Lab 3: GUI Tutorial

ECE 180D: Systems Design Laboratory

Contents

1 Introduction

This tutorial will point you toward actually creating a user interface for a game. The G (standing for graphics) may be a bit too generous for what we cover in this tutorial but this serves as a start. We start with a command-line interface and then move toward a PyGame tutorial, where you will learn the basics of how to design a graphics user interface for a game. Then, we will want to take what we have learned so far and try and create a multiplayer game.

As part of the project, we want user interaction with the software to determine the course of gameplay. Thus, we first begin with a command-line (sometimes referred to as a "text"-based) version of the game. However, while we can always hack together a project that is cool on the command-line for ease of programming, one of the goals is that we can share our projects with our friends and family. Thus, we would also like to move to something a bit more pleasing to the general person.

2 Designing a Game

Before we get into the user interface, one thing to consider is how the gameplay flow should work in the first place. One great way to do this is to consider the state machine. For the purposes of this tutorial, we will be referring to Rock-Paper-Scissors. For all the tasks, you may implement rock-paper-scissors or a simplified version of your project.

Consider the most basic state machine of rock-paper-scissors from a program's perspective, which we show in Figure 1. However, given the simplicity, we only have one possible place to interact. In fact, from the

player's perspective, the program is effectively always in the same state, asking it for the next input.

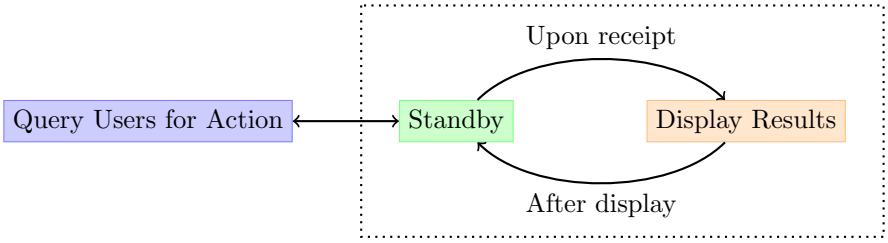


Figure 1: Most basic RPS state machine is drawn in the dotted box. Here, the green box represents an interactive state, while the orange box represents that this is simply a transition state.

We can add more states, for instance a menu state or a confirmation state. One such example is shown in Figure 2. As you add more interactions, your state space will also adjust and shift accordingly. We won't ask you to create a state machine for now, but please keep in mind that we will want to see one in your midterm presentation.

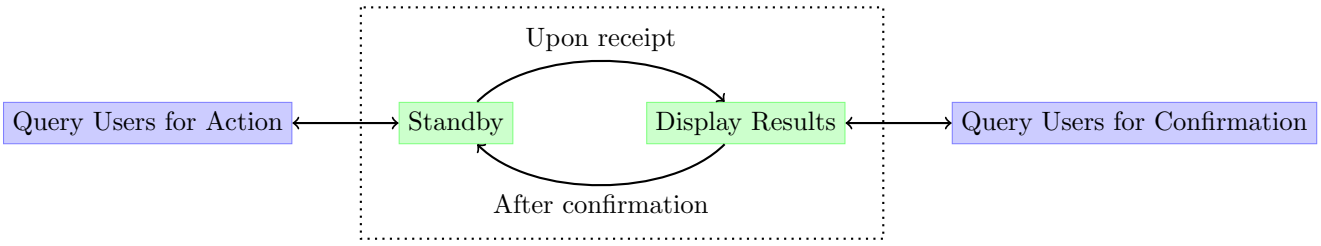


Figure 2: One step more complicated RPS state machine. Here, we now have two interactive states.

3 Command-Line Game

3.1 Against AI

Create a Rock-Paper-Scissors bot in command-line based on the state machine displayed in Figure 1. In the most basic formulation:

1. Accept user input that will map to rock, paper, or scissors.
2. Have your script randomly generate an action. Feel free to use one of the Python random functions.
3. Print out the result of the game. If you want you can tabulate your game results.
4. Ask for the next user input.

Please screenshot a picture of this basic version of the game and then continue playing.

3.2 Communicating Across MQTT

Partner up with someone in your group. Now, please combine the MQTT script from last week with the basic command-line version of rock-paper-scissors that you have just coded. There are two ways to do this:

1. Distributed: You will simply have two MQTT clients that send each other the actions. The results are computed individually on each computer and displayed.

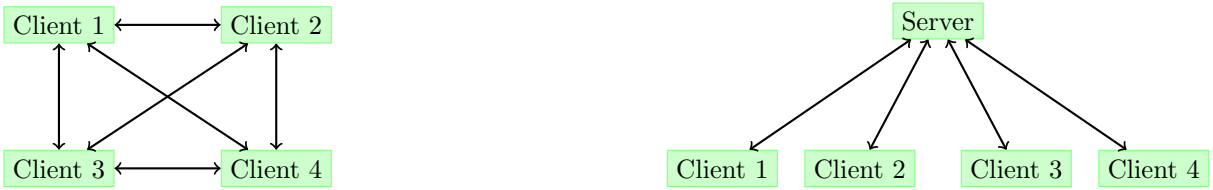
2. Centralized Server: You will have 3 total scripts running. Both players will each have a game client script and then there will be a server that takes care of all the game actions and then sends the results back to the clients. The server can run on any of the two computers (just pick one). The clients don't actually interact with each other whatsoever.

Depending on which one you pick, you will have a slightly different state machine flow.

While for a 2-player game the distributed solution sounds more appealing, keep in mind that the centralized solution is more scalable in terms of software complexity. On the other hand, with some extra code, distributed solutions can help by increasing the amount of resources used (2 computers rather than 1), which can reduce the amount of network overhead and processing required by the centralized node. Thus, as a comparison:

- If you require all pair-wise connections for a distributed solution (which is the most naïve solution), distributed networks would require $O(n^2)$ connections, while the centralized solution requires only $O(n)$.
- Distributed networks require code to be distributed amongst each client across all updates.
- Centralized networks require the server to receive all the data, handle all the data, and then send all the data back. This can incur quite a bit of networking and processing time.

A 4-player representation of each scheme is shown in Figure 3. In truth, you will probably use a mixture of distributed and centralized schemes depending on the lag experienced in your game, as each communication will incur some amount of communications lag to the system. For instance, it might be better to do IMU classification in a distributed way and then send the result to the server to reduce network traffic.



(a) Distributed clients (b) Centralized server

Figure 3: Communication Schemes

As specifications:

1. Have at least two computers run scripts that accept user input that will map to rock, paper, or scissors.
2. Send the input to another script. Also open communications to receive messages. If distributed, the other message will be the other player's input. If centralized, it will be the result.
3. Depending on the message, determine the result and print it out to the user.
4. Tabulate the result.
5. Ask for the next user input.

4 PyGame

4.1 Creating a GUI

Now, we want to turn this very simple command-line version of the game and create a graphical user interface. Please refer to this PyGame tutorial. This tutorial slowly walks you through how to build a

side-scrolling shooter game. While we have had a team do that game in the past, it is a relatively difficult task to integrate full multiplayer in a real-time game. Therefore, you can probably diverge from the tutorial somewhere halfway to two-thirds of the way down (feel free to keep going for your own reference).

For now, our goal is to implement a basic user interface of rock-paper-scissors. For now, we consider just the single player version again. Thus, consider the state machine in Figure 1 again. We still want to do the same steps presented in Section 3.1, but now more user-friendly.

1. Accept user input that will map to rock, paper, or scissors. This can be key presses or button clicks.
2. Have your script randomly generate an action just like before.
3. Display the result of the game (and what you selected). This can be a text image, some kind of vague arbitrary shape, or a sprite you find off Google. If you want you can tabulate your game results.
4. Ask for the next user input.

4.2 Create a Multiplayer Game with a GUI

Once again partner up with your teammate. Choose whichever single-player PyGame interface looks best and again try to integrate MQTT in so that you can play rock-paper-scissors across the Internet via MQTT, displayed on PyGame.

Again, As specifications:

1. Have at least two computers both show a GUI asking for user input that will map to rock, paper, or scissors.
2. Send the input to another script. Also open communications to receive messages. If distributed, the other message will be the other player's input. If centralized, it will be the result.
3. Depending on the message, determine the result and display the result (and the actions chosen by each user) to the player.
4. Tabulate the result.
5. Ask for the next user input.

5 Other Resources

While we recommend PyGame for this lab as we are doing games, there are many other resources for you to try out. Here, we list a couple other GUI libraries (game or not), with some pros and cons based on what we've seen in previous years. For the non-game libraries, you can take a look in more details here (more listed that people haven't used as well).

1. Basic: Tkinter. This is for the most basic of GUIs. If you just want sprites, buttons, and windows, Tkinter may be enough for you. However, Tkinter provides a noticeably old-fashioned look.
2. Intermediate: PyQT. This has quite a bit more functionality than Tkinter for only a bit more complexity. We've had groups in the past use this to great success.
3. Intermediate: Kivy. This is more designed for mobile development, but the customization is quite nice.
4. Text and Image-based with Networking included: Discord bot. While you might not be able to complete all of the project requirements via Discord, you could create a fun mini-version on Discord rather than through command-line as a trial version.

5. For those who are very interested in graphics, you can also try out Unity. Unity supports MQTT, so everything that we have done so far in this lab can still apply. However, it does run in C# and so someone will have to learn another programming language. Luckily with the MQTT combination, it only needs to listen for messages and then interact with the game, rather than necessarily needing to code the entire thing in a different programming language.

6 Task

1. Please submit a screenshot of you playing your command-line game and your PyGame game with another person.
2. Try reading a bit on one other GUI library. Sometimes you don't need a full game loop interface to implement the games that we are working on, so you can try an easier (or more flexible) interface.
3. Describe in a short blurb how you think you want to do graphics for your project.
4. As always, please remember to submit all new code into your public ECE180D-Warmup repository.