University of london  CM2020

ASP8 Tower defence documentation

ASP8 Team members:

* Alex Chu
* Dimitri Vlachos
* Freda Xiaoyun Yu
* Jeremy Matthews

Table of Contents

[audioManager 3](#_Toc99317702)

[audioManager.mjs 3](#_Toc99317703)

[Parameters 3](#_Toc99317704)

[Functions 3](#_Toc99317705)

[Map Objects 4](#_Toc99317706)

[enemy.js 4](#_Toc99317707)

[Functions 4](#_Toc99317708)

[Play Objects 4](#_Toc99317709)

[enemy.js 4](#_Toc99317710)

[gameStats.js 4](#_Toc99317711)

[hud.js 5](#_Toc99317712)

[hudLogic.js 6](#_Toc99317713)

[hudLogicTower.js 7](#_Toc99317714)

[map.js 7](#_Toc99317715)

[mapLogic.js 8](#_Toc99317716)

[mapLogicEnemy.js 8](#_Toc99317717)

[mapLogicTower.js 8](#_Toc99317718)

[map.js 8](#_Toc99317719)

[states 8](#_Toc99317720)

[bootstrap.js 8](#_Toc99317721)

[complete.js 9](#_Toc99317722)

[level.js 9](#_Toc99317723)

[menu.js 9](#_Toc99317724)

[over.js 9](#_Toc99317725)

[play.js 9](#_Toc99317726)

[map.js 9](#_Toc99317727)

[treasure.js 9](#_Toc99317728)

[utilities 10](#_Toc99317729)

[checkJSON.js 10](#_Toc99317730)

[Functions 10](#_Toc99317731)

[gameRecords.js 10](#_Toc99317732)

[Parameters 10](#_Toc99317733)

[Functions 10](#_Toc99317734)

[resources.js 10](#_Toc99317735)

# audioManager

## audioManager.mjs

The audio manager is used to manage the audio of the game. The music, SFX and so on.

### Parameters

* \_soundManager Phaser’s default soundManager for the audioManager to manage.
* \_sfxEvents A list of sfx ‘events’ to be called by the playEffect() function.
* \_musicEvents A list of music ‘events’ to be called by the playMusic() function.
* \_musicSounds A list of music ‘sounds’ to be called by the playMusic() function and looped.
* \_volume (default: 1) The audible volume of all sounds.
* \_musicVolume (default: 1) The audible volume of all music.
* \_sfxVolume (default: 1) The audible volume of all sfx.
* \_currentMusic (default: undefined) Music to be playing currently.

### Functions

#### pause()

Pauses all sounds being played by the audioManager.

#### resume()

Resumes all sounds being played by the audioManager.

#### stopAll()

Stops all sounds being played by the audioManager.

#### playMusic(musicEvent)

Plays the musicEvent, adding it to the list \_musicEvents if it is not already in it.

#### stopMusic()

Stops music from playing.

#### playEffect(sfxEvent)

Plays the sfxEvent, adding it to the list \_sfxEvents if it is not already in it.

#### getVolume()

Returns the value of \_volume.

#### setVolume(volume)

Sets the value of \_volume.

#### getMusicVolume()

Returns the value of \_musicVolume.

#### setMusicVolume(musicVolume)

Sets the value of \_musicVolume.

#### getSfxVolume()

Returns the value of \_sfxVolume.

#### setSfxVolume(sfxVolume)

Returns the value of \_sfxVolume.

#### \_getInstance()

Returns the \_instance. Unless no instance exists, in which case it returns null.

# Map Objects

## enemy.js

This class extends the Phaser class Phaser.GameObjects.PathFollower, since enemies need to follow a settled path.

Functions

constructor (config)

Sets configurations for the enemy.

follow(self)

is used to let enemies follow the specified path.

destroyEnemy(self)

is used to remove the enemy when its Hit-Point is 0.

# Play Objects

## enemy.js

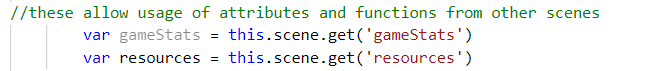
Same as mentioned above.

## gameStats.js

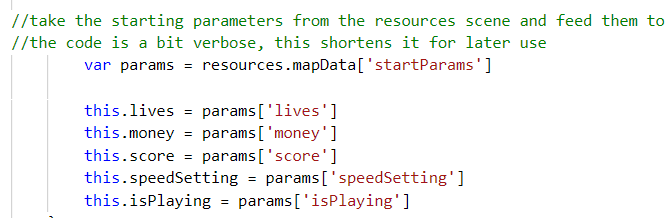
This file is a class which is extended from Phaser.Scene. It handles game statistics.

**Create()** method documentation: <https://newdocs.phaser.io/docs/3.55.2/Phaser.Types.Scenes.SceneCreateCallback>

Within it, firstly reuse data from other scenes:

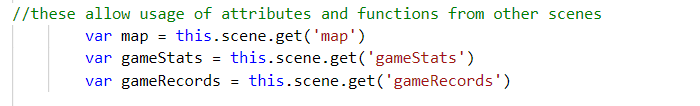


Then import map data from resources scene:

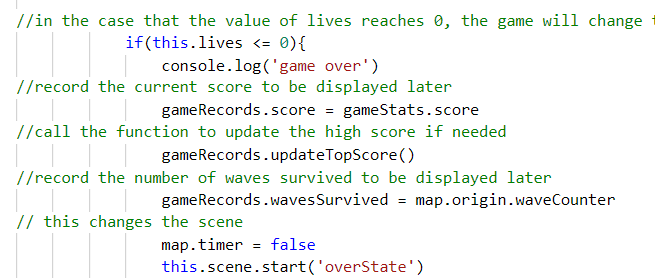


**Update()** method documentatation: <https://newdocs.phaser.io/docs/3.55.2/Phaser.Scene#update>

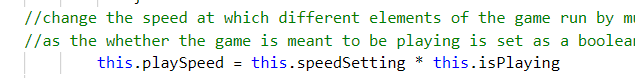
Within it, use attributes and functions from other scenes:



Set the game over state when lives go to 0.



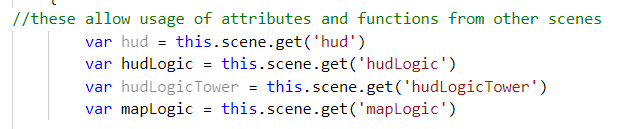
Set speed:



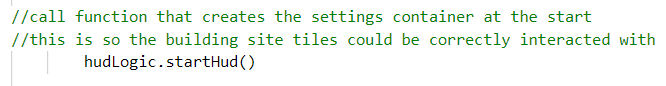
## hud.js

This is also a class extended from Phaser.Scene, which describes functionalities in the User Interface when playing the game where the map displays.

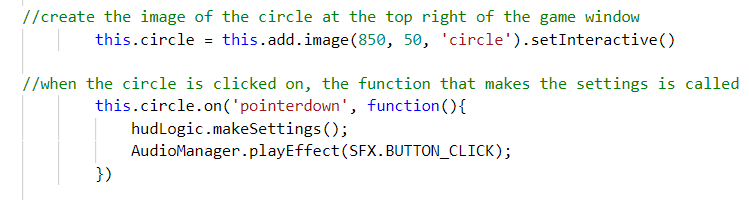
Load data from other scenes:



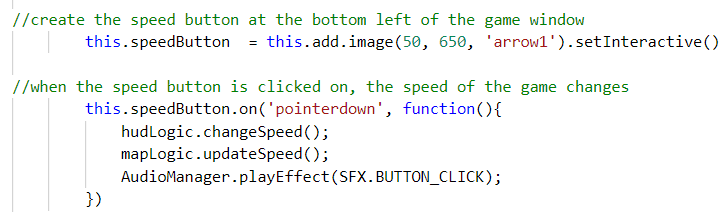
Import the tile logic which is how the map is built and is written in another file later:



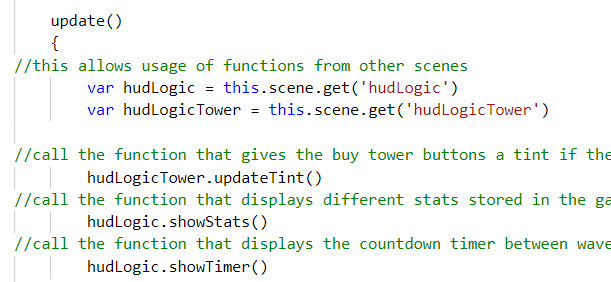
Add the menu button at top right:



Add the speed-changing button at the bottom:



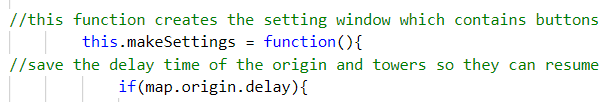
Within **update()** method, after loading data from other files, (1) add buy-tower functionality; (2) update different game statistics; (3) let the timer runs normally:



## hudLogic.js

A class extended from Phaser.Scene, which manipulates the logic of HUD elements.

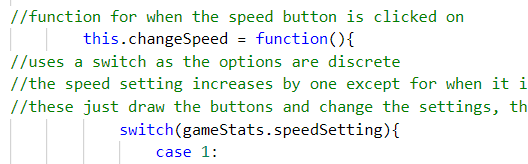
Set menu with 3 buttons, save the current map to be resumed:



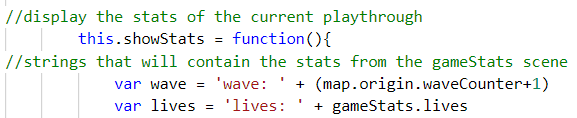


Then draw buttons and set interactions with specific outcomes. This step is difficult since it requires programmers to think about all the elements in each state comprehensively.

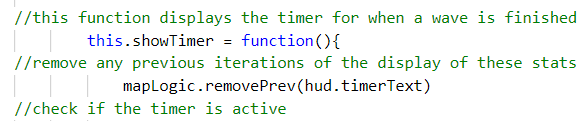
Then draw the speed-changing button and set options:



Then show the game stats:



And show the timer:



## hudLogicTower.js

This is a class extended from Phaser.Scene. Mainly manipulates the towers shown in the HUD, i.e. buy tower, sell tower, tower upgrades, etc.

After reusing data from other files,

#### this.makeTowerMenu = function(tile)

Function that creates the tower menu. Takes in as a parameter the tile that was clicked on which caused the function to be called.

#### this.makeMenuBuy = function(tile)

Function that displays all the towers that could be bought. Takes in the tile the that was clicked on.

#### this.updateTint = function()

Function that would dim the buy button in the tower menu if the cost of the tower exceeds the amount of money at the moment. The button reverts back to normal when there is enough money.

#### this.clickBuy = function(button, tile, tower)

Function that allows the buy buttons in the tower menu to be interacted with interactions involve hovering the mouse over is and clicking it to buy the tower. Takes in as parameters 1)the button in question, 2)the tile the tower menu is for and 3)the tower the button is for.

#### this.makeMenuStats = function(tile)

#### Shows the stats of the tower in the building site that was clicked on. Takes as a parameter the tile that was clicked on.

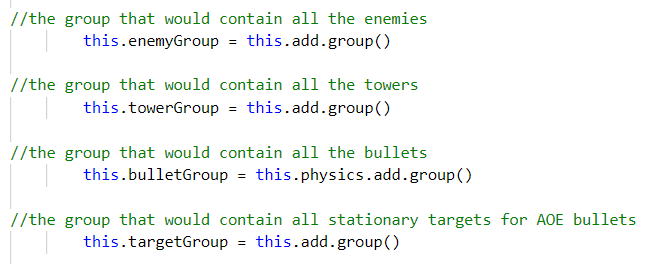
#### this.removeTower = function(tile)

Removes a tower from building site. Parameter: the tile the tower is to be removed from.

## map.js

This is a class extended from Phaser.Scene. Mainly summarises all of the interactions and logics done by other files.

Set all the towers, enemies, and bullets into groups:



Then import tile logic. Set enemies’ path.

Then add the ‘start’ button and set interactions.

In **update()** method:

Update (1) speed changes; (2) remove bullets; (3) determine the attack speed of towers; (4) searches for any enemies that are in a tower's range; (5) checks the state of the origin and creates an enemy if the state is the right one; (6) determine how fast the enemies spawn in a wave; (7) handles the countdown between waves; (8) call the next next wave when the rush button is clicked; (9) when a building site tile is clicked on, it opens up the tower menu in the HUD.

## mapLogic.js

extends from Phaser.Scene.

## mapLogicEnemy.js

## mapLogicTower.js

## map.js

# states

Each file is one class which is extended from Phaser.Scene.

## bootstrap.js

This file loads and initialises all of the Phaser objects for the audio manager in order to facilitate testing as the mocha.js testing suite crashes when it tried to load Phasers objects, meaning we had to remove this section from that code in order to run our tests.

## complete.js

## level.js

## menu.js

## over.js

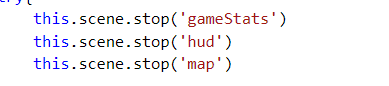
## play.js

## map.js

## treasure.js

This is the ‘treasure box’ page which shows lists of characters.

Firstly, stop interactions from other pages:



Then, add the background picture from the resource.js:



Then, create a ‘start’ button and set interactions.

# utilities

## checkJSON.js

This utility extends Phaser.io’s default Phaser.Scene. It allows for the application to validate JSON files, ensuring their integrity and compatibility with the application.

### Functions

#### checkDict(dict, keys)

This function checks that every relevant value of a given dictionary is the right data type.

It takes, as input:

* dict, the dictionary to check.
* keys, a second dictionary that contains the data types for all of the relevant values for any dictionary used in the game. Only those covered by the key dictionary are checked.

#### checkLevel(dict, data, key)

This function is used specifically for checking the inner arrays of the level dictionaries.

It takes, as input:

* dict, the dictionary to add the data to.
* data, the dictionary containing all of the data to check.
* key, the key of the dictionary to check.

By default, all dictionaries pass, therefore, the data may be added to the larger level dictionary. If any of the tests fail, then it will not be added to the larger level dictionary.

## gameRecords.js

This utility extends Phaser.io’s default Phaser.Scene. It allows for levels to be selected

### Parameters

* levelSelect (default: -1) This is how the levels are selected and the selection is preserved.
* wavesSurvived This records how many waves the player has survived.
* lives This records how many lives the player has remaining.
* score This records the highest score the player has attained during the current game session.
* topScore (default: 0) This records the highest score the player has attained during the current game session.
* instantWin (default: false)

### Functions

#### updateTopScore()

this function updates the top score by comparing it to the current score. If the current score is higher, then the top score is changed to the value of the current score.

## resources.js

This utility pre-loads all of the necessary assets for the game.