REPORT
SPACE LETTERS

Autores: David Ventas Sánchez.

Tutor: Dr. Antonio Castellanos López.

Diseño y desarrollo de videojuegos, Universidad Jaume I, Castellón de la Plana, España.

Julio 2022

UNIVERSITAT JAUME·I

INDEX

Introduction

Our game is based on a ship in the lower part that we can move with the keyboard, it will face ships that will come closer to collide and destroy it, these ships have words and in order to destroy them before they arrive we must write the word. This will be repeated for several waves, we also have different game modes to increase the difficulty and complexity of the game.
To be able to test the game from the server: teseo.act.uji.es/~al408208/SpaceLetters/

Developed sections

We have been developing the game together until we reach a "goal" set for that moment and then we will save the progress to start from that point when we want to advance in the development. We have copied the structure proposed by the game document, which once it reaches gameover or win, it starts over, we move through all the states with buttons.

On the initial screen, there is the title and several buttons, we can access an about state, with a button on the bottom right, where we have information about the game, we can also access as an extra state, settings, where we can change the configuration of the sound and we can also access preplay, where there are three buttons, all of them will take us to the play state, with the difference that each one will have a boolean assigned which will be set to true, once all the rounds are finished we go to the state of win, if instead of that, we die, we access the state of game over, of the two we can return to init by means of a button.

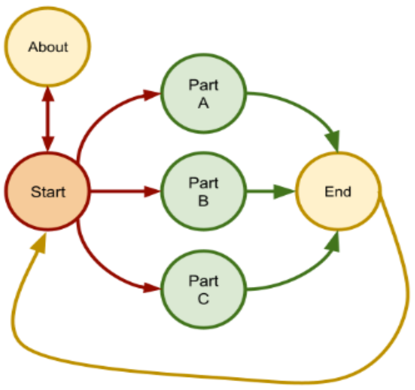
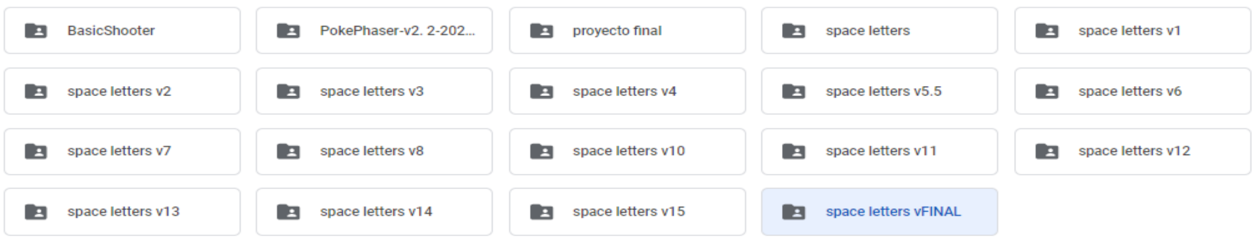


Figure 4: Game state transitions

For the sound we have declared an initial variable through which we will obtain the volume with which we left it and we will put it in the rest of the sounds, thus lowering all the sounds in the same way.

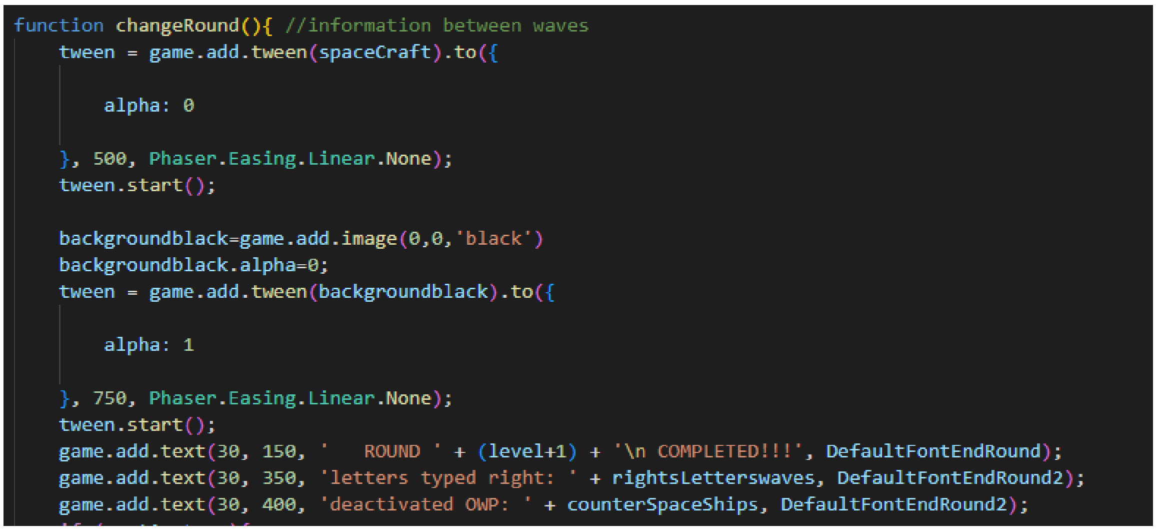


BasicShooter | PokePhaser-v2. 2-202... | proyecto final | space letters | space letters v1
space letters v2 | space letters v3 | space letters v4 | space letters v5.5 | space letters v6
space letters v7 | space letters v8 | space letters v10 | space letters v11 | space letters v12
space letters v13 | space letters v14 | space letters v15 | space letters vFINAL

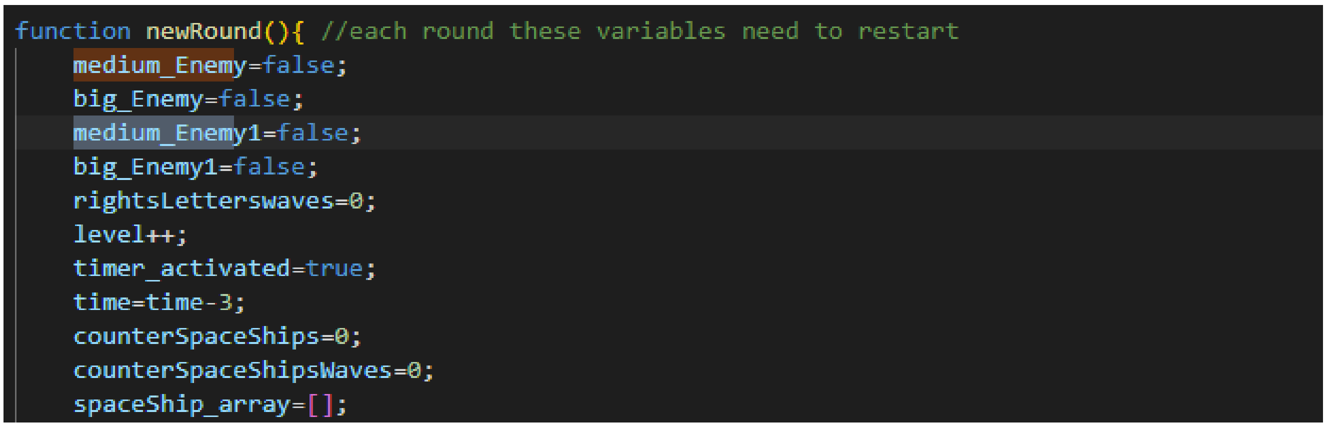
The menus are all very similar, with buttons to change state, implementing sounds, music and a moving background for volume.

Part A:

For part A there will be 5 rounds in which a number of ships defined in your json will be generated.



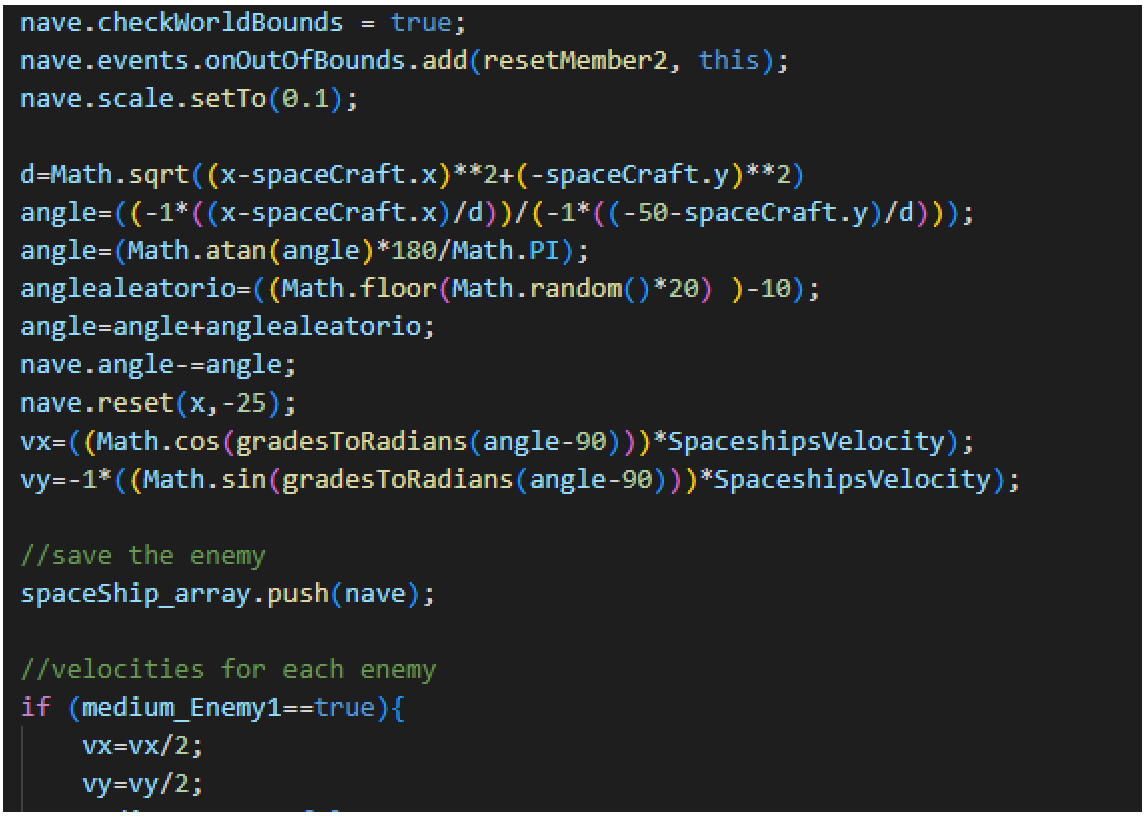
```
function changeRound(){ //information between waves
    tween = game.add.tween(spaceCraft).to({

        alpha: 0

    }, 500, Phaser.Easing.Linear.None);
    tween.start();

    backgroundblack=game.add.image(0,0,'black')
    backgroundblack.alpha=0;
    tween = game.add.tween(backgroundblack).to({

        alpha: 1

    }, 750, Phaser.Easing.Linear.None);
    tween.start();
    game.add.text(30, 150, '    ROUND ' + (level+1) + '\n COMPLETED!!!', DefaultFontEndRound);
    game.add.text(30, 350, 'letters typed right: ' + rightsLetterswaves, DefaultFontEndRound2);
    game.add.text(30, 400, 'deactivated OWP: ' + counterSpaceShips, DefaultFontEndRound2);
```



```
function newRound(){ //each round these variables need to restart
    medium_Enemy=false;
    big_Enemy=false;
    medium_Enemy1=false;
    big_Enemy1=false;
    rightsLetterswaves=0;
    level++;
    timer_activated=true;
    time=time-3;
    counterSpaceShips=0;
    counterSpaceShipsWaves=0;
    spaceShip_array=[];
```

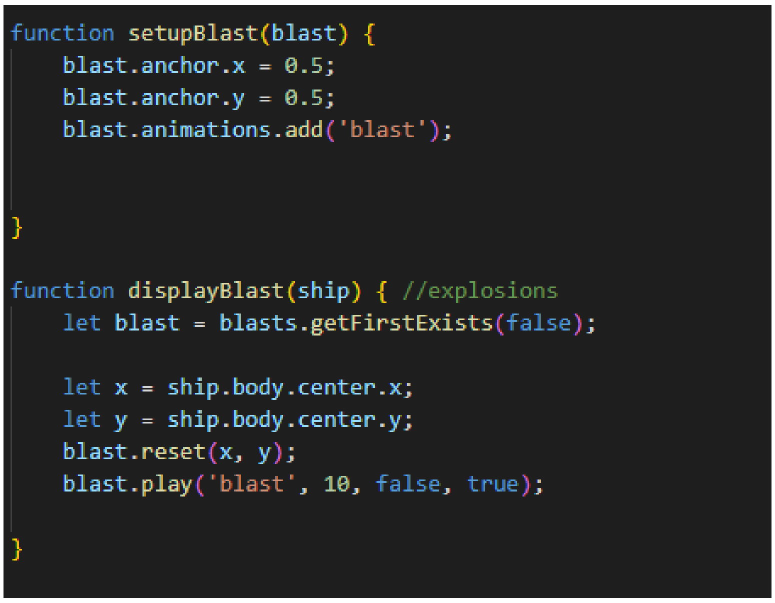
These ships will be generated and each time one is generated, it will be added to a counter and once that counter reaches the number determined by the ships json, the round will end, and so on until the game ends.

Each ship is generated pointing towards the ship, by means of a bit of trigonometry we obtain the angle, to which we will add an angle of deviation, and we also obtain the speed in the x axis and in the y axis.



```
nave.checkWorldBounds = true;
nave.events.onOutOfBounds.add(resetMember2, this);
nave.scale.setTo(0.1);

d=Math.sqrt((x-spaceCraft.x)**2+(-spaceCraft.y)**2)
angle=((-1*((x-spaceCraft.x)/d))/(-1*((-50-spaceCraft.y)/d)));
angle=(Math.atan(angle)*180/Math.PI);
anglealeatorio=((Math.floor(Math.random()*20) )-10);
angle=angle+anglealeatorio;
nave.angle-=angle;
nave.reset(x,-25);
vx=((Math.cos(gradesToRadians(angle-90)))*SpaceshipsVelocity);
vy=-1*((Math.sin(gradesToRadians(angle-90)))*SpaceshipsVelocity);

//save the enemy
spaceShip_array.push(nave);

//velocities for each enemy
if (medium_Enemy1==true){
    vx=vx/2;
    vy=vy/2;
```

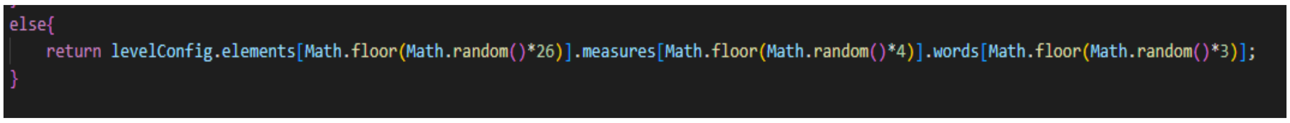
We will also start the explosions so that every time a ship dies, the explosion goes out



```
function setupBlast(blast) {
    blast.anchor.x = 0.5;
    blast.anchor.y = 0.5;
    blast.animations.add('blast');


}

function displayBlast(ship) { //explosions
    let blast = blasts.getFirstExists(false);

    let x = ship.body.center.x;
    let y = ship.body.center.y;
    blast.reset(x, y);
    blast.play('blast', 10, false, true);

}
```

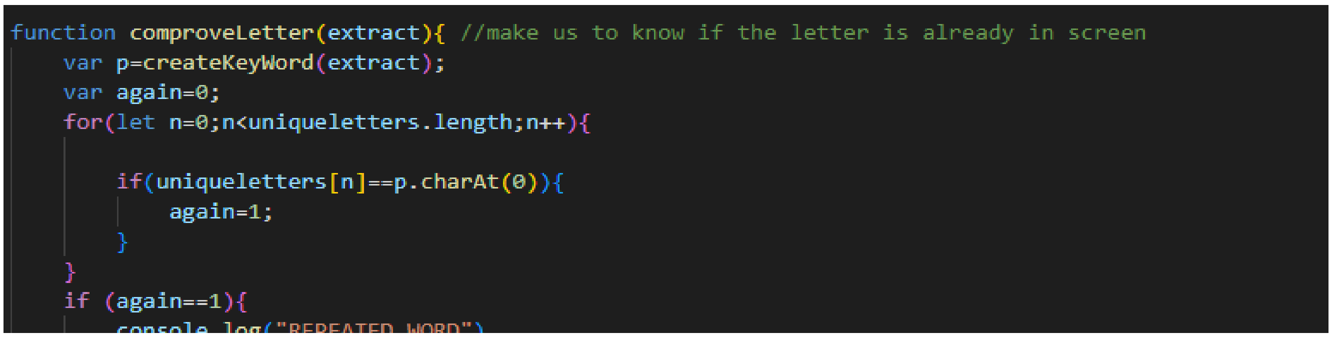
Also every time we hit writing a letter we will shoot towards the word, to simulate that we destroy a letter. We have implemented a tween that goes to the x and y position of the enemy ship.

To control what is related to the text, the first thing was to generate random words from the json:



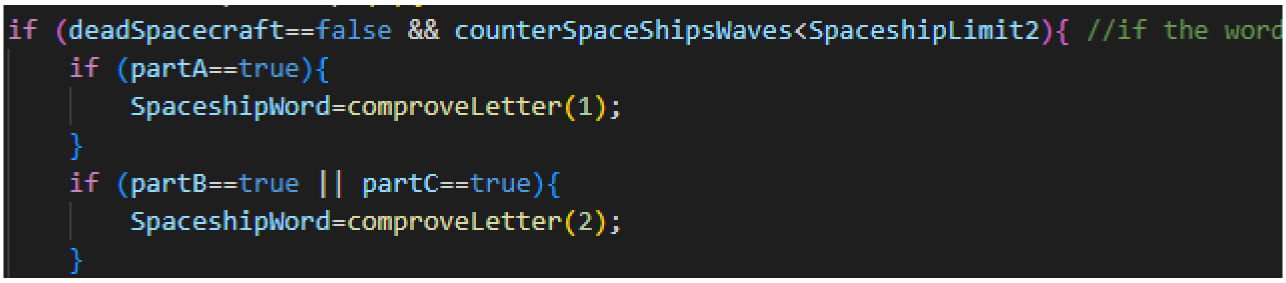
```
else{
    return levelConfig.elements[Math.floor(Math.random()*26)].measures[Math.floor(Math.random()*4)].words[Math.floor(Math.random()*3)];
}
```

The method depending on which game mode it will generate the words that later passed through a control so that there would be no repeated letters:



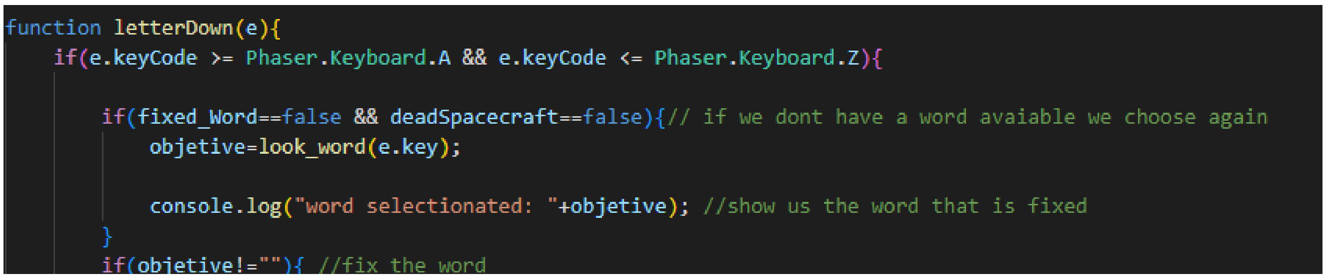
```
function comproveLetter(extract){ //make us to know if the letter is already in screen
    var p=createKeyWord(extract);
    var again=0;
    for(let n=0;n<uniqueletters.length;n++){

        if(uniqueletters[n]==p.charAt(0)){
            again=1;
        }
    }
    if (again==1){
        console.log("REPEATED WORD")
```

If the word was not there, then the initial letter was saved in a list of letters, and if it was, we called that function recursively again so that it would bring a new word. Finally this word is positioned in the given ship:



```
if (deadSpacecraft==false && counterSpaceShipsWaves<SpaceshipLimit2){ //if the word
    if (partA==true){
        SpaceshipWord=comproveLetter(1);
    }
    if (partB==true || partC==true){
        SpaceshipWord=comproveLetter(2);
    }
```

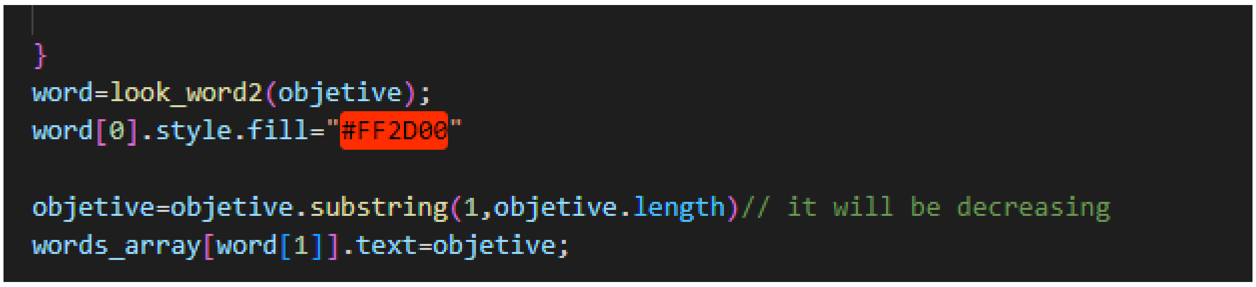
To be able to control the pulsations we use the code of the letter:



```
function letterDown(e){
    if(e.keyCode >= Phaser.Keyboard.A && e.keyCode <= Phaser.Keyboard.Z){

        if(fixed_Word==false && deadSpacecraft==false){// if we dont have a word avaiable we choose again
            objetive=look_word(e.key);

            console.log("word selectionated: "+objetive); //show us the word that is fixed
        }
        if(objetive!=""){ //fix the word
```

In this method, using Booleans, we control that if we have hit the first letter, we are going to set that word as our goal and therefore block searching for more words in the list with the letter that we press, that is, we can only continue with the word that we have started.

In this method we will also control that the letters disappear, that they change color and eliminate them if the word ends:

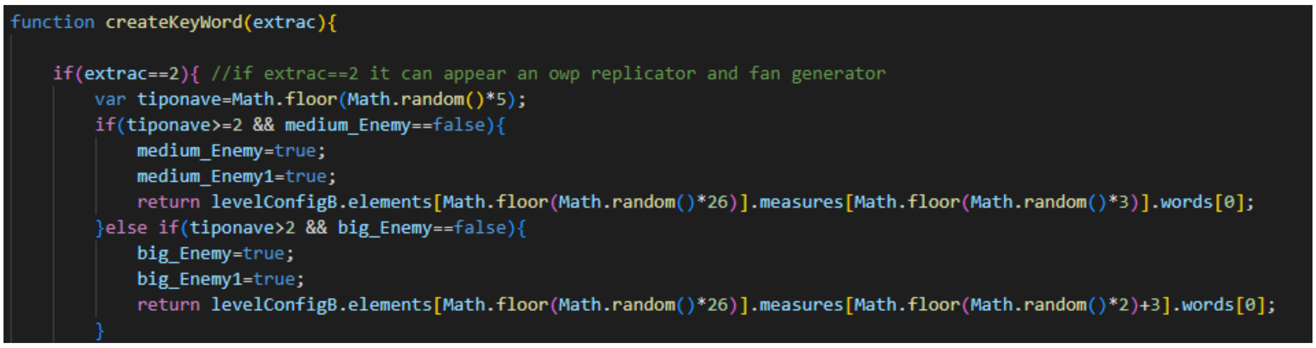


```
}
word=look_word2(objetive);
word[0].style.fill="#FF2D00"

objetive=objetive.substring(1,objetive.length)// it will be decreasing
words_array[word[1]].text=objetive;
```

And each time the round is changed, showing the information between round and round (current part, time, letters removed and dead ships). When it gets to the last round, instead of showing us the stats, we go into the win state.

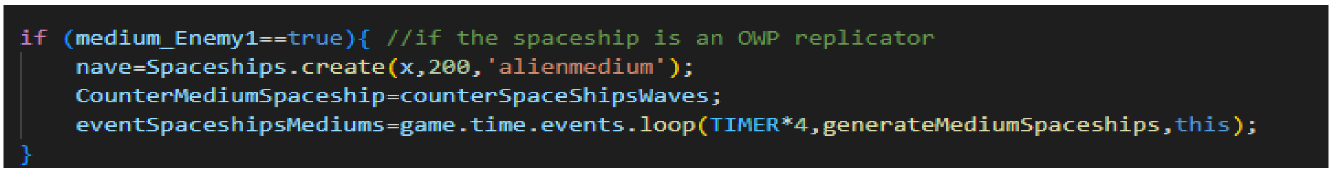
Part B

To create the different ships we have established a level of probability where knowing that the game is in mode B depending on what a random gives us we will create the corresponding ship also using jsonB:

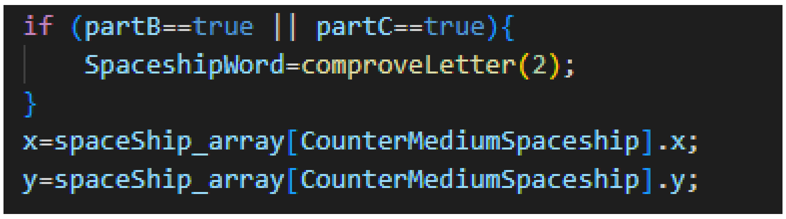


```
function createKeyWord(extrac){

    if(extrac==2){ //if extrac==2 it can appear an owp replicator and fan generator
        var tiponave=Math.floor(Math.random()*5);
        if(tiponave>=2 && medium_Enemy==false){
            medium_Enemy=true;
            medium_Enemy1=true;
            return levelConfigB.elements[Math.floor(Math.random()*26)].measures[Math.floor(Math.random()*3)].words[0];
        }else if(tiponave>2 && big_Enemy==false){
            big_Enemy=true;
            big_Enemy1=true;
            return levelConfigB.elements[Math.floor(Math.random()*26)].measures[Math.floor(Math.random()*2)+3].words[0];
        }
```

For medium ships to deploy normal ships we have simply replicated the original method but with the proper positions and angles, so that where the medium ship is generated, these generated words are also checked for repetition.

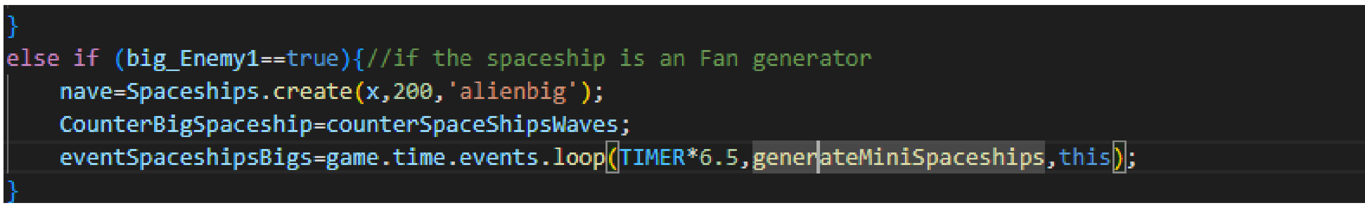


```
if (medium_Enemy1==true){ //if the spaceship is an OWP replicator
    nave=Spaceships.create(x,200,'alienmedium');
    CounterMediumSpaceship=counterSpaceShipsWaves;
    eventSpaceshipsMediums=game.time.events.loop(TIMER*4,generateMediumSpaceships,this);
}
```



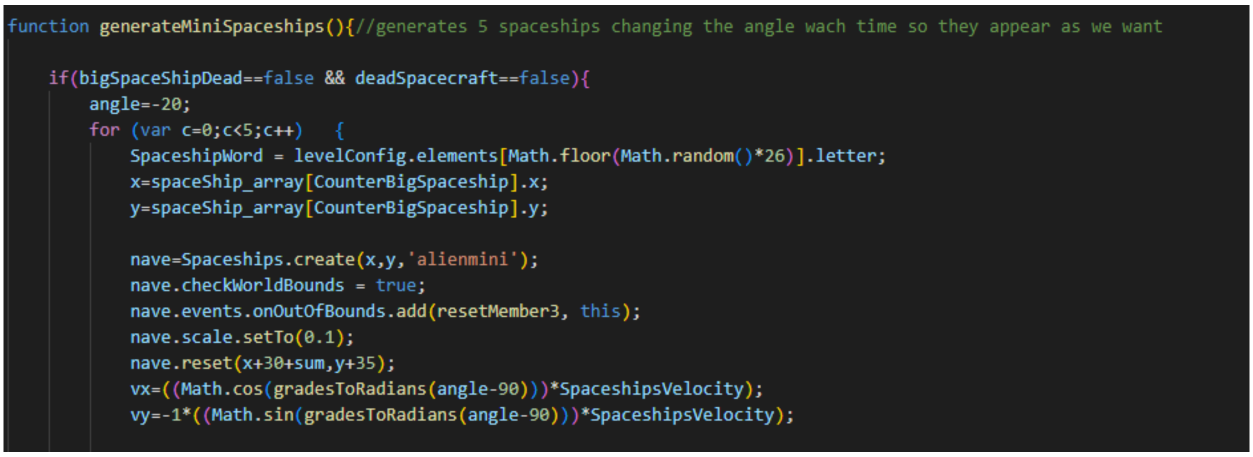
```
if (partB==true || partC==true){
    SpaceshipWord=comproveLetter(2);
}
x=spaceShip_array[CounterMediumSpaceship].x;
y=spaceShip_array[CounterMediumSpaceship].y;
```

For the big ship something similar, if it is generated, a loop function is added every 6 and a half seconds that shoots some small ships with a for, specifically 5, once one is generated the angle varies, thus forming a semicircle.



```
}
else if (big_Enemy1==true){//if the spaceship is an Fan generator
    nave=Spaceships.create(x,200,'alienbig');
    CounterBigSpaceship=counterSpaceShipsWaves;
    eventSpaceshipsBigs=game.time.events.loop(TIMER*6.5,generateMiniSpaceships,this);
}
```

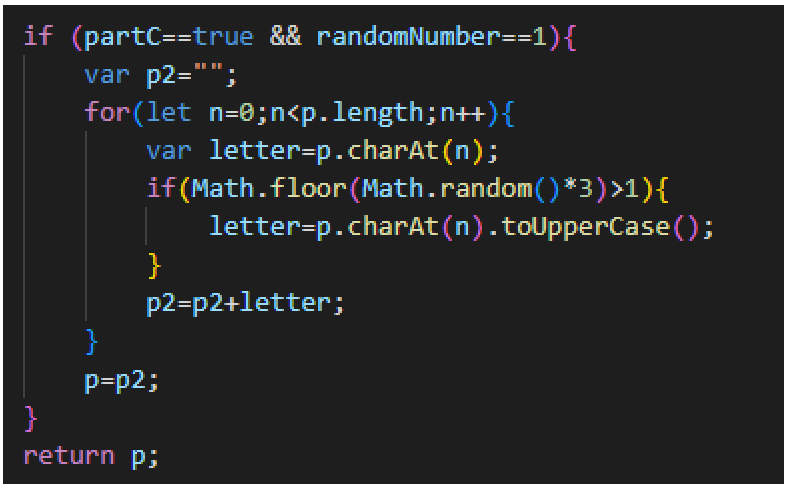
These ships are just one letter instead of one word.



```
function generateMiniSpaceships(){//generates 5 spaceships changing the angle wach time so they appear as we want

    if(bigSpaceShipDead==false && deadSpacecraft==false){
        angle=-20;
        for (var c=0;c<5;c++)   {
            SpaceshipWord = levelConfig.elements[Math.floor(Math.random()*26)].letter;
            x=spaceShip_array[CounterBigSpaceship].x;
            y=spaceShip_array[CounterBigSpaceship].y;

            nave=Spaceships.create(x,y,'alienmini');
            nave.checkWorldBounds = true;
            nave.events.onOutOfBounds.add(resetMember3, this);
            nave.scale.setTo(0.1);
            nave.reset(x+30+sum,y+35);
            vx=((Math.cos(gradesToRadians(angle-90)))*SpaceshipsVelocity);
            vy=-1*((Math.sin(gradesToRadians(angle-90)))*SpaceshipsVelocity);
```

Improvements part C

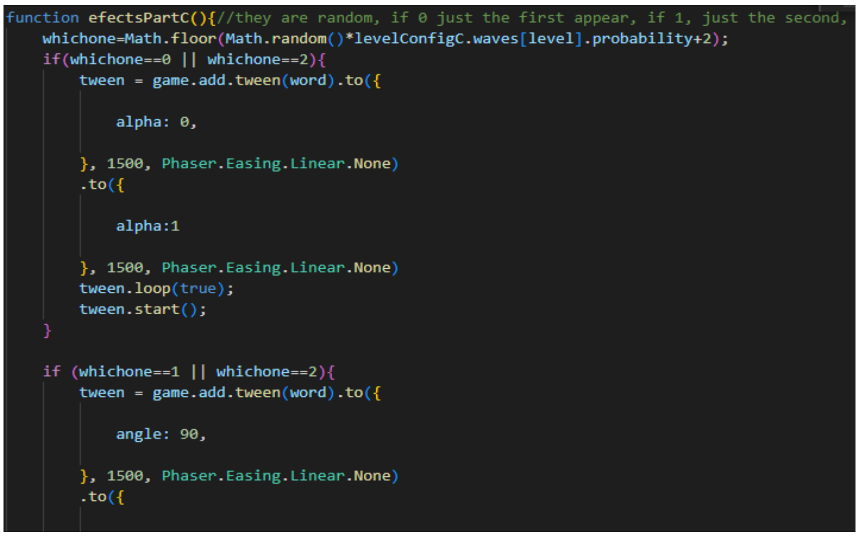
The first improvement that we have implemented in this part is to randomly capitalise words and thus complicate typing them quickly. For them, simply in the method that checked that the letters are not repeated, once we have already decided that the word can be written, we randomly put some letters in uppercase:



```
if (partC==true && randomNumber==1){
    var p2="";
    for(let n=0;n<p.length;n++){
        var letter=p.charAt(n);
        if(Math.floor(Math.random()*3)>1){
            letter=p.charAt(n).toUpperCase();
        }
        p2=p2+letter;
    }
    p=p2;
}
return p;
```

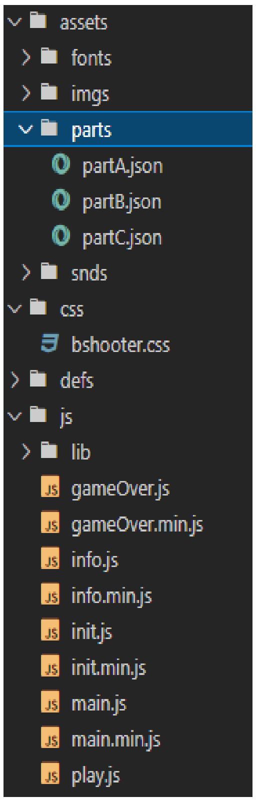
In addition, we have added a separate improvement which consists of making the ships appear from above if they have left the edges without hitting us, thus making the game not consist of dodge and it is impossible to advance like that.

This buff is not applied to ships spawned by the fan generator, if those ships leave the screen they are simply removed.



```
function efectsPartC(){//they are random, if 0 just the first appear, if 1, just the second,
    whichone=Math.floor(Math.random()*levelConfigC.waves[level].probability+2);
    if(whichone==0 || whichone==2){
        tween = game.add.tween(word).to({

            alpha: 0,

        }, 1500, Phaser.Easing.Linear.None)
        .to({

            alpha:1

        }, 1500, Phaser.Easing.Linear.None)
        tween.loop(true);
        tween.start();
    }

    if (whichone==1 || whichone==2){
        tween = game.add.tween(word).to({

            angle: 90,

        }, 1500, Phaser.Easing.Linear.None)
        .to({
```

Code structure



```
assets
  fonts
  imgs
  parts
    partA.json
    partB.json
    partC.json
  snds
css
  bshooter.css
defs
js
  lib
  gameOver.js
  gameOver.min.js
  info.js
  info.min.js
  init.js
  init.min.js
  main.js
  main.min.js
  play.js
```

- Info.js: gives us information about the game
- Init.js: the initial screen, through which we can move to info, preplay and settings. Settings.js: we can lower the volume
- Preplay.js – Various buttons with different parts.
- Win.js: screen when you finish all the rounds, show the information requested in the document, accuarcy
- Gameover.js: screen when you die by collision, you can go back to init with a button.
- Play.js: there is all the code of the different parts, with their waves and enemies