Multiuser Online Games
In Python

Charles Severance - www.dr-chuck.com

Textbook: Python Programming: An Introduction to Computer Science, John Zelle (www.si182.com)
MMORPG

- Massively Multiple Online Role Playing Games
- A blend of game action and social interaction
- Ultima Online - September 25, 1997
- Natural combination of Game technology and ubiquitous Internet

http://en.wikipedia.org/wiki/MMORPG
MultiPlayer LAN Games

- Precursor to MMORPG
- From 2-10 players
- Needed a Local Area Network (LAN) so that game state could be updated 10+ times per second
Primary Problem in MMORPG

- Most games run at 10 - 60+ frames per second
- Most internet connections can handle 10 or so server round trips per second
- Sometimes network delay has “jitter” - some round-trips take longer than others
Ping Times

charles-severances-macbook-air:~ csev$ ping www.dr-chuck.com
PING www.dr-chuck.com (74.208.28.177): 56 data bytes
64 bytes from 74.208.28.177: icmp_seq=0 ttl=51 time=30.038 ms
64 bytes from 74.208.28.177: icmp_seq=1 ttl=51 time=31.121 ms
64 bytes from 74.208.28.177: icmp_seq=2 ttl=51 time=31.149 ms
64 bytes from 74.208.28.177: icmp_seq=3 ttl=51 time=31.064 ms
64 bytes from 74.208.28.177: icmp_seq=4 ttl=51 time=30.937 ms
^C
--- www.dr-chuck.com ping statistics ---
9 packets transmitted, 9 packets received, 0% packet loss
round-trip min/avg/max/stddev = 30.038/30.657/31.149/0.401 ms
charles-severances-macbook-air:~ csev$
Overall Map Artificial Intelligence

Server

Map Updates

Console

Updates due to user action

Map Updates

Console
Periodic Updates

- Because of the delay in communicating with the game server, each console must simulate the whole game for a little while.

- When the global map is retrieved - the local game is adjusted if there is a discrepancy.

- If the data retrieval is too slow - then the local consoles get out of sync with the real map and the game seems to not work.

- This adds a step to “locally adjust reality” to match the central reality.
Single User
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

Online Game Program Outline

# Set up the game

# Run the game
while True:
    # Simulate the game
    # Update the display
    # Periodically:
        # Retrieve game state
        # Adjust local game data
    # Wait for a bit
    # Handle any user action
Game Starts
Orange Turns
Simulations Continue
OOPS!
Very different views
With Updates
Game Begins
Orange Turns
Orange Updates
Server
Server and Orange are Accurate.
Simulation continues
Green gets an update.
Green adjusts game and runs simulation step. Green perceives a player “jump”.
Neo notices a black cat, a yellow-green eyed shadow that slinks past them and pads quickly down the stairs. A moment later, Neo sees another black cat that looks and moves identically to the first one.

NEO: *Whoa. Deja vu.*

TRINITY: *What happened? What did you see?*

NEO: *A black cat went past us and then I saw another that looked just like it.*

TRINITY: *How much like it? Was it the same cat?*

NEO: *It might have been. I'm not sure.*

TRINITY: *A deju vu is usually a glitch in the Matrix. It happens when they change something.*
Goal

• Game designers go to great lengths to make the updates as quick as possible and at the same time tolerate imperfections in the global knowledge.

• Anomalies happen all the time - the goal is to write a lot of code so you don’t notice it.
Looking at the Game Code
Local Game

- No Server involved - no simulated server
- The fake server shares the simulation data with the game
- Makes the game very accurate and smooth
- One universal and shared view of game data means no anomalies due to out of date game date
Data Structures

- When we have something more complex than a single integer, float, string, etc - and we want to group them into what we call a “data structure”

- It can be as simple as a list of numbers [0.5, 0.25] where the first is the X coordinate and the second is the Y coordinate
Our Game Objects - A List

[id, xpos, ypos, dx, dy, graphics-object]

[0, 0.5, 0.5, 0.003, 0.003, <graphics.Image at 0xb49148>]

[5199, 0.5, 0.5, 0.01, 0.01, <graphics.Circle instance at 0xb497d8>]

[2, 0.823, 0.303, 0.01, 0.01, <graphics.Rectangle at 0xb49a80>]
mycircle = Circle(Point(0.5,0.5), 0.02)
img = Image(Point(0.5,0.5), "sakaiger-50.gif")

...  

gamedata = []
item = [0, 0.5, 0.5, speed/3, speed/3, img]
gamedata.append(item)

item = [myid, 0.5, 0.5, speed, speed, mycircle]
gamedata.append(item)
print "My item"
print item
Initial Game Configuration

[0, 0.5, 0.5, 0.00333, 0.00333, <graphics.Image at 0xb49148>]
[5199, 0.5, 0.5, 0.01, 0.01, <graphics.Circle instance at 0xb497d8>]
[2, 0.823, 0.303, 0.01, 0.01, <graphics.Rectangle at 0xb49a80>]
[3, 0.188, 0.784, 0.01, 0.01, <graphics.Rectangle at 0xb49b48>]
[4, 0.273, 0.681, 0.01, 0.01, <graphics.Rectangle at 0xb49cd8>]
[5, 0.909, 0.661, 0.01, 0.01, <graphics.Rectangle at 0xb49e68>]
[6, 0.945, 0.273, 0.01, 0.01, <graphics.Rectangle at 0xb4f120>]
[7, 0.278, 0.656, 0.01, 0.01, <graphics.Rectangle at 0xb4f288>]
[8, 0.258, 0.870, 0.01, 0.01, <graphics.Rectangle at 0xb4f350>]
[9, 0.830, 0.840, 0.01, 0.01, <graphics.Rectangle at 0xb4f580>]

# A function to determine if
# a point is within a rectangle centered
# at rpoint with height and width

def intersect(cpoint, ipoint, width, height):
    dx = abs(ipoint.getX() - cpoint.getX())
    dy = abs(ipoint.getY() - cpoint.getY())
    if dx < width/2 and dy < height/2:
        return True
    return False

Use the math function for absolute value to get the “distance”.
We loop through the gamedata many times in the program.

Sometimes we want to find or avoid ourselves or the Sakaiger.

# Simulate life for the robot players for item in gamedata:
    if item[0] == myid or item[0] == 0:
        continue

    # Check to see if the robots scored a point
    mex = item[1]
    mey = item[2]
    ......
0 1 2 3 4 5
[id, xpos, ypos, dx, dy, graphics-object]

# If the robot is in Sakaiger space - give them a point
if intersect(Point(mex,mey),Point(sax,say), imgwidth, imgheight):
    print "Robot got a point",item[0]
    obj = item[5]
    obj.undraw()
x = random()
y = random()
rect = Rectangle(Point(x,y),Point(x+0.01,y+0.01))
rect.setFill(randomColor())
rect.draw(win)
item[5] = rect
item[1] = x
item[2] = y

Re-Draw Pattern - because Zelle objects do not support “move to”
Simple Robot Chase AI

# Make the robots chase the Sakaiger
# The further away, the faster we chase
if random() < 0.10:
    dx = ( sax - item[1] ) * 0.01
    dy = ( say - item[2] ) * 0.01
    item[3] = dx
    item[4] = dy
# Check Boundaries - Bounce off walls
for item in gamedata:
    spd = speed
    if item[0] == 0:
        spd = speed / 3
    if item[1] > 0.98:
        item[3] = -1 * spd
    if item[1] < 0.02:
        item[3] = spd
    if item[2] > 0.98:
        item[4] = -1 * spd
    if item[2] < 0.02:
        item[4] = spd

# Do Physics - move one timestep
for item in gamedata:
    if item[3] == 0 and item[4] == 0:
        continue
    object = item[5]
    object.move(item[3],item[4])
# If we scored, update the score, and jump to a random
# spot, clearing out the old circle and drawing a
# new circle
if intersect(Point(mex, mey), Point(sax, say), imgwidth, imgheight):
    score = score + 1
    sctext.setText(score)
    meobj = gamedata[1][5]
    meobj.undraw()
    newx = random()
    newy = random()
    newcircle = Circle(Point(newx, newy), 0.02)
    newcircle.setFill('red')
    newcircle.draw(win)
    gamedata[1][1] = newx
    gamedata[1][2] = newy
    gamedata[1][5] = newcircle

Similar to Robot Scoring with the undraw / redraw pattern.
# Check for user input
pos = win.getLastMouse()
if pos != None :
    win.clearLastMouse()
    # Handle the quit in upper right corner
    if pos.getX() > 0.9 and pos.getY() > 0.9 :
        break

meitem = gamedata[1]
mex = meitem[1]
mey = meitem[2]
print mex, pos.getX(), pos.getY()
dx = ( pos.getX() - mex ) * speed * 2
dy = ( pos.getY() - mey ) * speed * 2
print "Click", dy, dy
gamedata[1][3] = dx
gamedata[1][4] = dy
Game Strategy

- Don’t just click on the Sakaiger
- Click *beyond* the Sakaiger to go faster
- Get in the Sakaiger’s way - you can stop yourself (hit the brakes)
- Go where the Sakaiger is going
Multi-User Server Version
Major Changes

• Inform the server of our location on the screen

• Retrieve data of the “official” global map of the world including the Sakaiger’s position and the other player’s positions

• Adjust our local view of the map - since between global updates - the Sakaiger and other players (or robots) keep moving - this leads to an anomaly
Looking at Multi-User Code

- Our own import statement
- Initial local robots are there to be deleted
- updateGame
  - delete roots gone missing
  - add newly discovered robots
  - update positions - redraw when there is a big jump
- Main loop - simulates all particles
  - Bounce / Physics / Score / Input
  - Contact the Server
- gamestate.py
  - Data structure has no graphic object
  - Url parameters - urllib
  - Retrieve and parse the response
serverurl = "http://www.dr-chuck.com/si182/murpg.php?"
theurl = serverurl + "myid="+str(myid)+"&x="+str(x)+"&y="+
        str(y)+"&dx="+str(dx)+"&dy="+str(dy)

f = urllib.urlopen(theurl)
contents = f.read()
f.close()
lines = contents.split("\n");
retval = []

for line in lines:
    if not line.startswith("data,"):
        continue
    # Slip up the data, convert to numbers, and add to the list
    words = line.split(',','
    item = [ int(words[1]), float(words[2]), float(words[3]), float(words[4]), float(words[5])]
    retval.append(item)
Adding local robot
Initial Game Configuration
[0, 0.5, 0.5, 0.003, 0.003, <graphics.Image at 0xb80328>]
[2117, 0.5, 0.5, 0.01, 0.01, <graphics.Circle at 0xb80788>]
[10, 0.08, 0.48, 0.01, 0.01, <graphics.Rectangle at 0xb80bc0>]
Point(0.5, 0.5) 0.1275 0.1275
Retrieving Server data from URL
http://www.dr-chuck.com/si182/murpg.php?
myid=2117&x=0.51&y=0.51&dx=0.01&dy=0.01

Server Data Retrieved 409 characters.
data,100000,0.200000,0.300000,0.010000,0.010000,0.711082
data,100001,0.300000,0.500000,0.010000,0.010000,0.711082
data,0,0.837659,0.090650,0.010000,0.010000,0.711082
data,2117,0.51,0.51,0.01,0.01,0.524112
Server returns
[100000, 0.20, 0.29, 0.01, 0.01]
[100001, 0.29, 0.5, 0.01, 0.01]
[0, 0.83, 0.09, 0.01, 0.01]
[2117, 0.51, 0.51, 0.01, 0.01]
Robot was not in server list 10
Deleting Robot 10
Appending [100000, 0.20, 0.29, 0.01, 0.01, <graphics.Rectangle at 0x668c6c0>]
Appending [100001, 0.29, 0.5, 0.01, 0.01, <graphics.Rectangle at 0x668c990>]
Redrawing 0 0.50 0.50 0.837659 0.09065
Summary

• The combination of high speed computers, high speed graphics, and ubiquitous Internet makes large-scale multi-user gaming a reality.

• There are many challenges - the goal is to make it look like there is one view of the “real world” even though each console must maintain its own view of the world and adjust that view as global information flows to the console through the server.