ATILIM ÜNİVERSİTESİ

Faculty of Engineering

Computer Engineering Department

CMPE 323 – Algorithms

HOMEWORK #2

Academic Year: Fall 2020-21
Due Date: 02.12.2020 (Wednesday), Hr: 23:59
Due Place: Upload to the Course Moodle Site
Instructor: Assoc. Prof. Dr. Hürevren Kılıç
Assistants: Buğra Yener Şahinoğlu, İbrahim Tarakçı

(Game Event Scheduler)

In multi-player games, the players generate various game events throughout a game session, independently and asynchronously. Each game event can be described by a unique event ID (i.e. **what**), event start time (i.e. **when**), the player ID who generated the event (i.e. **by whom**) and its location (i.e. **at where**) x, y, z coordinates). In order to make realistic execution of the game, we need a **scheduler** unit in our game engine that keeps track of the generated events and returns the next event to be executed to the **event executer** unit of the engine **in the order of event start times** when it is requested by the executer. Also, the executer must be able to get the **list of all information** about all the **events** kept by the scheduler **sorted** in the order of their potential execution at the time of the request and **display** the **list**.

In your implementation, assume that if two events happen at the **same time** slot then tie is **broken randomly**. Also, notice that because of possible **communication delays** and **asynchrony**, the generated events may not be received by the scheduler in their start order but as an **unordered sequence**. See Figure 1, for the visual description of the system.

Design and implement a **Game Event Scheduler** as a **priority queue** based on a **heap** data structure. For simplicity, you may assume that all the requests made by player generated events and the game executer can be **read from an input file** having the below format:

<request> → <insert_event> | <get_next_event> | <list_all_events>
<insert_event> → "I" **BLANK** <event_ID> **BLANK** <time> **BLANK** <player_ID> **BLANK** <location>
<event_ID> → **POSITIVE_INT**

<time> → **POSITIVE_INT**

<player_ID> → **POSITIVE_INT**

<location> → <x_coordinate><y_coordinate><z_coordinate>

<x_coordinate> → **INT**

<y_coordinate> → **INT**

<z_coordinate> → **INT**

<get_next_event> → "G"

<list_all_events> → "L"

SAMPLE INPUT:

I 3 **7** 3 1 2 -7
I 1 **3** 5 3 2 6
I 2 **5** 4 4 -4 4
G
L
I 4 **6** 4 -5 1 2
L
G
L

Your program must be able to generate the **corresponding output** given below.

CORRESPONDING OUTPUT:

2 **5** 4 4 -4 4
3 **7** 3 1 2 -7
---
2 **5** 4 4 -4 4
4 **6** 4 -5 1 2
3 **7** 3 1 2 -7
---
4 **6** 4 -5 1 2
3 **7** 3 1 2 -7

PS:

1. You are **required** to work either **alone or** in **at most two-person** group.
2. If you wish to work as a two-person group, both of the group members should send me (**NOT to the course assistants**!) an **e-mail** (**hurevren.kilic@atilim.edu.tr**) indicating the

name of his/her **agreed** group member until 22.11.2020 (Sunday) Hr:23:59. **Otherwise**, it is **assumed** that you will work **alone** (as **default)**.

3. **Late submissions** will be graded by using formula **100-10*d^2** where **d** is the number of **late** submission **days**.
4. Note that besides from submitting the homework, you are also required to **code review & demonstration** of your code.
5. Percentages of **submission** and **demo** parts are **%70** and **%30** of your overall Homework #2 grade, respectively. Submissions without online code review & demonstrations gets 0 (zero) grade from both parts.
6. **Time table** for the **code review & demos** is planned to be **announced later** at the course Moodle site.
7. Your implementation language **C, C++ or Python**. Do **NOT use** already **existing libraries** that implements the **heap data structure** but implement them by yourself that satisfies only the above requirements. (You may refer & implement the **pseudo-codes** given in **Chapter 6 –** Heapsort of our **textbook**)

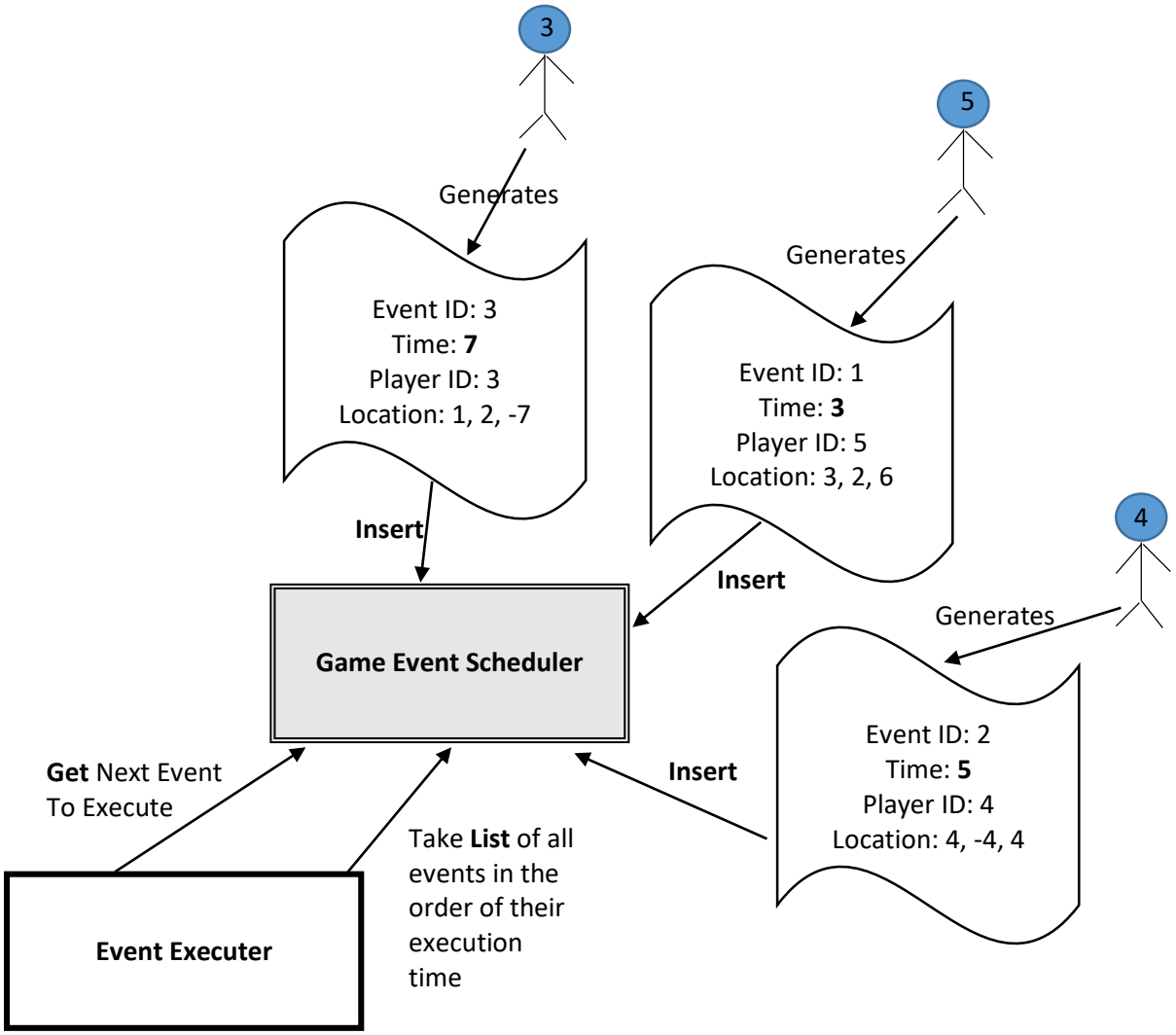

Figure 1. Visual description of the system.