Online Arcade Game

Praveen Vandeyar

pvandeyar@scu.edu

Abstract

I am very interested in video games and I wanted to use the knowledge I gained from COEN 241 to implement a multiplayer game. I have coded simple single player games that ran locally in the past, but for this project I wanted to create a multiplayer that is hosted in the cloud. I was able to learn about network programming in Python and high-level Docker concepts when creating this arcade-like game.

Introduction

Online gaming is a huge industry with a linear growth. That means it will undeniably continue to grow. With an already large player base that will be increasing, online gaming companies find ways to easily deploy and manage game servers. Traditionally, game servers were run on inhouse servers. This was expensive, difficult to maintain, and had high latency for gamers that did not reside near one of these servers. Cloud computing can be leveraged to overcome these issues.

Background and Related work

First I had to get the actual game client and game server for an online game. I wanted to gain more experience with network programming, so I decided to write the code for the game myself. My initial inspiration for the project was to learn Kubernetes, but as I kept running into issues with the Kubernetes set up, I decided to keep it simple

with just Docker. Once the game server was written I made a Docker image out of it. I thought this was a useful improvement in the deployment of game servers from the tutorials I found as it was easy to deploy, it was safely contained, and it was easily scalable. I hosted the game and the database on an AWS EC2 instance.

Approach

I decided to go with Python as the open source module Pygame simplifies making games. From my research, I found that onlines games generally have a Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) connection. UDP is used to send and receive player movements as it is quick. However, since UDP is connectionless, TCP connection is used to keep track of important data and check on the client's connection. I decided to only use TCP as my game was not exchanging large amounts of data and the speed was not too much of a concern for now. The game client and server code was written by me. I followed a tutorial to create the AWS instance with a Vagrant file (https://www.tothenew.com/blog/using-vagrant-to-deploy-aws-ec2-instances/) and followed the MySQL container set up from the Docker website (https://hub.docker.com/_/mysql).

To make the two containers speak with each other, I exposed the application ports on the container. Then, I forwarded the container ports to the EC2 instance ports. This way, I could easily access these containers from outside the EC2 instance while allowing the containers to communicate with each other. Figure 1 shows how the communication takes place between the applications

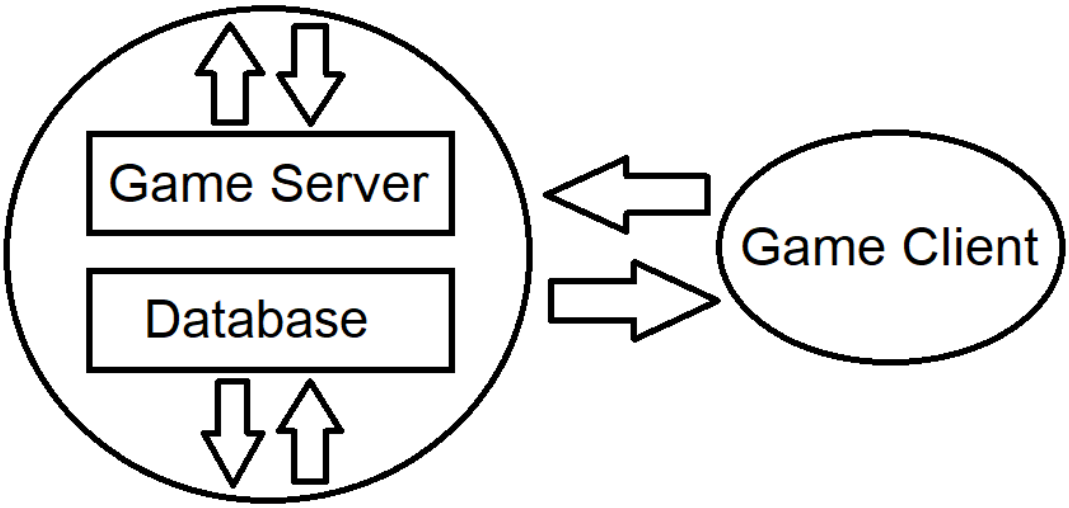

Figure 1

Outcome

The outcome of this project is a two player game. The blue blocks are the players, and they cannot cross the white divider. The yellow blocks are falling obstacles that the players have to dodge. In the middle, the highest score is displayed, which is obtained from the database. All of this is possible with multithreading. The players get their own threads to keep track of movement and to send coordinates. There is a thread for the obstacles and another to occasionally update the high score. Figure 2 shows the actual game.

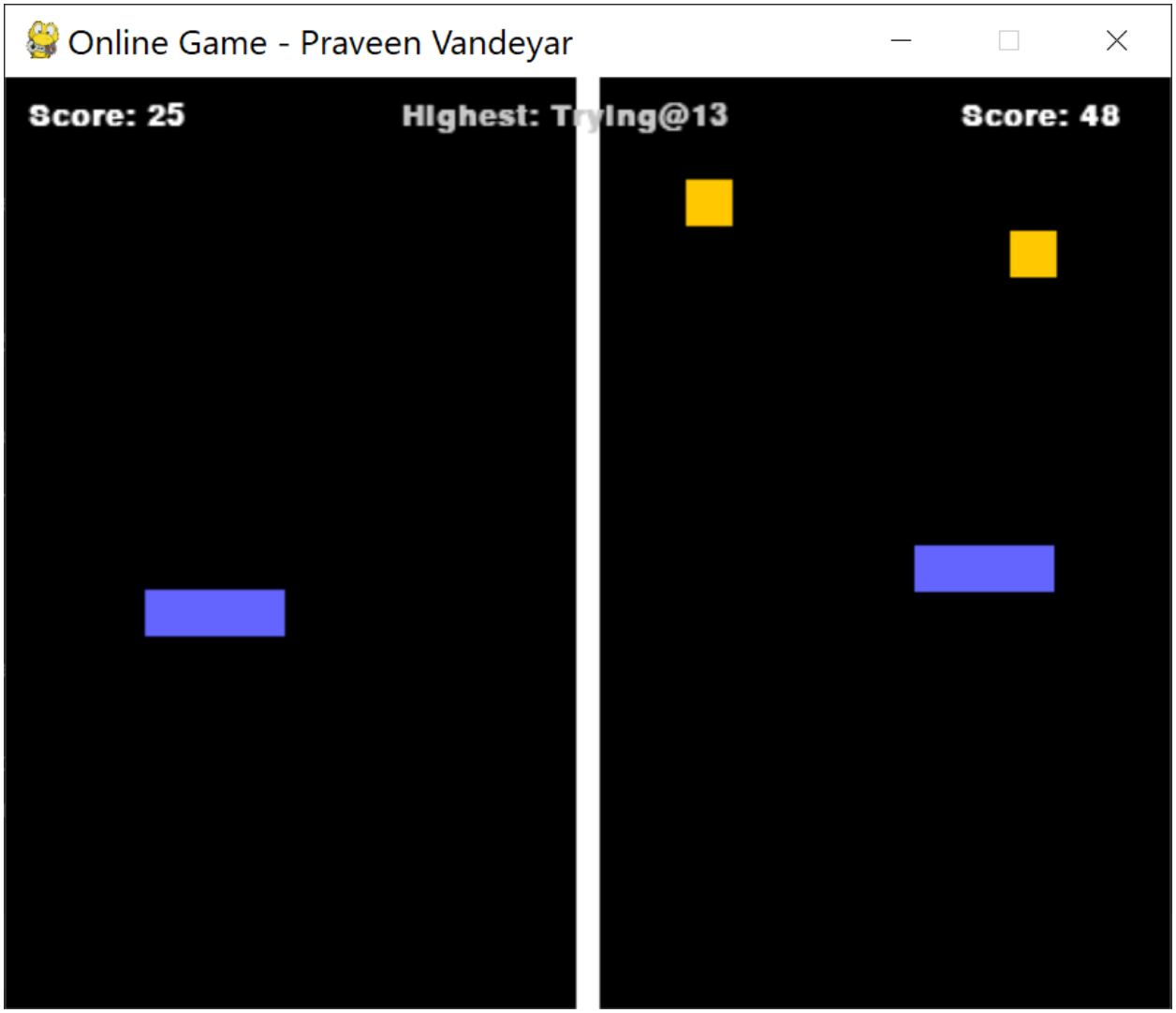

Figure 2

It takes 13ms to receive a response from the game hosted on my EC2 instance at ec2-54-215-117-138.us-west-1.compute.amazonaws.com on port 5555. This is seen in Figure 3.

```
ComputerName            : ec2-54-215-117-138.us-west-1.compute.amazonaws.com
RemoteAddress           : 54.215.117.138
RemotePort              : 5555
InterfaceAlias          : Wi-Fi
SourceAddress           : 192.168.0.20
PingSucceeded           : True
PingReplyDetails (RTT) : 13 ms
```

Figure 3

The size of the containers can be seen in Figure 4. The storage that the MySQL container uses is a persistent volume at 1 Gb so that the data is not lost if the container stops. The game server is identified by the container id ae1b98f77fb3.

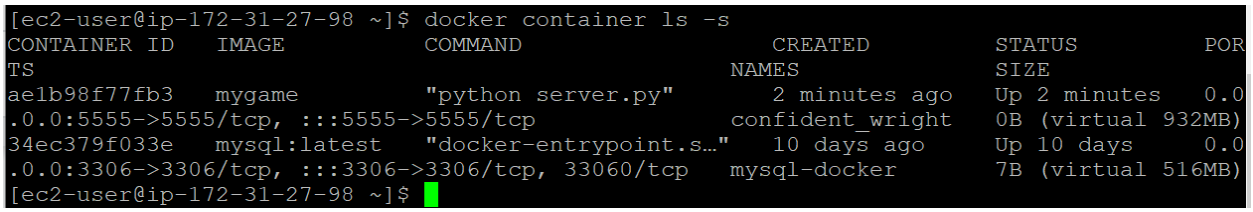

Figure 4

The CPU and memory usage can be seen from Figure 5.

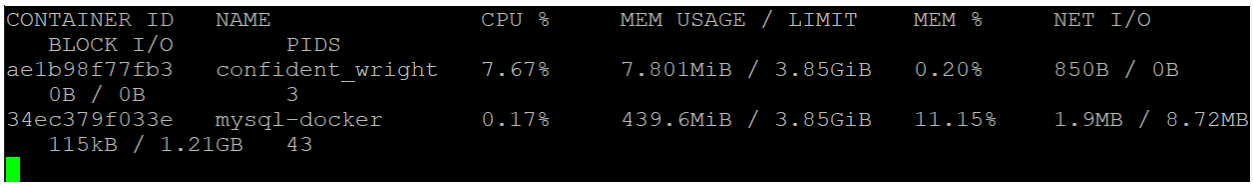

Figure 5

The EC2 instance itself has 2 CPU cores, 2 Gb of memory, and an 8 Gb storage. The entire application can be configured to run in different ways by adding more game server containers and forwarding them to different ports. This will allow for more concurrent players. Database storage can be increased by configuring the persistent volume that is being used for MySQL, and an EC2 instance with more resources can be created if needed to host more game servers with bigger database storage. With the required files downloaded, the commands to start up the containers are as follows:

Database	docker pull mysql docker run --detach --name=mysql-docker -p 3306:3306 --env="MYSQL_ROOT_PASSWORD=password" mysql:latest
Game Server	docker build --tag mygame mygame

	docker run -p 5555:5555 mygame python server.py

Analysis and Future Work

The game gets very “choppy” when two players are connected. The game performance might benefit from a UDP thread for player movements instead of TCP. While memory usage is minimal, CPU usage tends to hit 100% when there are players connected, so an EC2 instance with more CPU cores would help, though for such a simple game the CPU usage is too high. This is most likely due to the threads used. The multithreading is poorly implemented, and I need to do more research on how to properly implement threads.

Conclusion

I learned a lot about network programming and Docker usage. It was a difficult but rewarding experience to implement this simple game. With a better understanding of network programming and threading, I would be able to create a much smoother online game. I chose not to use Kubernetes as it was difficult to implement containers with Minikube and AWS EKS seemed too much like a shortcut, but in the future I hope to use Kubernetes to create an easily scalable multiplayer game..

References

I. https://www.techwithtim.net/tutorials/game-development-with-python/
II. https://www.pubnub.com/blog/why-you-should-run-your-game-servers-separate-from-your-chat/
III. https://realpython.com/python-sockets/
IV. https://hub.docker.com/_/mysql

V. https://www.tothenew.com/blog/using-vagrant-to-deploy-aws-ec2-instances/
VI. https://www.docker.com/blog/containerized-python-development-part-1/
VII. https://github.com/mitchellh/vagrant-aws