C1A_grid.py

```
# ================================================================================
"""GRID : create some grids with several color patterns"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
# ================================================================================
from ezTK import *
from random import randrange
# --------------------------------------------------------------------------------
def stripes(width:int=400, height:int=400, rows:int=40) -> None:
  """alternate black and white horizontal stripes"""
  win = Win(title='STRIPES') # create main window
  height = height // rows # set height for a single stripe
  colors = ('#000','#FFF') # store stripe colors in a tuple
  # ------------------------------------------------------------------------------
  for row in range(rows): # loop over the stripes
    color = colors[row % 2] # set color for new stripe (cycle on 2 colors)
    Brick(win, bg=color, width=width, height=height) # create new stripe
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def colorstripes(width:int=400, height:int=400, rows:int=40) -> None:
  """alternate color stripes using six colors"""
  win = Win(title='COLORSTRIPES') # create main window
  height = height // rows # set height for a single stripe
  colors = ('#F00','#0F0','#00F','#0FF','#F0F','#FF0') # store stripe colors
  # ------------------------------------------------------------------------------
  for row in range(rows): # loop over stripes
    color = colors[row % 6] # set color for new stripe (cycle on 6 colors)
    Brick(win, bg=color, width=width, height=height) # create new stripe
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def chessboard(width:int=400, height:int=400, cols:int=8, rows:int=8) -> None:
  """create black and white chessboard"""
  win = Win(title='CHESSBOARD', fold=cols) # set fold property to number of cols
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#000','#FFF') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    color = colors[(row+col) % 2] # set color for new cell (cycle on 2 colors)
    Brick(win, bg=color, height=height, width=width) # create new cell
  # ------------------------------------------------------------------------------
  # Alternative version using two nested loops on rows and cols
  # for row in range(rows): # loop over board rows
  #   for col in range(cols): # loop over board cols
  #     color = colors[(row+col) % 2] # set color for cell (cycle on 2 colors)
  #     Brick(win, bg=color, height=height, width=width) # create new cell
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def colorboard(width:int=400, height:int=400, cols:int=8, rows:int=8) -> None:
  """create six color checkerboard"""
  win = Win(title='COLORBOARD', fold=cols, op=2) # set 2-pixel outer padding
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#F00','#0F0','#00F','#0FF','#F0F','#FF0') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    color = colors[(row+col) % 6] # set color for new cell (cycle on 6 colors)
    Brick(win, bg=color, height=height, width=width, border=3) # create new cell
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def boxes(width:int=400, height:int=400, cols:int=13, rows:int=13) -> None:
  """create concentric black and white boxes"""
  win = Win(title='GRID', fold=cols) # set fold property to number of cols
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#000','#FFF') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over the cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    # for each cell, compute its distance to the topmost row, bottommost row,
    # leftmost col and rightmost col, then keep the smallest of the 4 distances
    distance = min(row, rows-row-1, col, cols-col-1)
    color = colors[distance % 2] # set color for new cell (cycle on 2 colors)
    Brick(win, bg=color, height=height, width=width) # create new cell
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def colorboxes(width:int=400, height:int=400, rows:int=13, cols:int=13) -> None:
  """create concentric color boxes"""
  win = Win(title='GRID', fold=cols) # set fold property to number of cols
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#F00','#0F0','#00F','#0FF','#F0F','#FF0') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    # for each cell, compute its distance to the topmost row, bottommost row,
    # leftmost col and rightmost col, then keep the smallest of the 4 distances
    distance = min(row, rows-row-1, col, cols-col-1)
    color = colors[distance % 6] # set color for new cell (cycle on 6 colors)
    Brick(win, bg=color, height=height, width=width) # create new cell
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def ballboard(width:int=600, height:int=600, cols:int=12, rows:int=12) -> None:
  """create board containing one random color ball for each cell"""
  win = Win(title='BALLBOARD', fold=cols, bg='#000') # set win background as b……
……lack
  width, height = width // cols, height // rows # set size for a single cell
  images = tuple(Image(f"Z{c}.gif") for c in 'RGBCMY') # store images for cells
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over the board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    image = images[randrange(6)] # set random image for new cell
    Brick(win, image=image, height=height, width=width) # create new cell
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# ================================================================================
if __name__ == "__main__":
  stripes()
  stripes(800, 600, 200) # do not use default values
  colorstripes()
  chessboard()
  colorboard()
  boxes()
  colorboxes()
  ballboard()
# ================================================================================
```

C1B_grid.py

```
# ================================================================================
"""GRID : create some grids with several color patterns"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # use multi-state widgets
__date__    = "2022-11-12"
# ================================================================================
from ezTK import *
from random import randrange
# --------------------------------------------------------------------------------
def stripes(width:int=400, height:int=400, rows:int=40) -> None:
  """alternate black and white horizontal stripes"""
  win = Win(title='STRIPES') # create main window
  height = height // rows # set height for a single stripe
  colors = ('#000','#FFF') # store stripe colors in a tuple
  # ------------------------------------------------------------------------------
  for row in range(rows): # loop over the stripes
    state = row % 2 # set state for new stripe (cycle on 2 states)
    Brick(win, bg=colors, state=state, width=width, height=height)
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def colorstripes(width:int=400, height:int=400, rows:int=40) -> None:
  """alternate color stripes using six colors"""
  win = Win(title='COLORSTRIPES') # create main window
  height = height // rows # set height for a single stripe
  colors = ('#F00','#0F0','#00F','#0FF','#F0F','#FF0') # store stripe colors
  # ------------------------------------------------------------------------------
  for row in range(rows): # loop over the stripes
    state = row % 6 # set state for new stripe (cycle on 6 states)
    Brick(win, bg=colors, state=state, width=width, height=height)
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def chessboard(width:int=400, height:int=400, cols:int=8, rows:int=8) -> None:
  """create black and white chessboard"""
  win = Win(title='CHESSBOARD', fold=cols) # set fold property to number of cols
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#000','#FFF') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over the board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    state = (row+col) % 2 # set state for new cell (cycle on 2 states)
    Brick(win, bg=colors, state=state, height=height, width=width)
  # ------------------------------------------------------------------------------
  # Alternative version using nested loops on rows and cols
#  for row in range(rows): # loop over board rows
#    for col in range(cols): # loop over board cols
#      state = (row+col) % 2 # set state for new cell (cycle on 2 states)
#      Brick(win, bg=colors, state=state, height=height, width=width)
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def colorboard(width:int=400, height:int=400, cols:int=8, rows:int=8) -> None:
  """create six color checkerboard"""
  win = Win(title='COLORBOARD', fold=cols, op=2) # set 2-pixel outer padding
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#F00','#0F0','#00F','#0FF','#F0F','#FF0') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over the board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    state = (row+col) % 6 # set state for new cell (cycle on 6 states)
    Brick(win, bg=colors, state=state, height=height, width=width, border=3)
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def boxes(width:int=400, height:int=400, cols:int=13, rows:int=13) -> None:
  """create concentric black and white boxes"""
  win = Win(title='BOXES', fold=cols) # set fold property to number of cols
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#000','#FFF') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over the board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    # for each cell, compute its distance to the topmost row, bottommost row,
    # leftmost col and rightmost col, then keep the smallest of the 4 distances
    distance = min(row, rows-row-1, col, cols-col-1)
    state = distance % 2 # set state for new cell (cycle on 2 states)
    Brick(win, bg=colors, state=state, height=height, width=width)
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def colorboxes(width:int=400, height:int=400, rows:int=13, cols:int=13) -> None:
  """create concentric color boxes"""
  win = Win(title='COLORBOXES', fold=cols) # fold is controlled by number of c……
……ols
  width, height = width // cols, height // rows # set size for a single cell
  colors = ('#F00','#0F0','#00F','#0FF','#F0F','#FF0') # store cell colors
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over the board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    # for each cell, compute its distance to the topmost row, bottommost row,
    # leftmost col and rightmost col, then keep the smallest of the 4 distances
    distance = min(row, rows-row-1, col, cols-col-1)
    state = distance % 6 # set state for new cell (cycle on 6 states)
    Brick(win, bg=colors, state=state, height=height, width=width)
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# --------------------------------------------------------------------------------
def ballboard(width:int=600, height:int=600, cols:int=12, rows:int=12) -> None:
  """create board containing one random color ball for each cell"""
  win = Win(title='BALLBOARD', fold=cols, bg='#000') # set win background as b……
……lack
  width, height = width // cols, height // rows # set size for a single cell
  images = tuple(Image(f"Z{c}.gif") for c in 'RGBCMY') # store images for cells
  # ------------------------------------------------------------------------------
  for loop in range(rows*cols): # loop over the board cells
    row, col = loop // cols, loop % cols # get cell coords by Euclidian division
    state = randrange(6) # set random state for new cell
    Brick(win, image=images, state=state, height=height, width=width)
  # ------------------------------------------------------------------------------
  win.loop() # start interaction loop
# ================================================================================
if __name__ == "__main__":
  stripes()
  stripes(800, 600, 200) # do not use default values
  colorstripes()
  chessboard()
  colorboard()
  boxes()
  colorboxes()
  ballboard()
# ================================================================================
```

C2A_counter.py

```
# ================================================================================
```

```
"""COUNTER : a user-controlled digital counter"""
# ============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
# ============================================================================
from ezTK import *
# ----------------------------------------------------------------------------
def main():
  """create the main window and pack the widgets"""
  global win; font1, font2 = 'Courier 16 bold', 'Arial 96 bold'
  win = Win(title='COUNTER', font=font1, op=5) # create main window
  # ------------------------------------------------------------------------
  frame = Frame(win) # create frame for the 7 buttons
  Button(frame, text='|<', width=5, command=lambda: on_but(0))
  Button(frame, text='<<', width=5, command=lambda: on_but(1))
  Button(frame, text='<',  width=5, command=lambda: on_but(2))
  Button(frame, text='0',  width=5, command=lambda: on_but(3))
  Button(frame, text='>',  width=5, command=lambda: on_but(4))
  Button(frame, text='>>', width=5, command=lambda: on_but(5))
  Button(frame, text='>|', width=5, command=lambda: on_but(6))
  Label(win, text=0, font=font2, border=2) # create counter widget
  # ------------------------------------------------------------------------
  win.counter = win[1] # friendly name for the counter widget
  win.loop() # start interaction loop
# ----------------------------------------------------------------------------
def on_but(index:int) -> None:
  """generic callback function for all seven buttons"""
  value = win.counter['text']    # get current counter value
  if   index == 0: value = -1000 # action for button: |<
  elif index == 1: value -= 10   # action for button: <<
  elif index == 2: value -= 1    # action for button: <
  elif index == 3: value = 0     # action for button: 0
  elif index == 4: value += 1    # action for button: >
  elif index == 5: value += 10   # action for button: >>
  elif index == 6: value = 1000  # action for button: >|
  win.counter['text'] = min(1000, max(-1000, value)) # clamp counter value
# ============================================================================
if __name__ == '__main__':
  main()
# ============================================================================
```

C2B_counter.py

```
# ============================================================================
"""COUNTER : a user-controlled digital counter"""
# ============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
# ============================================================================
from ezTK import *
# ----------------------------------------------------------------------------
def main():
  """create the main window and pack the widgets"""
  global win; font1, font2 = 'Courier 16 bold', 'Arial 96 bold'
  win = Win(title='COUNTER', font=font1, op=5) # create main window
  # ------------------------------------------------------------------------
  frame = Frame(win, font=font1) # create a frame for the 7 buttons widgets
  text = '|< << < 0 > >> >|'.split() # create a list for the 7 button strings
  for n in range(7): # loop to create the 7 control buttons
    Button(frame, text=text[n], width=5, command=lambda n=n: on_but(n))
  Label(win, text=0, font=font2, border=2) # create counter widget
  # ------------------------------------------------------------------------
  win.counter = win[1] # friendly name for the counter widget
  win.loop() # start interaction loop
# ----------------------------------------------------------------------------
def on_but(index:int) -> None:
  """generic callback function for all seven buttons"""
  slow, fast, minval, maxval = 1, 10, -1000, 1000 # set counter parameters
  value = win.counter['text'] # get current counter value
  # create a tuple containing all 7 new counter values, one for each button
  values = (minval, value-fast, value-slow, 0, value+slow, value+fast, maxval)
  value = values[index] # select counter value using the provided button index
  win.counter['text'] = min(maxval, max(minval, value)) # clamp counter value
# ============================================================================
if __name__ == '__main__':
  main()
# ============================================================================
```

C3_chrono.py

```
# ============================================================================
"""CHRONO : a user-controlled digital stopwatch"""
# ============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
# ============================================================================
from ezTK import *
# ----------------------------------------------------------------------------
def main():
  """create the main window and pack the widgets"""
  global win
  win = Win(title='CHRONO', op=5) # create main window
  font1, font2 = 'Arial 16', 'Times 120 bold' # define fonts for widgets
  # ------------------------------------------------------------------------
  frame = Frame(win, grow=False, font=font1) # create frame for buttons
  Button(frame, text=('START','STOP'), width=9, command=on_start) # multi state
  Button(frame, text='RESET', width=9, command=on_reset) # single state
  # ------------------------------------------------------------------------
  colors = ('#F00','#0F0','#00F','#0FF','#F0F','#FF0') # store colors for chrono
  Label(win, font=font2, width=5, fg=colors, border=2) # create chrono widget
  # ------------------------------------------------------------------------
  win.start, win.chrono = frame[0], win[1] # friendly names for widgets
  on_reset() # invoke callback for button 'RESET'
  win.loop() # start interaction loop
# ----------------------------------------------------------------------------
def on_reset() -> None:
  """callback function for the 'RESET' button"""
  win.chrono['text'] = win.chrono.state = 0 # reset value and state for chrono
# ----------------------------------------------------------------------------
def on_start() -> None:
  """callback function for the 'START/STOP' button"""
  win.start.state = 1-win.start.state # switch button state (state <--> 1-state)
  if win.start.state == 1: tick() # call 'tick' function when button state is 1
# ----------------------------------------------------------------------------
def tick() -> None:
  """increment counter and make recursive function call after 10ms"""
  win.chrono['text'] += 1 # increment counter value
  win.chrono.state = win.chrono['text']//50 # change chrono state every 50 steps
  if win.start.state == 1: win.after(10, tick) # call next tick after 10ms
# ============================================================================
if __name__ == '__main__':
  main()
# ============================================================================
```