

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

- 1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution
- 2 Handling Mouse Events
more about GUI programming
- 3 A GUI for Search
data attributes and layout
animating the search
percolation
- 4 Sliding Puzzles
rules of the game

MCS 275 Lecture 12
Programming Tools and File Management
Jan Verschelde, 8 February 2010

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

- 1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution
- 2 Handling Mouse Events
more about GUI programming
- 3 A GUI for Search
data attributes and layout
animating the search
percolation
- 4 Sliding Puzzles
rules of the game

8 Feb 2010

applying recursive backtracking

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

Goal: find a path in a maze (or labyrinth)
applying backtracking recursively, using

- `numpy` to represent the maze; and
- `Tkinter` for the GUI.

Two stages in the program:

- 1 modify a random distribution of blocked squares,
- 2 find a path through the maze.

Solution in two parts:

- 1 a plain interface for the essentials,
- 2 a GUI for a more enjoyable program.

applying recursive backtracking

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

Goal: find a path in a maze (or labyrinth)
applying backtracking recursively, using

- `numpy` to represent the maze; and
- `Tkinter` for the GUI.

Two stages in the program:

- 1 modify a random distribution of blocked squares,
- 2 find a path through the maze.

Solution in two parts:

- 1 a plain interface for the essentials,
- 2 a GUI for a more enjoyable program.

applying recursive backtracking

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

Goal: find a path in a maze (or labyrinth)
applying backtracking recursively, using

- `numpy` to represent the maze; and
- `Tkinter` for the GUI.

Two stages in the program:

- 1 modify a random distribution of blocked squares,
- 2 find a path through the maze.

Solution in two parts:

- 1 a plain interface for the essentials,
- 2 a GUI for a more enjoyable program.

8 Feb 2010

Making a Path

a Path through a Maze

running the program

generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

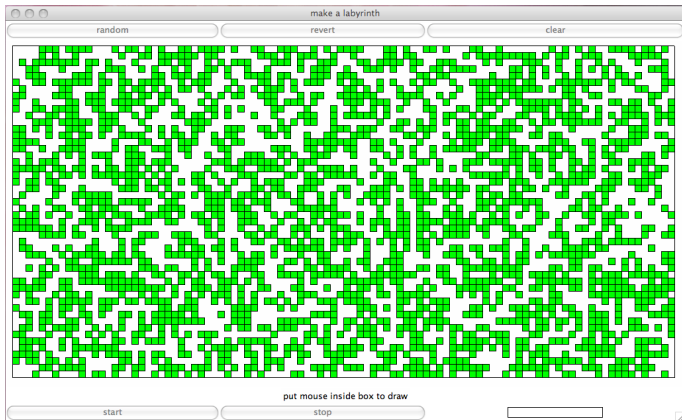
more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game



8 Feb 2010

Finding a Path

a Path through a Maze

running the program

generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

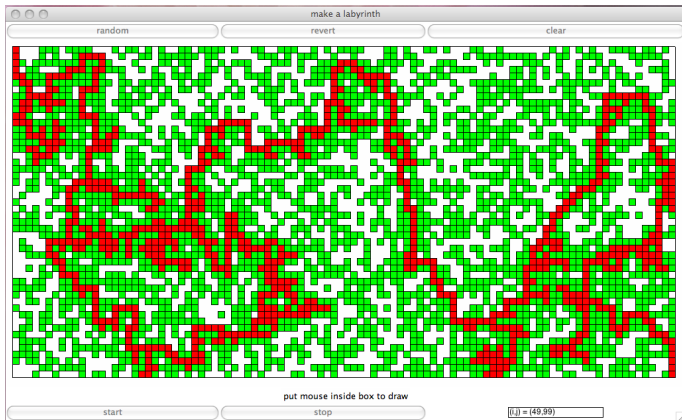
more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game



8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

- 1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution
- 2 Handling Mouse Events
more about GUI programming
- 3 A GUI for Search
data attributes and layout
animating the search
percolation
- 4 Sliding Puzzles
rules of the game

representing a maze

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

To represent the maze, we use a `numpy` matrix of *integer* entries.

Square $[i, j]$ of the maze can have value

- 1: blocked;
- 0: free and not visited on any path;
- k : visited as the k th square in a path.

By marking the visited squares we never walk along the same path and the walk terminates.

In a maze with n rows and m columns,

- a path starts at square $[0, 0]$; and
- $[n - 1, m - 1]$ is the destination of the path.

representing a maze

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

To represent the maze, we use a `numpy` matrix of *integer* entries.

Square $[i, j]$ of the maze can have value

- 1: blocked;
- 0: free and not visited on any path;
- k : visited as the k th square in a path.

By marking the visited squares we never walk along the same path and the walk terminates.

In a maze with n rows and m columns,

- a path starts at square $[0, 0]$; and
- $[n - 1, m - 1]$ is the destination of the path.

representing a maze

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

To represent the maze, we use a `numpy` matrix of *integer* entries.

Square $[i, j]$ of the maze can have value

- 1: blocked;
- 0: free and not visited on any path;
- k : visited as the k th square in a path.

By marking the visited squares we never walk along the same path and the walk terminates.

In a maze with n rows and m columns,

- a path starts at square $[0, 0]$; and
- $[n - 1, m - 1]$ is the destination of the path.

a random maze

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

```
from numpy import *
import random
```

```
def RandomMaze(rows,cols):
    """
    Returns a random integer matrix with dimensions
    in rows and cols, indicating free positions
    with 0 and occupied positions with -1.
    Free positions are at [0,0] and [rows-1,cols-1].
    """
    M = zeros((rows,cols),int)
    for i in range(0,rows):
        for j in range(0,cols):
            M[i,j] = random.randint(-1,0)
    M[0,0] = 0
    M[rows-1,cols-1] = 0
    return M
```

a random maze

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

```
from numpy import *  
import random
```

```
def RandomMaze(rows,cols):  
    """  
    Returns a random integer matrix with dimensions  
    in rows and cols, indicating free positions  
    with 0 and occupied positions with -1.  
    Free positions are at [0,0] and [rows-1,cols-1].  
    """  
    M = zeros((rows,cols),int)  
    for i in range(0,rows):  
        for j in range(0,cols):  
            M[i,j] = random.randint(-1,0)  
    M[0,0] = 0  
    M[rows-1,cols-1] = 0  
    return M
```

adjusting a random maze

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

```
def MakeMaze(rows,cols):
    """
    Returns a maze of given rows and columns.
    """
    M = RandomMaze(rows,cols)
    while True:
        PrintMaze(M)
        i = input('to change, give i (-1 if not) : ')
        if i < 0: break
        j = input('to change, give j (-1 if not) : ')
        if j < 0: break
        if M[i,j] == -1:
            M[i,j] = 0
        else:
            M[i,j] = -1
    return M
```

adjusting a random maze

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

```
def MakeMaze(rows,cols):
    """
    Returns a maze of given rows and columns.
    """
    M = RandomMaze(rows,cols)
    while True:
        PrintMaze(M)
        i = input('to change, give i (-1 if not) : ')
        if i < 0: break
        j = input('to change, give j (-1 if not) : ')
        if j < 0: break
        if M[i,j] == -1:
            M[i,j] = 0
        else:
            M[i,j] = -1
    return M
```

adjusting a random maze

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

```
def MakeMaze(rows,cols):
    """
    Returns a maze of given rows and columns.
    """
    M = RandomMaze(rows,cols)
    while True:
        PrintMaze(M)
        i = input('to change, give i (-1 if not) : ')
        if i < 0: break
        j = input('to change, give j (-1 if not) : ')
        if j < 0: break
        if M[i,j] == -1:
            M[i,j] = 0
        else:
            M[i,j] = -1
    return M
```

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

- 1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution
- 2 Handling Mouse Events
more about GUI programming
- 3 A GUI for Search
data attributes and layout
animating the search
percolation
- 4 Sliding Puzzles
rules of the game

a recursive solution

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

We define a function `Search`.

On input are (M, i, j, k) :

- M : the matrix representing the maze,
- (i, j) : next free position in M , i and j are indices respectively to a row and a column of M ,
- k : counter for square (i, j) visited along a path.

$\Rightarrow (i, j)$ will be the k th visited square along a path

On output are (M, b) :

- M : visited squares marked with increasing numbers,
- b : boolean is `True` if destination reached, `False` if stepping on (i, j) did not lead to the destination.

a recursive solution

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

We define a function `Search`.

On input are (M, i, j, k) :

- M : the matrix representing the maze,
- (i, j) : next free position in M , i and j are indices respectively to a row and a column of M ,
- k : counter for square (i, j) visited along a path.

$\Rightarrow (i, j)$ will be the k th visited square along a path

On output are (M, b) :

- M : visited squares marked with increasing numbers,
- b : boolean is `True` if destination reached, `False` if stepping on (i, j) did not lead to the destination.

a recursive solution

a Path through
a Maze

running the program
generating and
adjusting a matrix

a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

We define a function `Search`.

On input are (M, i, j, k) :

- M : the matrix representing the maze,
- (i, j) : next free position in M , i and j are indices respectively to a row and a column of M ,
- k : counter for square (i, j) visited along a path.

$\Rightarrow (i, j)$ will be the k th visited square along a path

On output are (M, b) :

- M : visited squares marked with increasing numbers,
- b : boolean is `True` if destination reached, `False` if stepping on (i, j) did not lead to the destination.

search for directions

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

There are four directions to go:

	$(i - 1, j)$	
$(i, j - 1)$	(i, j)	$(i, j + 1)$
	$(i + 1, j)$	

subject to the following conditions:

- go to row $i - 1$, only if $i > 0$,
- go to row $i + 1$, only if $i < n - 1$, $n = \#rows$,
- go to column $j - 1$, only if $j > 0$,
- go to column $j + 1$, only if $j < m - 1$, $m = \#columns$,
- square must be free and not visited.

search for directions

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

There are four directions to go:

	$(i - 1, j)$	
$(i, j - 1)$	(i, j)	$(i, j + 1)$
	$(i + 1, j)$	

subject to the following conditions:

- go to row $i - 1$, only if $i > 0$,
- go to row $i + 1$, only if $i < n - 1$, $n = \#rows$,
- go to column $j - 1$, only if $j > 0$,
- go to column $j + 1$, only if $j < m - 1$, $m = \#columns$,
- square must be free and not visited.

8 Feb 2010

a Path through
a Mazerunning the program
generating and
adjusting a matrix
a recursive solutionHandling
Mouse Eventsmore about GUI
programmingA GUI for
Searchdata attributes and
layout
animating the search
percolationSliding
Puzzles

rules of the game

the base case

```
def Search(M,i,j,k):
    """
    Search for a path starting at (i,j) in M.
    Markes visited tiles with increasing numbers.
    Returns M and a boolean indicating the arrival.
    """
    M[i,j] = k
    if i == M.shape[0] - 1 and j == M.shape[1] - 1:
        return (M,True)
    else:
        ...
```

a Path through
a Maze

running the program
generating and
adjusting a matrix

a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

backtracking calls

```
else:
```

```
    b = False
```

```
    if i < M.shape[0]-1:
```

```
        if M[i+1,j] == 0: (M,b) = Search(M,i+1,j,k+1)
```

```
    if not b and j < M.shape[1]-1:
```

```
        if M[i,j+1] == 0: (M,b) = Search(M,i,j+1,k+1)
```

```
    if not b and i > 0:
```

```
        if M[i-1,j] == 0: (M,b) = Search(M,i-1,j,k+1)
```

```
    if not b and j > 0:
```

```
        if M[i,j-1] == 0: (M,b) = Search(M,i,j-1,k+1)
```

```
    return (M,b)
```

a Path through
a Maze

running the program
generating and
adjusting a matrix

a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

backtracking calls

```
else:
    b = False
    if i < M.shape[0]-1:
        if M[i+1,j] == 0: (M,b) = Search(M,i+1,j,k+1)
    if not b and j < M.shape[1]-1:
        if M[i,j+1] == 0: (M,b) = Search(M,i,j+1,k+1)
    if not b and i > 0:
        if M[i-1,j] == 0: (M,b) = Search(M,i-1,j,k+1)
    if not b and j > 0:
        if M[i,j-1] == 0: (M,b) = Search(M,i,j-1,k+1)
    return (M,b)
```

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution

2 Handling Mouse Events
more about GUI programming

3 A GUI for Search
data attributes and layout
animating the search
percolation

4 Sliding Puzzles
rules of the game

Handling Mouse Events

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

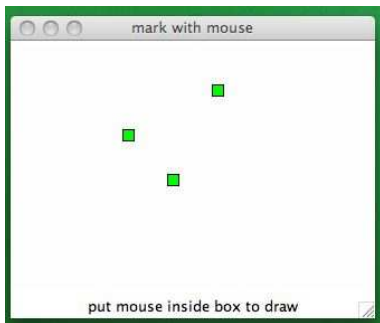
data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

Often the amount of data we generate is too huge for an orderly in a classical terminal window.

Much more data can be stored in an image on canvas and via the mouse we may interact with the data.



Filling Squares on a Grid

with the mouse

```

from Tkinter import *
from numpy import *

class FillSquares():
    """
    filling squares on canvas with mouse clicks
    """

    def __init__(self,wdw,r,c):
        """
        the mouse is bound to the canvas
        a label displays mouse position
        """

        wdw.title("mark with mouse")
        self.mag = 10 # magnification factor
        self.rows = r # number of rows on canvas
        self.cols = c # number of columns on canvas

```

__init__ continueda Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

```
def __init__(self,wdw,r,c):
    ....
    self.c = Canvas(wdw,\
                    width=self.mag*self.cols+2*self.mag,\
                    height = self.mag*self.rows+2*self.mag,\
                    bg='white')
    self.c.grid(row=1,column=0,columnspan=3)
    # to display mouse position :
    self.MousePosition = StringVar()
    self.MousePosition.set\
        ("put mouse inside box to draw")
    self.PositionLabel = Label(wdw, \
                               textvariable = self.MousePosition)
    self.PositionLabel.grid(row=2,column=0,\
                             columnspan=3)
    # bind mouse events
    self.BindMouseEvents()
    self.filled = zeros((r,c),bool)
```

__init__ continueda Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

```
def __init__(self,wdw,r,c):
    ....
    self.c = Canvas(wdw,\
                    width=self.mag*self.cols+2*self.mag,\
                    height = self.mag*self.rows+2*self.mag,\
                    bg='white')
    self.c.grid(row=1,column=0,columnspan=3)
    # to display mouse position :
    self.MousePosition = StringVar()
    self.MousePosition.set\
        ("put mouse inside box to draw")
    self.PositionLabel = Label(wdw, \
                               textvariable = self.MousePosition)
    self.PositionLabel.grid(row=2,column=0,\
                            columnspan=3)
    # bind mouse events
    self.BindMouseEvent()
    self.filled = zeros((r,c),bool)
```

__init__ continueda Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

```
def __init__(self,wdw,r,c):
    ....
    self.c = Canvas(wdw,\
                    width=self.mag*self.cols+2*self.mag,\
                    height = self.mag*self.rows+2*self.mag,\
                    bg='white')
    self.c.grid(row=1,column=0,columnspan=3)
    # to display mouse position :
    self.MousePosition = StringVar()
    self.MousePosition.set\
        ("put mouse inside box to draw")
    self.PositionLabel = Label(wdw, \
                               textvariable = self.MousePosition)
    self.PositionLabel.grid(row=2,column=0,\
                             columnspan=3)
    # bind mouse events
    self.BindMouseEvents()
    self.filled = zeros((r,c),bool)
```

Binding the Mouse Events

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

To bind the mouse events to canvas:

```
def BindMouseEvent(self):
    """
    binds mouse events to the canvas
    """
    self.c.bind("<Button-1>", self.ButtonPressed)
    self.c.bind("<ButtonRelease-1>", \
                self.ButtonReleased)
    self.c.bind("<Enter>", self.EnteredWindow)
    self.c.bind("<Leave>", self.ExitedWindow)
    self.c.bind("<B1-Motion>", self.MouseDragged)
```

The methods `ButtonPressed`, `ButtonReleased`, `EnteredWindow`, `ExitedWindow`, and `MouseDragged` are activated by the mouse.

8 Feb 2010

a Path through
a Mazerunning the program
generating and
adjusting a matrix
a recursive solutionHandling
Mouse Eventsmore about GUI
programmingA GUI for
Searchdata attributes and
layout
animating the search
percolationSliding
Puzzles

rules of the game

ButtonPressed and
ButtonReleased

```
def ButtonPressed(self, event):  
    """  
    display coordinates of button press  
    """  
    self.MousePosition.set("currently at [ " + \  
        str(event.x) + ", " + str(event.y) + " ]" + \  
        " release to fill, or drag")  
  
def ButtonReleased(self, event):  
    """  
    display coordinates of button release  
    """  
    self.MousePosition.set("drawn at [ " + \  
        str(event.x) + ", " + str(event.y) + " ]" + \  
        " redo to clear")  
    self.DrawRectangle(event.x, event.y)
```

ButtonPressed and ButtonReleased

```
def ButtonPressed(self, event):
    """
    display coordinates of button press
    """
    self.MousePosition.set("currently at [ " + \
        str(event.x) + ", " + str(event.y) + " ]" + \
        " release to fill, or drag")

def ButtonReleased(self, event):
    """
    display coordinates of button release
    """
    self.MousePosition.set("drawn at [ " + \
        str(event.x) + ", " + str(event.y) + " ]" + \
        " redo to clear")
    self.DrawRectangle(event.x, event.y)
```

The Other Methods

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

```
def EnteredWindow(self, event):
    "display message that mouse entered window"
    self.MousePosition.set("press mouse" + \
        "to give coordinates")

def ExitedWindow(self, event):
    "display message that mouse exited window"
    self.MousePosition.set("put mouse " + \
        "inside box to draw")

def MouseDragged(self, event):
    "display coordinates of moving mouse"
    self.MousePosition.set("dragging at [ " + \
        str(event.x) + ", " + str(event.y) + \
        " ]" + " release to draw")
```

The Other Methods

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

```
def EnteredWindow(self, event):
    "display message that mouse entered window"
    self.MousePosition.set("press mouse" + \
        "to give coordinates")

def ExitedWindow(self, event):
    "display message that mouse exited window"
    self.MousePosition.set("put mouse " + \
        "inside box to draw")

def MouseDragged(self, event):
    "display coordinates of moving mouse"
    self.MousePosition.set("dragging at [ " + \
        str(event.x) + ", " + str(event.y) + \
        " ]" + " release to draw")
```

The Other Methods

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

```
def EnteredWindow(self, event):
    "display message that mouse entered window"
    self.MousePosition.set("press mouse" + \
        "to give coordinates")

def ExitedWindow(self, event):
    "display message that mouse exited window"
    self.MousePosition.set("put mouse " + \
        "inside box to draw")

def MouseDragged(self, event):
    "display coordinates of moving mouse"
    self.MousePosition.set("dragging at [ " + \
        str(event.x) + ", " + str(event.y) + \
        " ]" + " release to draw")
```

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

**data attributes and
layout**
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution

2 Handling Mouse Events
more about GUI programming

3 **A GUI for Search**
data attributes and layout
animating the search
percolation

4 Sliding Puzzles
rules of the game

8 Feb 2010

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout

animating the search
percolation

Sliding Puzzles

rules of the game

A GUI for Search

The GUI uses Tkinter.

The numpy matrix that represents the maze
is a data attributed of the GUI.

Widgets of the GUI:

- canvas to draw the maze,
- random, revert, clear buttons to make maze,
- start and stop button for the animation,
- entry widget to show current position.

With mouse, user can make a path through the maze.

Squares are “magnified” pixels

A GUI for Search

The GUI uses Tkinter.

The numpy matrix that represents the maze is a data attributed of the GUI.

Widgets of the GUI:

- canvas to draw the maze,
- random, revert, clear buttons to make maze,
- start and stop button for the animation,
- entry widget to show current position.

With mouse, user can make a path through the maze.

Squares are “magnified” pixels

naming the squares

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

The button Random activates the function:

```
def RandomFill(self):
    """
    fills the canvas with boxes placed at random
    """
    for i in range(0,self.filled.shape[0]):
        for j in range(0,self.filled.shape[1]):
            name = '('+str(i)+','+str(j)+')'
            ...
```

Observe:

- `filled` is a numpy matrix,
- Every square at row `i` and column `j` has a unique name: `(i, j)`.

the initial maze

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search
percolation

Sliding
Puzzles

rules of the game

```
def RandomFill(self):
    """
    fills the canvas with boxes placed at random
    """
    for i in range(0,self.filled.shape[0]):
        for j in range(0,self.filled.shape[1]):
            name = '('+str(i)+','+str(j)+')'
            if randint(0,1) == 1:
                x0 = self.mag + i*self.mag
                y0 = self.mag + j*self.mag
                x1 = x0 + self.mag
                y1 = y0 + self.mag
                self.c.create_rectangle(y0,x0,y1,x1,\
                    fill="green",tags=name)
                self.filled[i,j] = -1
            else:
                self.c.delete(name)
                self.filled[i,j] = 0
```

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search

percolation

Sliding
Puzzles

rules of the game

solving puzzles

1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution

2 Handling Mouse Events
more about GUI programming

3 A GUI for Search
data attributes and layout
animating the search
percolation

4 Sliding Puzzles
rules of the game

animating the search

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search

percolation

Sliding
Puzzles

rules of the game

Recall the definition `Search(M, i, j, k)`.

The GUI exports the `animate` applied to `self`, with parameters `i`, `j`, and `k`.

The return of `animate` is a boolean to flag if the destination was reached.

Because the maze is a data attribute of the GUI, `animate` does not return a matrix.

animating the search

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search

percolation

Sliding
Puzzles

rules of the game

Recall the definition `Search(M, i, j, k)`.

The GUI exports the `animate` applied to `self`, with parameters `i`, `j`, and `k`.

The return of `animate` is a boolean to flag if the destination was reached.

Because the maze is a data attribute of the GUI, `animate` does not return a matrix.

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search
percolation

Sliding
Puzzles

rules of the game

```
def animate(self,i,j,k):
    """
    starts the search for a path
    """
    self.filled[i,j] = k
    self.e.delete(0,END)
    self.e.insert(INSERT,\
        '(i,j) = ('+str(i)+' ','+str(j)+'')')
    x0 = (i+1)*self.mag; y0 = (j+1)*self.mag
    x1 = (i+2)*self.mag; y1 = (j+2)*self.mag
    self.c.create_rectangle(y0,x0,y1,x1,\
        fill="red",tags='dot')
    self.c.after(self.delay)
    self.c.update()
    if self.go:
        if i == self.filled.shape[0] - 1 \
            and j == self.filled.shape[1] - 1:
            return True # found the end
```

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search
percolation

Sliding
Puzzles

rules of the game

```
def animate(self,i,j,k):
    """
    starts the search for a path
    """
    self.filled[i,j] = k
    self.e.delete(0,END)
    self.e.insert(INSERT,\
        '(i,j) = ('+str(i)+' ','+str(j)+'')')
    x0 = (i+1)*self.mag; y0 = (j+1)*self.mag
    x1 = (i+2)*self.mag; y1 = (j+2)*self.mag
    self.c.create_rectangle(y0,x0,y1,x1,\
        fill="red",tags='dot')
    self.c.after(self.delay)
    self.c.update()
    if self.go:
        if i == self.filled.shape[0] - 1 \
            and j == self.filled.shape[1] - 1:
            return True # found the end
```

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search
percolation

Sliding
Puzzles

rules of the game

```
def animate(self,i,j,k):
    """
    starts the search for a path
    """
    self.filled[i,j] = k
    self.e.delete(0,END)
    self.e.insert(INSERT,\
        '(i,j) = ('+str(i)+' ','+str(j)+'')')
    x0 = (i+1)*self.mag; y0 = (j+1)*self.mag
    x1 = (i+2)*self.mag; y1 = (j+2)*self.mag
    self.c.create_rectangle(y0,x0,y1,x1,\
        fill="red",tags='dot')
    self.c.after(self.delay)
    self.c.update()
    if self.go:
        if i == self.filled.shape[0] - 1 \
            and j == self.filled.shape[1] - 1:
            return True # found the end
```

animate continued

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search

percolation

Sliding
Puzzles

rules of the game

```
else:
```

```
    b = False
```

```
    if self.go and i < self.filled.shape[0]-1:
```

```
        if self.filled[i+1,j] == 0 and self.go:
```

```
            b = self.animate(i+1,j,k+1)
```

```
    if not b and self.go:
```

```
        if j < self.filled.shape[1]-1:
```

```
            if self.filled[i,j+1] == 0 and self.go:
```

```
                b = self.animate(i,j+1,k+1)
```

```
    if not b and self.go and i > 0:
```

```
        if self.filled[i-1,j] == 0 and self.go:
```

```
            b = self.animate(i-1,j,k+1)
```

```
    if not b and self.go and j > 0:
```

```
        if self.filled[i,j-1] == 0 and self.go:
```

```
            b = self.animate(i,j-1,k+1)
```

```
    return b
```

animate continued

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search
percolation

Sliding
Puzzles

rules of the game

```
else:
```

```
    b = False
```

```
    if self.go and i < self.filled.shape[0]-1:
```

```
        if self.filled[i+1,j] == 0 and self.go:
```

```
            b = self.animate(i+1,j,k+1)
```

```
    if not b and self.go:
```

```
        if j < self.filled.shape[1]-1:
```

```
            if self.filled[i,j+1] == 0 and self.go:
```

```
                b = self.animate(i,j+1,k+1)
```

```
    if not b and self.go and i > 0:
```

```
        if self.filled[i-1,j] == 0 and self.go:
```

```
            b = self.animate(i-1,j,k+1)
```

```
    if not b and self.go and j > 0:
```

```
        if self.filled[i,j-1] == 0 and self.go:
```

```
            b = self.animate(i,j-1,k+1)
```

```
    return b
```

animate continued

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout

animating the search
percolation

Sliding
Puzzles

rules of the game

```
else:
```

```
    b = False
```

```
    if self.go and i < self.filled.shape[0]-1:
```

```
        if self.filled[i+1,j] == 0 and self.go:
```

```
            b = self.animate(i+1,j,k+1)
```

```
    if not b and self.go:
```

```
        if j < self.filled.shape[1]-1:
```

```
            if self.filled[i,j+1] == 0 and self.go:
```

```
                b = self.animate(i,j+1,k+1)
```

```
    if not b and self.go and i > 0:
```

```
        if self.filled[i-1,j] == 0 and self.go:
```

```
            b = self.animate(i-1,j,k+1)
```

```
    if not b and self.go and j > 0:
```

```
        if self.filled[i,j-1] == 0 and self.go:
```

```
            b = self.animate(i,j-1,k+1)
```

```
    return b
```

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

- 1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution
- 2 Handling Mouse Events
more about GUI programming
- 3 A GUI for Search
data attributes and layout
animating the search
percolation
- 4 Sliding Puzzles
rules of the game

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

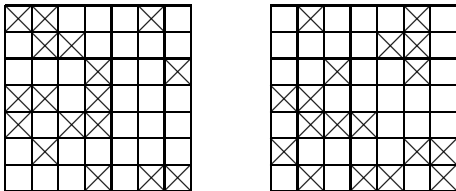
Sliding
Puzzles

rules of the game

Percolation

a variation on a path through a maze

Empty squares admit flow, crosses block the flow:



Both configurations percolate, but only the one on the left percolates vertically.

Percolation: there is a path starting at an open square at the top and ending at an open square at the bottom.

Application: study of porous materials, percolation theory.

8 Feb 2010

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

solving puzzles

- 1 a Path through a Maze
running the program
generating and adjusting a matrix
a recursive solution
- 2 Handling Mouse Events
more about GUI programming
- 3 A GUI for Search
data attributes and layout
animating the search
percolation
- 4 Sliding Puzzles
rules of the game

8 Feb 2010

A Sliding Puzzle

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

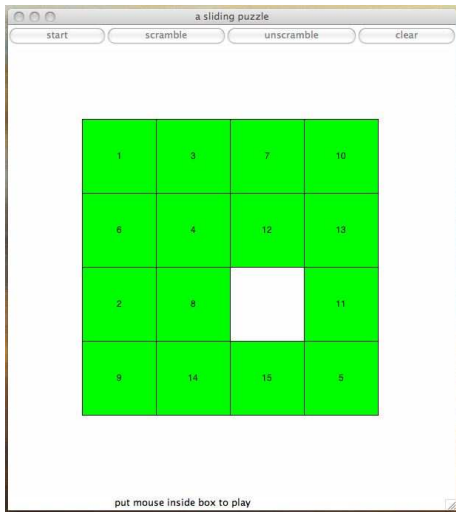
more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game



rules of the game

a Path through a Maze

running the program
generating and adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI programming

A GUI for Search

data attributes and layout
animating the search
percolation

Sliding Puzzles

rules of the game

Given is a n -by- n board of sliding pieces.

One open space where adjacent pieces may slide into.

Each piece has a unique number.

This number determines its right place on the board.

Goal of the sliding puzzle:

slide each piece to its right spot on the board.

Design of the GUI

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

The GUI contains several elements:

- 1 buttons to start, scramble, unscramble, and clear
- 2 a canvas widget to display the sliding puzzle
- 3 a numpy array stores the puzzle data
- 4 the scramble/unscramble buttons are animations
- 5 mouse events allow user to solve the puzzle

What defines a move?

→ piece (i, j) moves to its adjacent open spot.

Design of the GUI

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

The GUI contains several elements:

- 1 buttons to start, scramble, unscramble, and clear
- 2 a canvas widget to display the sliding puzzle
- 3 a numpy array stores the puzzle data
- 4 the scramble/unscramble buttons are animations
- 5 mouse events allow user to solve the puzzle

What defines a move?

→ piece (i, j) moves to its adjacent open spot.

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

Solving the Puzzle

A puzzle with n rows and m columns is solved if piece at (i, j) has number $i \times (n - 1) + j + 1$, for all pieces.

A function `solve` will first check whether the values for the pieces match their position. If so, return `True`.

Recursive calls:

→ for all pieces p adjacent to open spot:

- 1 slide p to the open spot
- 2 `b = solve()`, recursive call
- 3 if `b`: `break`, *puzzle solved!*

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

Solving the Puzzle

A puzzle with n rows and m columns is solved if piece at (i, j) has number $i \times (n - 1) + j + 1$, for all pieces.

A function `solve` will first check whether the values for the pieces match their position. If so, return `True`.

Recursive calls:

→ for all pieces p adjacent to open spot:

- 1 slide p to the open spot
- 2 `b = solve()`, recursive call
- 3 if `b`: `break`, *puzzle solved!*

Solving the Puzzle

a Path through a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling Mouse Events

more about GUI
programming

A GUI for Search

data attributes and
layout
animating the search
percolation

Sliding Puzzles

rules of the game

A puzzle with n rows and m columns is solved if piece at (i, j) has number $i \times (n - 1) + j + 1$, for all pieces.

A function `solve` will first check whether the values for the pieces match their position. If so, return `True`.

Recursive calls:

→ for all pieces p adjacent to open spot:

- 1 slide p to the open spot
- 2 `b = solve()`, recursive call
- 3 if `b`: `break`, *puzzle solved!*

a Path through
a Maze

running the program
generating and
adjusting a matrix
a recursive solution

Handling
Mouse Events

more about GUI
programming

A GUI for
Search

data attributes and
layout
animating the search
percolation

Sliding
Puzzles

rules of the game

Assignments

- 1 Modify the `Search` function to solve the percolation problem. Test if your modification works.
- 2 Adjust the GUI for finding a path through a maze to the percolation problem.
- 3 Instead of changing the length of the bars with scales in project one, explore how to change the length by allowing the user to drag the joints with the mouse.
- 4 Add a `solve` button to `sliding_puzzle.py` and write code for solving the puzzle.
- 5 Compare the difficulty of solving the sliding puzzles with finding a path in a maze. Which one is harder and why?
- 6 Suppose a graph is represented by a numpy matrix A of zeroes and ones. If there is an edge from node i to j , then $A[i,j] = 1$, otherwise $A[i,j] = 0$. Write a Python function to find a path between any two nodes.