Design Document

Wonderland Software Framework

Date: Nov 27, 2021

Table of Contents:

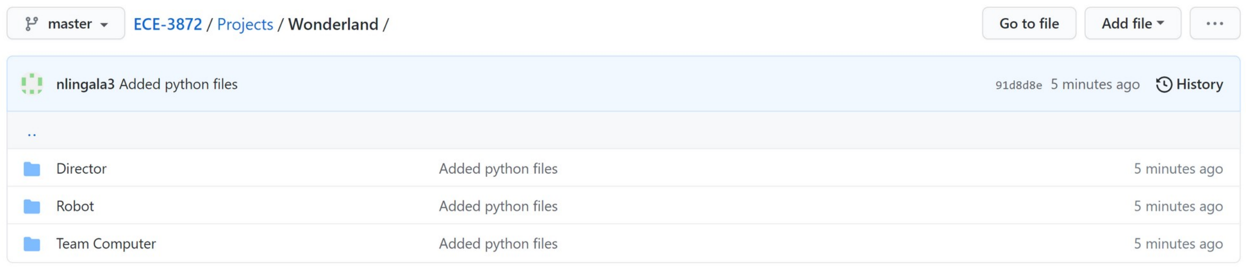
System Overview:

For the Wonderland project you will be using the software framework outlined in this document. Each team will construct a single robot thespian. The class will collaboratively write a version of *Alice in Wonderland*. Each team will write lines for their character and upload those to a server called the Director. Once the play starts the director will send the lines to the robot at the correct time. The robots will need to process/decode and then recite the lines to form a cohesive performance with all the robots.

System Design:

The system includes **team computers**, **the director**, and **team robots**. Inputs to the system includes scripts uploaded from the team computers to the director and the different commands enabling the users from the team computers to appropriately configure the scripts on the director system. The output of the system will include the transferring of files from the director to robots at the correct cue time. The robots will be responsible for processing/decoding these files to playback their scripts. The other inputs will include the starting and terminating of each subsystem.

Teams are responsible for encoding, uploading, and decoding of line files Line 1 and the construction of the robot. The software framework can be found on this GitHub page: https://github.gatech.edu/tbrothers3/ECE-3872. Teams must use the software provided and change only the parts of the code dictated by the comments.



master ECE-3872 / Projects / Wonderland /

Go to file | Add file | ···

nlingala3 Added python files — 91d8d8e 5 minutes ago — History

..

Director	Added python files	5 minutes ago
Robot	Added python files	5 minutes ago
Team Computer	Added python files	5 minutes ago

Figure 1. Folders containing software on GitHub

Specifications:

Machine	Raspberry Pi 3 B (Raspberry Pi 3, 4 and Zero W will also work)
OS	Raspberry Pi OS (Linux Distribution)
Python Version	Python 3.7 and newer
Dependencies	Wi-Fi Hotspot (Tested on Windows Laptop Hotspot)

The minimum specifications for the robot, director, and team computer can be seen in the specifications table.

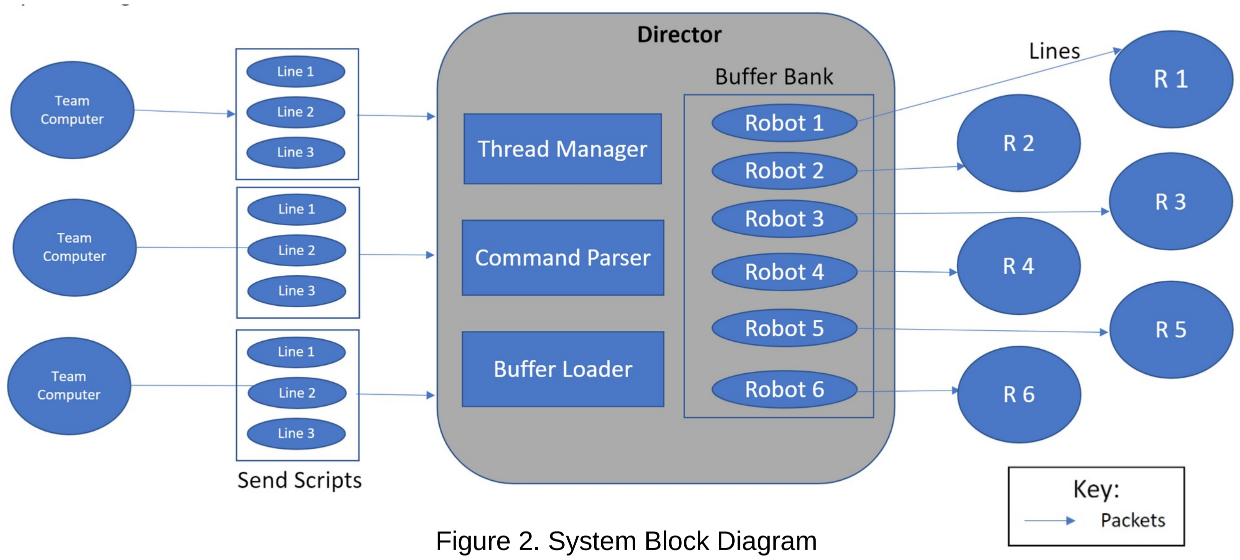


Figure 2. System Block Diagram

Subsystem Design:

The system's design incorporates two software subsystems: Team Computer – Director and Director – Robot applications. Each subsystem is written in Python3 and will rely on a private network with limited capacity and bandwidth. The director is the mediator between the two subsystems.

Team Computer - Director Subsystem:

This subsystem consists of a client.py file that runs on the team computer and a server.py file that runs on the director which is a Linux server. Once a successful connection is established between a team computer and the director, the software will read the commands that are sent from the team computer and process them on the director computer. The director will store the files and send messages back to the team computer when the process is completed.

To Start:

1. Run python3 server.py in the correct path on the director. (Use Thonny IDE)
2. Run python3 client.py

Commands: Use these commands to check and upload new scripts to the server.

1. LIST: List all the files on the director.
2. UPLOAD <path>: Upload a file to the director.
3. DELETE <filename>: Delete a file on the director.
4. LOGOUT: Disconnect from the director.
5. HELP: List all commands.
6. SEND: <Type_your_message> (no spaces) sends a message to the director
   i. At this time the director does not process received messages and will just output the message to the director terminal
   ii. Included for future custom commands for the director

File Naming Format:

- "<ROBOT#>_<CueTime in Milliseconds>.xxx"
  - For example: R01_1000.txt, R03_5500.mp3
- Each *file* is called a *line*
- The complete collection of *lines* for a robot is called the *script*

Notes:

- If sending multiple files, students can place all line files in a .zip
- Director will automatically extract all line files in the zip.
- **Caution**: Zip folder will be deleted after extraction on the director.

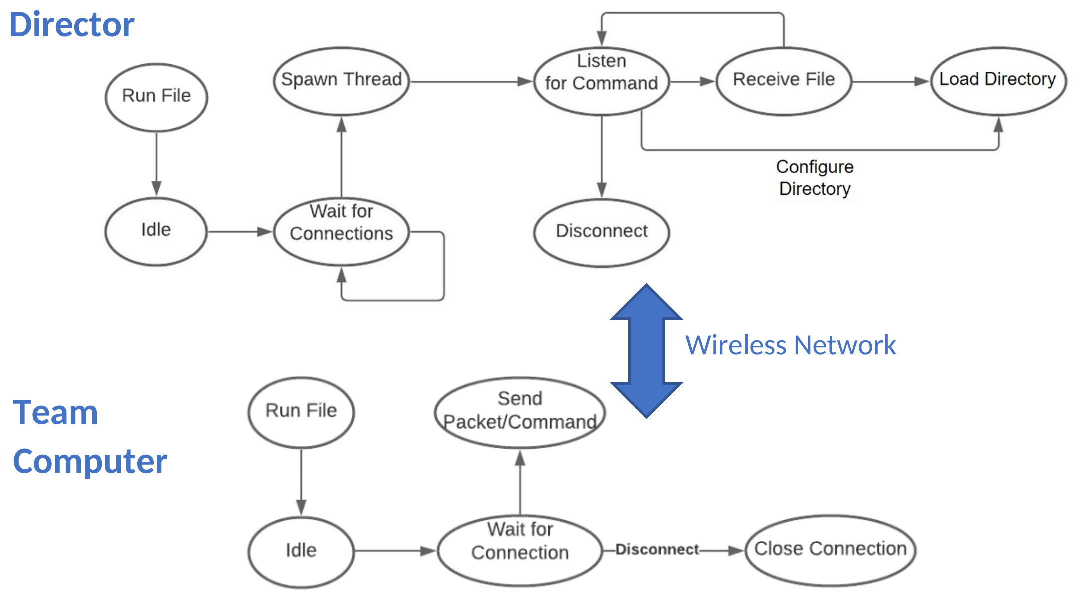


Figure 3. Team Computer – Director Software Block Diagram

Director - Robot Subsystem:

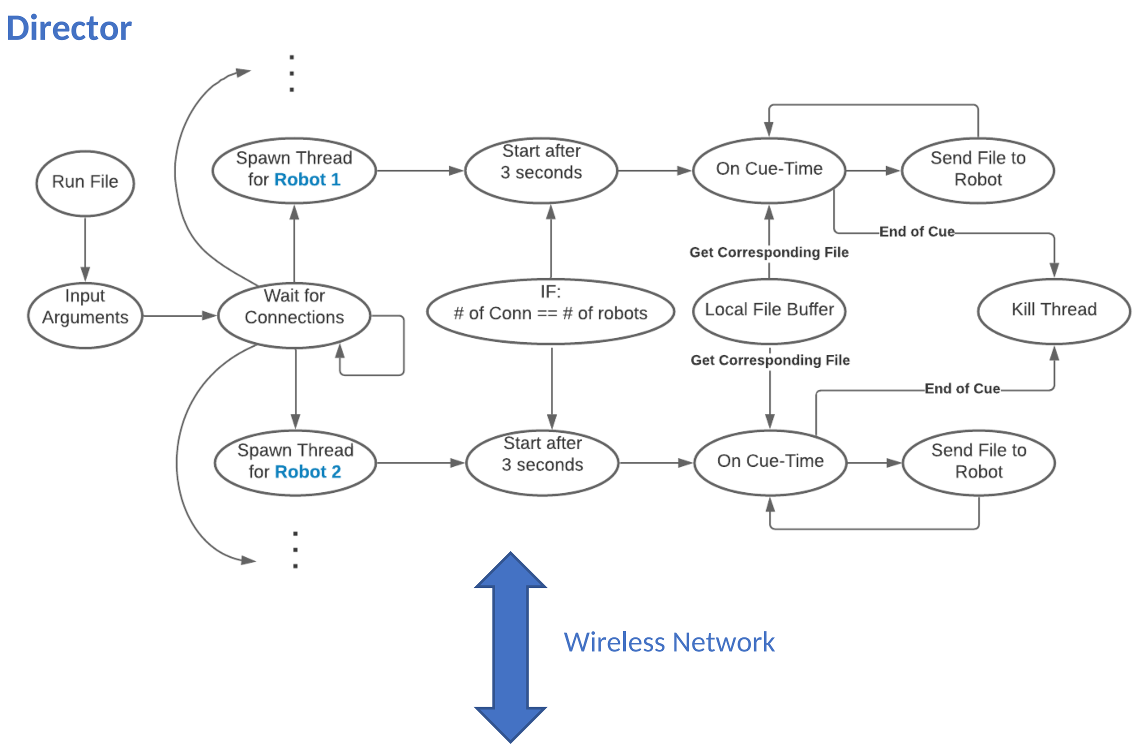
This subsystem consists of a play.py file that runs on the director and two files that run on the robot: rob.py and recite.py. Once the play.py file is started, the director software will automatically load the line files onto a local buffer in chronological order and then remain in the idle position to listen for robot connections. The director prompts the user to input the **number of robots** in the play and their corresponding **IP addresses** in order. This order establishes the numbering of robots when the director sends the files (see comments in play.py). Once this is inputted and the correct number of connections has been found, the director will automatically transfer the appropriate files to each robot on cue. The play will stop when all the files have been sent or the director's program is manually stopped.

To Start:

1. Run python3 play.py in the correct path on the director. (Use Thonny IDE)
2. Run python3 rob.py on robot.

Notes:

- Large files will add delay on the network
- Files are automatically sent 3 seconds after the correct number of robots have connected to the Director.



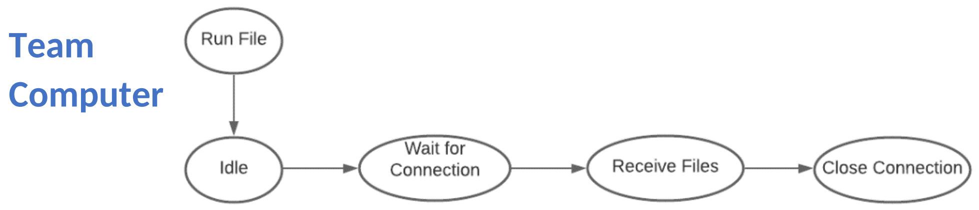


Figure 4. Director – Robot Software Block Diagram

Robot Student Software:

The students must complete recite.py to process and decode their scripts on the Robot side. This program will continuously poll the directory to search for new files that have been sent from the director. One example of reading the file has been provided to get you started.

Next Steps:

1. More flexible start time for director (commencement of the play)
2. Implement automatic robot ordering and selection for scenes. (no input of robot IPs)
3. Implement messaging for custom functions on the director
4. Create a web app to control the director
5. Create the ability for team computer to automatically load a variety of scripts.
6. Error checking for incorrect file formats