

# SpaceWar

**Titolo del progetto:** SpaceWar  
**Alunno/a:** Alessandro Perri  
**Classe:** Informatica 3BB  
**Anno scolastico:** 2022/2023  
**Docente responsabile:** Geo Petrini

1	Introduzione .....	3
1.1	Informazioni sul progetto .....	3
1.2	Abstract .....	3
	Situazione iniziale .....	3
	Approccio .....	3
	Risultati .....	3
1.3	Scopo .....	3
2	Analisi .....	4
2.1	Analisi del dominio .....	4
2.2	Analisi e specifica dei requisiti .....	4
2.2.1	Spiegazione elementi tabella dei requisiti: .....	7
2.3	Use case .....	8
2.4	Pianificazione .....	9
2.5	Analisi dei mezzi .....	10
2.5.1	Software .....	10
2.5.2	Hardware .....	10
3	Progettazione .....	11
3.1	Design del gioco .....	11
4	Implementazione .....	14
4.1	File index .....	14
4.1.1	Utilizzo Phaser 3 .....	14
4.2	Classe game .....	15
4.3	Scene .....	15
4.3.1	Menu .....	16
4.3.2	Playscene .....	17
4.3.3	InstructionScene .....	17
4.3.4	OptionScene .....	17
4.3.5	SpaceshipScene .....	18
4.3.6	DualPlayerScene .....	18
4.3.7	SinglePlayerScene .....	22
4.3.8	GameOverScene .....	23
4.4	Sprite .....	23
4.4.1	Laser .....	23
4.4.2	Missile .....	24
4.4.3	Planet .....	24
4.4.4	Ship .....	25
5	Test .....	28
5.1	Protocollo di test .....	28
5.2	Risultati test .....	31
5.3	Mancanze/limitazioni conosciute .....	31
6	Consuntivo .....	32
7	Conclusioni .....	33
7.1	Sviluppi futuri .....	33
7.2	Considerazioni personali .....	33
8	Bibliografia .....	33
8.1	Sitografia .....	33
8.2	Indice delle figure .....	34
9	Allegati .....	35

## **1 Introduzione**

---

### **1.1 Informazioni sul progetto**

Sono Alessandro Perri, allievo informatico presso la scuola Arti e Mestieri di Trevano e sono il produttore di questo progetto che è stato supervisionato dal professor Geo Petrini.

Il progetto è stato assegnato il 09.09.2022 con data di consegna entro il 23.12.2022.

### **1.2 Abstract**

#### **Situazione iniziale**

Inizialmente in questo progetto mi è stato chiesto di realizzare un gioco già esistente, ma migliorato. Questo gioco permette di giocare in due utilizzando una sola tastiera, oppure giocare da solo contro un bot.

#### **Approccio**

Per questo progetto, saranno necessarie buone conoscenze nella programmazione web, poiché per la creazione del gioco si utilizzerà il framework phaser che utilizza HTML5 Canvas per la visualizzazione e JavaScript per l'esecuzione. Grazie a questo framework sono riuscito a realizzare il gioco.

#### **Risultati**

Ho utilizzato il framework phaser che non avevo mai utilizzato prima, dunque ho avuto la possibilità di ampliare il mio bagaglio delle conoscenze. Uno dei punti fondamentali tenuti in considerazione durante il progetto riguarda lo studio per raggiungere la facilità di utilizzo per un utente medio. In conclusione, il progetto che ho realizzato è ora l'unico gioco moderno che simula il vecchio gioco Spacewar.

### **1.3 Scopo**

#### **Scopi**

- Didattici:
  - Saper creare e rispettare una progettazione.
  - Saper documentare il lavoro.
  - Saper creare i diari.
  - Saper creare Use-case.
- Operativi:
  - Saper creare una Gui.
  - Saper utilizzare phaser.

## 2 Analisi

### 2.1 Analisi del dominio

Per questo progetto mi è stato chiesto di creare un gioco dove si possa giocare sia da solo che con un amico e che abbia varie opzioni di scelta per poterlo rendere più bello e divertente. Questo gioco potrà essere utilizzato quando si avrà bisogno di un momento di svago e di divertimento.

### 2.2 Analisi e specifica dei requisiti

ID: REQ-01	
<b>Nome</b>	Modalità di gioco
<b>Priorità</b>	1
<b>Versione</b>	1.0
Sotto requisiti	
<b>001</b>	Single player (Player vs PC)
<b>002</b>	Dual player hotseat
ID: REQ-01	

ID: REQ-02	
<b>Nome</b>	Movimento
<b>Priorità</b>	1
<b>Versione</b>	1.0
Sotto requisiti	
<b>001</b>	Si può solo accelerare
<b>002</b>	Si può curvare
ID: REQ-02	

ID: REQ-03	
<b>Nome</b>	Navicelle che sparano laser e missili
<b>Priorità</b>	1
<b>Versione</b>	1.0
Sotto requisiti	
<b>001</b>	Delay tra i colpi
<b>002</b>	Lunghezza massima dei laser
<b>003</b>	Sparare diminuisce l'energia
<b>004</b>	Essere colpiti diminuisce la vita
ID: REQ-03	

ID: REQ-04	
<b>Nome</b>	Collisioni
<b>Priorità</b>	1
<b>Versione</b>	1.0
Sotto requisiti	
<b>001</b>	Ship con ship
<b>002</b>	Ship con laser e missili
<b>003</b>	Ship con pianeta
<b>004</b>	Laser e missili con pianeta
ID: REQ-04	

ID: REQ-05	
<b>Nome</b>	Ostacoli opzionali
<b>Priorità</b>	1
<b>Versione</b>	1.0
Sotto requisiti	
<b>001</b>	Pianeta al centro
<b>002</b>	Gravità
ID: REQ-05	

ID: REQ-06	
<b>Nome</b>	Poteri speciali
<b>Priorità</b>	1
<b>Versione</b>	1.0
Sotto requisiti	
<b>001</b>	Azzero l'energia e ricarica la vita
<b>002</b>	Turbo
ID: REQ-06	

ID: REQ-07	
<b>Nome</b>	Scelta navicella
<b>Priorità</b>	1
<b>Versione</b>	1.0
Sotto requisiti	
<b>001</b>	2 tipi di navicella
ID: REQ-07	

ID: REQ-08	
<b>Nome</b>	Exit
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note:</b>	Chiude il gioco
ID: REQ-08	

### 2.2.1 Spiegazione elementi tabella dei requisiti:

**ID:** identificativo univoco del requisito.

**Nome:** breve descrizione del requisito.

**Priorità:** l'importanza con la quale deve essere svolto un requisito.

**Versione:** indica la versione del requisito.

**Note:** eventuali osservazioni.

**Sotto requisiti:** elementi che compongono il requisito.

## 2.3 Use case

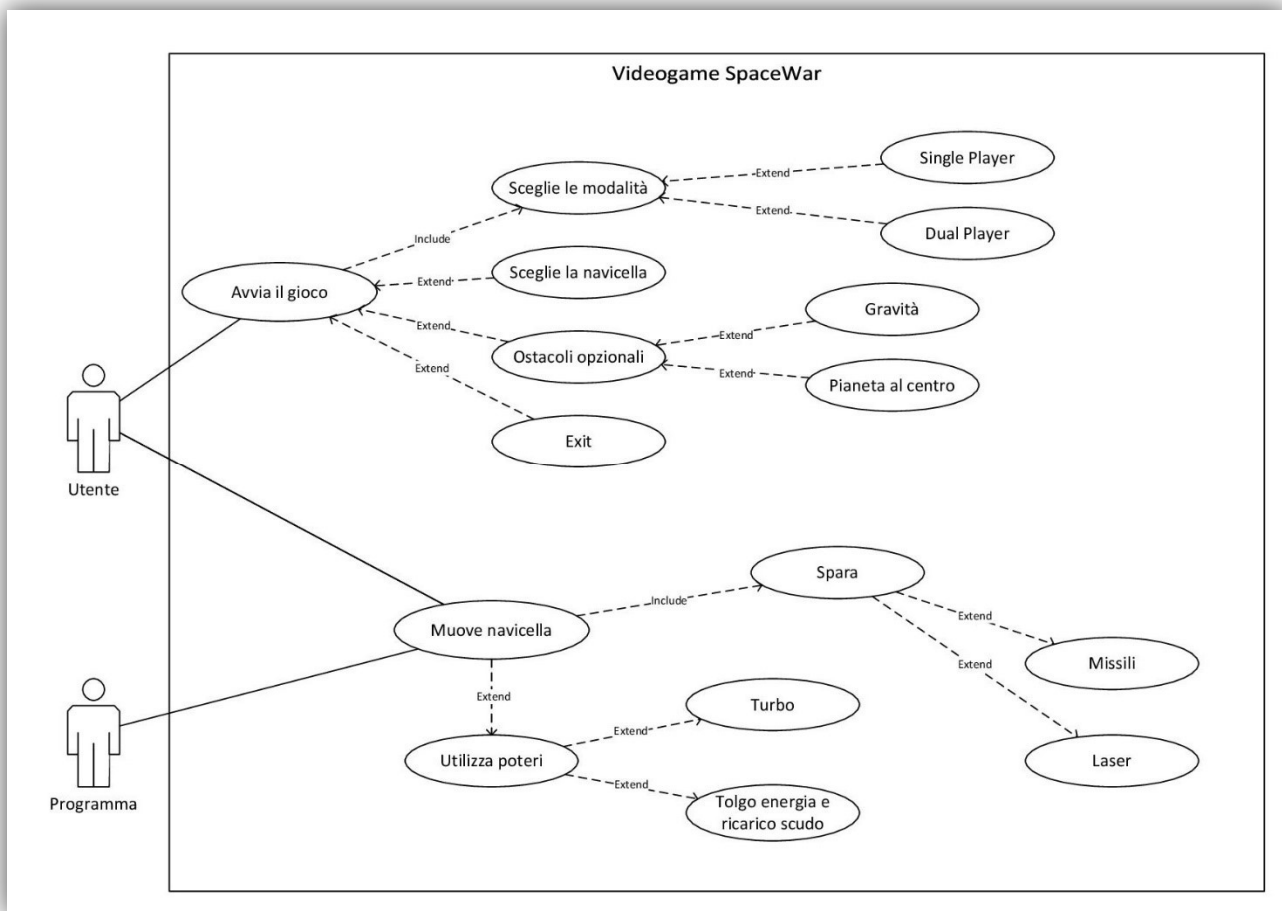


Figura 1: Use-Case



## 2.4 Pianificazione

Di seguito c'è la pianificazione preventiva, nel Gantt preventivo è stimato il tempo necessario per completare le task del progetto.

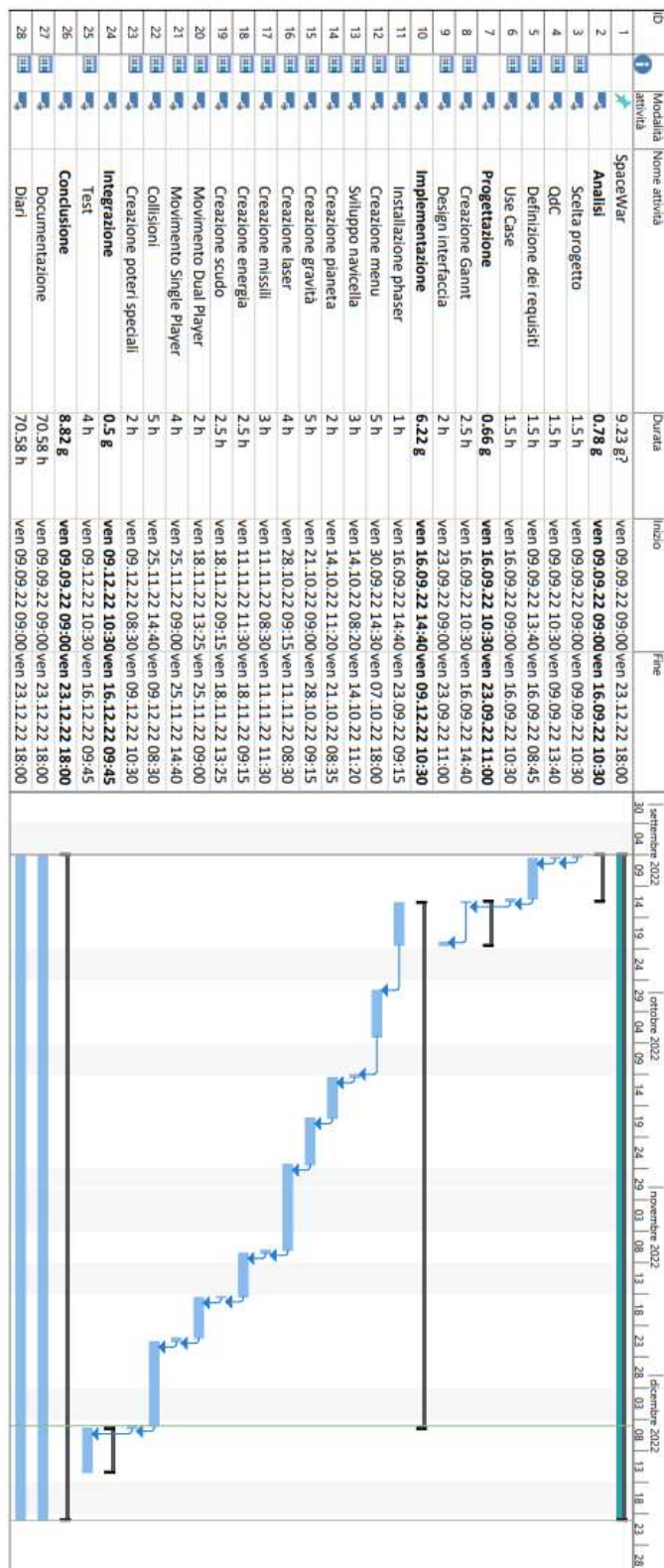


Figura 2: Gantt preventivo

## 2.5 Analisi dei mezzi

Per realizzare il progetto è stato utilizzato un computer con Visual Studio Code, Phaser3 come Framework, e un server php per visualizzare il gioco.

### 2.5.1 Software

Per realizzare questo progetto sono stati utilizzati i seguenti software:

- OS Computer - Windows 10: Per la creazione di tutto il progetto.
- Visual Studio Code - 1.72: Per tutto ciò che riguarda la scrittura del codice.
- Phaser – 3.55.2: Utilizzato come framework.
- Server php: Utilizzato per la visualizzazione del gioco.

### 2.5.2 Hardware

Un computer con:

- CPU Intel Core i7-7700
- RAM 16 GB

### 3 Progettazione

Questo capitolo descrive esaurientemente come deve essere realizzato il prodotto fin nei suoi dettagli. Una buona progettazione permette all'esecutore di evitare fraintendimenti e imprecisioni nell'implementazione del prodotto.

#### 3.1 Design del gioco

Il gioco è strutturato nel modo seguente:

##### Interfaccia home page

La home page è l'interfaccia che si visualizza non appena si avvia il gioco, da qui posso scegliere le varie opzioni.



Figura 3: Home page

### Interfaccia bottone play

Nell'interfaccia del bottone play si hanno le due scelte per poter giocare, single player o dual player.

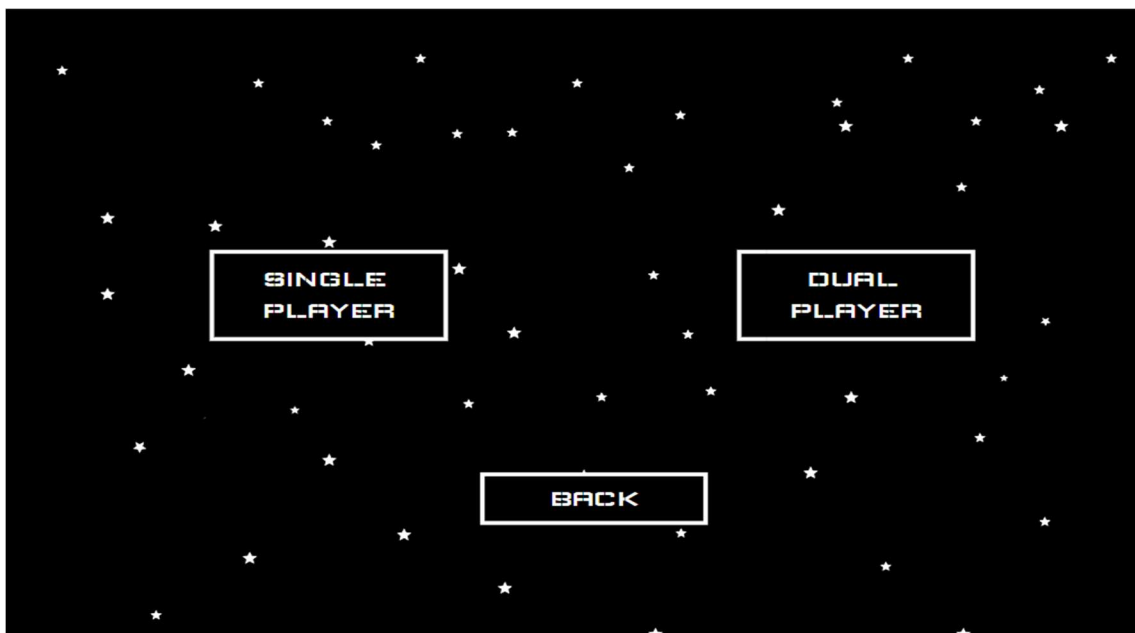


Figura 4: Interfaccia bottone play

### Interfaccia bottone options

Nell'interfaccia options ci sono le varie opzioni da selezionare, si può scegliere di utilizzare il pianeta e la gravità e quale navicella utilizzare

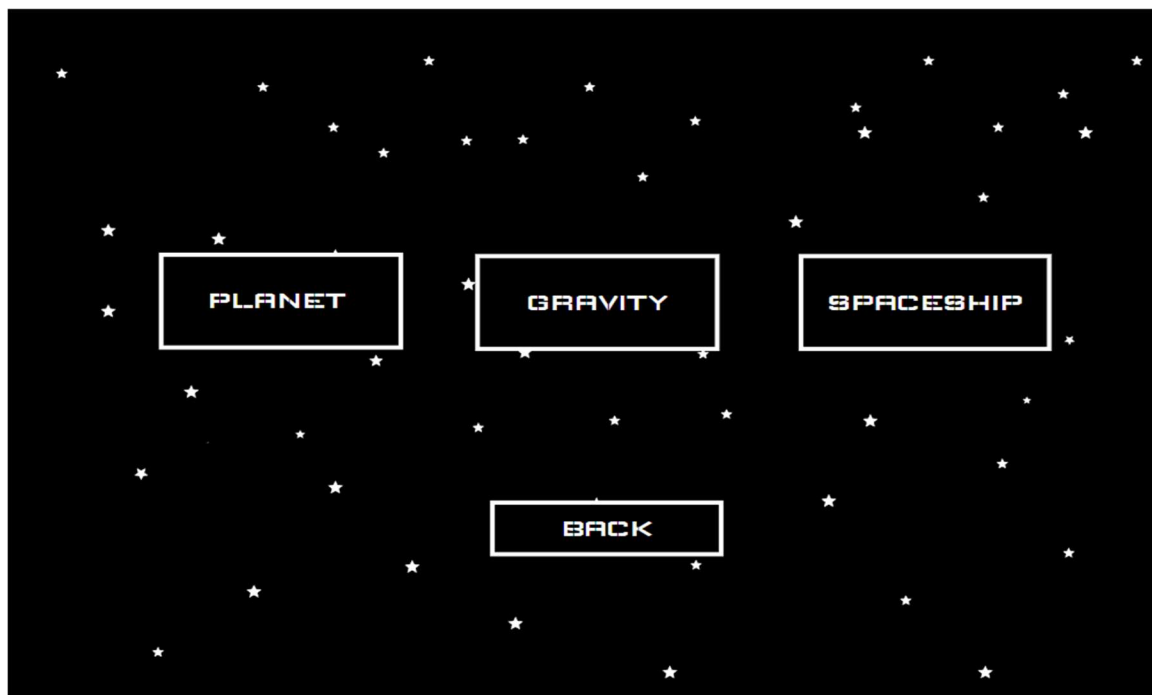


Figura 5: Interfaccia bottone options

### Interfaccia bottone instructions

Nell'interfaccia del bottone instructions vengono mostrati i comandi per poter muovere, far sparare e utilizzare il potere speciale della navicella.

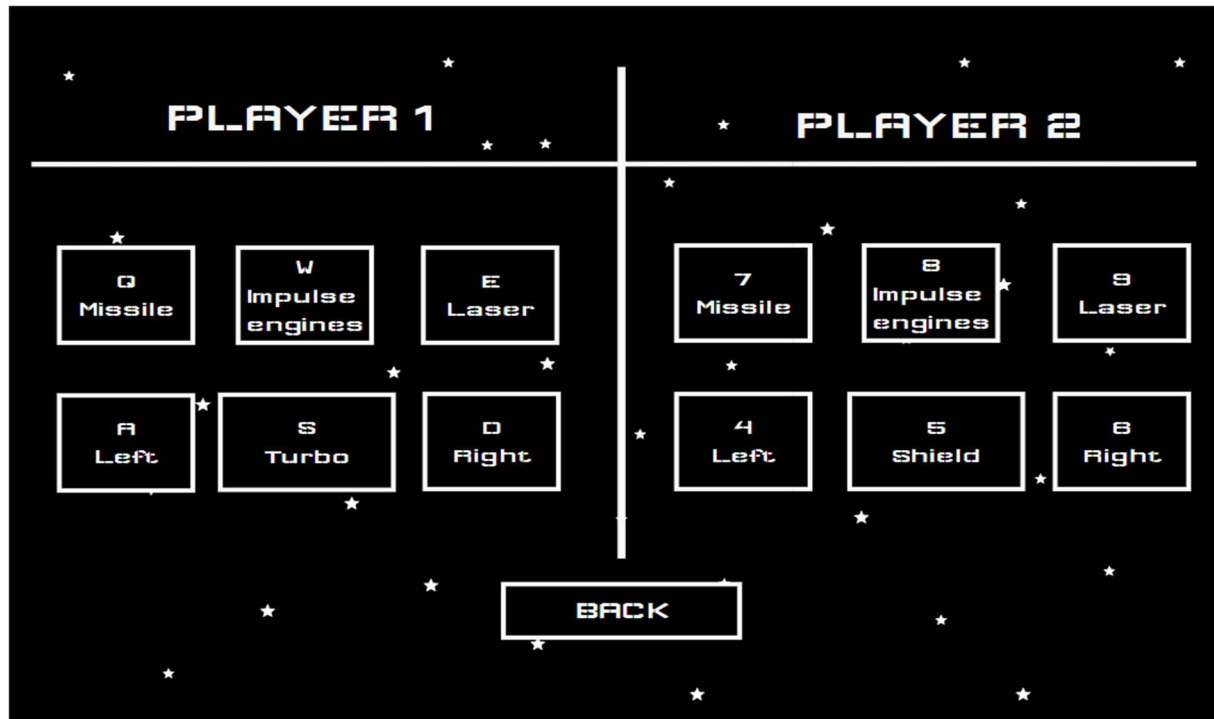


Figura 6: Interfaccia bottone instructions

### Interfaccia bottone exit

Appena il bottone exit viene cliccato si chiuderà la pagina.

## 4 Implementazione

### 4.1 File index

Nel file index vengono importate tutte le classi che ho implementato al fine di visualizzarle grazie alla classe presente nel body game.js.

```
<!DOCTYPE html>
<html>
<head>
  <title>SPACEWAR</title>
  <meta charset="UTF-8">
  <meta name="keywords" content="HTML,CSS,XML,JavaScript">
  <meta name="author" content="Alessandro Perri">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
  <link rel="icon" href="favicon.ico" type="image/x-icon">

  <script type="text/javascript" src="./assets/lib/phaser.js"></script>
  <script type="text/javascript" src="./assets/lib/phaser-arcade-physics.js"></script>
  <script type="text/javascript" src="./js/scenes/Menu.js"></script>
  <script type="text/javascript" src="./js/scenes/PlayScene.js"></script>
  <script type="text/javascript" src="./js/scenes/OptionsScene.js"></script>
  <script type="text/javascript" src="./js/scenes/SpaceshipScene.js"></script>
  <script type="text/javascript" src="./js/scenes/InstructionsScene.js"></script>
  <script type="text/javascript" src="./js/scenes/SinglePlayerScene.js"></script>
  <script type="text/javascript" src="./js/scenes/DualPlayerScene.js"></script>
  <script type="text/javascript" src="./js/scenes/GameOverScene.js"></script>
  <script type="text/javascript" src="./js/sprites/Ship.js"></script>
  <script type="text/javascript" src="./js/sprites/Missile.js"></script>
  <script type="text/javascript" src="./js/sprites/Laser.js"></script>
  <script type="text/javascript" src="./js/sprites/Planet.js"></script>
</head>
<body>
  <script type="text/javascript" src="./js/game.js"></script>
</body>
</html>
```

Figura 7: Codice file index.html

#### 4.1.1 Utilizzo Phaser 3

Per questo gioco, come già detto precedentemente ho utilizzato il framework Phaser nella sua ultima versione 3.55.2.

```
<script type="text/javascript" src="./assets/lib/phaser.js"></script>
<script type="text/javascript" src="./assets/lib/phaser-arcade-physics.js"></script>
```

Figura 8: Importazione librerie phaser

Ho importato le librerie di phaser nel codice per poter utilizzare tutte le sue funzionalità.

## 4.2 Classe game

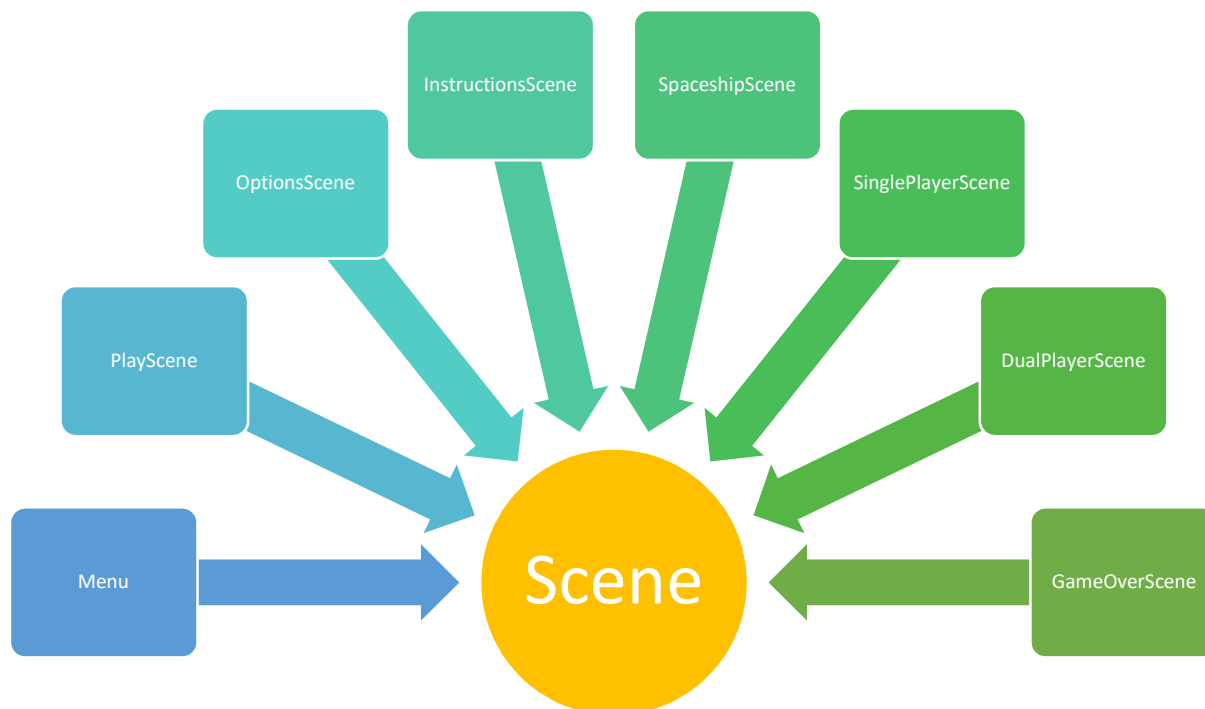
Nella seguente classe troveremo tutte le configurazioni per la creazione del gioco. Essa comprende una serie di variabili globali come pure le varie scene (menu, impostazioni, gioco).

### Configurazioni generali



## 4.3 Scene

Il codice del gioco è contenuto nelle diverse scene, esse ti permettono di rendere il codice più leggibile e nel mio caso di passare da una scena all'altra semplicemente cliccando un bottone.



Ogni scena ha bisogno di una key per essere riconosciuta e deve dichiarare i metodi `preload()`, `create()`, `update()`.

### **preload()**

In questo metodo carico tutto quello che riguarda i file e le immagini dei vari oggetti/sprite.

### **create()**

Nel create creo i vari oggetti, le sprite, e istanzio i metodi necessari per il codice.

### **update()**

Questo metodo invece gestisce il gameloop.

## **4.3.1 Menu**

Il menu è la scena principale che viene visualizzata appena si apre il gioco, da qui cliccando i vari pulsanti posso passare alle scene successive.

### **Constructor**

```
class OptionsScene extends Phaser.Scene {
    constructor() {
        super({key: 'OptionsScene'});
    }
}
```

Figura 9: Constructor menu

Inizialmente creo la classe estendo *Phaser.Scene* siccome la uso come scena. Dopodiché nel costruttore dichiaro la key che mi servirà per poter passare da una scena all'altra.

### **preload()**

```
preload(){
    this.load.image("background", "assets/images/background.png");
    this.load.image("ship1", "assets/images/ship1.png");
    this.load.image("ship2", "assets/images/ship2.png");
    this.load.image("title", "assets/images/Title.png");
    this.load.image("play", "assets/images/ButtonPlay.png");
    this.load.image("options", "assets/images/ButtonOptions.png");
    this.load.image("instructions", "assets/images/ButtonInstructions.png");
    this.load.image("exit", "assets/images/ButtonExit.png");
}
```

Figura 10: Preload menu

Per lo stile del pulsante ho utilizzato delle immagini dunque nel preload bisogna caricarle. Come primo argomento passo una key che mi permette poi di andare a prendere l'immagine, mentre come secondo argomento inserisco il percorso dell'immagine.

### **create()**

```
this.playButton = this.add.image(this.game.renderer.width / 2, this.game.renderer.height / 2, "play");
```

Figura 11: Create menu, aggiunta pulsanti

Dopo aver caricato le immagini, assegno l'immagine ad una variabile mettendo la posizione x-y e la chiave che ho inserito nel preload per poter prendere l'immagine.



```
//play
this.playButton.setInteractive();
this.playButton.on("pointerup", () => {
    this.scene.start("PlayScene");
})
```

Figura 12: Azione pulsanti

Successivamente metto *setInteractive()* alla variabile, perché questa funzione mi permette di trasformare l'immagine in un vero e proprio pulsante.

Dopodiché utilizzo la funzione "*pointerup*" che mi permette di passare alla scena play non appena clicco il pulsante.

Infine, eseguo lo stesso procedimento per i pulsanti options, instructions e exit che però al click non passerà a nessuna scena ma chiuderà l'applicazione.

#### 4.3.2 Playscene

In questa scena ci sarà la scelta di gioco, ovvero in single player o in dual player mode.

Per l'implementazione dai pulsanti singleplayer, dualplayer e back, ho eseguito esattamente gli stessi passaggi del punto 4.3.1.

#### 4.3.3 InstructionScene

La scena instruction ti permette di visualizzare tutti i comandi del player 1 e 2 per poter muovere e far sparare la navicella.

Per il pulsante back ho eseguito gli stessi passaggi del punto 4.3.1.

#### 4.3.4 OptionScene

Nella optionsScene ho la possibilità di scegliere le opzioni se utilizzare il pianeta, se utilizzare la gravità e quale navicella usare.

**create()**

```
//GravitySelected
this.gravityButtonSelected.setInteractive();
this.gravityButtonSelected.visible = false;
this.gravityButtonSelected.on("pointerdown", () => {
    //this.registry.set('gravityIndex', 0);
    this.game.config._gravityIndex = 0;
})

//Gravity
this.gravityButton.setInteractive();
this.gravityButton.on("pointerdown", () => {
    this.game.config._gravityIndex = 1;
    //this.registry.set('gravityIndex', 1);
})
```

Figura 13: Create OptionScene, utilizzo variabili globali

Per la scelta delle opzioni ho utilizzato le variabili globali create nella classe game nel seguente modo.

Per utilizzare le variabili globali mi basta fare riferimento al config scrivendo *this.game.config.\_gravityIndex* e successivamente vado ad assegnarli il valore 1 se viene cliccata.

update()

```
update() {
    //gravity
    if(this.game.config._gravityIndex == 1){
        //GravitySelected
        this.gravityButtonSelected.visible = true;
        this.gravityButton.visible = false;
    }else if(this.game.config._gravityIndex == 0){
        //GravitySelected
        this.gravityButtonSelected.visible = false;
        this.gravityButton.visible = true;
    }
}
```

Figura 14: Update OptionScene

Nell'update ho inserito il controllo per verificare se l'opzione è selezionata oppure no. Quando clicco il pulsante *gravityButton* esso sparirà e apparirà il pulsante *gravityButtonSelected* e stessa cosa per la gravità e per la navicella.

#### 4.3.5 SpaceshipScene

Nella scena spaceship ho la possibilità di scegliere quale navicella utilizzare, per l'implementazione del codice ho eseguito lo stesso procedimento che ho utilizzato nel punto 4.3.4

#### 4.3.6 DualPlayerScene

In questa scena è presente tutto il codice che riguarda il gioco vero e proprio, al suo interno vengono utilizzate diverse sprite che sono documentate nel punto 4.4.

create()

```
//Keys player1
this.player1Keys = {
    up: this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.W),
    left: this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.A),
    special: this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.S),
    right: this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.D),
    missile: this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.Q),
    laser: this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.E)
};
```

Figura 15: Create DualPlayerScene, chiavi tastiera

Creo la variabile *player1keys* che contiene le azioni che devono essere eseguite non appena si clicca il tasto della tastiera specifico. Successivamente creo anche la variabile *player2keys* con i relativi tasti.

```
//collide ship-ship
this.physics.add.overlap(this.ship, this.ship2, this.collideShipShip);

//collide missile
this.physics.add.overlap(this.ship, this.missiles2, this.collideMissileShip);

//collide laser
this.physics.add.overlap(this.ship, this.lasers2, this.collideLaserShip);

//collide laser-missile
this.physics.add.overlap(this.missiles2, this.lasers, this.collideLaserMissile);
```

Figura 16: Overlap

L'*overlap* ti permette di creare le collisioni desiderate, il primo argomento indica chi verrà colpito, il secondo chi colpisce e nel terzo inserisco il metodo che deve essere eseguito appena essi collidono.

### createShip()

Questo metodo mi permette di creare all'interno del gioco la ship documentata al punto 4.4.4 , configurando tutti i requisiti richiesti dal suo costruttore e assegnandoli i missili e i laser.

```
createShip(){
  //Se la ship selezionata = 1
  if(this.game.config._playerShip == 1){
    this.ship = new Ship({
      scene: this,
      texture: 'ship1',
      x: 100,
      y: 450,
    }, this.game.config._energyShip,
    this.game.config._lifeShip,
    this.game.config._gravityIndex,
    this.game.config._planetIndex,
    this.game.config._special1,
    this.game.config._velocita);

  //Se la ship selezionata = 2
  }else if(this.game.config._playerShip == 2){
    this.ship = new Ship({
      scene: this,
      texture: 'ship2',
      x: 100,
      y: 450,
    }, this.game.config._energyShip,
    this.game.config._lifeShip,
    this.game.config._gravityIndex,
    this.game.config._planetIndex,
    this.game.config._special1,
    this.game.config._velocita);
  }
}
```

Figura 17: CreateShip

Creo una nuova ship e assegno tutti i valori di cui ha bisogno, essa viene creata in base alla scelta della navicella nella scena *spaceShip*. La stessa cosa sarà fatta per i metodi *createShip2* e *createPlanet*.

```
//Gruppo oggetti di tipo missiles
this.missiles = this.physics.add.group({
  classType: Missile,
});
this.ship.assignMissiles(this.missiles);

//Gruppo oggetti di tipo laser
this.lasers = this.physics.add.group({
  classType: Laser,
});
this.ship.assignLasers(this.lasers);
```

Figura 18: Assegnazione missili e laser

Creo un gruppo di oggetti di tipo missile e di tipo laser, e gli assegno il tipo della classe. Successivamente dichiaro i metodi *assignMissiles* e *assignLasers* documentati al punto 4.4.1

## Collisioni

```
//collisioni ship
collideShipShip(ship, ship2){
  ship.vita = ship.vita - 10;
  ship2.vita = ship2.vita - 10;
}

collideMissileShip(ship, missile){
  missile.destroy();
  ship.vita = ship.vita -1;
}

collideLaserShip(ship, laser){
  laser.destroy();
  ship.vita = ship.vita -0.5;
}

//collisioni pianeta
collideShipPlanet(planet, ship){
  ship.vita = ship.vita - 1;
}

collideMissilePlanet(planet, missile){
  missile.destroy();
}

collideLaserPlanet(planet, laser){
  laser.destroy();
}
```

Figura 19: Collisioni

Questi sono tutti i metodi utili per la creazione delle collisioni tra i vari oggetti, ogni metodo ha le sue funzionalità.

Nella collisione delle due ship, non metto *destroy()* perché altrimenti da errore su tutto il resto siccome le due navicelle non esistono più.

## rechargeEnergy()

```
//Timer per ricaricare l'energia
this.timedEvent = this.time.addEvent({ delay: 1500, callback: this.rechargeEnergy, callbackScope: this, loop: true });
}
rechargeEnergy(){
    if(this.ship.energia < 3){
        this.ship.energia += this.ship.rechargeRate;
    }
    if(this.ship2.energia < 3){
        this.ship2.energia += this.ship2.rechargeRate;
    }
}
```



















Figura 20: RechargeEnergy






















In questo metodo semplicemente faccio un controllo che mi dice che se l'energia è minore di 3 devo ricaricarla.



Successivamente nel create, ho dichiarato il *timedEvent*, che mi permette di eseguire il metodo *rechargeEnergy* con il delay desiderato, in questo caso viene ricaricata l'energia di 0.5 ogni 1.5 secondi.

## updateHUD()

Nei metodi *updateHud* 1 e 2 c'è tutto quello che riguarda lo score del giocatore mentre sta giocando, di seguito viene mostrata la logica con la quale è strutturato.

Vite			
3			
2.5			
2			
1.5			
1			
0.5			
0	GAMEOVER		

Energia			
3			
2.5			
2			
1.5			
1			
0.5			
0			

Specialità	
1	
0	

**update()**

```

this.ship.update(this.player1Keys);
this.ship2.update(this.player2Keys);

this.updateHUDP1();
this.updateHUDP2();

//GAMEOVER
if (this.ship.vita <= 0 || this.ship2.vita <= 0){
    this.scene.start("GameOverScene");
}

//Pianeta
if(this.game.config._planetIndex == 1){
    this.planet.update(0.2);
}

```

Figura 21: Update Hud

Nell'update dichiaro:

- L'update delle ship con le variabili playerKeys che mi permetteranno di muovere le navicelle.
- I due HUD per lo score del giocatore.
- La scena gameOver che parte nel caso in cui la vita diventasse 0.
- L'update del pianeta con il valore 0.2 che mi permette di dare l'animazione che gira.

#### 4.3.7 SinglePlayerScene

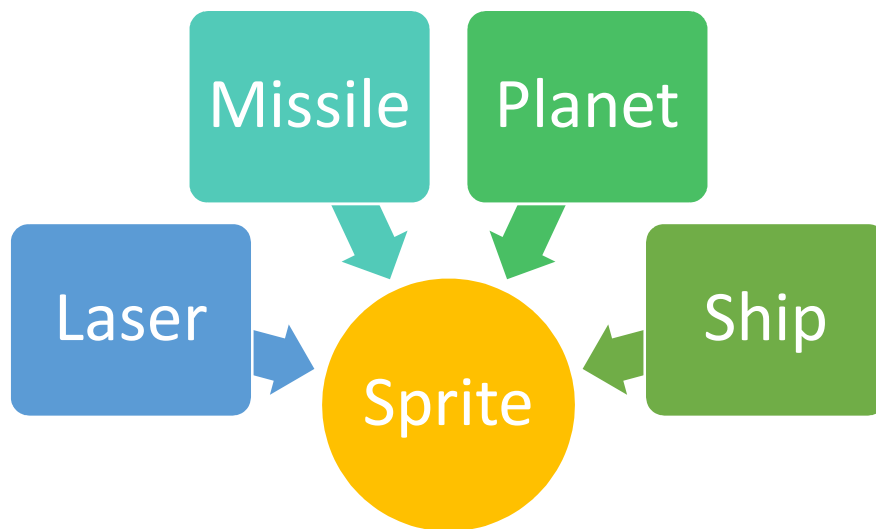
In questa scena, l'obiettivo era quello di permettere a colui che gioca di giocare da solo contro un bot. La struttura del codice è uguale a quella della *DualPlayerScene*, però non funzionante a causa del bot che non sono riuscito a creare.

#### 4.3.8 GameOverScene

Il codice della scena *gameover* è molto semplice, appena la vita di una delle due ship diventerà uguale a 0 essa entrerà in gioco.

#### 4.4 Sprite

Le sprite sono degli oggetti di gioco che possono essere immagini sia statiche che animate. Essi sono una parte fondamentale per il progetto perché riguardano tutti gli oggetti utili per giocare.



##### 4.4.1 Laser

```
function Laser (scene)
{
  Phaser.GameObjects.Image.call(this, scene, -3, -3, 'laser');
},
```

Figura 22: Laser function

Questa funzione mi permette di prendere l'immagine che verrà assegnata all'oggetto dalla classe in cui verrà dichiarata. Le coordinate dell'oggetto inizialmente erano  $x=0$  e  $y=0$ , le ho cambiate con  $x=-3$  e  $y=-3$ , dato che se la ship andava in alto a sinistra si scontrava con l'oggetto e perdeva vita.



```
fire: function (ship, speed)
{
    this.lifespan = 1000;
    this.setActive(true);
    this.setVisible(true);

    this.rotation = ship.rotation;

    const width = ship.width / 2 + this.width / 2;
    const posizione = new Phaser.Geom.Point(width, 0);
    Phaser.Math.Rotate(posizione, ship.rotation);
    this.setPosition(ship.x + posizione.x, ship.y + posizione.y);

    const angle = Phaser.Math.DegToRad(ship.body.rotation); // si può ruotare solo con i radianti
    this.body.world.scene.physics.velocityFromRotation(angle, speed, this.body.velocity);

    this.body.setCircle(1, 60, 12);
}
```

Figura 23: Laser fire

Il metodo *fire* mi permette di far sparare l'oggetto alla ship in base alla sua rotazione e direzione. Inizialmente dichiaro il *lifespan* che indica la durabilità del laser e dopodiché lo rendo attivo e visibile.

Grazie ai metodi *Phaser.Math.Rotate*, *setPosition* e *velocityFromRotation* riesco a sparare il laser in base al movimento e alla direzione della ship.

Infine, metto alla base dell'oggetto il *setCircle* che mi permette di dargli un bordo utile far collidere il laser.

#### update()

```
update: function ()
{
    this.lifespan -= 10;

    if (this.lifespan <= 0)
    {
        this.setActive(false);
        this.setVisible(false);
        this.body.stop();
    }
}
```

Figura 24: Update laser

L'update mi permette semplicemente di dare una durabilità al laser che una volta finita lo farà sparire.

#### 4.4.2 Missile

La classe del missile è uguale a quella del laser documentata nel punto precedente.

#### 4.4.3 Planet

```
config.scene.physics.world.enable(this);
config.scene.add.existing(this);

this.body.allowGravity = false;
this.body.setCircle(85, 7, 7);
this.body.setImmovable(true);
```

Figura 25: Codice planet

Questa è la classe del pianeta che rende il gioco più complicato se viene selezionato.

Per prima cosa aggiungo la scena esistente e la fisica, dopodiché metto l'*allowGravity* settato a false e il *setImmovable* settato a true che permettono al pianeta di rimanere immobile al posto di cadere in basso.



**update()**

```
update(rotazione){
    this.body.rotation += rotazione;
}
```

Figura 26: Update planet

Nell'update richiedo il parametro rotazione che inserisco al punto 4.3.6, che mi permette di animare il pianeta facendolo girare e rendere il tutto più bello.

```
//decelerazione attiva
this.body.setDamping(true);

//ship rallenta da sola
this.body.setDrag(0.5);

//collisione con i bordi
this.body.setCollideWorldBounds(true);
```

Figura 27: Variabili ship

#### 4.4.4 Ship

Il metodo *setDamping* mi permette di attivare la decelerazione della ship, seguita dal metodo *setDrag* che fa rallentare da sola la navicella se non clicco niente.

Infine, attivo la collisione con i bordi grazie al metodo *setCollideWorldBounds*.

```
assignMissiles(missiles){
    this.missiles = missiles;
}

assignLasers(lasers){
    this.lasers = lasers;
}
```

Figura 28: Assign laser e missili alla ship

Questi metodi permettono di poter utilizzare e di assegnare alla ship i missili e i laser documentati precedentemente.

```

if(this.vita > 0){
    if(keys == null){
    }else{
        //W
        if(keys.up.isDown) {
            this.scene.physics.velocityFromRotation(this.rotation, this.vel, this.body.acceleration);
        } else{
            this.body.setAccelerationX(0);
            this.body.setAccelerationY(0);
        }
        //A
        if(keys.left.isDown) {
            this.body.setAngularVelocity(-200);
        } //D
        } else if(keys.right.isDown) {
            this.body.setAngularVelocity(200);
        }else{
            this.body.setAngularVelocity(0);
        }
    }
}

```

Figura 29: Tasti W, A e D

Queste sono le azioni che riguardano il movimento della navicella non appena verrà cliccato e premuto un tasto specifico.

#### **Tasto W**

La ship accelererà in base alla rotazione.

#### **Tasto A**

La ship ruota verso sinistra.

#### **Tasto D**

La ship ruota verso destra

```
//S
if(keys.special.isDown) {
    if(this.special == 1){
        this.vel = this.vel + 100;
        this.body.setMaxVelocity(this.vel);
        this.special -= 1;
    }else if(this.special == 2){
        if(this.energia > 0){
            this.vita = 3;
            this.energia = 0;
            this.special -= 2;
        }
    }
}
//Q
} else if(Phaser.Input.Keyboard.JustDown(keys.missile)) {
    var missile = this.missiles.get();

    if (missile && this.energia > 0.5)
    {
        let offset = new Phaser.Geom.Point(0, -this.body.height / 2);
        Phaser.Math.Rotate(offset, this.body.rotation);

        missile.fire(this, 1000);
        this.energia = this.energia - 1;
    }
}
```

Figura 30: Tasti S e Q

Queste azioni invece ti permettono di sparare e di attivare il potere speciale.

### Tasto S

Attivo il potere speciale utilizzabile una sola volta.

In base al player che clicca il tasto viene eseguito un potere differente:

- Player 1: turbo.
- Player 2: azzero l'energia e ricarico la vita.

### Tasto Q

Sparo i missili.

Posso sparare se l'energia è maggiore di 0.5 e grazie al *math.rotate* sparo in base alla direzione della ship

### Tasto E

Sparo i laser.

Posso sparare se l'energia è maggiore di 0 e a livello di codice è uguale al tasto Q.

```
if(this.gravita == 1){
    this.body.gravity = this.body.gravity.normalize().multiply(500, 500);
}else if(this.gravita == 1 && this.planeta == 1){
    this.body.gravity = new Phaser.Point(planet.body.x - this.body.x, planet.body.y - this.body.y);
    this.body.gravity = this.body.gravity.normalize().multiply(500, 500);
}
```

Figura 31: Gravità

In questa parte provo a creare la gravità, che però non sono riuscito a farla diventare funzionante.

## 5 Test

### 5.1 Protocollo di test

<b>Test Case:</b>	TC-001	<b>Nome:</b>	Modalità di gioco
<b>Riferimento:</b>	REQ-01		
<b>Descrizione:</b>	Selezionare la modalità di gioco, ovvero single player o dual player.		
<b>Procedura:</b>	<ol style="list-style-type: none"> <li>1. Cliccare play</li> <li>2. Scegliere la modalità</li> <li>3. Se seleziono single player verificare che il player 2 si muova da solo</li> <li>4. Se seleziono dual player verificare che il player 2 si muova con i numeri</li> </ol>		
<b>Risultati attesi:</b>	In modalità single player il player 2 deve essere controllato dal pc, mentre in modalità dual player il player 2 deve essere controllato dall'utente tramite i numeri.		

<b>Test Case:</b>	TC-002	<b>Nome:</b>	Movimento
<b>Riferimento:</b>	REQ-02		
<b>Descrizione:</b>	La navicella può solo andare avanti e curvare.		
<b>Procedura:</b>	<ol style="list-style-type: none"> <li>1. Iniziare a giocare</li> <li>2. Verificare che posso andare solo avanti e curvare con i vari tasti.</li> <li>3. Verificare che il movimento sia graduale e non a scatti.</li> <li>4. Per il Movimento vedi GUI instructions.</li> </ol>		
<b>Risultati attesi:</b>	La navicella deve curvare e accelerare.		

<b>Test Case:</b>	TC-003	<b>Nome:</b>	Navicelle che sparano laser e missili
<b>Riferimento:</b>	REQ-03		
<b>Descrizione:</b>	Una navicella quando si sposta deve poter sparare sia laser che missili per eliminare l'avversario.		
<b>Procedura:</b>	<p>Player 1</p> <ul style="list-style-type: none"> <li>• Cliccare il tasto "Q" e verificare se spara i missili</li> <li>• Cliccare il tasto "E" e verificare se spara i laser</li> </ul> <p>Player 2</p> <ul style="list-style-type: none"> <li>• Cliccare il tasto "7" e verificare se spara i missili</li> <li>• Cliccare il tasto "9" e verificare se spara i laser</li> </ul>		
<b>Risultati attesi:</b>	La navicella deve sparare sia i missile che i laser.		

<b>Test Case:</b>	TC-004	<b>Nome:</b>	Specifiche laser e missili
<b>Riferimento:</b>	REQ-03		
<b>Descrizione:</b>	<p>I laser devono avere una lunghezza massima e se colpiscono un player esso perda mezza vita.</p> <p>I missili non hanno una lunghezza massima e se colpiscono un player esso perda una vita.</p>		
<b>Procedura:</b>	<p>Verifiche missile</p> <ul style="list-style-type: none"> <li>• Si distrugga quando tocca il pianeta e quando colpisce un player.</li> <li>• Se quando colpisce un player esso perda una vita.</li> </ul> <p>Verifiche laser</p> <ul style="list-style-type: none"> <li>• Se il laser abbia una lunghezza massima.</li> <li>• Si distrugga quando tocca il pianeta e quando colpisce un player.</li> <li>• Se quando colpisce un player esso perda mezza vita.</li> </ul>		
<b>Risultati attesi:</b>	I laser e I missili soddisfano le loro specifiche.		

<b>Test Case:</b>	TC-005	<b>Nome:</b>	Collisioni
<b>Riferimento:</b>	REQ-04		
<b>Descrizione:</b>	Le navicelle se si scontrano o sbattono contro il pianeta esplodono automaticamente.		
<b>Procedura:</b>	<ol style="list-style-type: none"> <li>1. Iniziare a giocare</li> <li>2. Andare contro l'altra navicella e verificare che entrambe esplodano.</li> <li>3. Andare contro il pianeta e verificare che la navicella esploda.</li> </ol>		
<b>Risultati attesi:</b>	La navicella deve soddisfare le collisioni che ha nella sua specifica.		

<b>Test Case:</b>	TC-006	<b>Nome:</b>	Ostacoli opzionali
<b>Riferimento:</b>	REQ-05		
<b>Descrizione:</b>	Negli ostacoli posso scegliere se avere il pianeta al centro o la gravità.		
<b>Procedura:</b>	<ol style="list-style-type: none"> <li>1. Cliccare options</li> <li>2. Scegliere se usare una di queste opzioni</li> <li>3. Entrare nel gioco e verificare se ci sono</li> </ol>		
<b>Risultati attesi:</b>	Le opzioni se scelte dovrebbero apparire quando inizio a giocare.		

<b>Test Case:</b>	TC-007	<b>Nome:</b>	Poteri speciali
<b>Riferimento:</b>	REQ-06		
<b>Descrizione:</b>	Ogni giocatore se clicca un tasto dovrebbe poter attivare la sua specialità.		
<b>Procedura:</b>	Player 1 <ul style="list-style-type: none"> <li>Cliccare il tasto “s” e vedere se la navicella utilizza la sua specialità.</li> </ul> Player 2 <ul style="list-style-type: none"> <li>Cliccare il tasto “5” e vedere se la navicella utilizza la sua specialità.</li> </ul>		
<b>Risultati attesi:</b>	Quando clicco il relativo tasto la navicella deve attivare la sua specialità.		

<b>Test Case:</b>	TC-008	<b>Nome:</b>	Scelta navicella
<b>Riferimento:</b>	REQ-07		
<b>Descrizione:</b>	Nelle opzioni si può scegliere la navicella da usare		
<b>Procedura:</b>	1. Cliccare option 2. Cliccare spaceship 3. Selezionare la navicella che si desidera		
<b>Risultati attesi:</b>	Quando inizio a giocare dovrei utilizzare la navicella che ho scelto.		

<b>Test Case:</b>	TC-009	<b>Nome:</b>	Exit
<b>Riferimento:</b>	REQ-08		
<b>Descrizione:</b>	Cliccando exit chiudo il gioco.		
<b>Procedura:</b>	1. Cliccare exit		
<b>Risultati attesi:</b>	Il gioco dovrebbe chiudersi.		

## 5.2 Risultati test

Test Case	Funzionamento	Commento	Data
TC-001	NON OK	Se seleziono la modalità single player il player 2 non si muove da solo.  La modalità dual player funziona correttamente.	09.12.2022
TC-002	OK	Le navicelle si muovono correttamente.	09.12.2022
TC-003	OK	Le navicelle sparano sia laser che missili.	09.12.2022
TC-004	OK	I laser e i missili si distruggono quando colpiscono un player, e se il giocatore viene colpito da un missile perde una vita, altrimenti mezza vita.	09.12.2022
TC-005	OK	Le collisioni tra le ship, il pianeta, i laser e i missili funzionano correttamente.	09.12.2022
TC-006	NON OK	Pianeta funzionante.  Gravità non funzionante.	09.12.2022
TC-007	OK	Specialità funzionanti.	09.12.2022
TC-008	OK	Posso scegliere la ship che desidero prima di iniziare a giocare.	09.12.2022
TC-009	OK	Se clicco exit il gioco si chiude correttamente.	09.12.2022

Non è possibile mettere le immagini dei risultati non funzionanti, siccome sono cose non visibili.

## 5.3 Mancanze/limitazioni conosciute

Gli elementi mancanti che non sono stati implementati in questo progetto sono due: il bot e la gravità. La prima mancanza è dovuta al fatto che non sono riuscito a trovare nessuna informazione utile su internet per poter implementare il bot. Per la seconda invece, sono riuscito a trovare varie soluzioni su internet, ma purtroppo non riuscendo a capire il motivo non sono funzionante

## 6 Consuntivo

Rispetto alla pianificazione preventiva, nel Gantt consuntivo sono cambiati diversi tempi di diverse attività, questo era dovuto dal fatto che sono stati calcolati male alcuni tempi e l'importanza di alcune attività. Nonostante ciò, sono comunque riuscito a stare nei tempi e a finire il progetto.

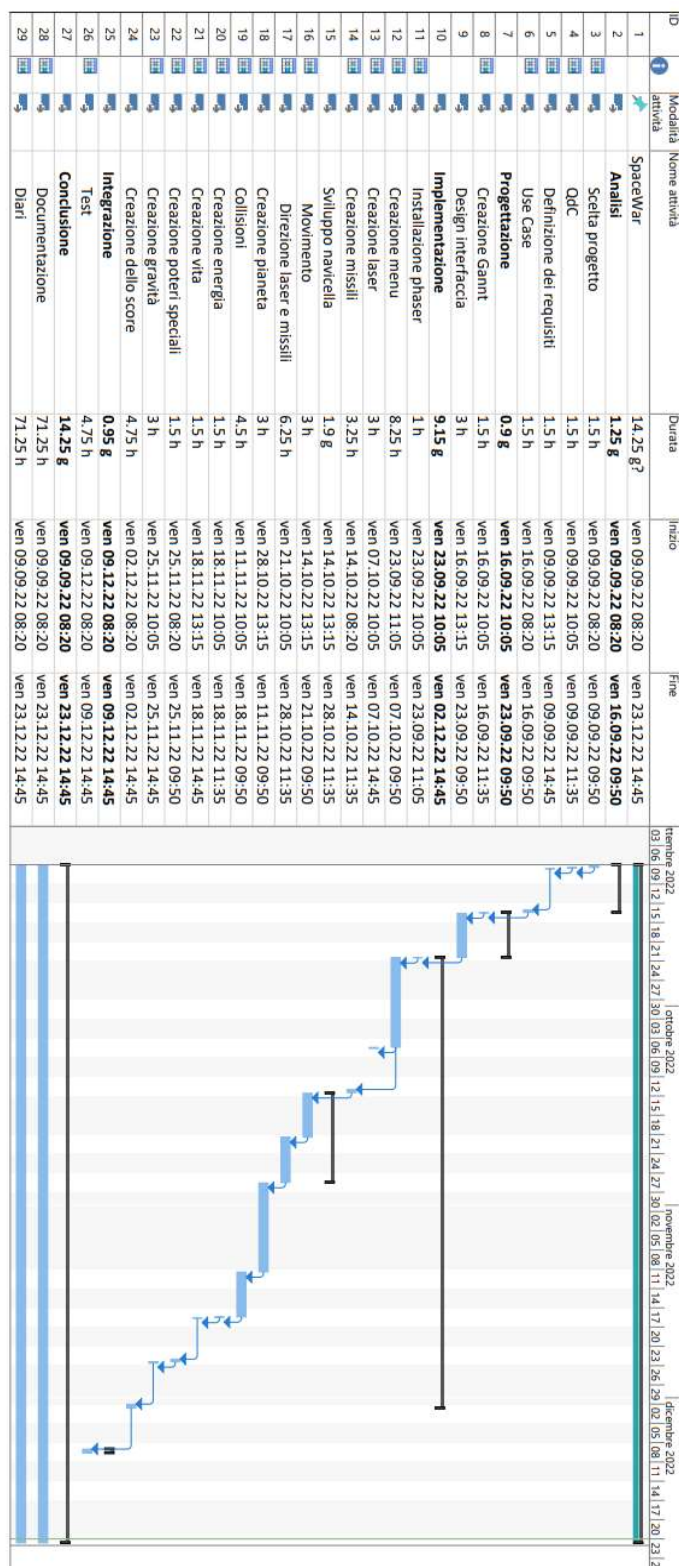


Figura 32: Gantt consuntivo



## 7 Conclusioni

### 7.1 Sviluppi futuri

Gli sviluppi futuri che ho pensato per questo gioco sono:

- L'aggiunta dello score finale non appena un player muore con l'eventuale classifica.
- Dare la possibilità all'utente di mettere in pausa.
- Aggiungere più navicelle da selezionare.
- Mettere la possibilità di scegliere il potere speciale che si desidera e anche i tasti da utilizzare.
- Si potrebbe creare in multiplayer.
- Aggiunta di oggetti volanti per complicare il gioco.
- Creare una classe proiettile che implementa altre sotto classi, come missile, laser,

### 7.2 Considerazioni personali

In conclusione, posso dire che il progetto mi ha insegnato moltissimo, sia a livello personale che a livello lavorativo. Sono contento di aver scelto questo progetto, grazie ad esso ho imparato molte cose nuove. Ho imparato a documentare un lavoro, creare i diari, creare Use Case, progettare diagrammi di gantt e ad utilizzare il framework phaser 3, utilizzato per creare i giochi che prima non conoscevo. Però ho anche avuto modo di mettere alla prova anche le mie abilità in cose che già conoscevo, come l'utilizzo della programmazione in javascript. Sono certo che andrò avanti ad implementare questo gioco, perché durante questo periodo mi sono divertito molto. A livello di tempistiche posso dire che sono soddisfatto del progetto, siccome sono riuscito a gestire al meglio tutto il tempo a mia disposizione.

## 8 Bibliografia

### 8.1 Sitografia

<https://phaser.io/phaser3>, *Sito di phaser*, consultato durante tutto il progetto.

<https://rexrainbow.github.io/phaser3-rex-notes/docs/site>, *Note of phaser*, consultato durante tutto il progetto.

<https://github.com/geo-petrini/phaserasteroids>, *Asteroids*, consultato durante tutto il progetto.

## 8.2 Indice delle figure

Figura 1: Use-Case.....	8
Figura 2: Gantt preventivo .....	9
Figura 3: Home page .....	11
Figura 4: Interfaccia bottone play .....	12
Figura 5: Interfaccia bottone options .....	12
Figura 6: Interfaccia bottone instructions.....	13
Figura 7: Codice file index.html .....	14
Figura 8: Importazione librerie phaser.....	14
Figura 9: Constructor menu .....	16
Figura 10: Preload menu .....	16
Figura 11: Create menu, aggiunta pulsanti .....	16
Figura 12: Azione pulsanti .....	17
Figura 13: Create OptionScene, utilizzo variabili globali .....	17
Figura 14: Update OptionScene .....	18
Figura 15: Create DualPlayerScene, chiavi tastiera.....	18
Figura 16: Overlap .....	19
Figura 17: CreateShip.....	19
Figura 18: Assegnazione missili e laser .....	20
Figura 19: Collisioni .....	20
Figura 20: RechargeEnergy .....	21
Figura 21: Update Hud .....	22
Figura 22: Laser function .....	23
Figura 23: Laser fire.....	24
Figura 24: Update laser .....	24
Figura 25: Codice planet.....	24
Figura 26: Update planet .....	25
Figura 27: Variabili ship .....	25
Figura 28: Assign laser e missili alla ship .....	25
Figura 29: Tasti W, A e D .....	26
Figura 30: Tasti S e Q.....	27
Figura 31: Gravità .....	27
Figura 32: Gantt consuntivo.....	32

## **9 Allegati**

---

- Mandato e/o QdC
- Prodotto
- Diari di lavoro
- Codici sorgente/documentazione
- Use-case
- Gui
- Gantt preventivo
- Gantt consultivo