Project Report

Master Of Computer Applications

Semester – II

Advanced Data Structures

Topic: ADSA GROUP PROJECT QUEUE

King-of-Pirates Cloud Gaming App

Alliance University

Chandapura, Anekal Main Road, Anekal

Bengaluru – 562106

April - 2025

Project Overview:

Title: Cloud Gaming App with Queue Management

Technology: Python

Data Structure Focus: Queue (Regular & Priority)

Dataset Used: Gamepass_Games_v1.csv (Game info from Xbox Gamepass)(Kaggle)

Course: Advance Data Structures and Algorithms in Python

Developer:

Nandhu S

Mutthu R

Girish K T

Abeed

Ajay M

Objective:

What is the goal?

We want to simulate a cloud gaming system where:

- **Users (VIP and Regular) wait in a queue to play games.**
- **Only a few users can play at a time.**
- **We prioritize VIP users over Regular users.**
- **When a user finishes playing, the next one from the queue is allowed.**

Dataset Fields (from your CSV):

We will extract useful fields like:

- **Game Title**
- **Platform**
- **Genre**
- **Publisher**

Key Features:

- **Load game list from CSV**
- **Simulate players joining the platform**
- **Players added to a queue (regular) or priority queue (for VIPs)**
- **Allocate players to available slots**
- **Track playing users, queue position, and wait time**
- **Show logs and current queue status**

Functional Workflow:

1. **Load games**
2. **Simulate adding players (Regular/VIP)**
3. **Assign games to users based on queue**
4. **Users play for a fixed time (simulated)**
5. **Release slot and assign to next player in queue**

Required Modules:

import pandas as pd

import time

import random

from queue import PriorityQueue, Queue

Data Structures Used:

- **Queue – for regular users**
- **PriorityQueue – for VIP users**
- **List/Dict – for currently playing users**

Complete Python Project Code:

I

import pandas as pd

import time

import random

```
from queue import PriorityQueue, Queue


# Load the dataset
def load_games(file_path):
    df = pd.read_csv(file_path)
    games = df[['Title', 'Platform', 'Genre', 'Publisher']].dropna()
    return games.to_dict(orient='records')


# Player class for the queue
class Player:
    def __init__(self, username, priority):
        self.username = username
        self.priority = priority  # 0 = VIP, 1 = Regular
        self.join_time = time.time()


    def __lt__(self, other):
        return self.priority < other.priority  # VIPs first


# Game Manager to simulate servers and queues
class CloudGamingManager:
    def __init__(self, game_list, server_limit=3):
        self.games = game_list
        self.vip_queue = PriorityQueue()
        self.regular_queue = Queue()
        self.playing_users = {}
        self.server_limit = server_limit
```

```
    def add_player(self, username, is_vip):
        player = Player(username, 0 if is_vip else 1)
        if is_vip:
            self.vip_queue.put(player)
            print(f"[+] VIP Player '{username}' joined the queue.")
        else:
            self.regular_queue.put(player)
            print(f"[+] Regular Player '{username}' joined the queue.")

    def assign_servers(self):
        while len(self.playing_users) < self.server_limit:
            next_player = None
            if not self.vip_queue.empty():
                next_player = self.vip_queue.get()
            elif not self.regular_queue.empty():
                next_player = self.regular_queue.get()
            else:
                break

            if next_player:
                game = random.choice(self.games)
                self.playing_users[next_player.username] = {
                    "start_time": time.time(),
                    "game": game
                }
                print(f"[ 🎮 ] '{next_player.username}' started playing '{game['Title']}' on
{game['Platform']}")
```

```
    def release_players(self, play_duration=10):
        finished_users = []
        for username, session in self.playing_users.items():
            if time.time() - session['start_time'] > play_duration:
                finished_users.append(username)

        for user in finished_users:
            print(f"[ ] '{user}' has finished playing.")
            del self.playing_users[user]

    def show_status(self):
        print("\\n--- Queue Status ---")
        print(f"VIP Queue: {self.vip_queue.qsize()}")
        print(f"Regular Queue: {self.regular_queue.qsize()}")
        print(f"Currently Playing: {list(self.playing_users.keys())}\\n")

# Simulation
def simulate_cloud_gaming():
    games = load_games('Gamepass_Games_v1.csv')
    manager = CloudGamingManager(games)

    # Add test players
    players = [
        ("Alice", False),
        ("Bob", True),
        ("Charlie", False),
```

```
        ("Dave", True),
        ("Eve", False),
        ("Frank", True),
    ]

    for name, vip in players:
        manager.add_player(name, vip)

    # Simulate multiple rounds
    for round in range(5):
        print(f"\\n===== Round {round + 1} =====")
        manager.release_players(play_duration=5)
        manager.assign_servers()
        manager.show_status()
        time.sleep(2)  # simulate delay

if __name__ == "__main__":
    simulate_cloud_gaming()
```

Output Sample:

```
[+] Regular Player 'Alice' joined the queue.
[+] VIP Player 'Bob' joined the queue.
...
[🎮] 'Bob' started playing 'Halo Infinite' on Xbox
[🎮] 'Dave' started playing 'Forza Horizon' on PC
...
[🔲] 'Bob' has finished playing.
```

Breakdown of Code & Explanation

---

class ArrayQueue

```
class ArrayQueue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)

    def is_empty(self):
        return len(self.items) == 0

    def size(self):
        return len(self.items)

    def display(self):
        return [player['username'] for player in self.items]
```

Explanation:

- **We built our own queue using arrays (lists).**

- **enqueue: Adds an item to the end.**
- **dequeue: Removes an item from the front.**
- **is_empty: Checks if the queue is empty.**
- **display: Shows usernames currently in the queue.**

---

load_games()

```
def load_games(file_path):
    df = pd.read_csv(file_path)
    df = df[['Title', 'Platform', 'Genre']].dropna()
    return df.to_dict(orient='records')
```

Explanation:

- **Reads the Gamepass_Games_v1.csv file.**
- **Takes only 3 columns: Title, Platform, and Genre.**
- **Converts DataFrame into a list of dictionaries for easy use.**

---

class CloudGamingApp

This is the heart of the project. It handles the entire logic of how players join, play, and leave the system.

__init__()

```
def __init__(self, games, server_limit=3):
    self.games = games
    self.vip_queue = ArrayQueue()
    self.regular_queue = ArrayQueue()
    self.server_limit = server_limit
    self.playing = {}
```

Explanation:

- **games: Game dataset.**
- **vip_queue / regular_queue: Two separate queues.**
- **server_limit: Max number of users who can play at once.**
- **playing: A dictionary storing who is playing what and when.**

---

add_player()

```
def add_player(self, username, is_vip=False):
    player = {'username': username, 'join_time': time.time()}
    if is_vip:
        self.vip_queue.enqueue(player)
    else:
        self.regular_queue.enqueue(player)
```

Explanation:

- **Adds player to either VIP or Regular queue based on input.**

---

assign_servers()

```
def assign_servers(self):
    while len(self.playing) < self.server_limit:
        player = None
        if not self.vip_queue.is_empty():
            player = self.vip_queue.dequeue()
        elif not self.regular_queue.is_empty():
            player = self.regular_queue.dequeue()
        else:
```

```
            break

        if player:
            game = random.choice(self.games)
            self.playing[player['username']] = {
                'game': game,
                'start_time': time.time()
            }
```

Explanation:

- **If server has free slots:**
  - **Pick VIP player first, then Regular.**
  - **Assign a random game to the player.**
  - **Store their game and start time.**

---

release_players()

```
def release_players(self, play_duration=5):
    finished = []
    for username, session in self.playing.items():
        if time.time() - session['start_time'] > play_duration:
            finished.append(username)

    for user in finished:
        del self.playing[user]
```

Explanation:

- **Checks if a user has played long enough.**

- **Removes them from the playing list so that someone else can join.**

---

show_status()

```
def show_status(self):
    print(f"Currently Playing: {list(self.playing.keys())}")
    print(f"VIP Queue: {self.vip_queue.display()}")
    print(f"Regular Queue: {self.regular_queue.display()}")
```

Explanation:

- **Displays which players are:**
    - **Currently playing.**
    - **Waiting in the VIP queue.**
    - **Waiting in the Regular queue.**

---

run_simulation()

```
def run_simulation():
    games = load_games("Gamepass_Games_v1.csv")
    app = CloudGamingApp(games)

    players = [
        ("Alice", False),
        ("Bob", True),
        ("Charlie", False),
        ("Dave", True),
        ("Eve", False),
        ("Frank", True),
    ]
```

```
    for name, vip in players:
        app.add_player(name, vip)

    for round in range(5):
        app.release_players(play_duration=4)
        app.assign_servers()
        app.show_status()
        time.sleep(2)
```

Explanation:

- **Loads games.**
- **Adds players with VIP or Regular flag.**
- **Simulates 5 rounds:**
    - **Releases players who finished.**
    - **Assigns new players to servers.**
    - **Shows current status.**
    - **Waits 2 seconds between each round.**

Key Learnings from This Project:

Concept	Where It's Used	Why It Matters
Queue (Array)	VIP and Regular player handling	Helps simulate waiting systems
Random Selection	Assigning games	Adds dynamic behavior
Timers	Game play simulation	Helps visualize real-time systems
Priority Handling	VIPs get assigned first	Reflects real-world systems

Concept	Where It's Used	Why It Matters
CSV with Pandas	Loading game data	Prepares you for data handling

---

References:

- **Python Official Documentation: https://docs.python.org/3/**
- **Pandas Documentation: https://pandas.pydata.org/docs/**
- **Xbox Game Pass Games Library(kaggle):**
- **https://www.kaggle.com/datasets/deepcontractor/xbox-game-pass-games-library/data**
- **DSA Textbooks and Lecture Notes: Class Notes**