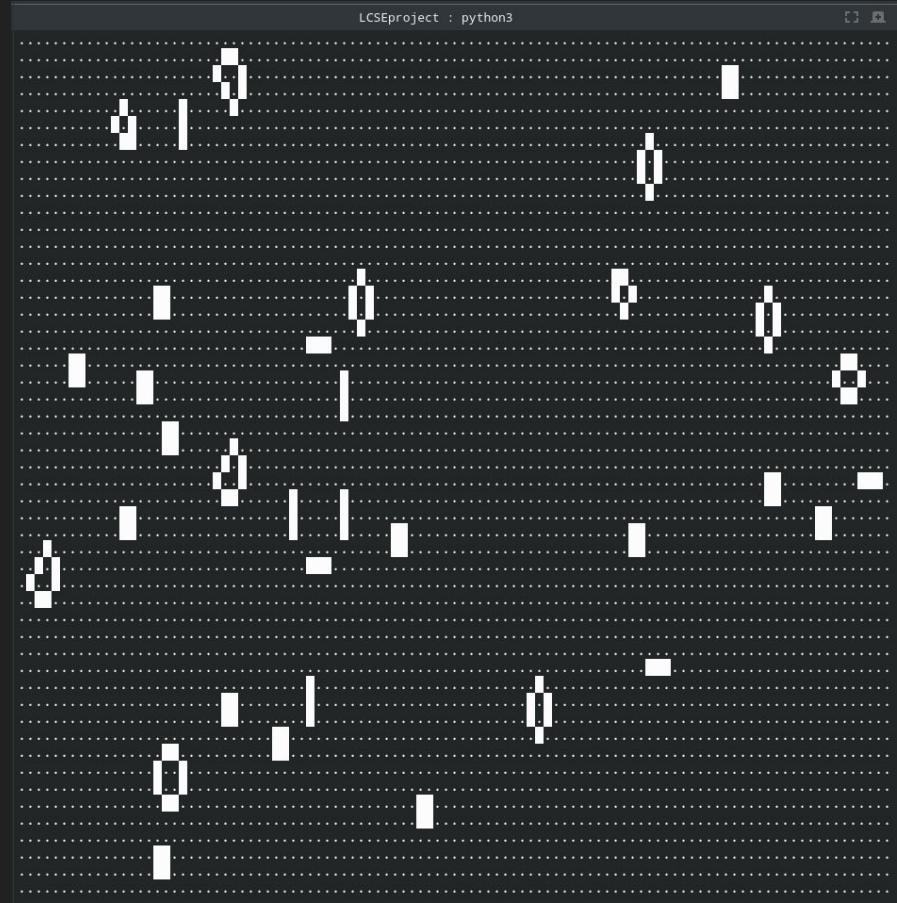
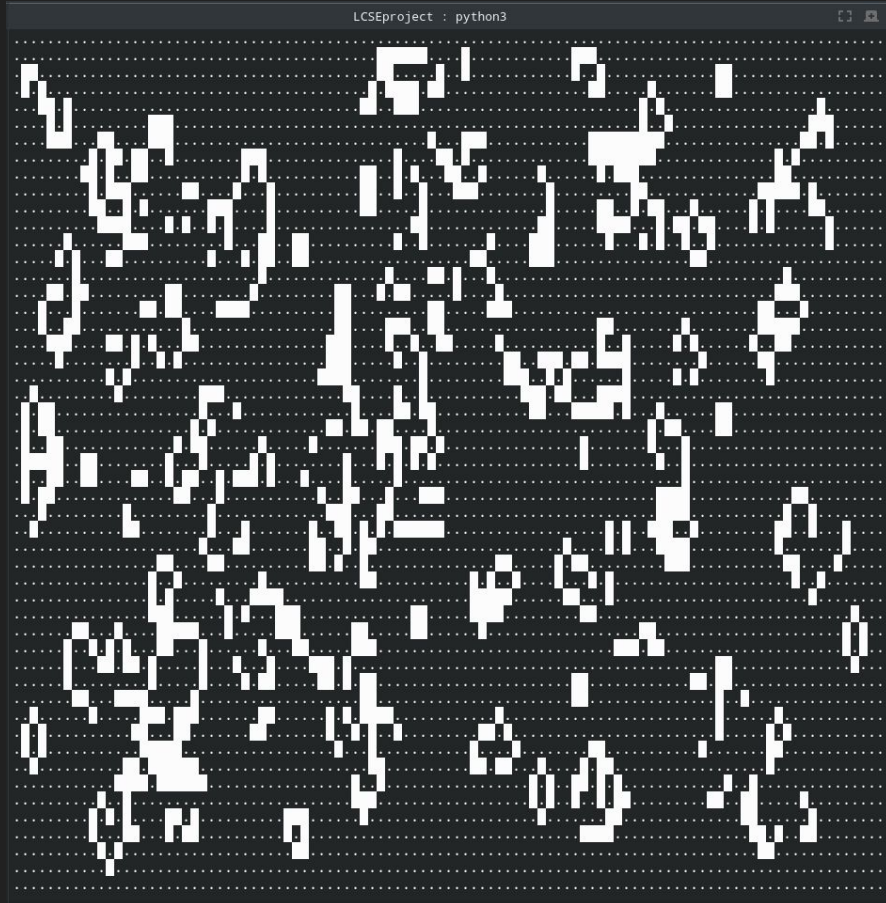
```
56 def evolve(matrix):
57     ##CODE HERE##
58     replacement = np.zeros((len(matrix), len(matrix[0])), dtype=int)
59
60     for x in range(1, len(matrix)-1):
61         for y in range(1, len(matrix[0])-1):
62             total = -matrix[x][y]
63             for a in range(x-1, x+2):
64                 for b in range(y-1, y+2):
65                     total += matrix[a][b]
66
67             if matrix[x][y] != 0:
68                 if total < 2:
69                     replacement[x][y] = 0
70                 elif total == 2 or total == 3:
71                     replacement[x][y] = 1
72                 elif total > 3:
73                     replacement[x][y] = 0
74
75
76
77
78
79     return replacement
80
```

AUTOMA — EVOLVE

E finalmente quelle per le **cellule spente**: con esattamente 3 nasce, diversamente rimane spenta

```
56 def evolve(matrix):
57     ##CODE HERE##
58     replacement = np.zeros((len(matrix), len(matrix[0])), dtype=int)
59
60     for x in range(1, len(matrix)-1):
61         for y in range(1, len(matrix[0])-1):
62             total = -matrix[x][y]
63             for a in range(x-1, x+2):
64                 for b in range(y-1, y+2):
65                     total += matrix[a][b]
66
67             if matrix[x][y] != 0:
68                 if total < 2:
69                     replacement[x][y] = 0
70                 elif total == 2 or total == 3:
71                     replacement[x][y] = 1
72                 elif total > 3:
73                     replacement[x][y] = 0
74             else:
75                 if total == 3:
76                     replacement[x][y] = 1
77                 else:
78                     replacement[x][y] = 0
79     return replacement
80
```

AUTOMA — EVOLVE



AUTOMA - CAVE

Cambiando un paio di regole e aggiungendo le generazioni, possiamo utilizzare l'automa per generare dei terreni simili a caverne:

```
def cave(matrix, generations):  
    ##CODE HERE##
```

```
    return matrix
```

AUTOMA - CAVE

Iniziamo togliendo il ritorno di `matrix` come in `evolve()`. Aggiungiamo anche un ciclo che esegua le `n` generazioni richieste.

```
def cave(matrix, generations):  
    ##CODE HERE##  
    replacement = []  
    for z in range(generations):  
        replacement = np.ones((len(matrix), len(matrix[0])), dtype=int)  
  
        matrix = replacement  
    return replacement
```

AUTOMA - CAVE

Iniziamo togliendo il ritorno di **matrix** come in **evolve()**. Aggiungiamo anche un ciclo che esegua le **n** generazioni richieste.

```
def cave(matrix, generations):  
    ##CODE HERE##  
    replacement = []  
    for z in range(generations):  
        replacement = np.ones((len(matrix), len(matrix[0])), dtype=int)  
  
        matrix = replacement  
    return replacement
```

AUTOMA - CAVE

Esattamente come in `evolve()` calcoliamo il totale di celle adiacenti. Il codice è identico (**copia-incolla**).

```
def cave(matrix, generations):  
    ##CODE HERE##  
    replacement = []  
    for z in range(generations):  
        replacement = np.ones((len(matrix), len(matrix[0])), dtype=int)  
        for x in range(1, len(matrix)-1):  
            for y in range(1, len(matrix[0])-1):  
                total = -matrix[x][y]  
                for a in range(x-1, x+2):  
                    for b in range(y-1, y+2):  
                        total += matrix[a][b]  
  
        matrix = replacement  
    return replacement
```

AUTOMA - CAVE

Implementiamo le nuove regole: con **più di 5 cellule** vive attorno la cellula si attiva. Così chiudiamo i buchi.

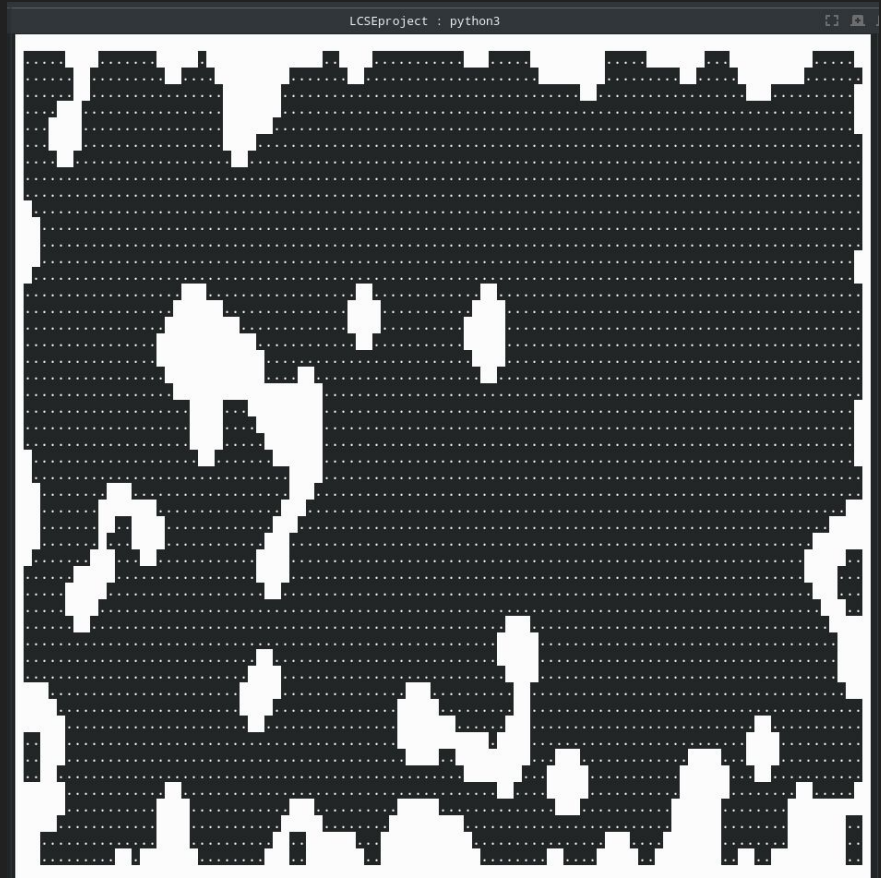
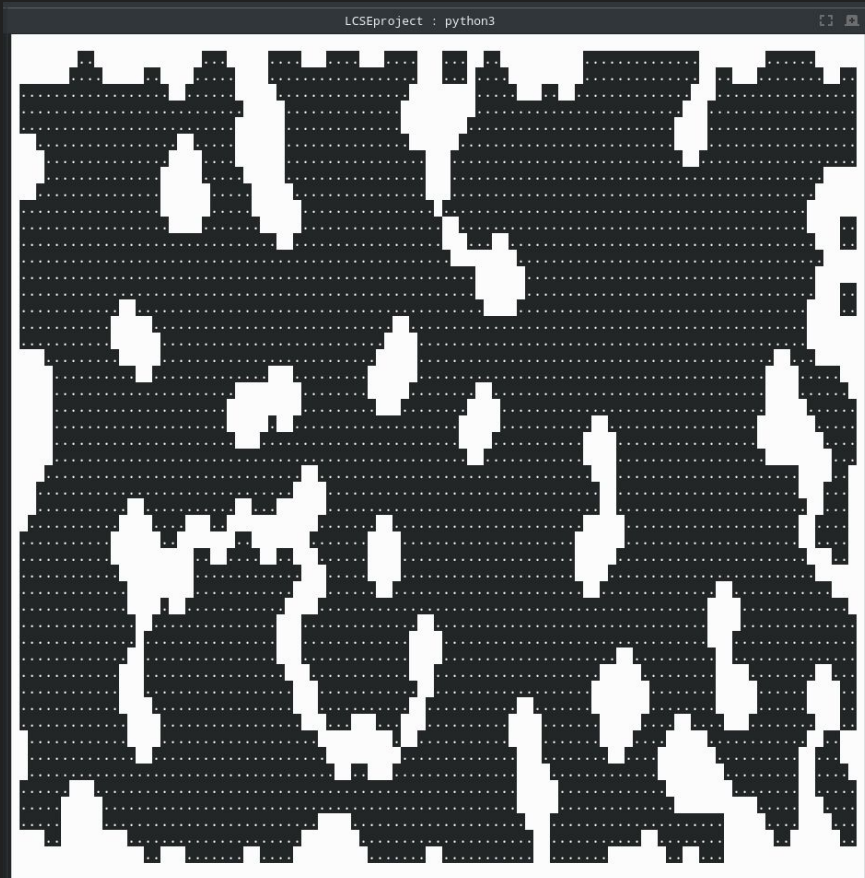
```
def cave(matrix, generations):  
    ##CODE HERE##  
    replacement = []  
    for z in range(generations):  
        replacement = np.ones((len(matrix), len(matrix[0])), dtype=int)  
        for x in range(1, len(matrix)-1):  
            for y in range(1, len(matrix[0])-1):  
                total = -matrix[x][y]  
                for a in range(x-1, x+2):  
                    for b in range(y-1, y+2):  
                        total += matrix[a][b]  
  
                if total > 5:  
                    replacement[x][y] = 1  
  
        matrix = replacement  
    return replacement
```


AUTOMA - CAVE

Infine, se le **cellule vive** adiacenti sono 4 o 5, il loro valore resta immutato.

```
def cave(matrix, generations):  
    ##CODE HERE##  
    replacement = []  
    for z in range(generations):  
        replacement = np.ones((len(matrix), len(matrix[0])), dtype=int)  
        for x in range(1, len(matrix)-1):  
            for y in range(1, len(matrix[0])-1):  
                total = -matrix[x][y]  
                for a in range(x-1, x+2):  
                    for b in range(y-1, y+2):  
                        total += matrix[a][b]  
  
                if total > 5:  
                    replacement[x][y] = 1  
                elif total < 4:  
                    replacement[x][y] = 0  
                else:  
                    replacement[x][y] = matrix[x][y]  
        matrix = replacement  
    return replacement
```

AUTOMA - CAVE



GAME - CAVE

Le ultime cose da fare sono caricare il codice all'interno di `game.py`, nella funzione `cave()`:

```
25  
26     def cave(self):  
27         return self.blank()  
28         ##CODE HERE##  
29
```

GAME - CAVE

Le ultime cose da fare sono caricare il codice all'interno di `game.py`, nella funzione `cave()`:

```
25  
26     def cave(self):  
27         return self.blank()  
28         ##CODE HERE##  
29
```



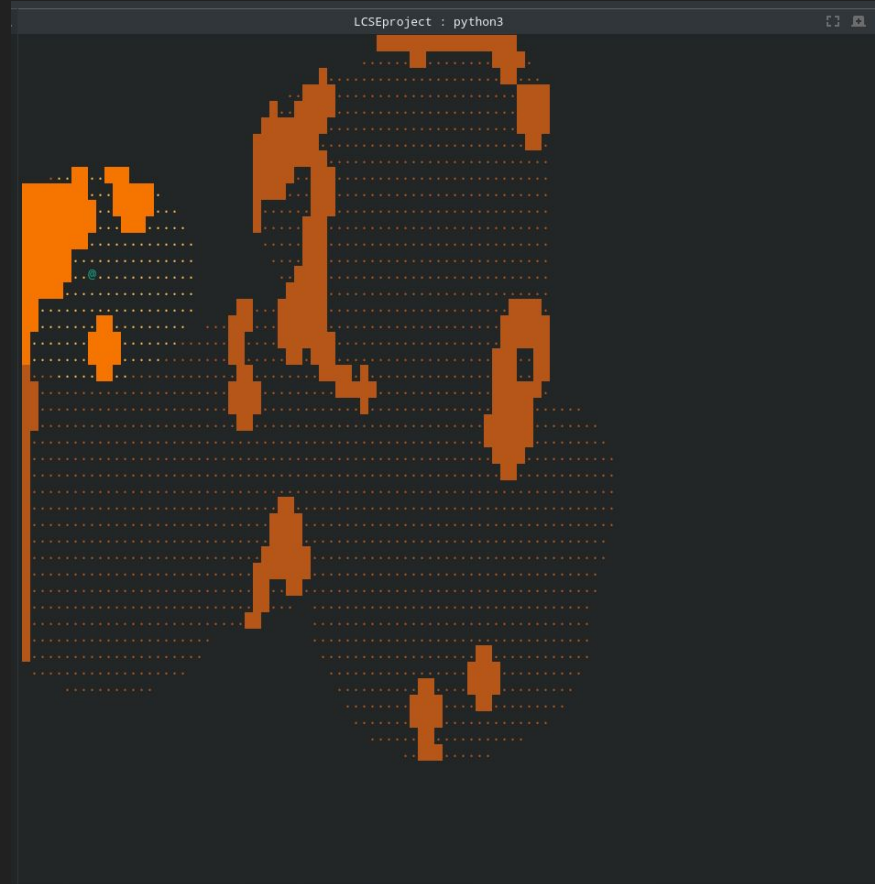
```
25  
26     def cave(self):  
27         self.matrix = automa.init(self.rows, self.columns)  
28         self.matrix = automa.cave(self.matrix, 30)  
29  
30         self.player = Player(self, int(self.rows/2), int(self.columns/2))  
31
```

GAME - CAVE

E infine aggiungere, a scelta, la **nebbia** (**fog**) nel file **main.py**, come parametro della funzione **show()**:

```
13     dung.show(stdscr)
14     dung.show(stdscr, fog = True)
15
16     while True:
17         choice = chr(stdscr.getch())
18
19         if choice == "0":
20             break
21
22         if (choice == "W" or choice == "w") and not dung.player.getWalls("u"):
23             dung.player.moveUp()
24
25         if choice == "S" or choice == "s" and not dung.player.getWalls("d"):
26             dung.player.moveDown()
27
28         if choice == "D" or choice == "d" and not dung.player.getWalls("r"):
29             dung.player.moveRight()
30
31         if choice == "A" or choice == "a" and not dung.player.getWalls("l"):
32             dung.player.moveLeft()
33
34     dung.show(stdscr, fog = True)
35
```

GAME - CAVE



**Sviluppare un videogioco
un (po' meno) retrò**



...PARTIAMO DAI
VIDEOGIOCHI 2D

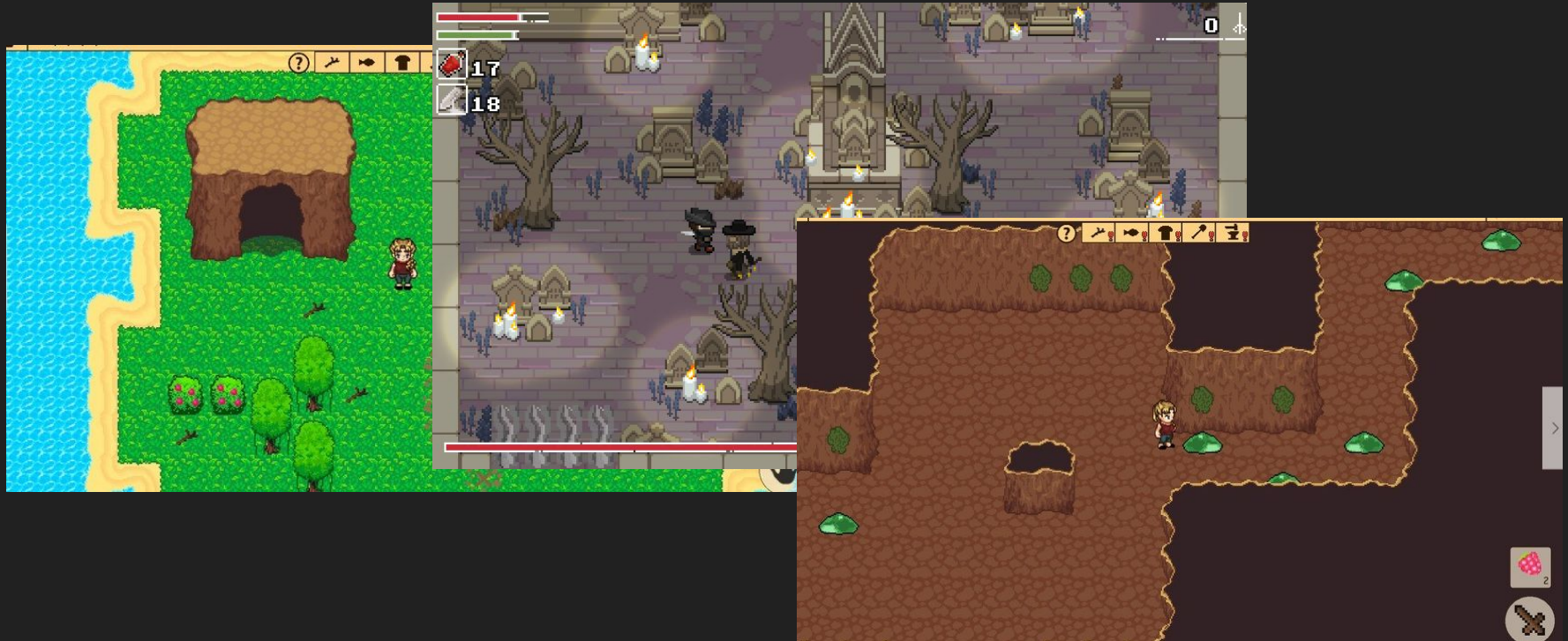
...PARTIAMO DAI VIDEOGIOCHI 2D



...PARTIAMO DAI VIDEOGIOCHI 2D



...PARTIAMO DAI VIDEOGIOCHI 2D



STRUMENTI UTILIZZATI

Materiale e concetti utilizzati nei laboratori precedenti

- Mappa e personaggi
- Creazione dell'ambiente casuale tramite un **Automa Cellulare**

Processing

- **Processing.py** (è stato scelto Python come linguaggio di programmazione per continuità rispetto ai lab precedenti)

A pixel art landscape featuring a bright blue sky with large, fluffy white and light blue clouds. Below the sky is a range of dark green, forested hills. In the foreground, there is a vibrant green field. On the right side, a small body of water with blue and white pixelated waves is visible, with a few tall, thin green trees standing near its edge.

Piccola introduzione a Processing

COS'è PROCESSING

- è un linguaggio di programmazione
- e permette di sviluppare **applicazioni grafiche** (giochi, animazioni, contenuti interattivi, ecc..)

COS'è PROCESSING

- è un linguaggio di programmazione
- e permette di sviluppare **applicazioni grafiche** (giochi, animazioni, contenuti interattivi, ecc..)
- Viene utilizzato un **Processing Development Environment**
 - Text editor: dove si scrive il codice
 - Console: per eventuali errori
 - Bottoni: per avviare il programma

```
sketch_210525b
1
2
3 def setup():
4     size(300,300)
5     background(32, 110, 213)
6
7 def draw():
8     ellipse(50, 50, 80, 80)
9
10
11
12
```

```
22
NameError: name 'ghj' is not defined

processing.app.SketchException: NameError: name 'ghj' is not defin
    at jycessing.mode.run.SketchRunner.convertPythonSketchEr
    at jycessing.mode.run.SketchRunner.lambda$startSketch$3(
    at java.lang.Thread.run(Thread.java:748)

Console
```



Processing.py

QUALCHE DETTAGLIO...

- Si scrive codice in Python
 - concetti di variabili, costrutti di selezione, cicli e funzioni
- Due funzioni principali di Processing:
 - **setup()**
 - viene chiamato automaticamente 1 volta
 - serve per le inizializzazioni
 - **draw():** viene chiamato ripetutamente
 - serve per le animazioni
- Libreria di Processing ...
 - **background(32, 110, 213)**
 - **ellipse(50,50,80,80)**
 - **rect(50,50,80,80)**
 - ...



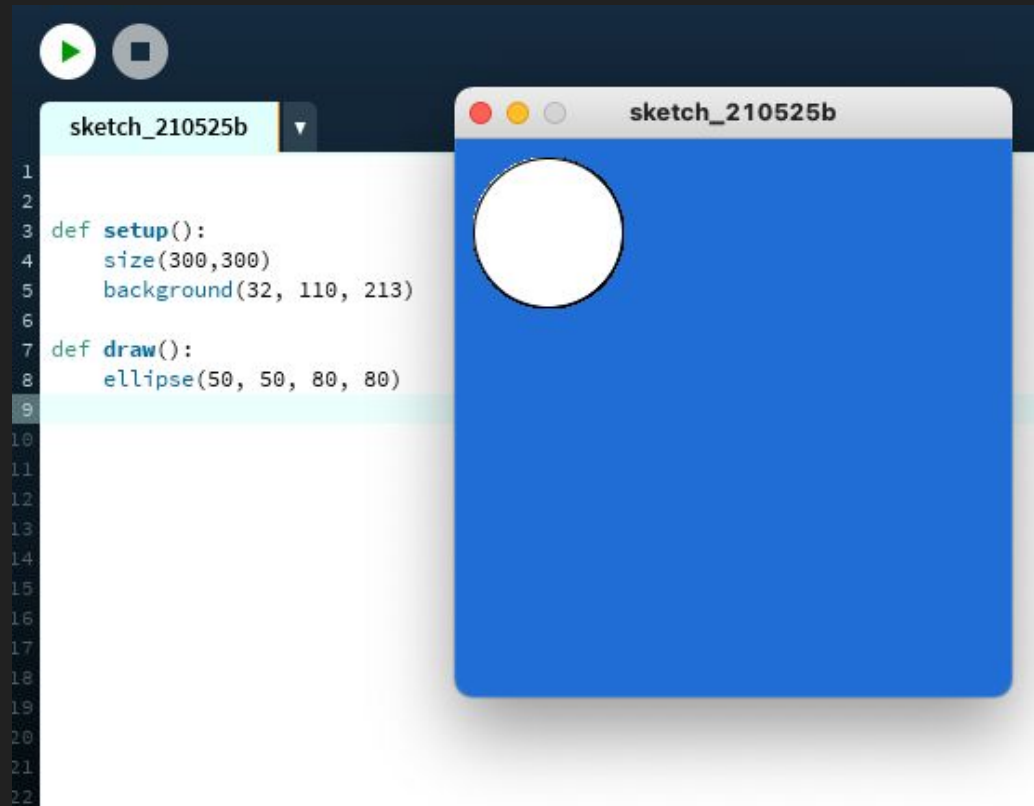
```
1
2
3 def setup():
4     size(300,300)
5     background(32, 110, 213)
6
7 def draw():
8     ellipse(50, 50, 80, 80)
9
10
11
12
```


QUALCHE DETTAGLIO...

- Le funzioni principali che useremo di più:
 - colorazioni:
 - background()
 - fill()
 - gestione eventi:
 - MousePressed()
 - KeyPressed()
 - shapes:
 - line(x, y, width, height)
 - rect(x, y, width, height)
 - ellipse(x, y, width, height)
 - Caricamento immagini:
 - loadImage(path)
 - image(img_loaded, x, y)

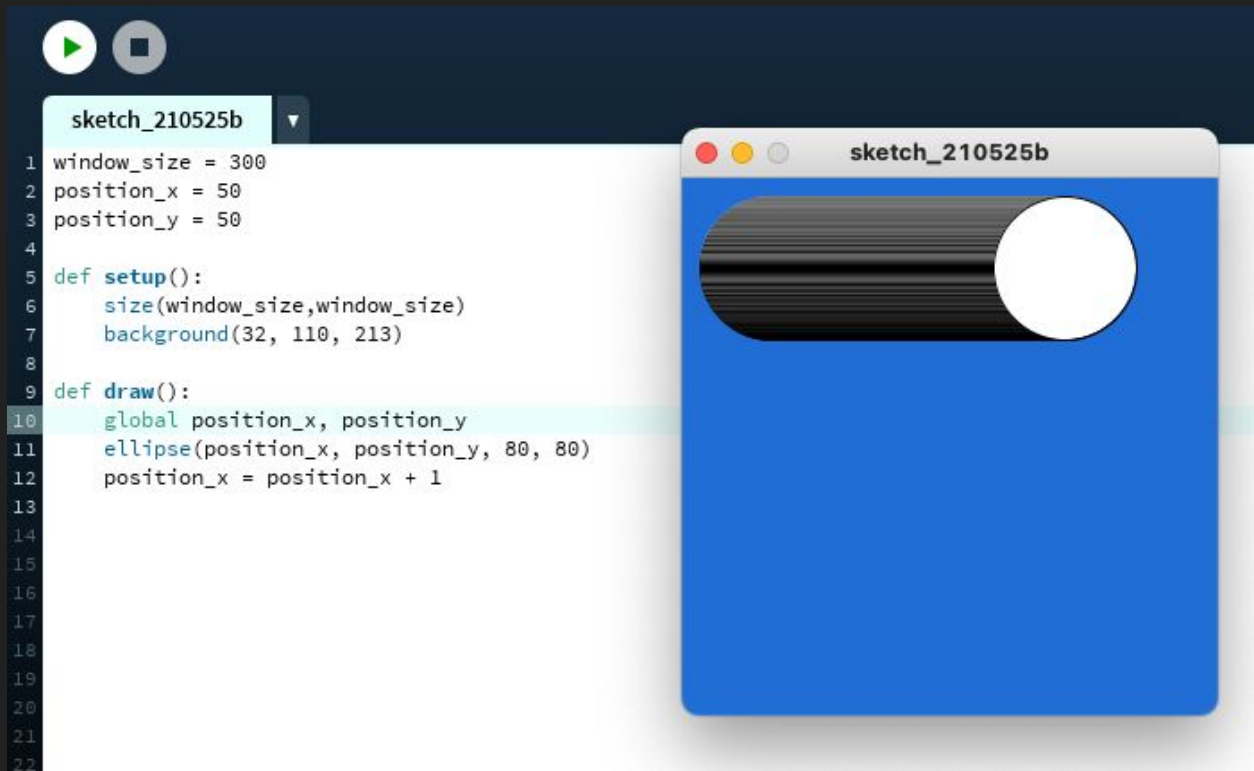
All references: <https://py.processing.org/reference/>

ESEMPIO BASE



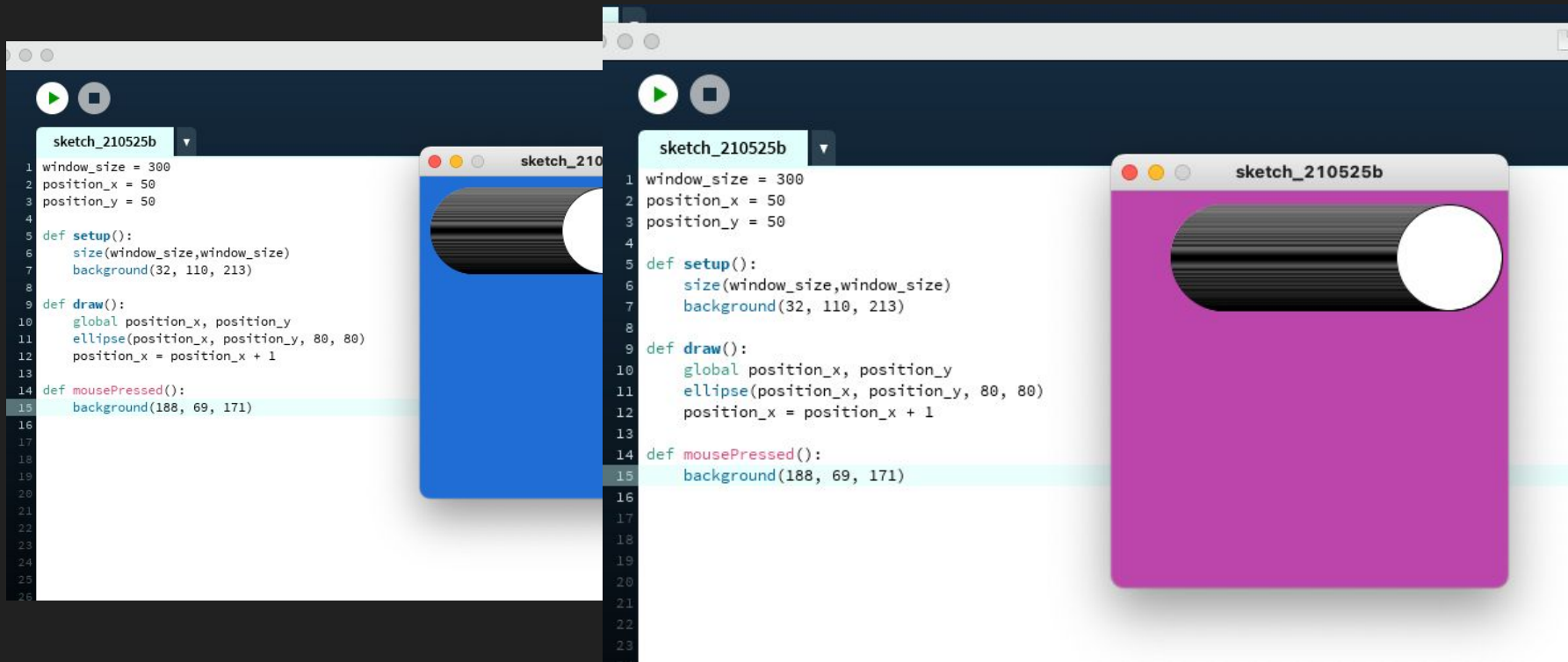
ESEMPIO BASE

USO DI VARIABILI



ESEMPIO BASE

USO DEL MOUSE



**Continuiamo con il
nostro videogioco**



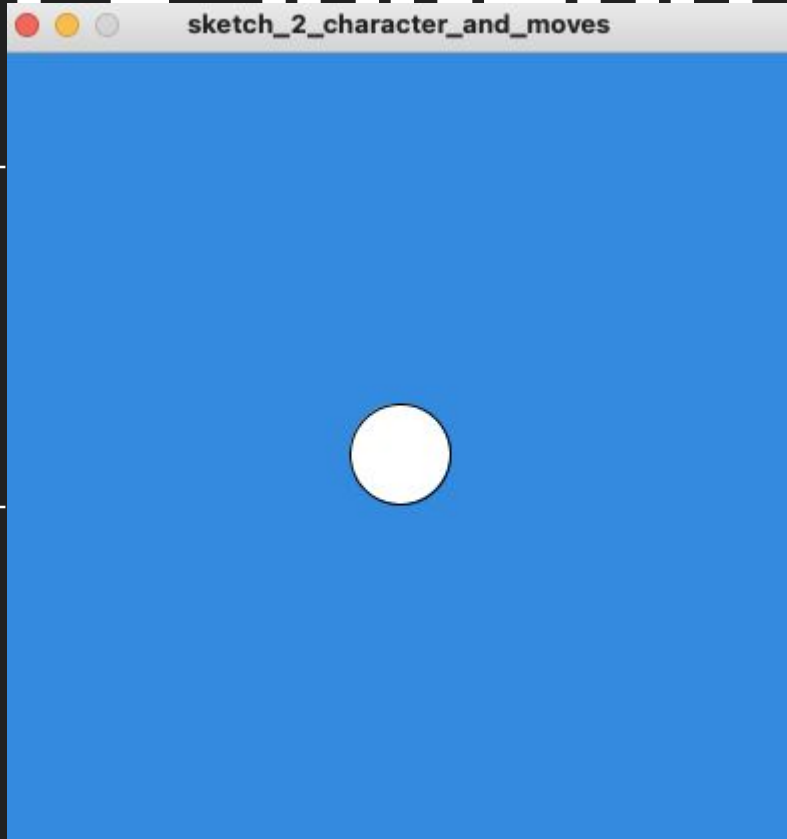
CREAZIONE PERSONAGGIO

- Creare un primo "personaggio"
 - può essere un cerchio, un rettangolo, o qualunque forma vogliate..
 - esempio: `rect()`, `ellipse()` ecc..
 - Provare a cambiare dimensione, colore ...

CREAZIONE PERSONAGGIO

- Creare un primo "personaggio"
 - può essere un cerchio, un rettangolo, o qualunque forma vogliate...
 - esempio: `rect()`, `ellipse()` ecc..
 - Provare a cambiare dimensione, colore ...
- Scegliamo ora un'immagine che vogliamo usare come personaggio
 - con la funzione `loadImage()` carichiamo l'immagine
 - con la funzione `image()` la usiamo come personaggio

CREAZIONE PERSONAGGIO



rettangolo, o qualunque forma vogliate...

se() ecc..

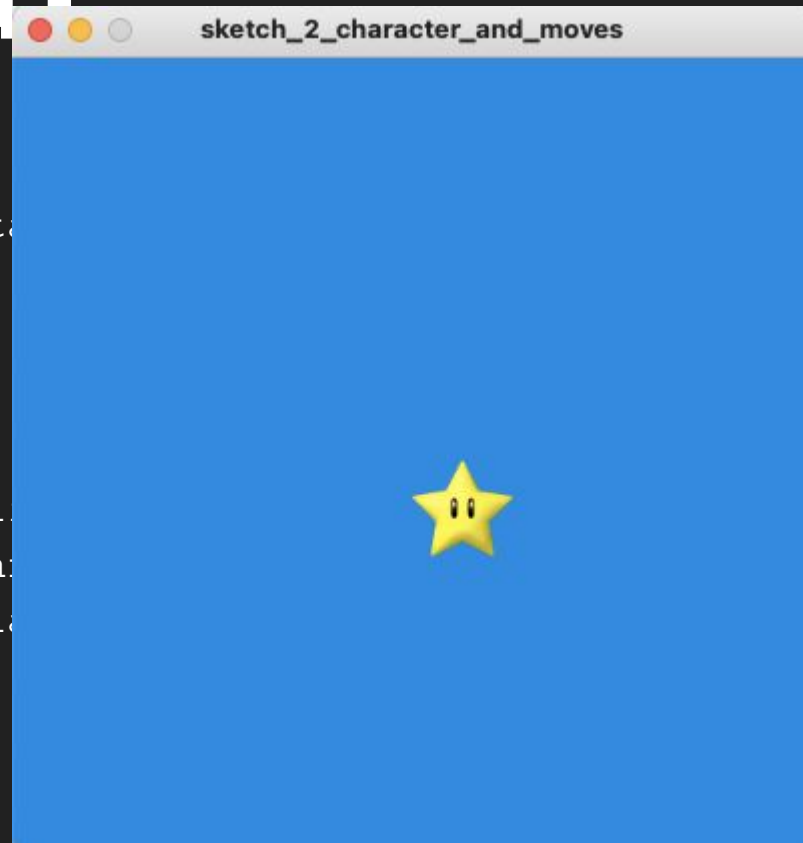
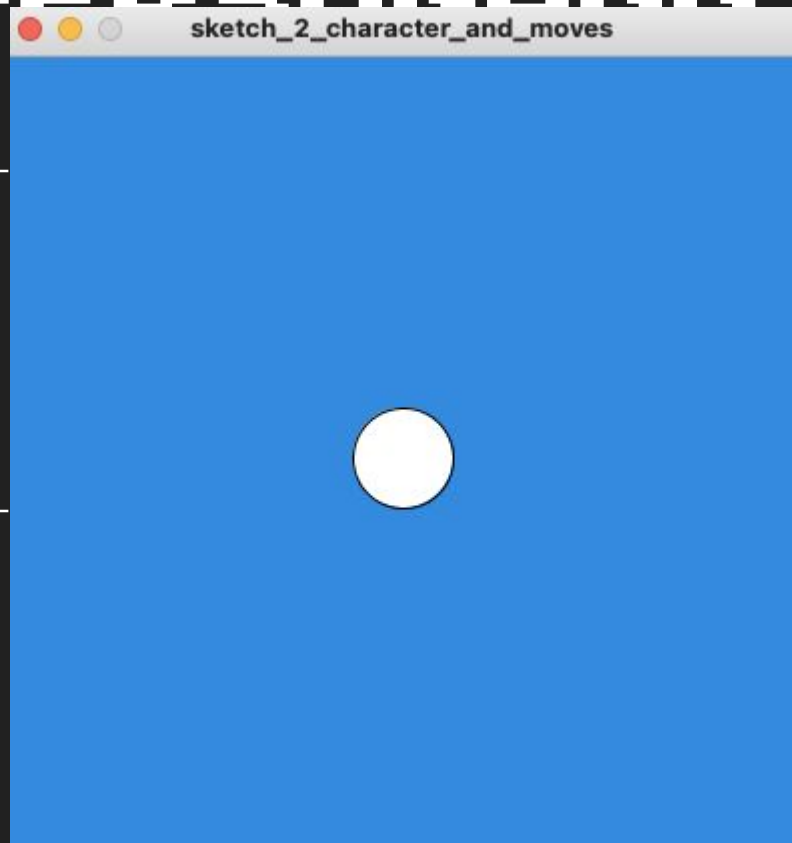
one, colore ...

vogliamo usare come personaggio

carichiamo l'immagine

usiamo come personaggio

CREAZIONE PERSONAGGIO



MOVIMENTI PT3

Ora cerchiamo di farlo muovere con la tastiera

Funzioni per l'evento:

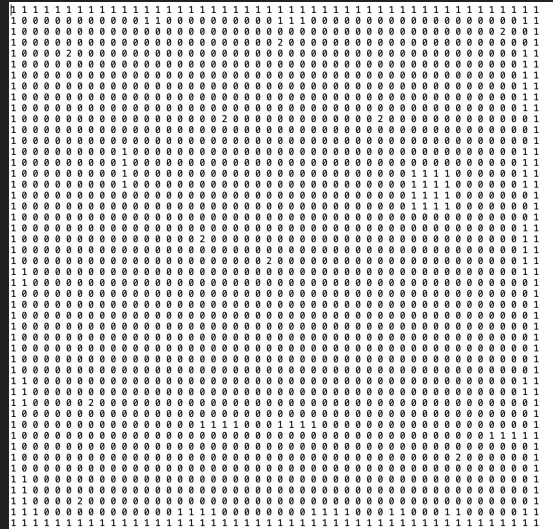
- **keyPressed()**
 - suggerimento: per controllare che tasto è stato premuto, usare:
 - `keyCode == UP`: freccette e simboli
 - `key == 'a'`: lettere e numeri
 - <https://py.processing.org/tutorials/interactivity/>

MAPPA E AMBIENTE

- **Input:** matrice generata dall' automa cellulare
- **Output:** rappresentazione grafica con Processing

MAPPA E AMBIENTE

- **Input:** matrice generata dall' automa cellulare
- **Output:** rappresentazione grafica con Processing



MAPPA E AMBIENTE

Lettura della **matrice**:

- 0 = spazio vuoto
- 1 = muri/oggetti rimovibili
- 2 = muri
- 3 = tesori da raccogliere

PERSONALIZZAZIONE E TEMI

- Personalizziamo il personaggio e l'ambiente
- Diversi temi
 - Scegliere 2 immagini per il personaggio
 - Scegliere 2 immagini per lo sfondo
 - Scegliere 2 immagini per i muri
 - Scegliere diversi tesori

PERSONALIZZAZIONE E TEMI

- Personalizziamo il personaggio e l'ambiente
- Diversi temi
 - Scegliere 2 immagini per il personaggio
 - Scegliere 2 immagini per lo sfondo
 - Scegliere 2 immagini per i muri
 - Scegliere diversi tesori

"La creatività è contagiosa. Trasmettila."

ALBERT EINSTEIN

ESEMPIO FINALE



ULTERIORI EVOLUZIONI

Algoritmica

- **Posizionamento iniziale** del giocatore
- Algoritmi per la **connessione** di aree della caverna irraggiungibili
- Introduzione di **Mostri**: Intelligenze artificiali (basilari) e pathfinding
-

Logica di gioco

- Aggiunta di un inventario/equipaggiamento
- Aggiungere più livelli e parametrizzare la loro difficoltà
- Aggiungere tesori/oggetti casuali e determinare regole per la generazione
-

Grazie per l'attenzione

