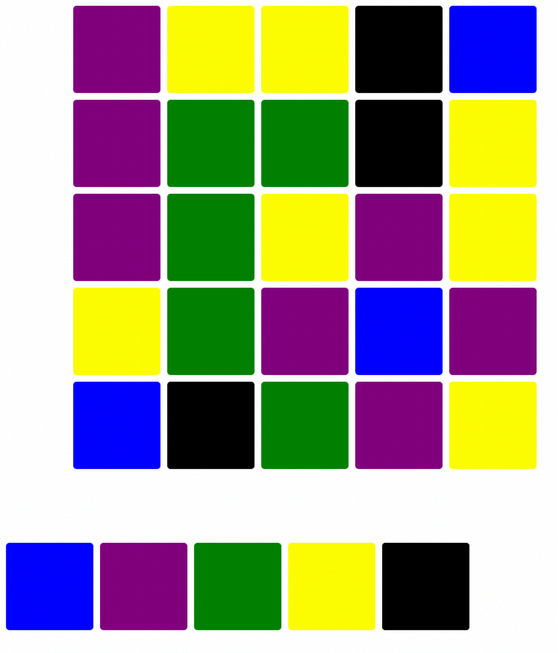
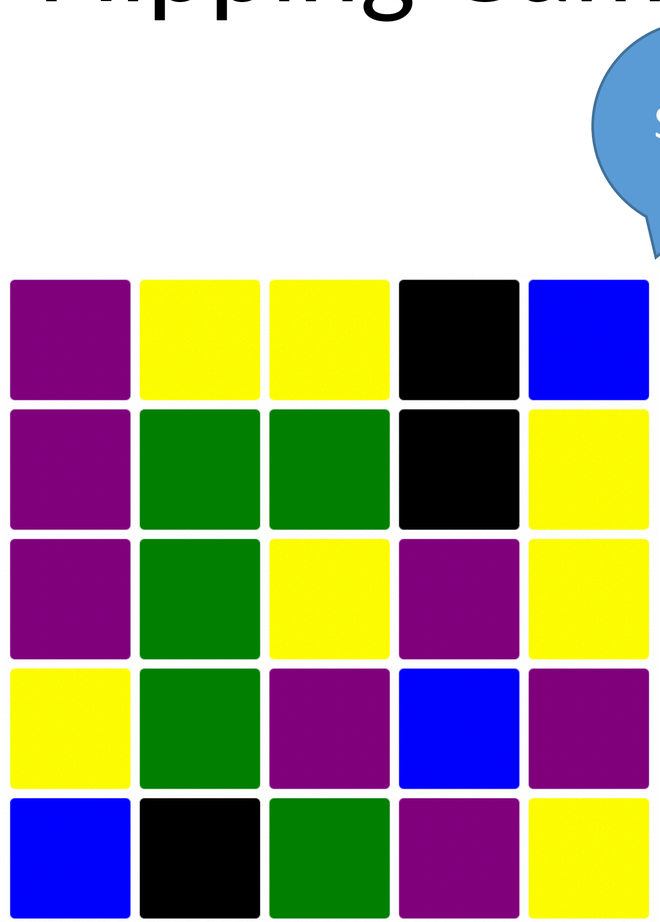


CSC 1002 Week 6 & 7

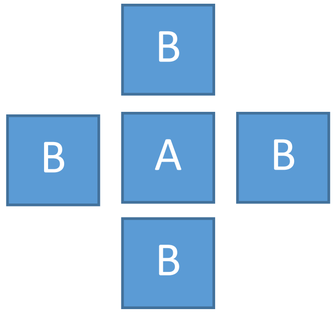
Flipping Color – Scope, Spec, Design, Implementation

Color-Flipping Game - GUI mode

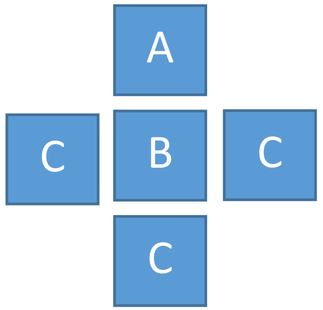


Connected Neighbors

- Two positions (A,B) are connected if the digits are identical and position of B is in one of the A's direct left, right, up or down position.



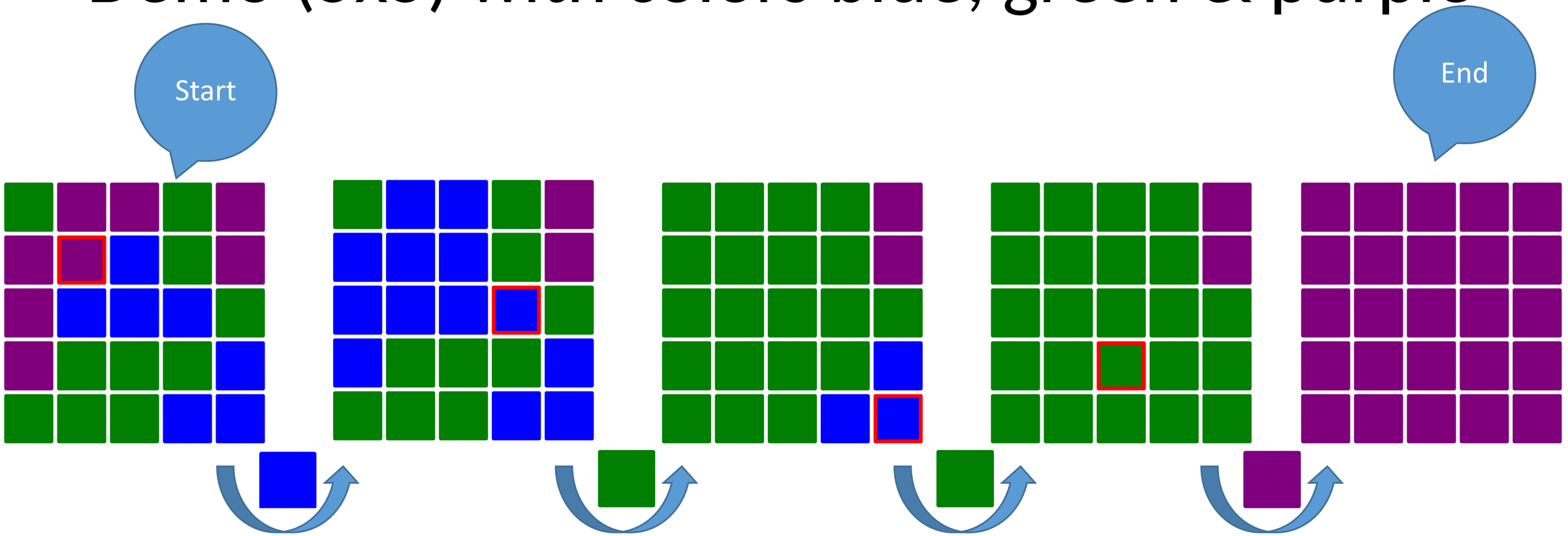
- Two positions (A,C) are connected if digits are identical and A and C are connected via other connected neighbors.



Number-Flipping Game (GUI, Event Based)

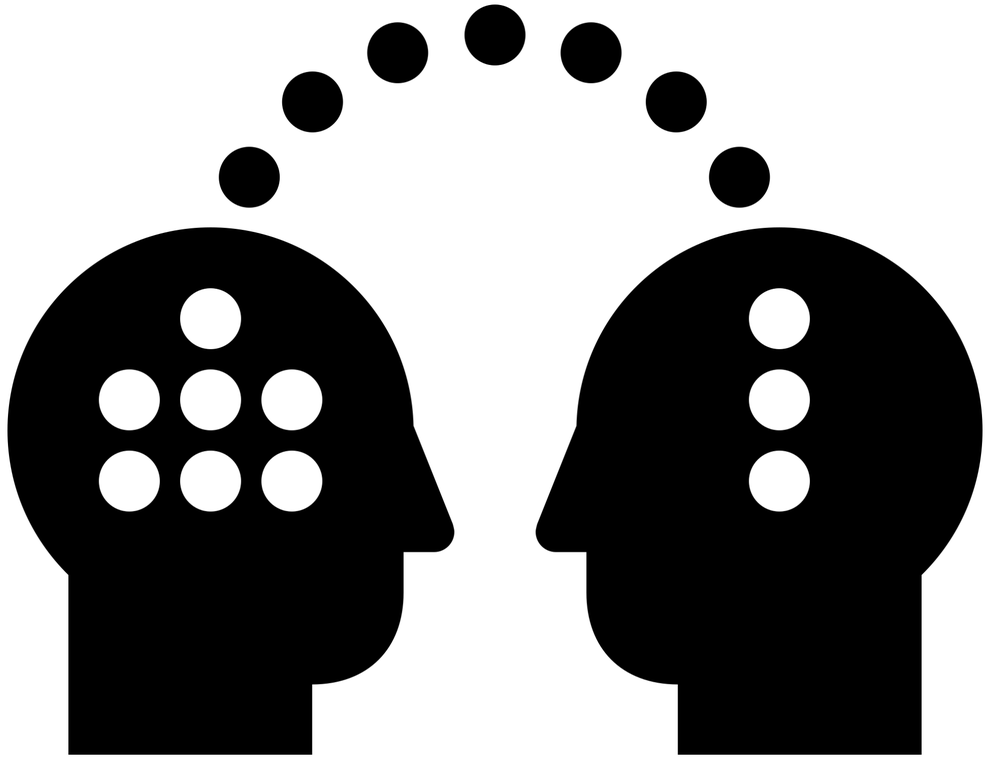
- A rectangular board of an NxN array of randomly chosen colors
- Goal: change all tiles to one single color
- Rule:
  - Player first “click” any color tile on the game board
  - Then choose a color from the color bar to change the colors of the connected neighbors.
  - After the change is completed player are free to choose another tile and another color
  - The process repeats until all tiles have the same color

Demo (5x5) with colors blue, green & purple



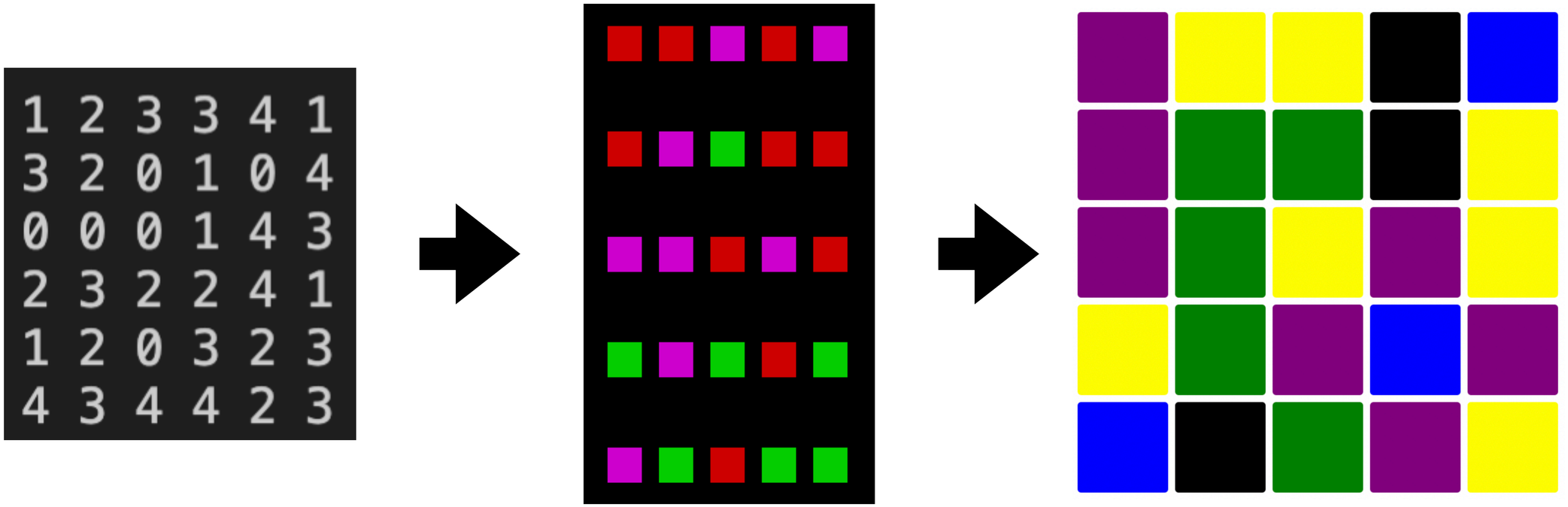
Show a demo

Goal #3 - Knowledge Transfer



Refactoring

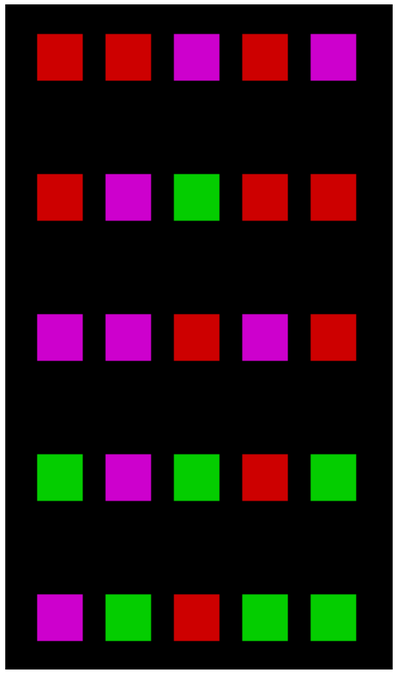
- Refactoring or Code Refactoring is defined as systematic process of improving existing computer code, without adding new functionality or changing external behaviour of the code.
- The goal of refactoring is NOT to add new functionality or remove an existing one.
- It is intended to enhance extensibility, maintainability, and readability of software without changing what it actually does.
- Why should we refactor our code when it works fine?
- We do refactor because we understand that getting design right in first time is hard and also you get the following benefits from refactoring:
  - Code size is often reduced
  - Confusing code is restructured into simpler code
  - Reduce repetitive code; better code reuse



Generalization - Refactoring

re-design + implementation

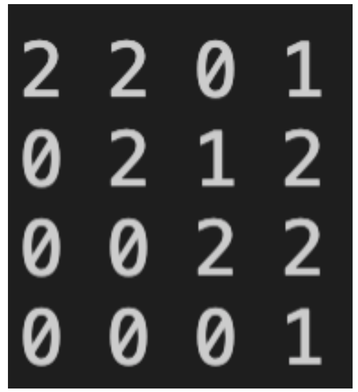
```
1 2 3 3 4 1
3 2 0 1 0 4
0 0 0 1 4 3
2 3 2 2 4 1
1 2 0 3 2 3
4 3 4 4 2 3
```



Demo - Console Color

Step 1 - Refactoring

Design – Data Model



- How do we model the game board ?

String

"2201021200220001"

List

[2, 2, 0, 1, 0, 2, 1, 2, 0, 0, 2, 2, 0, 0, 0, 1]

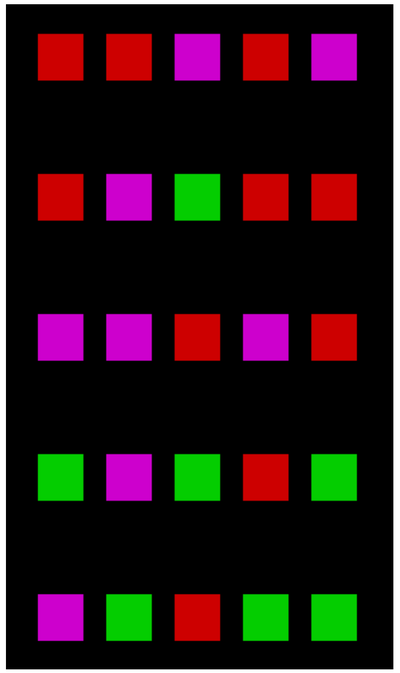
Nested List

[[2, 2, 0, 1], [0, 2, 1, 2], [0, 0, 2, 2], [0, 0, 0, 1]]

Dictionary

[0:[2, 2, 0, 1], 1:[0, 2, 1, 2], 2:[0, 0, 2, 2], 3:[0, 0, 0, 1]]

create_game(dim, digit_range)



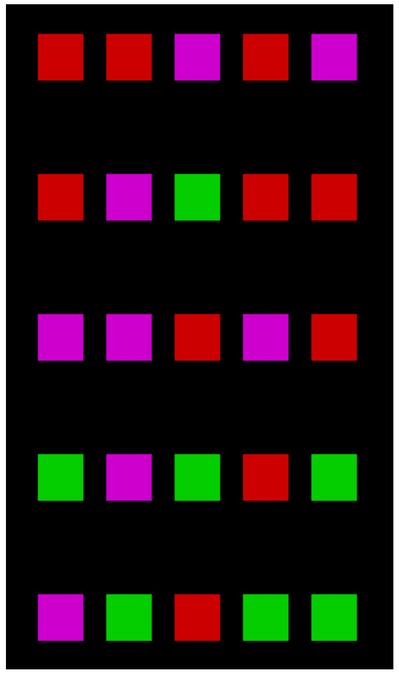
```
def create_game(dim, digit_range):
    b = []
    for _ in range(dim*dim):
        b.append( random.randint(0,digit_range) )
    return b
```

create_game(dim, digit_range) vs create_game(dim, data)

```
def create_game(dim, digit_range):
    b = []
    for _ in range(dim*dim):
        b.append( random.randint(0,digit_range) )
    return b
```

```
def create_game(dim, data):
    b = []
    for _ in range(dim*dim):
        b.append( random.sample(data,1)[0] )
    return b
```

prompt_player()



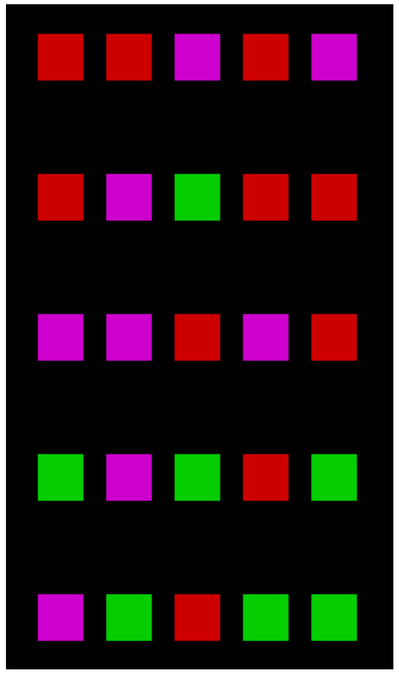
```
def prompt_player():
    while True:
        n = input('Enter number to flip to?')
        if n.isdigit() and len(n) == 1:
            break
    return int(n)
```

prompt_player() vs prompt_player(game, prompt)

```
def prompt_player():
    while True:
        n = input('Enter number to flip to?')
        if n.isdigit() and len(n) == 1:
            break
    return int(n)
```

```
def prompt_player(game, prompt):
    while True:
        n = input(prompt)
        if n in game:
            return n
```

refresh_screen(game)



```
def refresh_screen(game):
    print('')
    for r in range(g_dim):
        for c in range(g_dim):
            print(game[r*g_dim+c], end=' ')
        print('')
```

refresh_screen(game) vs refresh_screen(game, display_hdlr)

```
def refresh_screen(game):
    print('')
    for r in range(g_dim):
        for c in range(g_dim):
            print(game[r*g_dim+c], end=' ')
        print('')
```

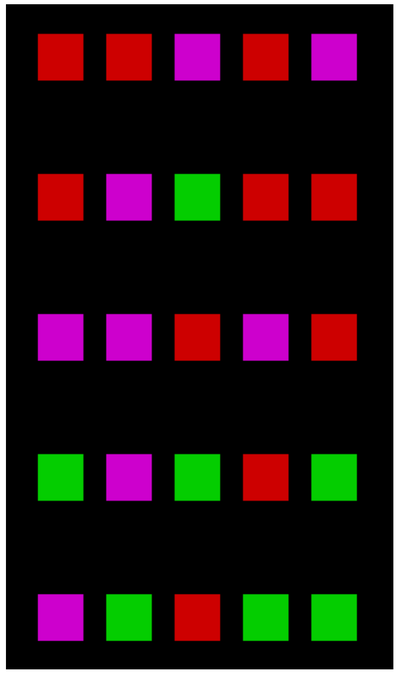
```
def refresh_screen(game, disp_hdlr):
    for r in range(g_dim):
        for c in range(g_dim):
            disp_hdlr(game, r, c)
```

Display handler - console_txt_hdlr(game, r, c)

```
def refresh_screen(game, disp_hdlr):
    for r in range(g_dim):
        for c in range(g_dim):
            disp_hdlr(game, r, c)
```

```
def console_txt_hdlr(game, r, c):
    '''
    display a single tile on the console
    '''
    print(game[r*g_dim+c], end=' ')
    if c == g_dim-1:
        print('')
```

Main Program



```
g_game = create_game(g_dim, g_digit_range)
while True:
    refresh_screen(g_game)
    n = prompt_player()
    flip_number(0, 0, g_game, g_game[0], n)
```

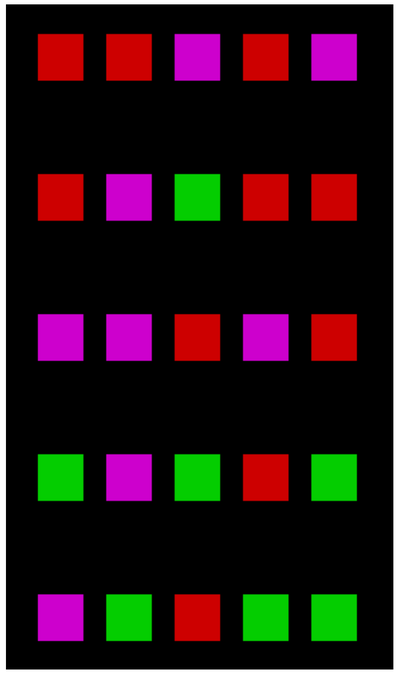
Main Program

```
g_game = create_game(g_dim, g_digit_range)
while True:
    refresh_screen(g_game)
    n = prompt_player()
    flip_number(0, 0, g_game, g_game[0], n)
```

```
g_game = create_game(g_dim, ['0','3','6'])
while True:
    refresh_screen(g_game, console_txt_hdlr)
    n = prompt_player(g_game, 'Enter a digit to flip to?')
    flip_number(0, 0, g_game, g_game[0], n)
```

Quick File Comparison

Demo - Console Generalized Version



Step 2 - Implementation (Color)

Demo - Text Color

start here

```
purple purple red green purple
purple green green red purple
red purple purple purple red
red purple green purple red
red red red green purple
Enter a color to flip to?green

green green red green purple
green green green red purple
red purple purple purple red
red purple green purple red
red red red green purple
Enter a color to flip to?red

red red red green purple
red red red red purple
red purple purple purple red
red purple green purple red
red red red green purple
Enter a color to flip to?purple
```

```
purple purple purple green purple
purple purple purple purple purple
purple purple purple purple red
purple purple green purple red
purple purple purple green purple
Enter a color to flip to?green

green green green green green
green green green green green
green green green green red
green green green green red
green green green green purple
Enter a color to flip to?red

red red red red red
red red red red red
red red red red red
red red red red red
red red red red purple
Enter a color to flip to?purple
```

```
purple purple purple purple purple
purple purple purple purple purple
purple purple purple purple purple
purple purple purple purple purple
purple purple purple purple purple
```

Using Escape Sequence to print text in color

```
reset = '\033[0m'
bold = '\033[01m'
disable = '\033[02m'
underline = '\033[04m'
reverse = '\033[07m'
strikethrough = '\033[09m'
invisible = '\033[08m'
```

foreground

```
black = '\033[30m'
red = '\033[31m'
green = '\033[32m'
orange = '\033[33m'
blue = '\033[34m'
purple = '\033[35m'
cyan = '\033[36m'
lightgrey = '\033[37m'
darkgrey = '\033[90m'
lightred = '\033[91m'
lightgreen = '\033[92m'
yellow = '\033[93m'
lightblue = '\033[94m'
pink = '\033[95m'
lightcyan = '\033[96m'
```

background

```
black = '\033[40m'
red = '\033[41m'
green = '\033[42m'
orange = '\033[43m'
blue = '\033[44m'
purple = '\033[45m'
cyan = '\033[46m'
lightgrey = '\033[47m'
```

Demo - Using Escape Sequence

```
reset = '\033[0m'
bg_red = '\033[41m'
bg_green = '\033[42m'
fg_red = '\033[31m'
```

```
print(f'{bg_red}Hello World!{reset}\n')

print(f'{bg_red}      {reset}\n')

print(f'{fg_red}{bg_green}Hello World!{reset}')
```

Display handler - console_color_hdlr(game, r, c)

```
g_color = {
        'red':'\033[41m',
        'green':'\033[42m',
        'orange':'\033[43m',
        'blue':'\033[44m',
        'purple':'\033[45m',
        'reset':'\033[0m'
    }
```

```
def console_color_hdlr(game, r, c):
    '''
    display a single tile on the console
    '''
    idx = r*g_dim + c
    color = g_color[game[idx]]
    reset = g_color["reset"]
    print(f'{color}  {reset}', end=' ')
    if c == g_dim-1:
        print('\n\n')
```

Game Processing Logic

Flipping Digit

```
g_game = create_game(g_dim, ['0','3','6'])
while True:
    refresh_screen(g_game, console_txt_hdlr)
    n = prompt_player(g_game, 'Enter a digit to flip to?')
    flip_number(0, 0, g_game, g_game[0], n)
```

Flipping Color

```
g_game = create_game(g_dim, ['red','green','purple'])
while True:
    refresh_screen(g_game, console_color_hdlr)
    n = prompt_player(g_game, 'Enter a color to flip to?')
    flip_number(0, 0, g_game, g_game[0], n)
```

Combining digit and color into one function

```
def play_game(data, prompt, hdlr):
    g_game = create_game(g_dim, data)
    while True:
        refresh_screen(g_game, hdlr)
        n = prompt_player(g_game, prompt)
        flip_number(0, 0, g_game, g_game[0], n)
```

```
play_game(['red','green','purple'], \
    'Enter a color to flip to?', \
     console_color_hdlr)
```

```
play_game(['0','3','6'], \
    'Enter a digit to flip to?', \
     console_txt_hdlr)
```

Design Process - Architecture

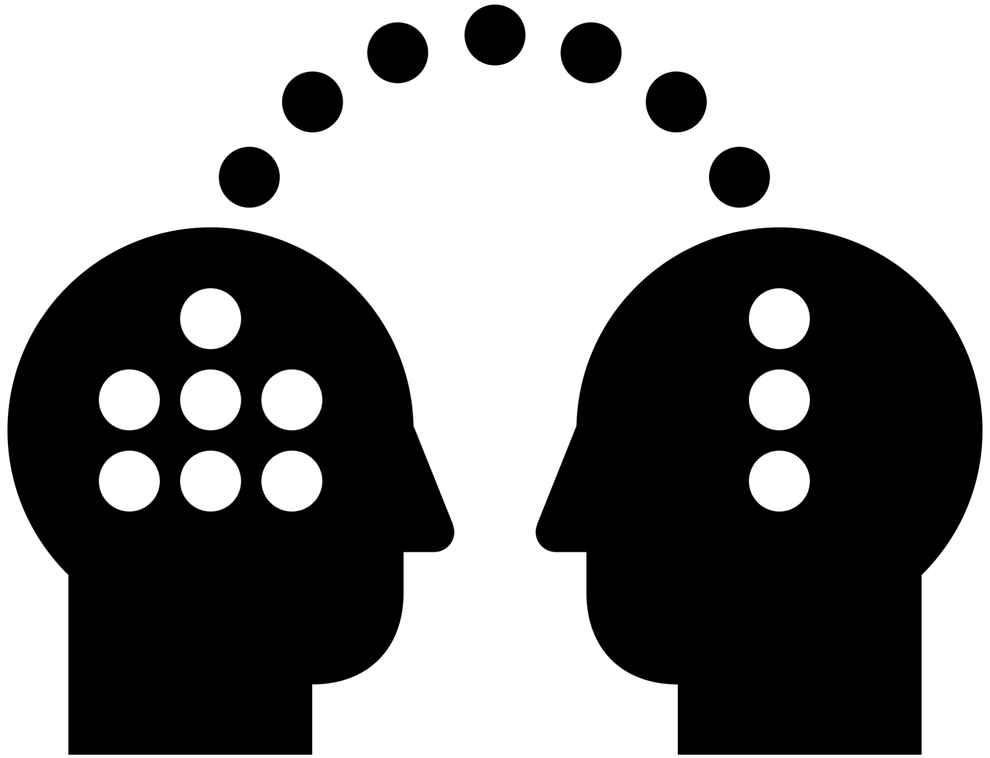
Design Principles

- Design is On-going Process, a decision-making process to finding a Trade-off between technical and bussiness decisions (performance, throughput, time, resource, skills, opportunity cost, market timing, …etc)
- A few design principles
  - Seperate what varies from what static (Decoupling)
    - Identify the aspects of your application that vary and separate them from what stays the same.
  - Program to interfaces, not implementation
  - Single Responsibility
    - seperate or combine: get_correct_cnt(), get_misplaced_cnt()

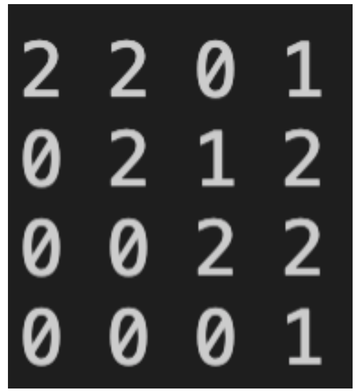
Quick File Comparison

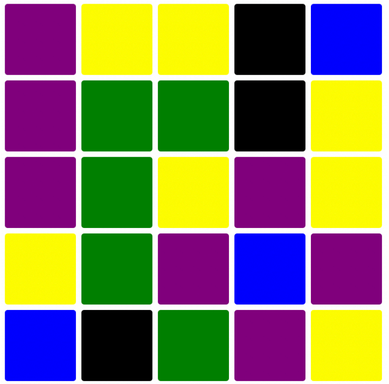
Implementation - GUI

Goal #3 - Knowledge Transfer



Design – Data Model





- How do we model the game board ?

String

"2201021200220001"

Flatted List

[2, 2, 0, 1, 0, 2, 1, 2, 0, 0, 2, 2, 0, 0, 0, 1]

Nested List

[[2, 2, 0, 1], [0, 2, 1, 2], [0, 0, 2, 2], [0, 0, 0, 1]]

Dictionary

[0:[2, 2, 0, 1], 1:[0, 2, 1, 2], 2:[0, 0, 2, 2], 3:[0, 0, 0, 1]]

Function Design

Parameters (Input) and Return (Output)

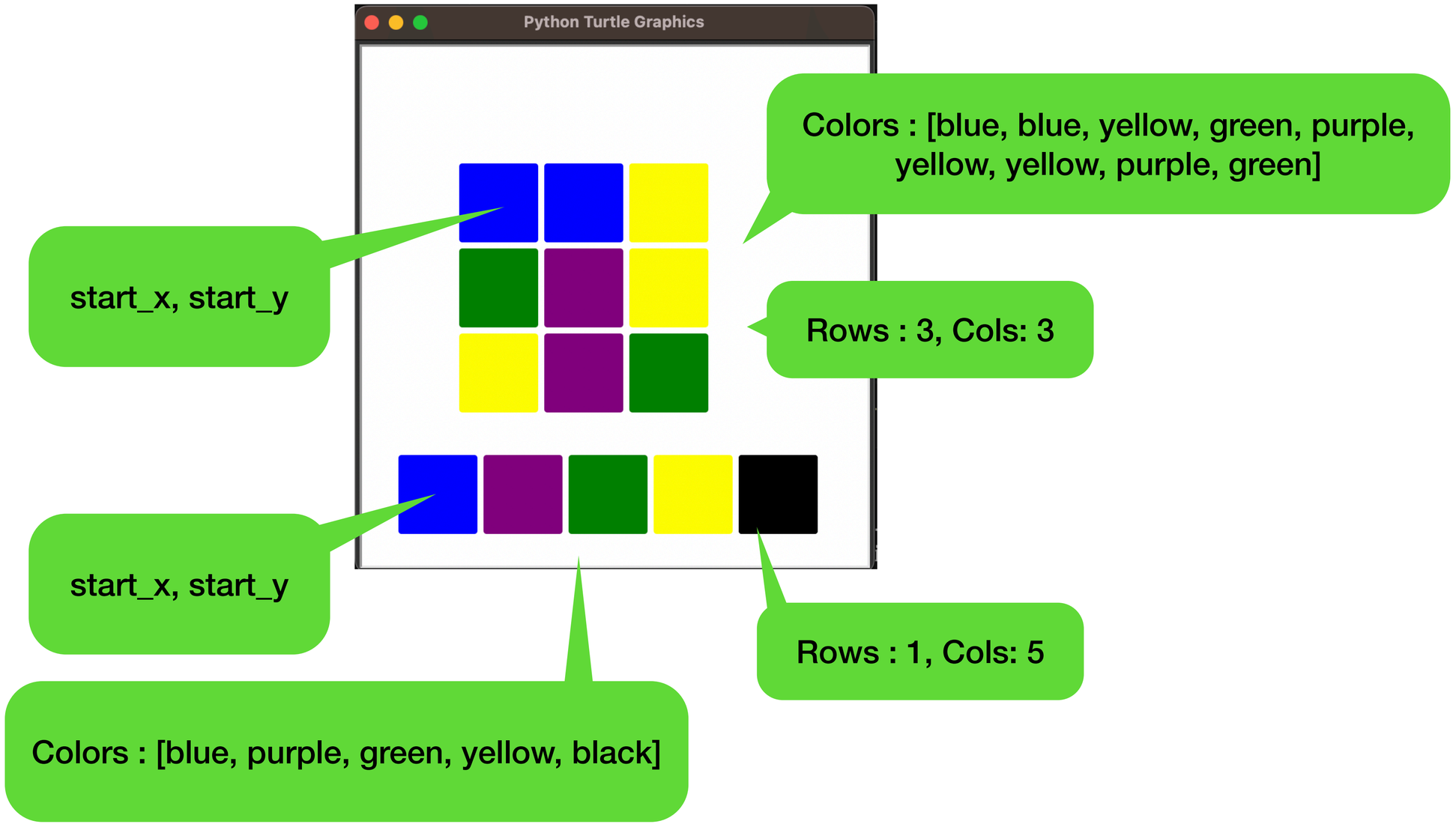
Single Responsibilty

Decoupling

- From Console Implementation:
  - create_game(), refresh_screen(), flip_number(), console_hdlr()
- display_tiles(start_x, start_y, rows, cols) -> list[turtle]
- create_a_tile(size, border) -> turtle
- set_mouse_click(tiles:list[turtle], type) -> none
- on_mouse_click(type, idx, x, y)
- gui_hdlr(game, x, y)

Encapsulate What Varies

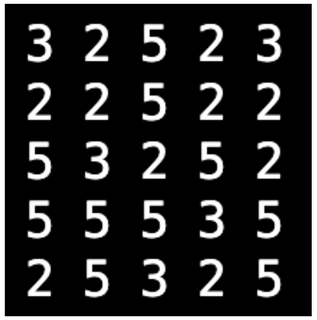
display_tiles(start_x, start_y, colors: list[str], rows, cols) -> list[turtle]

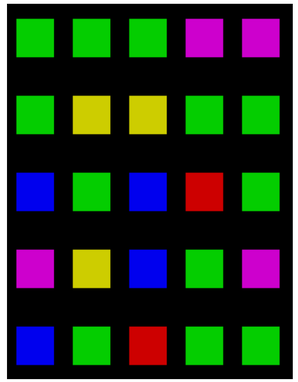


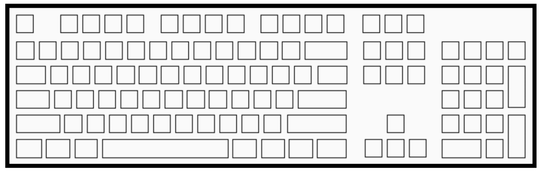
Game Setup - GUI

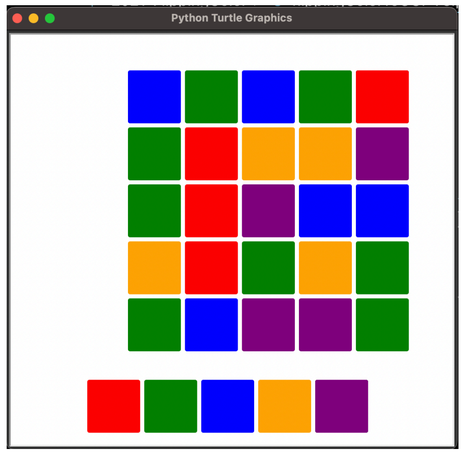
```
s = turtle.Screen()
s.setup(600,600)

colors = ['red','green','blue','orange','purple']

g_game = create_game(g_dim, colors)
g_tiles = display_tiles(-100, -50, g_dim, g_dim)
g_color_bar = display_tiles(-150, -150, 1, len(colors))

setMouseClick(TYPE_TILE, g_tiles)
setMouseClick(TYPE_BAR, g_color_bar)

refresh_screen(g_game, gui_color_hdlr)
refresh_screen(colors, gui_bar_hdlr, 1, len(g_color_bar))

turtle.Screen().mainloop()
```

Summary









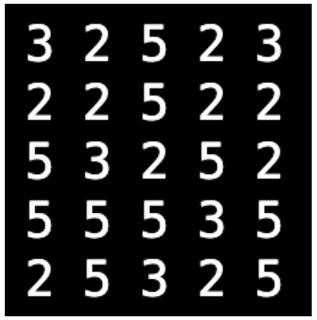
Relationship ???

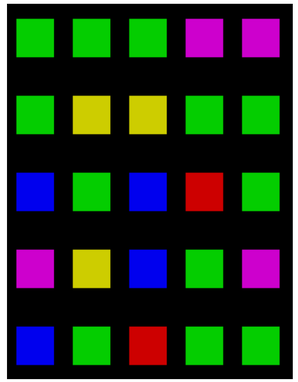
```
g_dim = 5
g_game = []
g_color = {
        'red':'\033[41m',
        'green':'\033[42m',
        'orange':'\033[43m',
        'blue':'\033[44m',
        'purple':'\033[45m',
        'reset':'\033[0m'
    }

g_tiles = None
g_color_bar = None
g_selected_tile = None
TYPE_TILE = 'TILE'
TYPE_BAR = 'BAR'
```

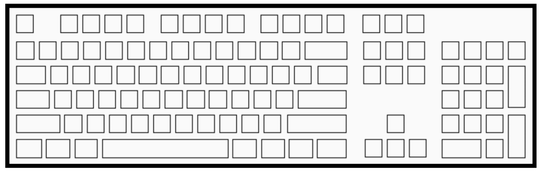
Problem Decomposition

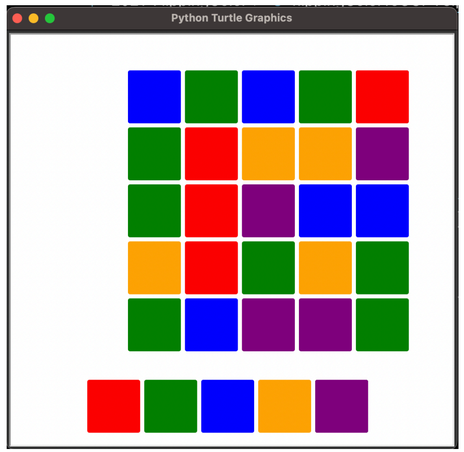
Main Processing Logic

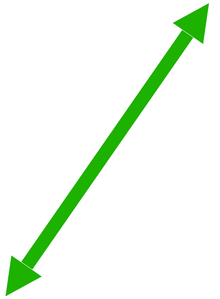
Summary











```
g_dim = 5
g_game = []
g_color = {
        'red':'\033[41m',
        'green':'\033[42m',
        'orange':'\033[43m',
        'blue':'\033[44m',
        'purple':'\033[45m',
        'reset':'\033[0m'
    }

g_tiles = None
g_color_bar = None
g_selected_tile = None
TYPE_TILE = 'TILE'
TYPE_BAR = 'BAR'
```

Problem Decomposition

Main Processing Logic

refresh_screen - further refactoring

- Unlike the console based versions (digit and color), the entire game is refreshed.
- However, it’s true for the console based versions.
- It’s not NOT required for GUI based version.
- flip_number() could be further Refactored to return the changes stored in a global variables
  - Refactor the display handler to update the changes as needed

The End – Thank You