Hurry up and Save the World - Code documentation.

Here you'll find simple explanations of different responsibilities of different files in approximate order of appearance in the execution of the code.

util.js
Defines multiple functions used all around the code, most of the functions are well documented in file, and don't require special mention here.

game.js
Responsible for the initialization of the main game skeleton. Adds the different states to Phaser.Game etc.

effectManager.js
Responsible for the creation of spawn effects, death effects, score text e.g. "-100" after player death, powerup emitters,

boot.js
Responsible for initializing connectionsManager. Also prompts the player if the main game will be muted or not, by showing a mute button. (Effects of muting the game: All effect sounds which were meant to be played on the main screen are broadcasted to player devices)

connectionsManager.js
Defines all RPC functions exposed from the screen side.

waiting.js
This is the state in which, as the name suggests, we wait for players. Here a demo of the game is played in the form of a video. Game transitions from this state when at least one player has been connected for ten seconds.

play.js
This is the state, in which the game is actually played, as the name suggests. It defines the roundManager used, and updates the roundManager. It also acts as a pipeline for commands such as newPlayer, which are routed to roundManager.

roundManager.js
Manages all the game logic, with the help of below mentioned managers. Main responsibility is the movement of the room backgrounds, and the loading and positioning of dynamically created room objects. It reads the round*.js record, and creates the round according to it.
Also defines the draw order of the different graphical layers of the game and shows QR code.

player.js

Defines player objects. Each player class (6) has its own bullets, weapons and sprites. Responsible for setting (controller) inputs for player object, player score, and all player logic e.g. shooting, moving, taking damage, dying, healing, spawning. Updates also the angle of the weapon. Also handles collision checks for player and enemy bullets. Provides functions for setting powerups, adding and losing points.

hud.js
Defines general hub object which is added as a child to parent object. Hud contains healthbar and possible name of the parent object.

playerPatternList.js
Contains all the values of player features. These are “baseMovementSpeed” (movement speed), “baseFireRate” (shooting fire rate, the bigger the slower), “bulletSpeed” (speed of bullet), “bulletLifespan” (how long the bullet stays alive), “maxHealth” (maximum health of player)

weapon.js
Defines weapons for players. Responsible for updating the position and flipping the sprite according to the player.

bulletManager.js
Responsible for creating all the bullets for enemies and players. It sets the speed and the direction of the bullet. Provides functions related to managing of bullets, e.g. killing a bullet.

enemyManager.js
Responsible for spawning enemies according to spawn rules defined in it. Also updates enemy movement.

enemy.js
Defines enemy objects and enemy logic. Handles also collision checks between enemy and player bullets.

powerupManager.js
Responsible for managing all the powerups in the game. Provides functions for spawning powerups.

powerup.js
Responsible for creating a new powerup. Provides a function for triggering the powerup.

room.js
Defines a room in the game, e.g. background, colliders (collidable sprites), moving direction (the direction where the camera will go next), moving speed (of the camera).
Provides functions related to managing of background of the room.
scoreBoard.js
Responsible for creating the scoreboard of the game. During the game the scoreboard shows the first five of the round.

roundOver.js
Responsible for showing scores, credits, “Time remaining until next round” text and QR code.

creditsBoard.js
Responsible for creating credits.