Programming Games
In Python
Charles Severance - www.dr-chuck.com

Textbook: Python Programming: An Introduction to Computer Science, John Zelle (www.si182.com)
History of Games
Computers and Games

• Games have been part of computer innovation from *almost* the very beginning

• Each time a new display technology appeared - new games appeared

• These days - games drive display innovation - the market is so large and the rewards are so great for display manufacturers
http://anita-calculators.info/
http://en.wikipedia.org/wiki/Nixie_tube
http://vintagecalculators.com/html/calculator_displays.html
My final project in my first programming class (in 1975) was a hangman game which you played on a teletype.

http://en.wikipedia.org/wiki/ASR33

http://www.youtube.com/watch?v=E4IztV7M3jI
The first 2D graphics game.
Lunar Lander.
1973

http://en.wikipedia.org/wiki/Lunar_Lander_%28computer_game%29
The First Commercial Video Game - Atari
http://en.wikipedia.org/wiki/Pong
Billions of Dollars - One of the main sources of innovation and R/D in consumer computer technology today
Computer Hardware is fast enough that you can do it too. In Python if you like.

http://www.youtube.com/watch?v=S7PkSFFpkYI
A Simple Example Game Using Zelle Graphics
Game Programs

• Data oriented programs run until the data is completely handled and then stop

• Game Programs are inherently “interactive” - they run as long as gameplay continues - they are ended by “game over” or “user decides to quit”

• The core structure of a game program is different than a data program
Data Program Outline

# Open a File

# Read through the file
for line in file:
    # Process each line

# Print Results

Data loops are bounded. Games are set up as “infinite” loops.

Game Program Outline

# Set up the game

# Run the game
while True:
    # Simulate the game
    # Update the display
    # Wait for a bit
    # Handle any user action
    if userQuit:
        break
    if userAction:
        # Modify the game variables
# Simulate the game
# Update the display
# Handle any user action
# Wait a bit
One Player Pong

- Physics
- Ball has velocity
- Bounces off walls and paddle
- User moves paddle
- Game over when ball misses paddle
Physics - Moving

- The ball has a position
- The velocity is kept separately in x and y. Position is to the right and up
- The speed is 0.01 - the ball moves 1% of the screen each time the game loop runs

\[ dx = -0.01 \quad (0.55, 0.42) \]
\[ dy = -0.01 \]

Before

\((0,0)\)
Physics - Moving

- The ball has a position
- The velocity is kept separately in x and y. Position is to the right and up
- The speed is 0.01 - the ball moves 1% of the screen each time the game loop runs

\[ dx = -0.01 \]
\[ dy = -0.01 \]

\[(0.55, 0.42) \rightarrow (0.54, 0.41) \]

After

\[(0,0) \rightarrow (1,1) \]
Physics of Moving

circle = Circle(Point(0.3, 0.5), 0.02)
circle.setFill('red')
circle.draw(win)

speed = 0.01
dx = speed
dy = speed

while True:
circle.move(dx, dy)
where = circle.getCenter()
print playing, dx, dy, where, paddlecenter
....
Debug Print - Ball Moving

Play  dx  dy  Ball Position  Paddle Position
True -0.01 -0.01 Point(0.89, 0.37) 0.441471571906
True -0.01 -0.01 Point(0.88, 0.36) 0.441471571906
True -0.01 -0.01 Point(0.87, 0.35) 0.441471571906
True -0.01 -0.01 Point(0.86, 0.34) 0.441471571906
True -0.01 -0.01 Point(0.85, 0.33) 0.441471571906
True -0.01 -0.01 Point(0.84, 0.32) 0.441471571906
True -0.01 -0.01 Point(0.83, 0.31) 0.441471571906

The console is cool because you can debug while playing the game.
Wall Bounce

- When we get near a wall we need to bounce or we will just go off of the screen and disappear.
- Depending on which wall you are hitting, just flip the sign of the velocity:
  - Before:
    - $dx = -0.01$
    - $dy = 0.01$
  - After:
    - $dx = 0.01$
    - $dy = -0.01$

Note: Sometimes you debug with graphics.
Wall Bounce

- When we get near a wall, we need to bounce or we will just go off of the screen and disappear.

- Depending on which wall you are hitting, just flip the sign of the velocity.

Note: Sometimes you debug with graphics.
Debug Print - Bounce

Play  dx  dy  Ball Position  Paddle Position
True -0.01 0.01 Point(0.04, 0.36) 0.250836120401
True -0.01 0.01 Point(0.03, 0.37) 0.250836120401
True -0.01 0.01 Point(0.02, 0.38) 0.250836120401
True 0.01 0.01 Point(0.03, 0.39) 0.250836120401
True 0.01 0.01 Point(0.04, 0.4) 0.250836120401
True 0.01 0.01 Point(0.05, 0.41) 0.250836120401
True 0.01 0.01 Point(0.06, 0.42) 0.250836120401
True 0.01 0.01 Point(0.07, 0.43) 0.250836120401
Wall Bounce Code

```python
while True:
    circle.move(dx, dy)
    where = circle.getCenter()
    print playing, dx, dy, where, paddlecenter
    # Bounce off vertical walls
    if where.getX() < 0.02:
        dx = speed * 1.0
    if where.getX() > 0.98:
        dx = speed * -1.0
```

Note: Why use < and not ==  ???
Why not Use Equals?

- Conservative programming - don’t assume the rest of the code is perfect - somehow X might get off the screen - this gets it back on.

- Equals will not work - try it - your ball will fly off the screen and not come back

- Floating point numbers are not exact - they are very close approximations

# Good
if where.getX() < 0.02:
    dx = speed * 1.0

# Bad
if where.getX() == 0.02:
    dx = speed * 1.0
Game Over?

• When the center of the ball is lower than the bottom of the paddle - the player missed the ball

• In this game the paddle goes from 0.05 to 0.1 vertically

\[ y = 0.05 \]
gameover = Text(Point(0.5,0.5), "Game Over")

playing = True
while True:
    .....  
    if playing and where.getY() < paddletop-paddleheight :
        print "Game over"
        gameover.draw(win)
        dx = 0
        dy = 0
        playing = False
        continue
Game Over

- Note where the ball stops moving
- “Game Over” appears
### Debug Print - Game Over

<table>
<thead>
<tr>
<th>Play</th>
<th>dx</th>
<th>dy</th>
<th>Ball Position</th>
<th>Paddle Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>-0.01</td>
<td>-0.01</td>
<td>Point(0.27, 0.07)</td>
<td>0.538461538462</td>
</tr>
<tr>
<td>True</td>
<td>-0.01</td>
<td>-0.01</td>
<td>Point(0.26, 0.06)</td>
<td>0.538461538462</td>
</tr>
<tr>
<td>True</td>
<td>-0.01</td>
<td>-0.01</td>
<td>Point(0.25, 0.05)</td>
<td>0.538461538462</td>
</tr>
</tbody>
</table>

Game over

<table>
<thead>
<tr>
<th>False</th>
<th>0</th>
<th>0</th>
<th>Point(0.25, 0.05)</th>
<th>0.538461538462</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>0</td>
<td>0</td>
<td>Point(0.25, 0.05)</td>
<td>0.538461538462</td>
</tr>
</tbody>
</table>

The game may be over for the user - but the game program is still running full tilt - the ball is not moving and playing is False.
Hit the Paddle Vertically?

- When the ball bottom is touching the paddle we check to see if we got a hit
- Paddle top = 0.1
- Paddle bottom = 0.05
- Circle radius = 0.02
- We check $y < 0.12$

\[
y = 0.12 \\
y = 0.05
\]
Hit the Paddle Horizontally?

- We know paddle center and paddle width
- When the ball center is between center - width/2 and center + width / 2 we have a hit
while True:
circle.move(dx,dy)
where = circle.getCenter()
print playing, dx, dy, where, paddlecenter

.....

if playing and where.getY() < paddletop+radius :
    if where.getX() > paddlecenter - paddlewidth/2 and
        where.getX() < paddlecenter + paddlewidth/2 :
        print "Hit the Paddle"
        dy = speed * 1.0

All we have to do is send the ball upwards and let the physics take care of it.
**Debug Print - Paddle Hit**

<table>
<thead>
<tr>
<th>Play</th>
<th>dx</th>
<th>dy</th>
<th>Ball Position</th>
<th>Paddle Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>-0.01</td>
<td>-0.01</td>
<td>Point(0.32, 0.14)</td>
<td>0.267558528428</td>
</tr>
<tr>
<td>True</td>
<td>-0.01</td>
<td>-0.01</td>
<td>Point(0.31, 0.13)</td>
<td>0.267558528428</td>
</tr>
<tr>
<td>True</td>
<td>-0.01</td>
<td>-0.01</td>
<td>Point(0.3, 0.12)</td>
<td>0.267558528428</td>
</tr>
</tbody>
</table>

Hit the Paddle

<table>
<thead>
<tr>
<th>Play</th>
<th>dx</th>
<th>dy</th>
<th>Ball Position</th>
<th>Paddle Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>-0.01</td>
<td>0.01</td>
<td>Point(0.29, 0.12)</td>
<td>0.267558528428</td>
</tr>
<tr>
<td>True</td>
<td>-0.01</td>
<td>0.01</td>
<td>Point(0.28, 0.13)</td>
<td>0.267558528428</td>
</tr>
<tr>
<td>True</td>
<td>-0.01</td>
<td>0.01</td>
<td>Point(0.27, 0.14)</td>
<td>0.267558528428</td>
</tr>
</tbody>
</table>

Paddle Width is 0.2 (20% of the width of the screen).
0.3 is between 0.26-0.1 and 0.26+0.1
Wait a Bit

- So far
- Physics of ball movement
- Game over
- Bouncing off the paddle
- But computers are fast and people are slow so we have to give them a chance

# Set up the game

# Run the game
while True:
    # Simulate the game
    # Update the display
    # Wait a bit
    # Handle any user action
    if userQuit:
        break
    if userAction:
        # Modify the game variables
from graphics import *
import time

# Set up Game

while True:
    circle.move(dx,dy)
    where = circle.getCenter()
    print playing, dx, dy, where, paddlecenter
    # Do Simulation, Game Over, etc

    time.sleep(0.05)

    # Check for User Input

We wait 1/20 of a second so the ball moves slow enough for the user to react.

Comment out the sleep and see how fast the program runs :) P.S. Quit does work
What about the User?

- Start the game and do not press a key at all
- The game plays until “Game Over” and then just sits there waiting for Quit or Reset
User Input
User Input

• All we have for input is a mouse click.

• If this were a real game console we would have many buttons, joysticks, etc etc.

# Set up the game

# Run the game
while True:
    # Simulate the game
    # Update the display
    # Wait a bit
    # Handle any user action
    if userQuit:
        break
    if userAction:
        # Modify the game variables
Getting a Click

- `getLastMouse()` gives us the position of the most recent mouse click or "None"
- `clearLastMouse()` resets the last mouse click value to None
- We use these to only handle each click once

```python
win.clearLastMouse()
while True:
    # Simulate Game
    time.sleep(0.05)
    pos = win.getLastMouse()
    if pos != None:
        # Only want one click
        win.clearLastMouse()
        print "Click", pos

    # Handle User Input
```
<table>
<thead>
<tr>
<th>Play</th>
<th>dx</th>
<th>dy</th>
<th>Ball Position</th>
<th>Paddle Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>0.01</td>
<td>0.01</td>
<td>Point(0.44, 0.64)</td>
<td>0.642140468227</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Click Point(0.63545, 0.40134)</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>0.01</td>
<td>0.01</td>
<td>Point(0.45, 0.65)</td>
<td>0.642140468227</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Click Point(0.27759, 0.47826)</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>0.01</td>
<td>0.01</td>
<td>Point(0.46, 0.66)</td>
<td>0.642140468227</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Click Point(0.29766, 0.55184)</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>0.01</td>
<td>0.01</td>
<td>Point(0.47, 0.67)</td>
<td>0.642140468227</td>
</tr>
<tr>
<td>True</td>
<td>0.01</td>
<td>0.01</td>
<td>Point(0.48, 0.68)</td>
<td>0.642140468227</td>
</tr>
<tr>
<td>True</td>
<td>0.01</td>
<td>0.01</td>
<td>Point(0.49, 0.69)</td>
<td>0.642140468227</td>
</tr>
</tbody>
</table>

Clicks are “independent” of the simulation. Some steps see a click and others do not and the ball just keeps moving...
Meaning???

- There are three active areas of the screen
- Quit
- Reset
- Move Paddle

In a real game system, we would have multiple inputs and a low-level object would tell us which button was clicked.
Meaning???

- We simply look at where the single click happened and check to see which rectangle it is “in”

- Outside these three areas we simply ignore the click as it has “no meaning”

Different clicks mean different things based on where the click happened.

Maybe we want the user to try to hit the ball and blow it up....

\[ y = 0.2 \]
Quitting

- Quitting is easy
- If we detect that the click is in the “quit button” area - we simply break the “While True” loop

```python
win.clearLastMouse()
while True:
    pos = win.getLastMouse()
    if pos != None:
        # Only want one click
        win.clearLastMouse()
        print "Click", pos
        if pos.getX() > 0.8 and pos.getY() > 0.9:
            print "Quitting"
            break
```
Play dx dy Ball Position Paddle Position
True 0.01 0.01 Point(0.71, 0.91) 0.5
True 0.01 0.01 Point(0.72, 0.92) 0.5
True 0.01 0.01 Point(0.73, 0.93) 0.5
Click Point(0.88629, 0.94649)
Quitting
$

Wherever you are... You click (0.8,0.9) or higher - you are done.
Paddle Move

- If we click lower than $y=0.2$, we interpret this as a paddle move.
- The new paddle center is the $X$ value of the click.
- We must undraw and redraw the Rectangle at the new center.

\[
y = 0.2 \quad (0.6, 0.12)
\]
if pos.getY() < 0.2 :
    paddlecenter = pos.getX()
    print "Moving paddle to",paddlecenter
    paddle.undraw()
    paddle = Rectangle(Point(paddlecenter-paddlewidth/2,paddletop-paddleheight),
                       Point(paddlecenter+paddlewidth/2,paddletop) )
    paddle.setFill('blue')
    paddle.draw(win)

Move the paddlecenter to the X-value of the click, undraw the old rectangle. Make and draw a new rectangle drawn on centered on the new paddle center.
Resetting the Game

- Reset is the trickiest bit of this game
- Reset must work in “Game Over” as well as while the game is playing
- Try pressing “Reset” while the game is playing - it works
Detecting Reset

- Reset is when the click is above and to the left of (0.2, 0.9)

```python
print "Click", pos
if pos.getX() < 0.2 and pos.getY() > 0.9:
    print "Restarting"
```
Reset

- Move the circle to start position
- Get rid of the “Game Over” message
- Get the ball moving again
- Indicate we are once again “in play”
- Let the physics take over

if pos.getX() < 0.2 and pos.getY() > 0.9:
    print "Restarting"
    circle.undraw()
    circle = Circle(Point(0.3,0.5), 0.02)
    circle.setFill('red')
    circle.draw(win)
    gameover.undraw()
    dx = speed
dx = speed
    playing = True
    continue
Play    dx    dy    Ball Position    Paddle Position
False 0 0 Point(0.25, 0.05) 0.90635451505
False 0 0 Point(0.25, 0.05) 0.90635451505
False 0 0 Point(0.25, 0.05) 0.90635451505
Click Point(0.10702, 0.94983)
Restarting
True 0.01 0.01 Point(0.31, 0.51) 0.90635451505
True 0.01 0.01 Point(0.32, 0.52) 0.90635451505
True 0.01 0.01 Point(0.33, 0.53) 0.90635451505

Recovering from Game Over.
Play dx dy Ball Position Paddle Position
True 0.01 0.01 Point(0.39, 0.59) 0.5
True 0.01 0.01 Point(0.4, 0.6) 0.5
Click Point(0.07692, 0.9699)
Restarting
True 0.01 0.01 Point(0.31, 0.51) 0.5
True 0.01 0.01 Point(0.32, 0.52) 0.5
True 0.01 0.01 Point(0.33, 0.53) 0.5
Resetting while running...
Summary

• Games are infinite loops
• Simulate a time step
• Show the user the game state
• Wait for a bit
• Handle User Actions

# Set up the game

# Run the game
while True:
    # Simulate the game
    # Update the display
    # Wait a bit
    # Handle any user action
    if userQuit:
        break
    if userAction:
        # Modify the game variables
More on Python Gaming

There are much better frameworks to use for real 3D, games, etc.
www.pyglet.org

- High Performance Graphics and Multimedia

**pyglet**: a cross-platform windowing and multimedia library for Python.

**news**

**pyglet 1.1 alpha 1 released.** pyglet 1.1 alpha 1 adds more features than any previous release, including fast graphics routines, formatted text layout, animated GIF support, resource loading, and even some bug fixes. If you're not working in a production environment and can afford some instability, go straight to the download server to grab a source or egg release, and start reading up on all the new features.

Submitted by Alex on 1-March-2008.

Older news...
www.pygame.org

- Build all of your game logic in Python
- Physics
- Intelligence
- Run-Time
- Pygame does the graphics
import sys, pygame

pygame.init()

size = width, height = 320, 240

speed = [2, 2]

black = 0, 0, 0

screen = pygame.display.set_mode(size)

ball = pygame.image.load("ball.bmp")

ballrect = ball.get_rect()

while 1:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: sys.exit()

    ballrect = ballrect.move(speed)

    if ballrect.left < 0 or ballrect.right > width:
        speed[0] = -speed[0]

    if ballrect.top < 0 or ballrect.bottom > height:

    screen.fill(black)

    screen.blit(ball, ballrect)

    pygame.display.flip()

This is event style programming. Pygame does the fast loop with the “sleep” - we are only called when something happens (like a click) or time passes.
while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT: sys.exit()

  ballrect = ballrect.move(speed)
  if ballrect.left < 0 or ballrect.right > width:
    speed[0] = -speed[0]
  if ballrect.top < 0 or ballrect.bottom > height:

  screen.fill(black)
  screen.blit(ball, ballrect)
  pygame.display.flip()
Summary

• Games are an important part of computers

• Games are analogs for all kinds of highly interactive applications such as data visualization, drawing programs, etc

• Games pour billions of investment dollars into technology research and development

• And they are fun too...