Activity 8 : Front-end Web Development II

Group #	18

Participating group members

Student ID	Name
6733015821	Kongphop Jitsongserm
6733014121	Kawin Seangsuwan
6733013521	Kritsada Limsripraphan
6733008421	Kristana Areerob

Objective

- To get more familiar with JavaScript by tackling a more challenging web application development.
- To get familiar with calling web APIs and handling JSON results.

Scenario

In this activity, your group will complete a web application (let's call it "CEDT Room") that simulates a dynamic, interactive multiplayer game environment where players can join rooms and move around.

- In this room, there will be 63 players. The player number runs from 1 - 63.
- Each group will develop their own version of a front-end of this CEDT Room, in which the user of the front-end can control the position of their player according to the group number. E.g.: The front-end developed by Group # 1 will control Player # 1.
- The front-end can be differ in details but the common functionalities are:
  - The UI shows the room whose size is 800x600 pixels.
  - In any front-end, each of the 63 players is shown in the room at the same position. (It might not be that real-time though!)
  - When clicked at a destination in the room from a front-end, the player associated with that front-end will smoothly move toward the destination, and, at the same time, tell the room server (via the CEDT room API) to update its current position as well as the new targeted destination.
- Each player will be shown in the front-end with a simple shape (E.g.: a circle, a square) in a specific color. The description of the look of each player must be according to what the room server provides (via the CEDT room API).

The CEDT room API

The CEDT room API provides 3 endpoints, each is served with a different PHP file on the CEDT room server. The URL will be provided in class. It is a temporary IP address and might change over time. So be ready to update the URL in your code when notified.

Endpoints

There are 3 endpoints which are listed below.

Endpoint / Description	Method	Sample successful response body (JSON) / POST request body
`roomapi/getplayerlook.php` Get the look description of all players	GET	`{ "p1": { "shape":"circle", "color":"red" } "p2": { "shape":"square", "color":"blue" } }`
`roomapi/getplayerxy.php` Get the positions/targets of all players	GET	`{ "p1": { "pos_x": "122.2", "pos_y": "271.5", "target_x": "701.5", "target_y": "206.5" }, "p2": { "pos_x": 145, "pos_y": 155, "target_x": 145, "target_y": 155 } }`
`roomapi/update.php` Update the position/target of the player based on the secret	POST	`Request body: { "pos_x" : 120, "pos_y" : 234, "target_x" : 200, "target_y" : 89 }` `Response: {"success":"Player data saved successfully"}`

Authentication

In order to be authenticated when calling the CEDT room API, each group will be given a "secret" which will be checked by the CEDT room server. Each secret is associated with a player. The secret must be added in the header of the request with the field name being "X-Secret".

The fetch statement could start with this:

```
fetch(`${RoomUtil.roomApiURL}[the endpoint PHP file]`, {
     method: '[GET or POST]',
     headers: {
         'X-Secret': RoomUtil.getSecret(),
     },
  })
```

Rate Limit and Banning

Making requests to the CEDT room API too frequently could crash the server, as a result, interrupt the class activity. Therefore, the rate limit of the requests made with a secret is set at approximately **1 request in 5 seconds**. Going over the rate limit will cause the secret to be **banned for 10 minutes**.

The Starter Pack

The foundation code for a front-end web application can be downloaded from myCourseVille.

Development Roadmap

Part 0

- Setting up the Phaser application.

Part 1

- Getting the look descriptions of all players from the CEDT Room API in the 'create' phase of the scene life cycle.
- Using the look data obtained to create instances of the Player class and keeping references to all Player objects in the RoomUtil.players array.
- Adding a method to the Player class responsible for adding the graphics object corresponding to the look data of its Player instance to the scene.
- Iterating through the RoomUtil.players array in order to set the positions of graphics objects associated with all Player objects in the 'update' phase of the scene life cycle.

Part 2

- Creating an instance of the MyPlayer which extends from the Player class.
- Adding a method to the MyPlayer class responsible for updating the position/target of your group's player to the CEDT room server.
- Setting up the event listener to handle the clicking which will:
    - Set the target of the Player object associated with your group's player so that your group's player moves to the target in the next update phase.
    - Post an update request to the CEDT room server.

Part 3

- Making requests to obtain the positions/targets of all players while keeping the numbers of requests under the rate limit.
- Fine tuning your front-end logic so that it works well.
- (Optional) Adding extra spices to your UX/UI.

Submission Instructions

- When you complete each * Checkpoint *, show the result to one of the TAs.
- Submit the zipped folder that contains your front end in myCourseVille.

Part 0: Setting up

1. Download the starter pack from myCourseVille.
2. Set up a web server on your machine so that the web server can serve index.html from the starter pack.
3. Remove `this.add.text(50, 50, …;` from the create phase of the scene life cycle.
4. * Checkpoint 0.4 * Show the resulting (running localhost) Phaser app to one of the TAs to check your group's progress.

Part 1: Rendering all players in the room

1. Go through all the available code in game.js and try to understand the class definition. If you do not understand something in the class definition, feel free to ask one of the TAs or the instructors.

2. In the create phase of the scene life cycle, we will begin by making sure that our application has enough information about all the players before the game starts.

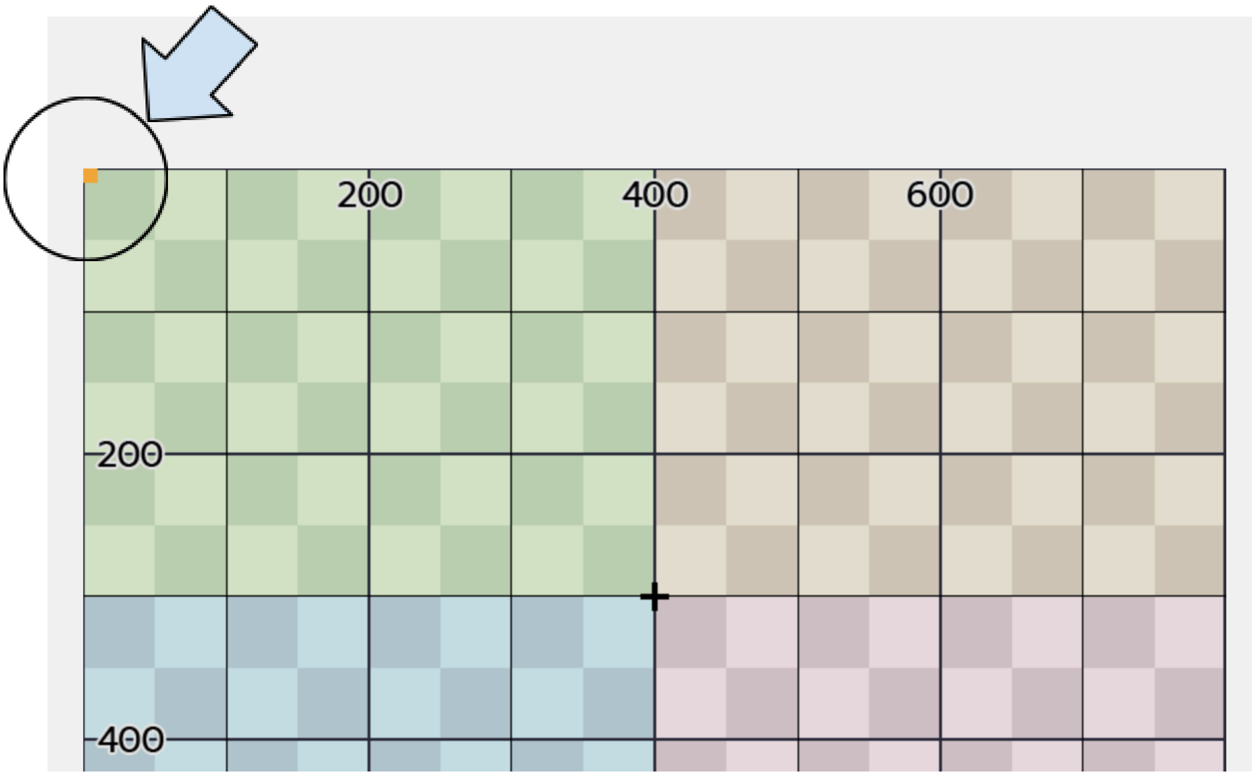
   A statement `RoomUtil.getInitAndStartGames(this,myPlayer);` has been placed there. What this method does is to fetch all player's look descriptions via the `roomapi/getplayerlook.php` endpoint.

Once the look data is obtained,
`RoomUtil.initAllPlayers(scene,data,myPlayer);` will then be called.

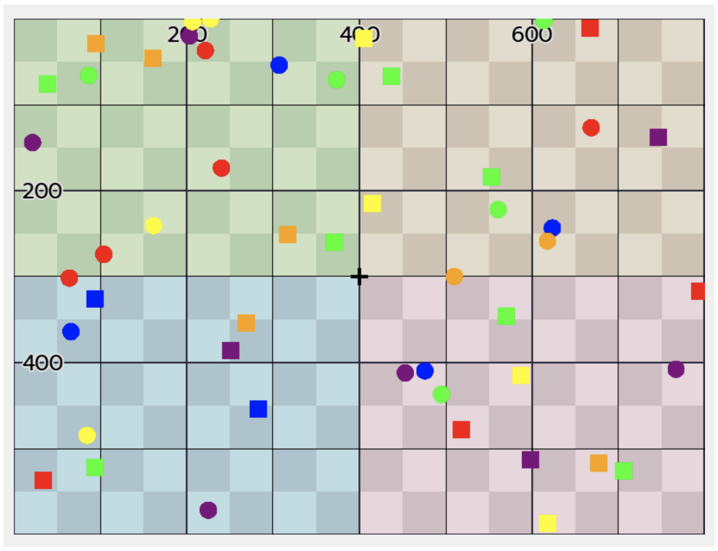
Complete `RoomUtil.initAllPlayers()` as well as other necessary parts so that, after the statement is done, all Player objects are instantiated, their visual attributes are added to the scene, and they have been added to the `RoomUtil.players` array to be used later on in the program.

* Checkpoint 1.2 * From the web browser console, show the content of `RoomUtil.players` to one of the TAs to check your group's progress.
Hint: Method `initAllPlayers` should call method `initLookAndVisual`, please make sure that you have finished both methods before calling the xTA.

3. From the previous step, if you have added the Phaser graphics object to the scene correctly, you will see all the visuals of the players in the room. However, with the given Player class definition, the constructor sets the position as well as the target of the object to (0,0). Therefore, all the visuals will be piled up at the top left corner of the room. It might appear like this:



Modify the constructor of the Player class so that it assigns a random position (within the 800 x 600 dimension) to the Player object. All player in front end will be scattered like this:



* Checkpoint 1.3 * Show the UI of the room that has all the visuals of the players scattering in the room.
Hint: Use `Math.random()` to random number (Try this method to see its' result)

Part 2: Handling your player

1. In this part, we will start making a distinction between other players and the player associated with your group. Notice that we have already created the instance of MyPlayer with:

   ```
   myPlayer = new
   MyPlayer(RoomUtil.getSecret(),RoomUtil.getPlayerNumber());
   ```

   The MyPlayer class is a subclass of the Player class. Therefore, an instance of MyPlayer contains everything that an instance of Player has.

   Remember the `RoomUtil.players` array? It holds all instances of the Player class in your room. In Part 1, we did not distinguish between other players and the player associated with your group. From now on, we would like to fix that.

   Modify your logic in `RoomUtil.initAllPlayers()` so that, for the player whose player number is the one belonging to your group, instead of creating a new Player instance, put myPlayer in the `RoomUtil.players` array.

2. We also would like to make the visual of your player different from the visuals of other players. The visual of your player will still use the shape and color based on the look description obtained from the room API but **your player will have a thick black stroke** while other players do not have.

   To achieve this visual, in the class definition of MyPlayer, we have already overridden `initLookAndVisual(scene, look)` so that when the method is called from an instance of MyPlayer, this overridden version will be used instead of the one provided in the class definition of Player.

Complete `initLookAndVisual(scene, look)` in the class definition of MyPlayer so that your player has a thick black stroke.

3. Next, we will handle the mouse click events. When the user click on a position in the room, this is what should happen:
    - Update the values of targetX and targetY of the base class to the coordinate of the pointer when the click happens.
    - Make a request to update.php so that the CEDT room server records the new target of your player.

   The code to do the first bullet is there already in the `setTarget()` method of the MyPlayer class.

   Complete the `updateMyselfToServer()` method in MyPlayer class so that your front-end calls the fetch() function to post the position/target of your player via the CEDT room API (`roomapi/update.php`).

   In order to also see your player moves in your front-end, this might be a good time to implement something for `Object.values(RoomUtil.players).forEach()` in the update phase and use `setTarget` in function pointer in create phase of the scene life cycle.

   * Checkpoint 2.3 * Show to one of the TAs that when you click to move your player, your player in the TA's front end moves accordingly.
   Hint: Many methods needed to be fixed to complete this step. For example, `updatePlayerInRoom`, and others.
   Be careful: “Content-Type” should be ***"application/x-www-form-urlencoded.”***

Part 3: Syncing all players and Fine tuning

1. The only thing left to do is to obtain all players' positions/targets from the CEDT room server. You might want to put `RoomUtil.fetchAllPlayersXY(this);` in the `if (this.lastFetchTime >= fetchInterval)` condition in the update phase of the scene life cycle and complete the details of `RoomUtil.fetchAllPlayersXY()`.

   Complete your front-end. Do whatever modifications that you think necessary.

   * Checkpoint 3.1 * Show your masterpiece to one of the TAs. (Every group who is able to complete the front-end will obtain the outstanding factor.)

2. If you would like to challenge yourself more, improve the UX/UI of your front-end. E.g. Animated sprites would be awesome.