Online Tetris Game

Group 5

Dujian Ding, Rouyi Wang, Jiaqi Yang, Zhikui Xu, Wenqian Chen

Introduction

We are committed to implement an online 2-player Tetris game. In our implementation, you will enter into the game lobby once signed in. And you can see your personal information on the left top, while all the spare players' usernames are shown on the left bottom. It is free to chat in the lobby to find your opponent or just for fun. By clicking the "+" button, you can create a room and ready for your rival coming. The game logic is the same as the classic Tetris, and the one who survives wins. When the game is over, you can start a new game or leave the room instantly. Apparently, you can always stay there and chat with your new friend in the private room channel.

Introduction Queue

The problem:

In the game, there are multiple source of operation instructions, move from the player and drop from the timer (, and perhaps disturbance from other players). And these instructions should be handled properly.

Our solution:

We unified the multiple sources with a message deque. By sending instructions, players put their operation at the bottom and timer at the top. And there is another worker thread popping the queue and perform the instructions. In this way, massive player messages will not block other instructions from executing, and unifying the sources makes it easier to add more utilities in the future.

Fowarding Queue

The problem:

In our way to deal with multiple instructions, we introduced worker children threads. They will produce the results of instructions, but the socket is owned by the parent thread.

Our solution:

To communicate between threads and not to block each other, a message queue service is used to coordinate operations such as broadcasting. The socket extension we use supports Redis message queue for forwarding broadcasts, and thus we used it to simplify our work.

Reference:

Flask-socketIO document: https://flask-socketio.readthedocs.io/en/latest/

Web Socket

the problem:

In the game each player needs to be constantly updated of the newest game state by the server. The problem is in HTTP, server cannot send the user anything the user doesn't request for. How can the server push data to the user and notify him?

Wrong ways to do this:

1. Let the user request the server constantly. This approach would obviously create a huge, unnecessary burden on the server, and the response time wouldn't be satisfactory.
2. Set up a custom TCP connection. This approach is not very viable in a browser.
3. Use Comet tricks, such as a hidden iframe. This approach is obviously slow and ugly.

Our way to deal with the problem is to leverage the new communication protocol, WebSocket, defined in RFC 6455. WebSocket is a standardized TCP protocol different from HTTP that meets every aspect of our need. It provides two-way communication and can invoke callback functions whenever a new piece of data arrives. The connection is established by an HTTP handshake, and uses the same port as HTTP does. Almost all modern browsers support WebSocket.

We leveraged a library, python-socketio, as an interface to leverage the WebSocket protocol. In the frontend the library socketio.js is used. python-socketio not only supports most functionalities of WebSocket, but also automatically degrades to Comet if the user browser does not support WebSocket.

References:

RFC 6455 https://tools.ietf.org/html/rfc6455

Game Rendering

There are several approaches to render a game on web platform, but the most popular and appropriate methods are these three: Unity3D, Flash and HTML5+JS.

Unity3D, indeed, is powerful enough for rendering a Tetris on web. However, it is too heavy, considering our task is just render a 2D game, especially all the game logics are on server side. Besides, Flash has the same problem, and Adobe already claimed that Flash will be abondoned since 2020. Moreover, the new protocol for rendering graphics on web browsers in HTML5, WebGL, is both effective and simple. Therefore, we chose HTML5+JS to implement our game.

Given that, the smart way to render the game is that using a game engine, or a renderer engine, considering there is no game logic on client side. We have looked up several HTML5 game or renderer engines, like three.js, cocos2D, PlayCanvas, but they both more or less have their problems. Finally, we decided to use phaser.io. One reason is that Phaser.io is a light game engine, and it supports detailed API documents and tons of examples, which makes it quite easy to learn. Besides, it is based on pixi.js, a powerful HTML5 2D rendering engine, so it provides high performance on 2D graphics. That meets our requirements prefectly.

UI