Example Project

Thomas Schwarz

A simple game

- We use Tkinter to generate a simple game
  - Player has to keep a moving ball inside a rectangle
  - Player can use mouse clicks to change the movement of the ball

View - Model Paradigm

- Split programming task into two pieces
  - View: What we can see
    - The canvas and the controls
  - Model:
    - The behavior of the ball
- This is a very simple paradigm:
  - Better ones:
    - Model-View-Controller
    - Model-View-ViewModel

View - Model Paradigm

- Designing the View
  - Create the top window

```
def __init__(self):
    self.top = tk.Tk()
    self.top.geometry('700x500')
    self.top.title('keep the ball')
    self.ball = Ball(200, 200)
    self.top.after(1, self.animate)
    self.make_widgets()
    self.top.mainloop()
```

This is our connection to the model

View - Model Paradigm

- Designing the View
  - Create the top window

```
def __init__(self):
    self.top = tk.Tk()
    self.top.geometry('700x500')
    self.top.title('keep the ball')
    self.ball = Ball(200, 200)
    self.top.after(1, self.animate)
    self.make_widgets()
    self.top.mainloop()
```

Select the window size with geometry

View - Model Paradigm

- Designing the View
  - Create the top window

```
def __init__(self):
    self.top = tk.Tk()
    self.top.geometry('700x500')
    self.top.title('keep the ball')
    self.ball = Ball(200, 200)
    self.top.after(1, self.animate)
    self.make_widgets()
    self.top.mainloop()
```

Set up the animation

View - Model Paradigm

- Designing the View
  - Create the top window

```
def __init__(self):
    self.top = tk.Tk()
    self.top.geometry('700x500')
    self.top.title('keep the ball')
    self.ball = Ball(200, 200)
    self.top.after(1, self.animate)
    self.make_widgets()
    self.top.mainloop()
```

Main loop

View - Model Paradigm

- Creating a title label

```
def make_widgets(self):
    self.label = tk.Label(text='Keep the ball inside',
                          font=('Avant Garde', 30))
    self.label.pack(side='top')
```

View - Model Paradigm

- Creating a canvas and bind it to mouse clicks

```
self.canvas = tk.Canvas(self.top,
                        height=400,
                        width=700,
                        bg = '#104060')
self.canvas.pack(side='bottom')
self.canvas.bind("<Button-1>",
                 lambda e: self.handler(e))
```

View - Model Paradigm

- Creating a canvas and bind it to mouse clicks
  - The event handler gets the coordinates to the mouse click and passes them on to the model

```
def handler(self,e):
    self.ball.adjust(e.x, e.y)
```

View - Model Paradigm

- We need to make the ball visible

```
def make_widgets(self):
    …
    self.display_ball()
…
```

View - Model Paradigm

- We make a reset button

```
self.button=tk.Button(self.top,
                      text='reset',
                      command=self.reset)
self.button.pack(side='bottom')
```

- The callback just resets the ball

```
def reset(self):
    self.ball = Ball(350, 250)
    print('reset')
```

View - Model Paradigm

- Reset just creates a new ball
  - which gets rid of the old one

```
def reset(self):
    self.ball = Ball(350, 250)
    print('reset')
```

View - Model Paradigm

- Animation code is standard, just display the ball
  - After updating

```
def animate(self):
self.ball.move()
self.display_ball()
self.top.after(100, self.animate)
```

View - Model Paradigm

- Model: Class Ball
  - Balls have a size, a speed limit, a position, and a change of position

```
class Ball:
    max_speed = 10
    def __init__(self, x, y):
        self.x, self.y = x, y
        self.dx, self.dy = rd.random()*10-5, rd.random()*10-5
```

View - Model Paradigm

- Balls need to move

```
def move(self):
    self.x += self.dx
    self.y += self.dy
```

View - Model Paradigm

- Balls adjust to mouse clicks
  - This code can stand improvements

```
def adjust(self, a,b):
        deltax = a - self.x
        deltay = b - self.y
        distance = ((a-self.x)**2+(b-self.y)**2)**1/2
        self.dx = 0.5 *self.dx + 0.5 * deltax
        self.dy = 0.5 *self.dy + 0.5 * deltay
        length = (self.dx**2+self.dy**2)**1/2
        #self.dx *= Ball.max_speed / length
        #self.dy *= Ball.max_speed / length
        print(self.dx, self.dy)
```