گزارش فنی شبیه‌ساز پارک تفریحی (Theme Park Simulator)

دانشجو: مهدیار طالبی

۱. مقدمه و اهداف پروژه

هدف این پروژه، طراحی و پیاده‌سازی یک سیستم مدیریت پارک تفریحی با استفاده از ساختمان داده‌های پایه است. در این سیستم، مدیریت زمان به صورت گسسته (Event-driven) انجام شده و مفاهیمی نظیر اولویت‌بندی (VIP)، سیستم صبر (Patience)، امکان ذخیره و بازیابی کل سیستم ازطریق فایل، امکان ورود گروهی و قابلیت بازگشت به عقب (Undo) پیاده‌سازی شده است.

۲. معماری شیءگرا (Object-Oriented Architecture)

سیستم از چندین کلاس اصلی تشکیل شده که به شرح زیر با هم در ارتباط هستند:

- کلاس Visitor: نگهداری اطلاعات فردی (ID، نام، نوع عضویت، میزان صبر).
- کلاس Ride: مدیریت اطلاعات هر دستگاه (ظرفیت، مدت زمان بازی) و نگهداری دو صف (صف انتظار و صف افراد در حال بازی).
- کلاس Event: نمایش یک رخداد زمانی (پایان بازی یا اتمام صبر).
- کلاس Simulator (هسته مرکزی): مدیریت کل جریان برنامه، ارتباط بین ساختمان داده‌ها و هماهنگی توابع TICK و UNDO.
- کلاس LogManager: ثبت و نگهداری وقایع سیستم برای ارائه گزارش نهایی.

۳. توجیه انتخاب ساختمان داده‌ها

در این پروژه تمامی ساختارها به صورت دستی پیاده‌سازی شده‌اند:

- درخت AVL (مدیریت بازدیدکنندگان): برای جستجوی بازدیدکننده بر اساس ID از AVL استفاده شد. به دلیل خاصیت خود-توازنی، عملیات جستجو، درج و حذف همیشه در زمان O(log n) انجام می‌شود که برای تعداد بالای بازدیدکنندگان بسیار بهینه است.
- Min-Heap (مدیریت رویدادها): قلب تپنده شبیه‌ساز است. رویدادها بر اساس زمان وقوع در هیپ ذخیره می‌شوند تا همیشه رویدادی که زودتر رخ می‌دهد در ریشه باشد (O(۱) برای دسترسی و O(log n) برای درج/حذف).
- لیست پیوندی / Linked List (صف انتظار): صف هر دستگاه با لیست پیوندی پیاده‌سازی شد تا امکان حذف از وسط (برای سیستم صبر) و درج در مکان خاص (برای قانون اولویت VIP) فراهم باشد.

- پشته / Stack (قابلیت Undo): برای لغو آخرین دستورات کاربر، معکوس هر عملیات در پشته ذخیره می‌شود تا در صورت فراخوانی UNDO با پیچیدگی O(۱) بازیابی گردد.

---

۴. تحلیل پیچیدگی زمانی O

طبق خواسته پروژه، تحلیل عملیات‌های اصلی به شرح زیر است:

عملیات	پیچیدگی زمانی	علت
ADD_VISITOR	O(log n)	درج در درخت AVL و حفظ توازن آن.
JOIN_QUEUE	O(n)	در بدترین حالت (ورود VIP) باید لیست پیوندی را برای یافتن جایگاه مناسب پیمایش کرد.
TICK	O(k /log m)	k تعداد رویدادهای همزمان و m تعداد کل رویدادها در هیپ است.
UNDO	O(۱)	بازخوانی آخرین عملیات از بالای پشته (بدون احتساب اجرای خود دستور معکوس).

---

۵. قابلیت‌های پیشرفته (Advanced Features)

برای دریافت امتیاز ویژه، موارد زیر پیاده‌سازی شد:

۱. سیستم صبر (Patience): به صورت خودکار با هر TICK بررسی شده و در صورت اتمام صبر، بازدیدکننده با یک رویداد داخلی از صف خارج می‌شود.

۲. Save/Load سیستم: وضعیت کامل پارک (شامل درخت بازدیدکنندگان، صف‌ها و افراد در حال بازی) در فایل ذخیره و بازیابی می‌شود.

۳. ورود گروهی (Family Join): امکان افزودن همزمان چندین نفر (خانواده)

تشریح دقیق معماری و روابط کلاس‌ها (Class Relations)

در این پروژه، از معماری چندلایه استفاده شده است تا منطقِ مدیریت داده (AVL/Heap) از منطقِ شبیه‌سازی (Simulator) جدا باشد.

۱. کلاس Visitor (موجودیت پایه)

این کلاس «شناسنامه» هر فرد در سیستم است.

- وظیفه: نگهداری وضعیت فردی بازدیدکننده شامل شناسه (ID)، نام، نوع (VIP/Normal) و میزان صبر (Patience).

- فیلد کلیدی isBusy: این فیلدِ منطقی، تضمین می‌کند که یک بازدیدکننده همزمان در دو صف نباشد (قانون یکتایی وضعیت).
- رابطه: به صورت اشاره‌گر (Pointer) در گره‌های AVL و گره‌های لیست پیوندی (Queue) قرار می‌گیرد تا از کپی‌های اضافی در حافظه جلوگیری شود .

۲. کلاس Ride (مدیریت دستگاه بازی)

هر دستگاه یک واحد پردازشی مستقل است.

- وظیفه: مدیریت ظرفیت (Capacity) و مدت زمان سرویس‌دهی (Duration) .
- متد processQueue: در هر چرخه، به اندازه ظرفیت دستگاه، افراد را از صف انتظار (Queue) خارج و به لیستِ در حال بازی (Serving) منتقل می‌کند .
- رابطه: شامل یک شیء از RideQueue برای مدیریت صف انتظار است.

۳. کلاس AVLTree (ایندکسینگ و جستجو)

این کلاس نقشِ دیتابیسِ سریع پارک را ایفا می‌کند.

- توابع کلیدی:
  - insert(Visitor*): درج گره جدید و انجام چرخش‌های (Left/Right) برای حفظ توازن درخت در زمان O(log n) .
  - find(int id): جستجوی دودویی برای دسترسی سریع به اطلاعات فرد جهت ورود به صف یا تبدیل به VIP.
- رابطه: تمام کلاس‌ها برای پیدا کردنِ یک Visitor ابتدا به این درخت مراجعه می‌کنند.

۴. کلاس RideQueue (لیست پیوندی اولویت‌دار)

پیاده‌سازی سفارشی صف که فراتر از یک FIFO معمولی عمل می‌کند .

- وظایف و توابع:
  - enqueueVIP(Visitor*): پیاده‌سازی قانون درج پشت آخرین VIP و قبل از اولین فرد عادی .
  - removeById(int id): حذف یک گره از وسط لیست (برای دستور LEAVE_QUEUE یا سیستم صبر) .
- رابطه: توسط کلاس Ride برای مدیریت صف استفاده می‌شود.

۵. کلاس MinHeap (زمان‌بندی و موتور رویداد)

مدیریتِ ترتیبِ وقوعِ اتفاقات در آینده را بر عهده دارد .

- وظیفه: رویدادها را بر اساس زمان وقوع (Timestamp) مرتب می‌کند تا همیشه رویدادی که زودتر رخ می‌دهد در ریشه باشد.
- متد push(Event) و pop(): درج و استخراج رویداد با پیچیدگی O(log n).
- رابطه: شبیه‌ساز در هر TICK از این کلاس می‌پرسد که «الان وقتِ انجام چه کاری است؟».

۶. کلاس UndoStack (پشته بازگشت)

وظیفه ذخیره تاریخچه اقدامات معکوس‌ناپذیر کاربر را دارد .

- استراتژی: به جای ذخیره کل وضعیت، «عملگر معکوس» را ذخیره می‌کند؛ مثلاً برای ADD عمل DELETE را ذخیره می‌کند .
- محدودیت: دستور TICK به دلیل تغییرات فیزیکی در سیستم، قابل ذخیره در پشته نیست.

۷. کلاس Simulator (شبیه ساز مرکزی)

هماهنگ‌کننده تمام ساختمان داده‌های فوق است.

- متد tick(int t): قلب شبیه‌ساز که زمان را جلو برده و رویدادهای سررسیده را از Heap استخراج و اجرا می‌کند .
- متد report(): جمع‌آوری آمار نهایی شامل تعداد کل سرویس‌دهی‌ها و میانگین زمان انتظار .

تحلیل پیچیدگی و روابط

روابط بین کلاس‌ها به گونه‌ای تنظیم شده است که اشاره‌گر به Visitor نقطه اتصال همه است. اگر یک بازدیدکننده در AVL به VIP تبدیل شود ، به دلیل استفاده از Pointer، این تغییر بلافاصله در صفِ دستگاه (RideQueue) و هیپِ رویدادها (MinHeap) نیز اعمال می‌شود .

تحلیل نهایی:

این معماری تضمین می‌کند که سیستم حتی با افزایش تعداد بازدیدکنندگان (به سمت بینهایت)، به دلیل استفاده از ساختارهای لگاریتمی مانند AVL و Heap، دچار افت کارایی نشود.