DPS Project 2024

Distributed and Pervasive Systems Lab Course

May 2024

1 Project Description

WatchOut is a real-life game that merges modern technology with the classic thrill of hide-and-seek [1]. Each player is provided with a smartwatch containing an application (Java process) specifically designed for the game. Players' smartwatches coordinate to choose the hiders and the seeker. Moreover, since WatchOut is a game that requires a certain intense physical activity, the smartwatches are equipped with a photoplethysmography sensor, capable of detecting the heart rate by low-intensity infrared light. Therefore, a game manager, thanks to an administration client, can control the health status of the participants and intervene if needed. Figure 1 shows the overall architecture of the WatchOut game.

The goal of the project is to implement the Administration Server, the Administration Client, and a peer-to-peer system of players. The Players autonomously organize themselves when they need to elect a seeker and when they concurrently want to reach the home base. Moreover, they periodically send heart-rate measurements to the Administration Server.

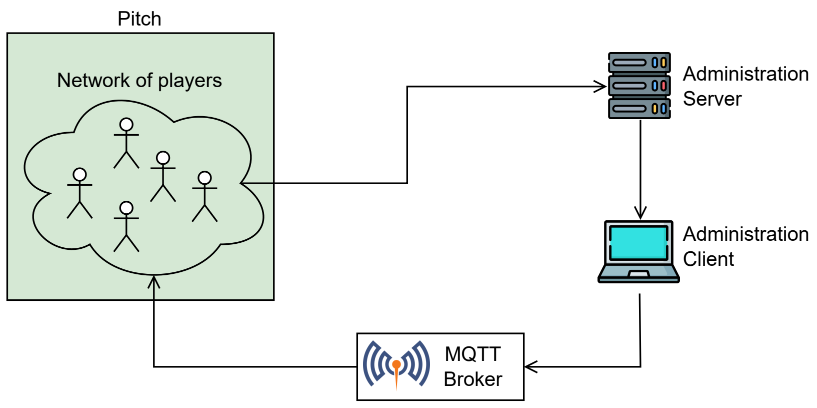

Figure 1: Overall architecture of WatchOut.

[1] https://en.wikipedia.org/wiki/Hide-and-seek

1.1 General Rules

Players are divided into two categories: one seeker, i.e. the one who has to catch, and an indefinite number of hiders, i.e. those who have to escape. Both hiders and the seeker can move within the pitch. If the seeker moves to a point where a hider is located, the hider is considered tagged and therefore eliminated from the game. The goal of the seeker is to tag as many hiders as possible. The goal of the hiders instead is to reach the home base without getting tagged by the seeker. Once a hider reaches the home base, they need to wait there for 10 seconds after which they are considered safe. Only one player at a time can wait at the home base, so, if a hider wants to reach the home base while another hider is waiting there, they need to wait until the home base is cleared. The game ends when all the hiders are either safe or eliminated.

1.2 Game Phases

1.2.1 Preparation

When a new player enters the game (i.e. their process is launched), they registers on the administration server, if the registration is successful the player's process receives from the server their initial position on the perimeter of the pitch, together with the list of other players already present. When the game starts each player receives a notification from the game manager, at which point phase 0 of the game begins.

1.2.2 Phase 0

During phase 0 the players' processes coordinate to choose who is the seeker. Once a consensus is reached on who the seeker is, phase 1, or the actual game, begins.

1.2.3 Phase 1

During phase 1 the goal of the hiders is to reach the home base without getting tagged by the seeker. Upon reaching home base a player must stay there for 10 seconds after which that player is considered safe. Only one player at a time can wait in the home base. If the home base is free the hider can go there. Every player knows the position of the others. At the beginning of phase 1, the seeker starts trying to tag hiders, they do that by moving toward the player they are closest to. When the seeker reaches the point toward which they were moving there are two possibilities:

- The player they were looking for is still there. In this case, the player is tagged and thus eliminated from the game (the process is till running).
- The player they were looking for acquired permission to go to the home base and thus fled. In this case, that player is not considered tagged.

In any case, as the next step the seeker starts again trying to tag a new player, moving toward the closest player still in the game.

1.2.4 End of the Game

When a player is tagged, they are eliminated from the game. When a player remains for 10 seconds at the home base, they are safe. Once a player is safe or eliminated, they communicates it in broadcast to all other players. When all hiders are either safe or eliminated the game ends. It is the network of players themselves who understand when the game is over.

1.2.5 New players

New players can enter the game at any phase. If they enter when the seeker election has not yet taken place or is in progress, they must be able to participate in the election, if they enter during phase 1, they are automatically hiders.

1.3 Internal Representation of the Pitch

The pitch of WatchOut is represented as a 10×10 grid (see Figure 2), each cell of the grid represents a point with the respective coordinates. For simplicity, assume that the value of the coordinate is expressed in tens of meters. The home base is located in a squared area defined by points $(4,4)$, $(4,5)$, $(5,5)$ and $(5,4)$ included. To move from one point to another, the players move along the diagonal. Therefore, the distance between a point $A(x_a, y_a)$ and a point $B(x_b, y_b)$ is defined, according to the Euclidean distance metric, as follows:

$$d(A,B) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

The players move at a speed of 2 meters per second.

2 Applications to Be Implemented

For this project, you are required to develop the following applications:

- Player: the process that runs on the smartwatches of the WatchOut players.
- Administration Server: a REST server that dynamically adds/removes players of WatchOut and computes statistics about the health status of participants.
- Administration Client: a client that allows the game manager to query the Administration Server to obtain information about the health status of participants. The Administration client is also in charge of notifying the players via MQTT of the start of the game.

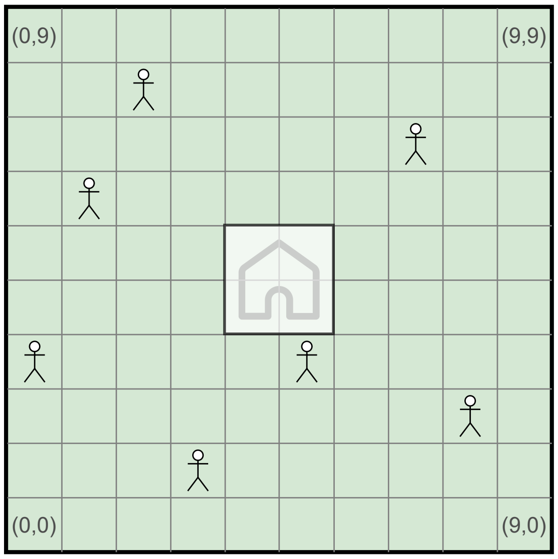

Figure 2: Internal representation of the WatchOut pitch.

Please note that each player is a stand-alone process and so, it must not be implemented as a thread. In the following, we provide more details about the applications that must be developed.

3 Player

Each player has a smartwatch on which is running a Java process, which is responsible for:

- Coordinating themself with the other players by using gRPC to decide which player will be the seeker (the other players will be the hiders), and to understand which player will be able to access the home base.
- Informing the other players when permission to go to the home base is acquired. For simplicity assume that once a player acquire permission to go to the home base it cannot be tagged, so once reached the home base, after being there for 10 seconds, they are automatically safe.
- Sending to the server information about the values detected by the heart-rate sensor.

3.1 Initialization

A player is initialized by specifying:

- ID.

- Listening port for the communication with the other players.
- Administration server's address.

Once it is launched, the player process must register itself to the system through the Administration server. If its insertion is successful (i.e., there are no other players with the same ID), the player receives from the Administration server:

- Their randomly generated starting position on the pitch (The initial position is on the perimeter of the pitch, therefore, at least one of X or Y must have a value of either 0 or 9).
- The list of the other players already present in the game (i.e. address and port number of each player).

Once the player receives this information, it starts acquiring data from the heart-rate sensor. If there are already other players in the game, the new player presents themself to the other players by sending them their position on the pitch. Finally, the player subscribes to the MQTT topic in which the game manager will communicate the start of the game and additional text messages. When the game manager decrees the start of the game, each player is notified by the Administration Client that the game has started, at which point phase 0 of the game begins.

3.2 Distributed synchronization

3.2.1 Seeker election

The players of the game must use a distributed and decentralized algorithm to decide who will be the seeker. Specifically, the seeker will be the player whose starting position is the closest to the home base. If two or more players are the same distance from home base, the seeker will be the player with the highest ID.

3.2.2 Mutual exclusion

During phase 1 of the game, each player can try to reach the home base. Once reached, they must wait 10 seconds, after which they are considered safe. Only one player at a time can access the home base, so, you have to implement one of the distributed algorithms of mutual exclusion introduced in the theory lessons in order to coordinate the access to the home base. For the sake of simplicity (and since the processes run on the same machine), you can assume that the clocks of the players are properly synchronized and that the timestamps of their requests will never be the same (like Lamport total order can ensure). When a player acquires rights to go to the home base, they starts moving towards it according to the mechanism described in 1.3.

3.3 Heart-Rate Sensor

The smartwatches are equipped with a photoplethysmography sensor (a sensor that uses light to detect heartbeat). During the game, the players' smartwatches collect heartbeat information so that the game manager can monitor the players' health status and possibly stop the game if abnormal values are detected. Each heart-rate sensor periodically produces measurements of the heart rate (HR) value of the player. Every single measurement is characterized by:

- HR value.
- Timestamp of the measurement, expressed in milliseconds.

The generation of such measurements is produced by a simulator. In order to simplify the project implementation, it is possible to download the code of the simulator directly from the page of the course on MyAriel, under the section Projects. Each simulator assigns the number of seconds after midnight as the timestamp associated with a measurement. The code of the simulator must be added as a package to the project, and it must not be modified. During the initialization step, each player launches the simulator thread that will generate the measurements for the heart-rate sensor. Each simulator is a thread that consists of an infinite loop that periodically generates (with a pre-defined frequency) the simulated measurements. Such measurements are added to a proper data structure. We only provide the interface (`Buffer`) of this data structure that exposes two methods:

- `void add(Measurement m)`.
- `List <Measurement> readAllAndClean()`.

Thus, it is necessary to create a class that implements this interface. Note that each smartwatch is equipped with a single sensor. The simulation thread uses the method `addMeasurement` to fill the data structure. Instead, the method `readAllAndClean`, must be used to obtain the measurements stored in the data structure. At the end of a read operation, `readAllAndClean` makes room for new measurements in the buffer. Specifically, you must process sensor data through the sliding window technique that was introduced in the theory lessons. You must consider a buffer of 8 measurements, with an overlap factor of 50 %. When the dimension of the buffer is equal to 8 measurements, you must compute the average of these 8 measurements. A player will send these averages to the Administrator Server.

3.4 Send Data to the Server

Every 10 seconds, each smartwatch has to communicate to the Administrator Server the list of the averages of the heart rate measurements computed after the last communication with the server. This list of averages must be sent to the server associated with:

- The ID of the player.
- The timestamp in which the list was computed.

This communication takes place through a rest endpoint of the Administration Server, as will be explained below.

3.5 leaving the Game

For the sake of simplicity, it is assumed that no player, once they enter the game, leaves it.

4 Administration Server

The Administration Server collects the IDs of the players registered to the system and also receives from them the heart-rate sensor values. This information will then be queried by the game manager (through an Administrator Client). Thus, this server has to provide different REST interfaces for:

- Managing the initialization of the players.
- Receiving the local heart rate statistics.
- Enabling the game manager to execute the queries from the Administration Client.

4.1 Player Rest Interface

4.1.1 Players initialization

The server has to store the following information for each player joining the game:

- ID.
- Address (in your case it will be localhost).
- The Port Number on which it is available to handle the communication with the other player processes.

Moreover, the server is in charge of assigning to each joining player a randomly chosen position on the perimeter of the pitch. A player can be added to the network only if there are no other players with the same identifier. If the insertion succeeds, the Administrator Server returns to the player:

- The starting position of the player.
- The list of players that already joined the game, including for each of them the ID, the address and the port number for communication.

4.2 Heart Rate Statistics

The Administrator Server must provide an interface to receive the local heart rate statistics from the smartwatches of the players. These data have to be stored in proper data structures that will be used to perform subsequent analysis. During the development of the project, make sure that you correctly synchronize read and write operations made on these data structures. Indeed, the players of the game could send their local statistics while the Administrator Client is requesting to the Administrator Server to perform some computations on such statistics.

4.3 Administration Client Rest Interface

When requested by the Administration Client through the interface described in Section 5, the Administrator Server must be able to compute the following statistics:

- The list of the players currently in the game.
- The average of the last n heart rate values sent to the server by a given player.
- The average of the heart rate values sent by all the Players to the server that occurred between timestamp $t1$ and timestamp $t2$.

5 Administration Client

The Administration Client consists of a simple command-line interface that enables the interaction with the REST interface provided by the Administration Server. Hence, this application prints a straightforward menu to select one of the services offered by the administration server described in Section 4.3 and to enter possible required parameters. The Administration Client should also allow the game manager to send custom text messages to all players.

6 Simplifications and Restrictions

It is important to recall that the scope of this project is to prove the ability to design and build distributed and pervasive applications. Therefore, all the aspects that are not strictly related to the communication protocol, concurrency, and sensory data management are secondary. Moreover, it is possible to assume that no nodes behave maliciously. On the contrary, you should handle possible errors in data entered by the user. Furthermore, the code must be robust: all possible exceptions must be handled correctly. Although the Java libraries provide multiple classes for handling concurrency situations, for educational purposes, it is mandatory to use only the methods and classes explained during the laboratory course. Therefore, any necessary synchronization data structures (such as lock, semaphores, or shared buffers) should be implemented from scratch and will be discussed during the project presentation. Considering the communication between the Players' processes, it is necessary to use the gRPC framework. If broadcast communications are required among Players, these must be carried out in parallel and not sequentially.

7 MQTT Broker

The MQTT Broker on which WatchOut relies is online at the following address: `tcp://localhost:1883`. The game manager use this broker through the Administration Client to notify the players when the game starts and to broadcast custom text messages to the players for additional communications.

8 Project Presentation

You must develop the project individually. During the evaluation of the project, we will ask you to discuss some parts of the source code and we will also check if it runs correctly considering some tests. Moreover, we will ask you one or more theoretical questions about the theoretical content of the lab lessons. You will run your code on your machine, while the source code will be discussed on the tutor's machine. You must submit the source code before discussing the project. For the submission, you should archive your code in a .zip file, renamed with your university code (i.e., the "matricola"). For instance, if your university code is 012345, the file should be named 012345.zip. Then, the zip file should be uploaded at http://upload.di.unimi.it. It will be possible to submit your project a week before the exam date. You must complete the submission (strictly!) two days before the exam date at 11:59 PM (e.g., if the exam is on the morning of the 15th, you must complete the delivery before 11:59 PM of the 13th). We suggest the students to use, during their project discussion, the Thread.sleep() instructions to show how the synchronization problems have been correctly handled. We recommend analyzing all the synchronization issues and creating schemes to illustrate how the components of your project communicate. These schemes may represent the format of the messages and the sequence of the communications that occur among the components during the different operations involved in the project. Such schemes can be very helpful during the project discussion. It is not necessary to show the structure of the classes of your project, but only the main aspects regarding the synchronization and the communication protocols you have implemented. During the presentation of the project, we will ask you to test your code by launching at least 4 or 5 Players, the Administration Server, and a single Administration Client.

9 Plagiarism

The reuse of code written by other students will not be tolerated. In such a case, we will apply the sanctions described on the course website, in the section Student Plagiarism of General Information. The code developed for the project must not be published (e.g., GitHub) at least until March of the next year.

10 Optional Parts

In order to encourage the presentation of the project in the first exams sessions, the development of the first optional part becomes mandatory from the September session (included), while both the first and the second optional parts are mandatory from the January session (included).

10.1 First Optional Part: Leaving the Game

For this optional part, you must also consider the possibility of a player leaving the game. You can assume that each player process terminates only in a controlled way, meaning that the player explicitly exits the game. A player can leave the game in any of the phases described in 1.2, by entering the "exit" command into the command line of the process. When leaving the game, the player process must follow these 2 steps:

- Notify the other players.
- Request the Administrator Server to leave the game.

Note that when a player receives communication of another player's exit, he must handle it accordingly.

10.2 Second Optional Part: uncontrolled exit

For the second optional part, you can no longer assume that each player process terminates only in a controlled way. Hence, all the distributed synchronization mechanisms implemented for the project must take into account that a player could leave the system in an uncontrolled way (i.e. Without the ability to notify the server or other players).

11 Updates

If needed, the current text of the project will be updated. Changes in the text will be highlighted. Please, regularly check if new versions of the project have been published on MyAriel.