Online Multiplayer Video Game Framework

By Jeff Parla and Ben O'Neill

OUTLINE

01 **INTRODUCTION**
Our motivation and problem

02 **OUR SOLUTION**
How we solved our problem

03 **RESULTS**
Did it work how we expected?

04 **CONCLUSIONS**
Lessons we learned and what comes next

INTRODUCTION

Our Goal

Create an open-source framework for online multiplayer games

- Fast and simple
- Promotes good practices
- Abstracts networking concepts for the developer
- Available for anyone to edit and use

THE PROBLEM

- Online multiplayer games are not easy to develop without the right networking knowledge
- Some naive approaches may result in a slow game
- How can we present an easy to use networking solution for video games?

Solution

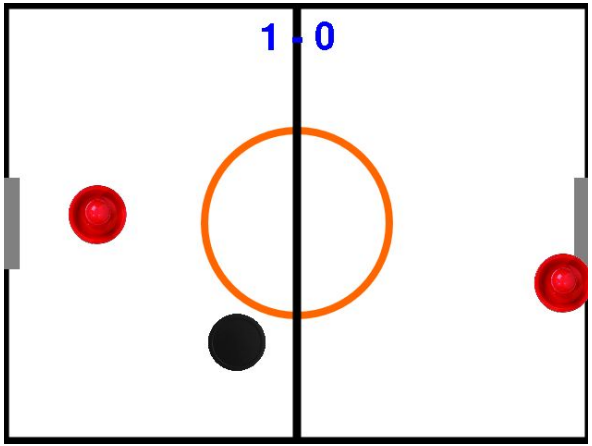
- Using the Client/Server model
- The game client sends inputs to the server
- The server processes these inputs and runs the game logic
- The server sends information about the game state to the client, which draws the representation of that information
- We can create a framework that handles the networking aspect of this model, leaving the developer to write game logic

WHY?

- The developer should spend the most time writing the game, without worrying how to write a networking solution
- We want to offer a solution that works well and is beginner-friendly
- We chose PyGame as it is a simple game engine that uses Python. Python has low level socket programming, so we hope our framework's code will help beginners understand how socket programming works

Summary of Results

- We've created an example game, Air Hockey, using our framework to show how traditional game code (update functions, drawing functions) can be translated to an online multiplayer game
- It works!

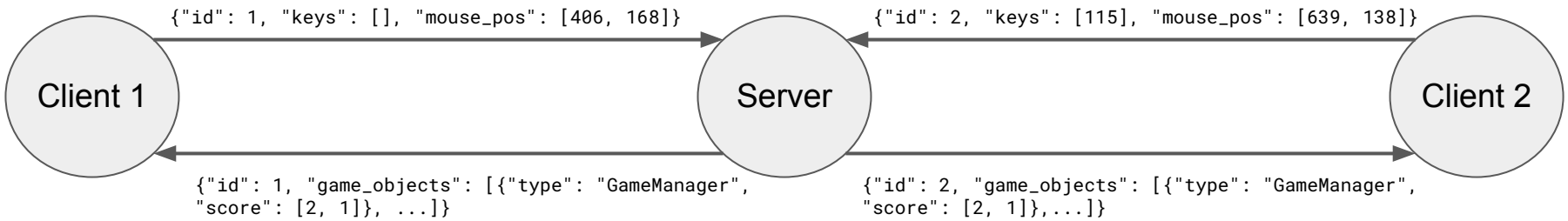


Our Solution

Client/Server Model

- The client only sends inputs and receives updated positions and variables (like score)
- It will then display these updated positions, such as a draw function in a traditional game
- The server will take all inputs and “play” the game as if it were a local multiplayer game
- The server then sends the updated state back to the clients
- This prevents situations where one client may have a fast connection and another client has a slow connection and the slow connection slows down the game for everyone

Basic Network Layout



Initial Connection

Client-side:

- Send the client connect message to the server
- Create GameObjects from data in the server response packet

Server-side:

- Store client ID, IP address, and port
- Create necessary GameObjects for the new user
- Send all game state information to the client

Client and Server Classes

GameClient

- Draws to screen
- Gets keyboard and mouse input
- Gets game state data from the server

GameServer

- Runs all game logic code
- Processes packets from clients
- Handles new connections

Client and Server Classes

```
# convert dict to GameObject (used in GameClient)
def serialize_game_objects(self, data):
    self.game_objects = []
    for o in data['game_objects']:
        if o['type'] == "Player":
            self.game_objects.append(Player(o))
        elif o['type'] == "Puck":
            self.game_objects.append(Puck(o))
        elif o['type'] == 'GameManager':
            self.game_objects.append(GameManager(o))

# convert GameObject to dict (used in GameServer)
def deserialize_game_objects(self):
    if self.game_objects == None:
        return None
    result = []
    for o in self.game_objects:
        result.append(o.get_dict())
    return result

# in each GameObject
def get_dict(self):
    return {'type': 'Puck', 'x': self.x, 'y': self.y}

```

GAMEOBJECT

- The base class for all objects the game logic uses
- In our Air Hockey demo, the Player and Puck classes are DrawableObjects. This is a subclass of GameObject with a sprite and (x,y) coordinates.
- Built-in methods for converting to and from dictionaries, as well as updating on the client and server side.

Writing a game

- Write GameObject subclasses for objects in the game:
  - Players
  - Items
  - Enemies
- Implement server_update() and client_update() in GameObjects
  - server_update() refers to game logic, like checking collision between a bullet and a player, then decreasing health
  - client_update() sends the game state from the server, so it would show the user that their health decreased
- Implement add_client() method in GameServer subclass
- Implement draw() and serialize_game_objects() methods in GameClient subclass

OTHER NOTABLE FEATURES

- Command line arguments to change host and IP on client and server side
- Debug mode which prints extra data for developers

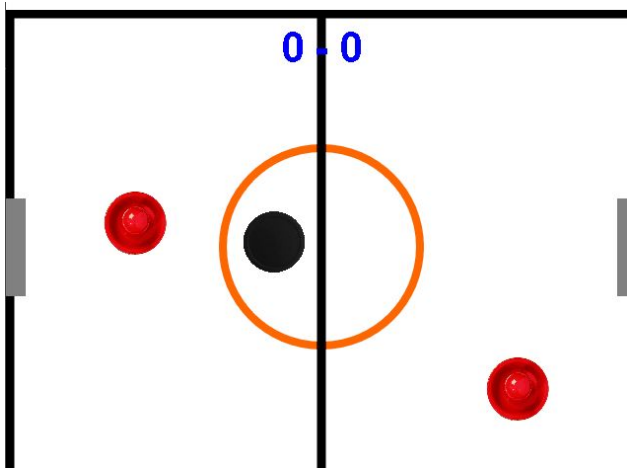
Results

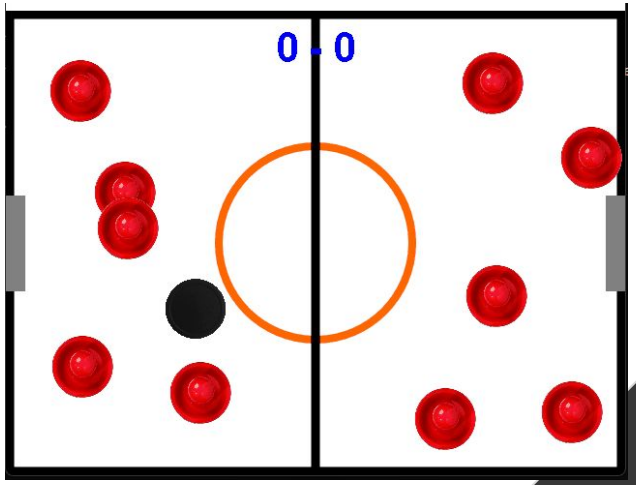
Air Hockey Demo

We've created a small game to demonstrate our framework.
It features client and server side updating of objects, along with objects that are not drawn but keep track of score.
Unlimited amounts of players are supported as a way of stress testing the network code

STRESS TEST

- Realistically, this engine is meant for 2-4 player multiplayer games, but it supports unlimited amounts of players without much slowdown





NOTES

- While we tested with multiple players on one machine, and two players from different machines, we have not had the chance to test across networks, so we cannot definitively say how low the latency is
- As long as the server is not too far from each client, and every connected client has a good network connection, in theory, latency is low.

Conclusions

Conclusions

- We have succeeded in our attempt to make a simple networking package
- It gives developers a good starting point for creating a multiplayer game
- The client_update() and server_update() functions provide a good abstraction for the client/server model

Lessons Learned

- How to design an efficient system for UDP packet transfer
- More advanced Python and Pygame programming
- Designing an abstract framework for game development

WHAT'S NEXT?

- Implementing client-side prediction to give the user smoother visual feedback
- Optimizing packet size for better speed with many objects
- Separation of client only objects from server objects

QUESTIONS?