FALMOUTH
UNIVERSITY

Lecture 8: Data Persistence with SQL

COMP260: Distributed Systems
BSc (Hons) Computing for Games

- Today’s session:
  - Assignment 2 overview
  - Working with Digital Oceans
  - Development and Operations for GaaS
  - Data persistence with SQL

- Assignment 2 overview
  - – Assignment 2 builds on Assignment 1 to make your distributed MUD:
    - Remote
    - Secure
    - Persistent
  - – Server will remain Python-based
  - – Free choice on client technology (though I would steer away from C++)

- Working with Digital Oceans
  - Andy is in the process of setting up your Digital Oceans server accounts
  - Should be ready by next week's lecture
    - We can make the workshop about setting them up
  - Can use your own Linux server for development if you fancy
    - Have a look at VirtualBox & run off your laptop
    - Did this last year for dealing with C# compatibility issues in Ubuntu
    - Python server should be trivial to maintain on your normal laptop / lab machines
    - Will still need to viva on DObox though

- Development and Operations for GaaS

- Development and Operations for GaaS
  - GaaS (Games as a service) introduces a new concept for us as games developers
    - Persistence
    - As a service, a game is expected to be always on, always running and never breaking or failing

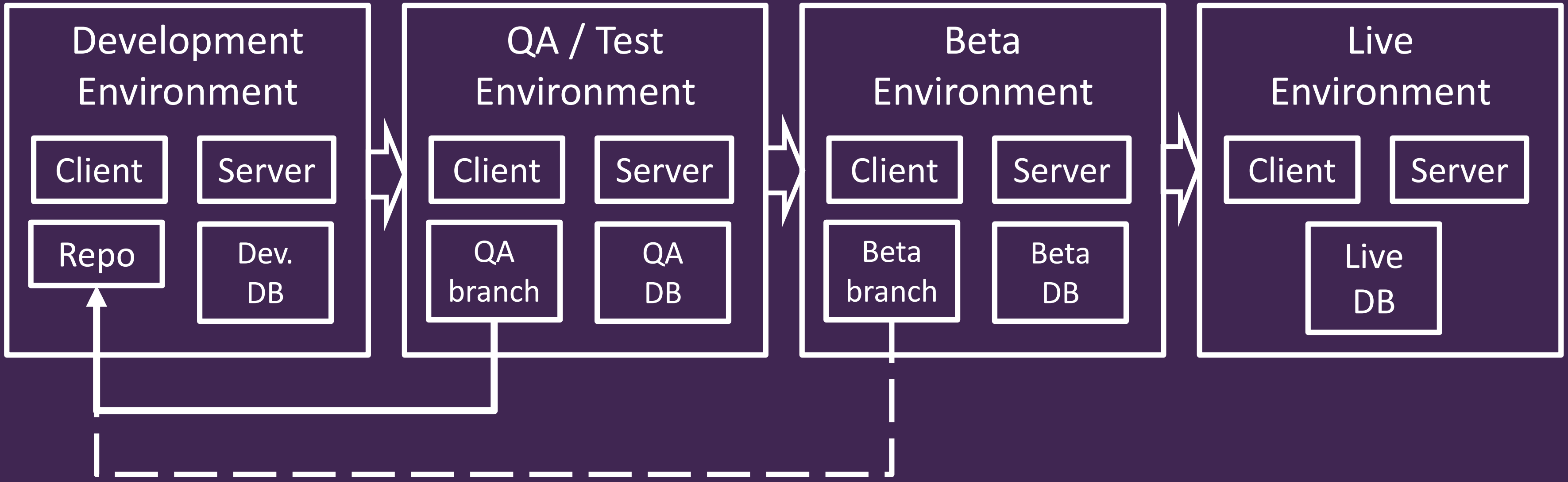
- Development and Operations for GaaS
  - GaaS (Games as a service) introduces a new concept for us as games developers
    - To a degree, console developers are somewhat used to this with the soak test as part of the approval process
      - Typically a console game should run for 48hrs without failing
      - Forces console developers to manage limited resources (memory) properly
      - Traditionally, not an issue for PC development given virtual memory
    - However, 48hrs is not long-term for GaaS

- Development and Operations for GaaS
  - GaaS (Games as a service) introduces a new concept for us as games developers
    - What stops a GaaS from running?
      - Software failure / bugs / scaling issues
      - Hardware failure
      - Resource depletion (memory leaks, filling up disks, thread over-use slowdown)
      - Human error (should I unplug this to do the vacuuming?)
      - Malicious activities from 3rd parties
      - Planned upgrades
      - Planned upgrades that go wrong

- Development and Operations for GaaS
  - How do developers look to minimise downtime?

    - Environmental management
      - Unlike traditional development, GaaS development tends to be far more gated
      - Environments for:
        - » Game development
        - » Game testing
        - » Game operations
      - New features are not developed on the live service
      - QA is not performed on a live service
      - Data is managed so that it can be rolled back if things go haywire

- Development and Operations for GaaS
  - How do developers look to minimise downtime?
    - Environmental management



Development Environment

Client

Server

Repo

Dev. DB

QA / Test Environment

Client

Server

QA branch

QA DB

Beta Environment

Client

Server

Beta branch

Beta DB

Live Environment

Client

Server

Live DB

- Development is gated
  - Development has to go through a number of stages (with checks and balances) to make it into the live service

- Development and Operations for GaaS
  - How do developers look to minimise downtime?
    - Environmental management
    - Typically, when a new build goes live, the service is taken down for a short period
    - This means that data has to persist beyond the scope of the game
      - It can’t ‘just’ live in variables in the code

- Development and Operations for GaaS
  - How do developers recover from issues in the live build?
    - Typically, issues that make it into the live build are difficult issues to deal with
      - Survived QA and Beta testing
    - Often tend to be systemic exploits / issues
      - WoW plague
      - MMO hyper inflation

- Development and Operations for GaaS
  - How do developers recover from issues in the live build?
    - Generally, issues that massively bork the game ecosystem
      - Need to be able to roll back the game to when things worked properly
      - Game needs to be able to store everything important that happens as a starting point

- Development and Operations for GaaS
  - How do developers recover from issues in the live build?
    - Hotfixes & bugfixes
      - There’s the notion of developers applying bugfixes directly to the live environment, usually from a console or something
      - This may occur, but is not good practice
      - Generally, hotfixes are bugfixes that are applied outside of normal support iterations
        - » So, a GaaS may normally apply bugfixes on a weekly basis, but a hotfix would be applied ASAP (assuming it has gone through some form of testing)
      - See this a lot with client-side patching on starting a GaaS client

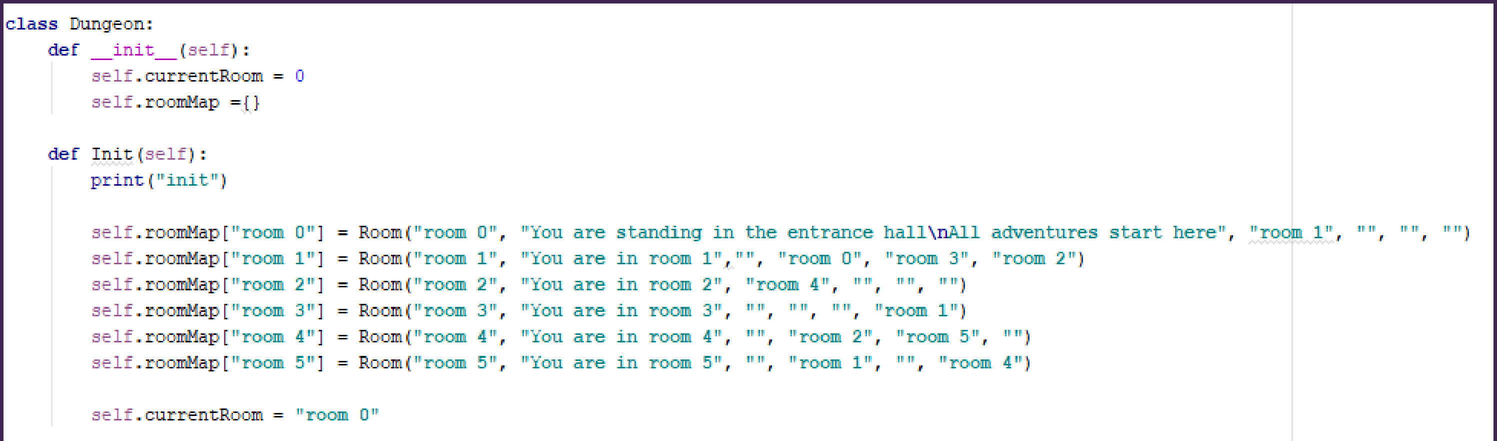
- Development and Operations for GaaS
  - How do developers recover from issues in the live build?
    - The challenge for GaaS operators is that any down time looks bad
      - Particularly if it is a paid-for GaaS
        - » Pay monthly – subscribers will expect refunds
        - » Transaction / IAP – revenue will be lost
      - More importantly, customer may go to other GaaS
      - Frequent issues & downtime gives the impression that you don’t know what you’re doing as a GaaS provider

- Data persistence with SQL

- Data persistence with SQL
  - Databases are the solution
    - Used in commercial data processing for decades to create the digital economy we are used to
      - Managing bank, utility and shopping financial and other data
    - GaaS are just really another form of data processing
      - But with better graphics
      - And gameplay

- Data persistence with SQL
  - – Databases are the solution
    - As we saw last year, databases can:
      - – 1. Store data securely
      - – 2. Provide transactional information that can be used to roll back part or all of a database to a known good state
        - » This is what git /google drive and OneDrive do
      - – 3. Provide access to data
      - – 4. Allow users (programmers) to quickly and efficiently search for arbitrary data
      - – 5. Allow users (programmers) to make completely arbitrary collections of data and manage their relationships

- Data persistence with SQL
  - Back to the SUD



```
class Dungeon:
    def __init__(self):
        self.currentRoom = 0
        self.roomMap ={}

    def Init(self):
        print("init")

        self.roomMap["room 0"] = Room("room 0", "You are standing in the entrance hall\nAll adventures start here", "room 1", "", "", "")
        self.roomMap["room 1"] = Room("room 1", "You are in room 1","", "room 0", "room 3", "room 2")
        self.roomMap["room 2"] = Room("room 2", "You are in room 2", "room 4", "", "", "")
        self.roomMap["room 3"] = Room("room 3", "You are in room 3", "", "", "", "room 1")
        self.roomMap["room 4"] = Room("room 4", "You are in room 4", "", "room 2", "room 5", "")
        self.roomMap["room 5"] = Room("room 5", "You are in room 5", "", "room 1", "", "room 4")

        self.currentRoom = "room 0"
```

- In my SUD, the room the player is in is stored in the current room
  - When the game is stopped, that memory is lost
  - If that data was stored on file, it would never get lost
  - When the game restarts, the player will be just where I left them

- Data persistence with SQL
  - Back to the SUD
    - We could just manually write everything to disk, but that seems a bit of bind
    - Sqlite will do that for us

- Data persistence with SQL
  - Back COMP130
    - As we saw with the high score table last year
      - Sqlite is part of Python
      - It’s fine for what we want to do
    - To include sqlite in a Python file
      - `Import sqlite3`

- Data persistence with SQL
  - For today’s workshop
    - 1. Get to grips with sqlite and sql in general
      - Make a Python application that will demonstrate data lifecycle (create, retrieve, update, delete & search) on a database
      - Use the same kind of approach from the SUD to have command line entry
    - 2. Embed sqlite into the SUD
      - Do this for both the player and the dungeon rooms
      - This will give you a persistent player
      - And the ability to change the dungeon layout without touching the code base
    - 3. Roll those changes into your MUD
      - This is the first part of the assignment

- Data persistence with SQL
  - For today’s workshop
    - For an intro to SQL in Python
      - https://www.pythoncentral.io/introduction-to-sqlite-in-python/
    - For SQL in general
      - https://www.w3schools.com/sql/

- Questions