Code Specification

May 17, 2019

Contents

1 Deploy and Run the System **2**

1.1 Install Dependencies 2

1.1.1 Install node.js, npm, and yarn 2

1.1.2 Install node packages on server side 2

1.1.3 Install node packages on client side 2

1.2 Run the System Locally 2

1.2.1 Start Server 2

1.2.2 Start Client Application 2

1.3 Push Updates to Heroku Server 3

2 Code File Architecture **3**

3 Front-end Code **4**

4 Back-end Code **6**

1 Deploy and Run the System

You may find all code for our project in our Github repository and git clone it to your local machine. The following is the instruction for installing dependencies, running the system locally and pushing updates to heroku server.

1.1 Install Dependencies

1.1.1 Install node.js, npm, and yarn

1. Install node.js@11.6.0 and npm@6.5.0 from https://nodejs.org/en/.
2. Install yarn@1.13.0 from https://yarnpkg.com/en/docs/install#windows-stable.

1.1.2 Install node packages on server side

1. In a bash terminal, change directory to the outmost folder.
2. Type command **npm install** which will install required node packages.

1.1.3 Install node packages on client side

1. In a bash terminal, change directory to the `client` folder.
2. Type command **yarn install** which will install required node packages.

1.2 Run the System Locally

1.2.1 Start Server

1. In a bash terminal, change directory to the out most folder.
2. Type command **npm start** that will run the server on http://localhost:5000.

1.2.2 Start Client Application

1. In a bash terminal, change directory to the `client` folder.
2. Type command **yarn start** that will run the server on http://localhost:3000. You may copy paste this address to a browser to test our application.

1.3 Push Updates to Heroku Server

1. Open a bash terminal in the project directory.
2. Type command **heroku login** to log in provided Heroku cloud account.
3. Commit all the changes you made.
4. Type command **git push heroku master** to push updates to heroku server.

2 Code File Architecture

We developed a Node.js Express Application as the back-end of our system and built the front-end with the React.js framework. The code of the back-end is in out most folder (mostly in the `bin` folder), and that of the front-end is in the folder named `client`.

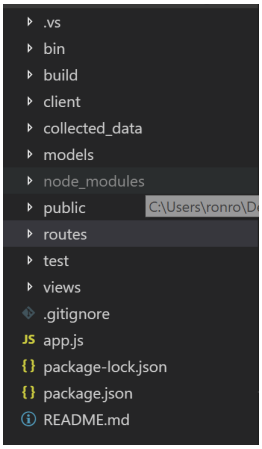

Figure 1: Project Main Folders

As shown in Figure 2, most of back-end code, including game logic, game data gathering, and player matching system, is in the folder named `bin`.

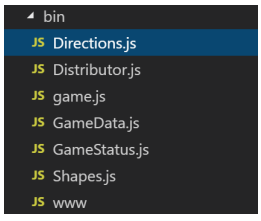

Figure 2: Bin Folder

As shown in Figure 3, most of front-end code, including React.js component (`client/src/components`), and css style sheet (`client/src/styles`), is in the folder named `client/src`.

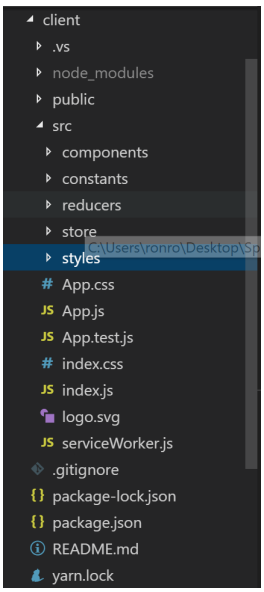

Figure 3: Client Source Folder

3 Front-end Code

Our front-end javascript code mainly locates inside the `client/src/component` directory of the project folder. It was mainly built with React framework. Corresponding style sheets locate in the `client/src/styles` directory.

- `CanvasComponent.js` The file displays the game data including player's individual score, team score and team's performance percentile. It is called by the GameOverPopup-Components.

- `Cell.jsx` This is the file to render each cell of the Tetris game grid. It implements the function of changing cell color according to its status. The stylesheet is `Cell.css`.

- `GameGrid.js` This file renders the game grid (the Tetris game field) and game panel (a table shows game status and player status), and the next block. The file implements functions that updates scores, checks game status, and updates the current control player. This file also implements a arrow keys handler function, which will send the direction to the back-end. The stylesheet is `GameGrid.css`.

- `GameOverPopupComponents.js` The game over pop up component renders game over interface. A window pops up when the game is over. The interface shows the result of team score, individual score, and percentile of the score in history. There are also two

functional buttons in the interface. One is playing again, users can click the ”play again” button, then there are two options for users to choose. If ”yes” there will be options to play again with same user or not, if ”no” it will redirect to survey page. The stylesheet is `Popup.css`.

- `GamePanel.js` It calls the GameGrid component and passes JSX attributes to it.
- `HeaderComponent.js` This file renders the header component of the game.
- `HomeComponent.js` This file renders a Tetris game introduction YouTube video and short introduction of this game. The page also implements a button to redirect to lobby page. The stylesheet is `Home.css`.
- `LobbyComponent.js` This file renders the lobby page of the system. It implements Socket.IO APIs to interact with the back-end, including whether the player is paired with a partner, whether the player is renamed, whether the player is ready to play the game, and whether the player is disconnected or has left the game. This file contains a second component called Instruct, which renders the game instructions and is called by the lobby component. The stylesheet is `Lobby.css`.
- `MainComponent.js` This file specifies routes with its corresponding component. The main component also initializes each page.
- `OfflinePopupComponent.js` This file renders a pop-up when the partner of the player is offline. The stylesheet is `Popup.css`.
- `Row.jsx` This file renders the component of each row of the Tetris game grid. This is done by rendering a row of the Cell component.
- `Shapes.js` This file defines all possible shapes of a single Tetris block.
- `Survey.js` This file renders component contains survey questions for users to fill and submit.
- `Tetris.js` This file renders game instructions in the game page, and functions to check if game is over and if one of the two player is offline. This file implements listeners to get back-end data and trigger pop-ups. The stylesheet is `Tetris.css`.

4 Back-end Code

All back-end code is under directory `bin`. This section would explain further what each code file does.

- `www` This file serves like the ”main method” of the back-end. It would set up the server side and listen on events. It also manages all games and players, and calls corresponding methods on them when some events happen.
- `game.js` This file contains the `Game` class, which contains all the information of a game as well as all methods. To be more specific, it keeps the socket information, the game field, current/next player, and current/next block. When the players start a new game, a new object will be created. It uses an infinite loop to simulate the block falling and will stop the loop when the game is over.
- `GameData.js` This file contains the `GameData` class. It has all the information from a game and is mainly for data collection purposes. Each `GameData` object links to a `Game` object.
- `Direction.js` This file contains a constant array which represents the four directions of the arrow keys.
- `Distributor.js` This file contains the method deciding the distribution. We separated this method to a single file for greater flexibility.
- `GameStatus.js` This file contains a constant array which represents all possible statuses in a game.
- `Shape.js` This file contains the `Shape` class, which represents all possible shapes in a Tetris game. It also contains a constant array that lists all shapes in 2D arrays.