Documentazione SPACEWAR

Titolo del progetto: SPACEWAR
Alunno/a: Alessandro Castelli
Classe: I3BB
Anno scolastico: 2022/2023
Docente responsabile: Geo Petrini

1 Introduzione

1.1 Informazioni sul progetto

Allievo: Alessandro Castelli
Docente coinvolto: Geo Petrini
Data inizio: 09.09.2022
Data fine 23.12.2022

1.2 Abstract

Questo progetto è una ricreazione del vecchio gioco Spacewar, questo gioco è una battaglia spaziale tra due navicelle che si devono sparare a vicenda e riuscire a distruggersi e in questa documentazione sono descritte le varie fasi per la realizzazione di questo progetto.

1.3 Scopo

Lo scopo del progetto è quello di ricreare un vecchio gioco chiamato appunto SpaceWar; quindi, creare un gioco ambientato nello spazio dove due navicelle si devono sparare tra di loro. Il gioco finisce quando una delle due navicelle esaurisce le vite disponibili. In questo gioco sono presenti la modalità con due giocatori, questi due giocatori muovono la navicella e sparano dalla stessa tastiera per rispettare il più possibile la versione originale del gioco, la modalità in singleplayer, la modalità con il pianeta al centro della schermata e l'attrazione gravitazionale verso il centro dello schermo.

Analisi

1.4 Analisi del dominio

In questo progetto dovrò ricreare il gioco SpaceWar e per farlo ho deciso di utilizzare python come linguaggio di programmazione con l'ausilio di pygame.

1.5 Analisi e specifica dei requisiti

Questi sono i requisiti che sono riuscito a raccogliere

ID: REQ-001	
Nome	Muovere la navicella con la tastiera
Priorità	1
Versione	1.0
Note	"WAD" per il player 1 e "IJL" per il player 2

ID: REQ-002	
Nome	Sparare con la tastiera
Priorità	1
Versione	1.0
Note	Player 1: laser "q" e missili "e" Player 2: laser "u" e missili "o"

ID: REQ-003	
Nome	Possibilità di giocare da solo o con un altro giocatore dalla stessa macchina
Priorità	1
Versione	1.0

ID: REQ-004	
Nome	Possibilità di scegliere se avere la gravità oppure no
Priorità	2
Versione	1.0

ID: REQ-005	
Nome	La navicella deve poter sparare sia missili che laser
Priorità	1
Versione	1.0

ID: REQ-006	
Nome	Laser con una determinata dimensione
Priorità	2
Versione	1.0

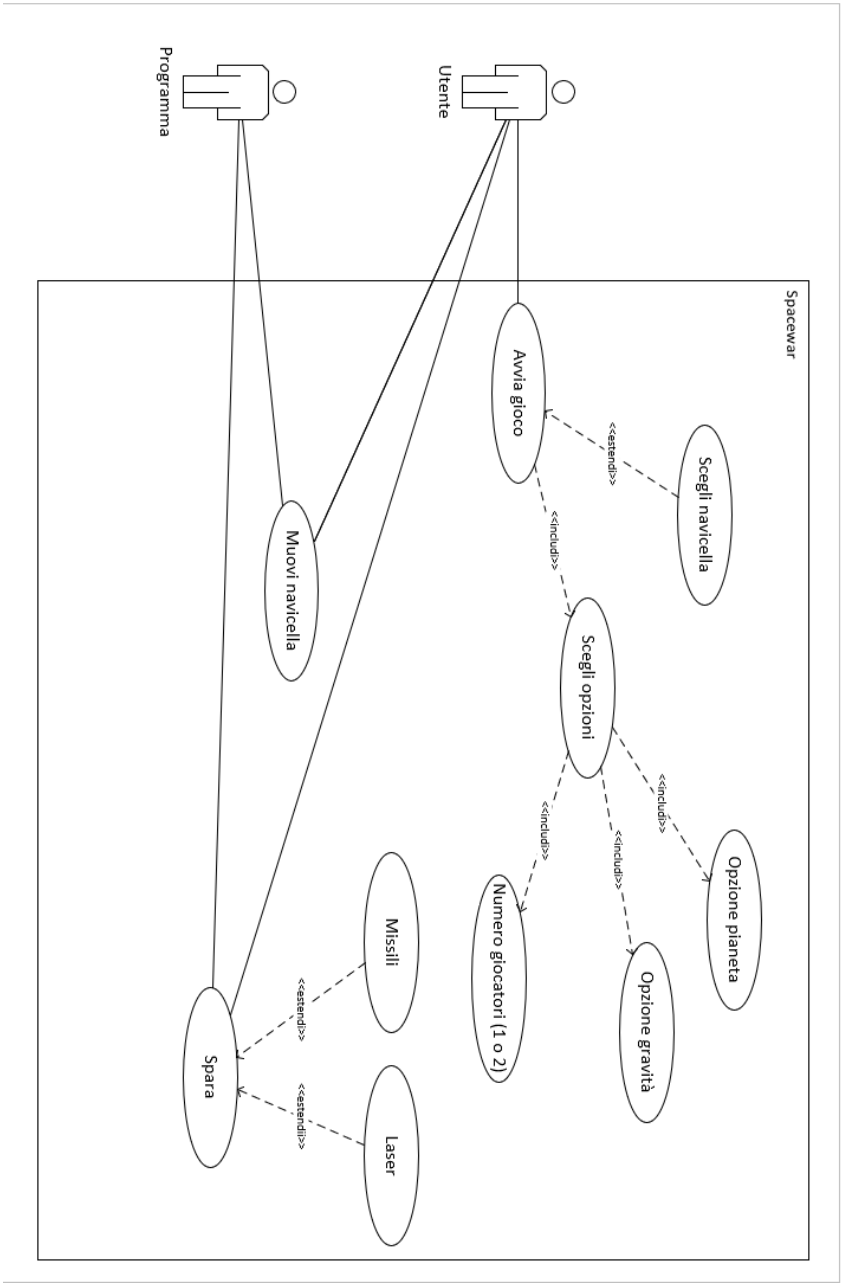
ID: REQ-007	
Nome	La navicella deve avere l'energia che si scarica sparando
Priorità	3
Versione	1.0

ID: REQ-008	
Nome	La navicella deve essere dotata di scudi (vite)
Priorità	2
Versione	1.0

ID: REQ-009	
Nome	Possibilità di scegliere se avere il pianeta oppure no
Priorità	2
Versione	1.0

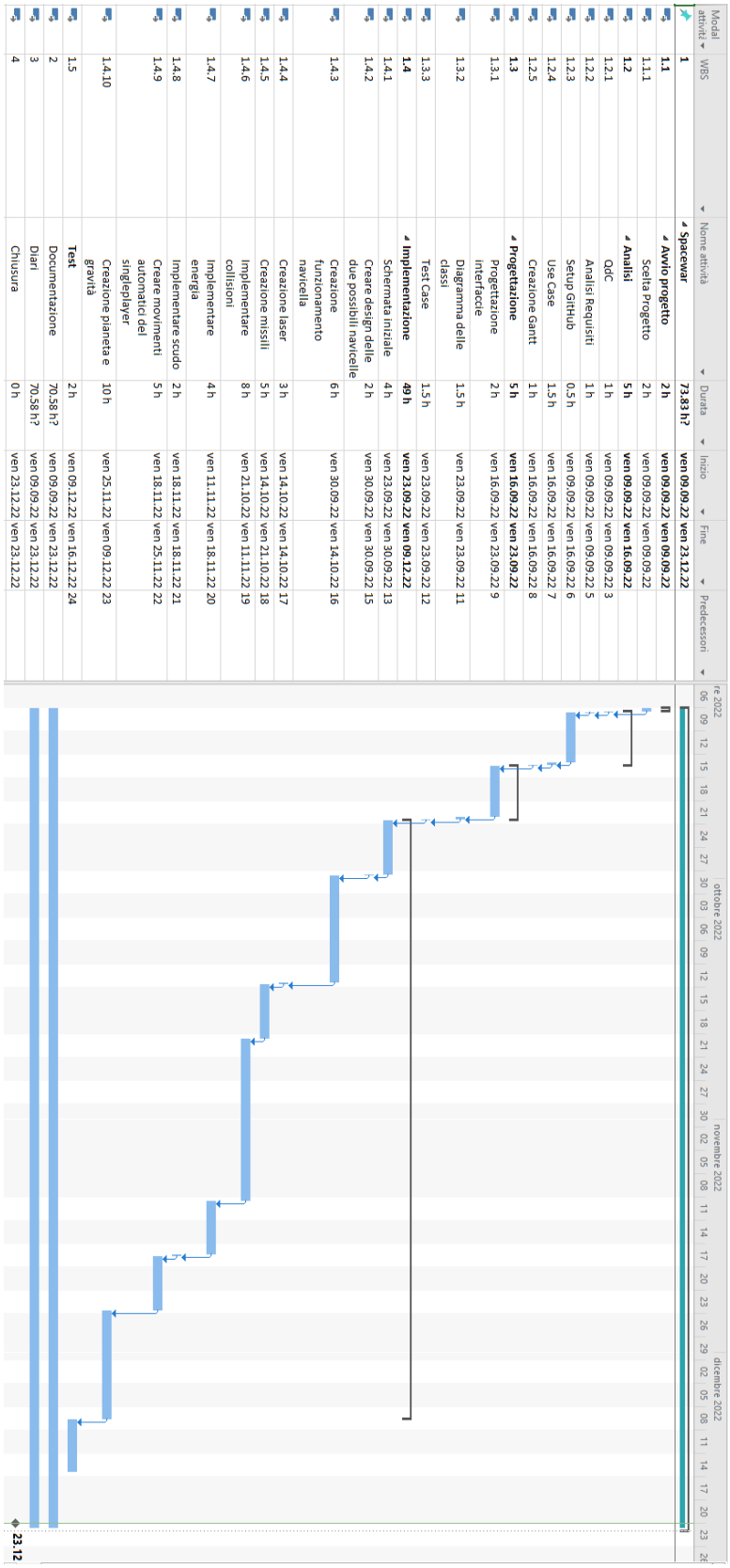
1.6 Use case

Questo è il mio use case:



1.7 Pianificazione

Questo è il gantt che ho realizzato, ho utilizzato un modello waterfall



Modal attività	WBS	Nome attività	Durata	Inizio	Fine	Predecessori
	1	**Spacewar**	**73.83 h?**	**ven 09.09.22**	**ven 23.12.22**	
	1.1	**Avvio progetto**	**2 h**	**ven 09.09.22**	**ven 09.09.22**	
	1.1.1	Scelta Progetto	2 h	ven 09.09.22	ven 09.09.22	
	1.2	**Analisi**	**5 h**	**ven 09.09.22**	**ven 16.09.22**	
	1.2.1	QdC	1 h	ven 09.09.22	ven 09.09.22	3
	1.2.2	Analisi Requisiti	1 h	ven 09.09.22	ven 09.09.22	5
	1.2.3	Setup GitHub	0.5 h	ven 09.09.22	ven 16.09.22	6
	1.2.4	Use Case	1.5 h	ven 16.09.22	ven 16.09.22	7
	1.2.5	Creazione Gantt	1 h	ven 16.09.22	ven 16.09.22	8
	1.3	**Progettazione**	**5 h**	**ven 16.09.22**	**ven 23.09.22**	
	1.3.1	Progettazione Interfaccie	2 h	ven 16.09.22	ven 23.09.22	9
	1.3.2	Diagramma delle classi	1.5 h	ven 23.09.22	ven 23.09.22	11
	1.3.3	Test Case	1.5 h	ven 23.09.22	ven 23.09.22	12
	1.4	**Implementazione**	**49 h**	**ven 23.09.22**	**ven 09.12.22**	
	1.4.1	Schermata iniziale	4 h	ven 23.09.22	ven 30.09.22	13
	1.4.2	Creare design delle due possibili navicelle	2 h	ven 30.09.22	ven 30.09.22	15
	1.4.3	Creazione funzionamento navicella	6 h	ven 30.09.22	ven 14.10.22	16
	1.4.4	Creazione laser	3 h	ven 14.10.22	ven 14.10.22	17
	1.4.5	Creazione missili	5 h	ven 14.10.22	ven 21.10.22	18
	1.4.6	Implementare collisioni	8 h	ven 21.10.22	ven 11.11.22	19
	1.4.7	Implementare energia	4 h	ven 11.11.22	ven 18.11.22	20
	1.4.8	Implementare scudo	2 h	ven 18.11.22	ven 18.11.22	21
	1.4.9	Creare movimenti automatici del singleplayer	5 h	ven 18.11.22	ven 25.11.22	22
	1.4.10	Creazione pianeta e gravità	10 h	ven 25.11.22	ven 09.12.22	23
	1.5	**Test**	2 h	ven 09.12.22	ven 16.12.22	24
	2	Documentazione	70.58 h?	ven 09.09.22	ven 23.12.22	
	3	Diari	70.58 h?	ven 09.09.22	ven 23.12.22	
	4	Chiusura	0 h	ven 23.12.22	ven 23.12.22	

1.8 Analisi dei mezzi

1.8.1 Software

- Python
- Pygame

1.8.2 Hardware

- PC scolastico
- Hard Disk personale
-

2 Progettazione

2.1 Design delle interfacce

Ho iniziato creando per prima cosa con i design delle interfacce che voglio realizzare utilizzando un sito che mi ha permesso di disegnarle, queste sono le interfacce:

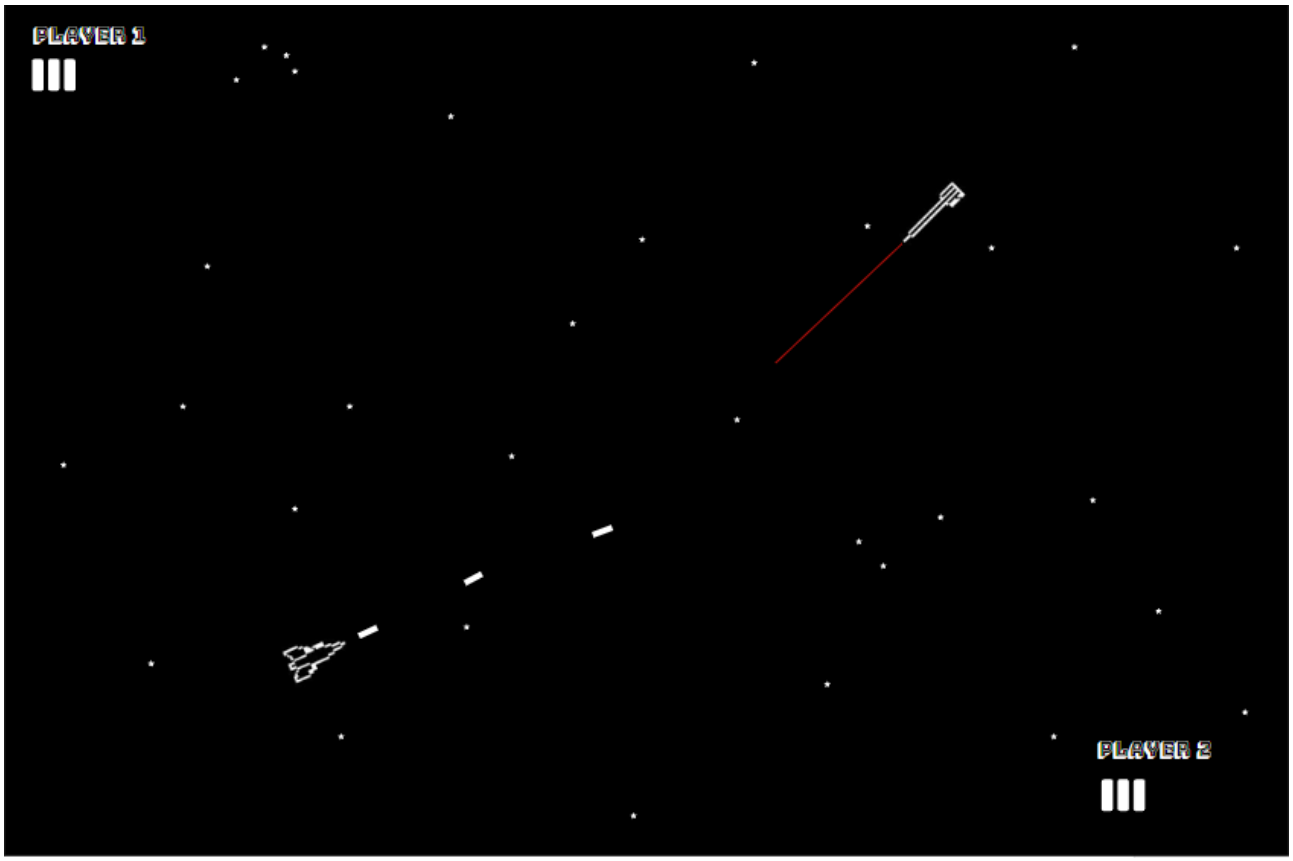
Schermata home:

Ho deciso di mettere le modalità di gioco cambiabili tramite i tasti della tastiera; quindi, in questa schermata appariranno le scritte con definito quale tasto bisogna premere per cambiare una determinata impostazione del gioco.

Schermata informazioni:

In questa schermata durante il design avevo pensato di mettere l'immagine delle due navicelle presenti in gioco ma durante l'implementazione ho deciso di scrivere solo i tasti che l'utente dovrà utilizzare per giocare.

Schermata di gioco:



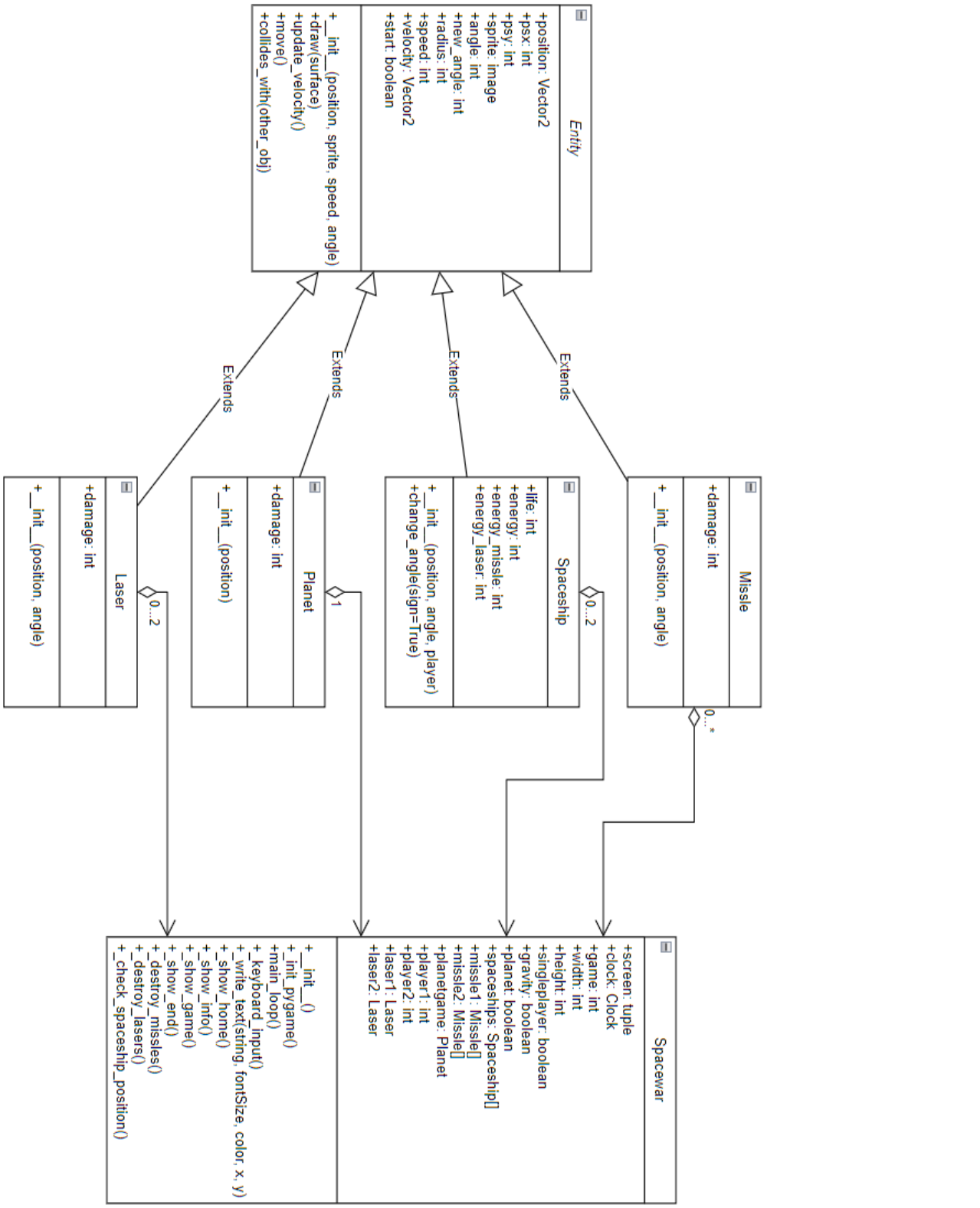
Questa è la schermata di gioco con presenti le vite dei giocatori. In questo design il laser sparato dalla navicella è un semplice laser dritto, nell'implementazione però ho optato per una sfera laser, più veloce e molto più grande dei proiettili che ha bisogno di arrivare all'energia massima per essere sparato.

Schermata finale:

In questa schermata viene semplicemente rappresentato quale giocatore ha vinto e il tasto da schiacciare per tornare alla schermata home.

2.2 Design procedurale

Questo è il diagramma delle classi che ho realizzato per il codice sorgente di questo progetto:



3 Implementazione

3.1 Methods.py

In questo file è presente un unico metodo:

```
def _get_sprite(name, with_alpha=True):
    logging.debug(f'loading sprite {name}')
    try:
        path = f"./sprites/{name}.png"
        loaded_sprite = load(path)
        if with_alpha:
            return loaded_sprite.convert_alpha()
        else:
            return loaded_sprite
    except Exception as e:
        logging.exception(f'error loading sprite {name}')
        return None
```

Questo metodo passando il nome del file come parametro ritorna l'immagine.

3.2 Classe Entity

Questa classe è la classe generica creata per realizzare le navicelle, i missili, i laser e il pianeta.
All'interno di questa classe ci sono tutti gli attributi necessari al funzionamento dei diversi oggetti presenti nel gioco.
Position: contiene un vettore con le coordinate x e y dell'oggetto.
Psx: contiene il valore della coordinata x dell'attributo position.
Psy: contiene il valore della coordinata y dell'attributo position.
Angle: contiene il valore dell'angolo.
New_angle: contiene il valore del nuovo angolo dopo la rotazione della navicella.
Radius: contiene il valore del raggio dell'oggetto.
Speed: contiene il valore della velocità dell'oggetto.
Velocity: contiene il vettore formato dalla velocità e dalla rotazione dell'angolo per poter muovere l'oggetto.
Start: contiene un valore boolean per controllare l'inizio del movimento dell'oggetto.

```
class Entity:
    def __init__(self, position, sprite, speed, angle):
        self.position = Vector2(position)
        self.psx, self.psy = self.position
        self.sprite = sprite
        self.angle = angle
        self.new_angle = angle
        self.radius = sprite.get_width() / 2
        self.speed = speed
        self.velocity = Vector2(self.speed, 0).rotate(-self.angle)
        self.start = False

    def draw(self, surface):
        self.psx, self.psy = self.position
        #rotazione immagine
        rotated_image = rotozoom(self.sprite, self.new_angle, 1.0)
        #coordinate immagine ruotata
        blit_position = Vector2(self.psx, self.psy) - (Vector2(self.radius))

        surface.blit(rotated_image, blit_position)

    def update_velocity(self):
        self.angle = self.new_angle
        self.velocity = Vector2(self.speed, 0).rotate(-self.angle)

    def move(self):
        if self.start:
            self.position += self.velocity

    def collides_with(self, other_obj):
        distance = self.position.distance_to(other_obj.position)
        return distance < self.radius + other_obj.radius
```

Il metodo **draw** permette la rotazione e il cambiamento della posizione dell'immagine, questo metodo utilizza il metodo rotozoom() che ruota l'immagine in base all'angolo passato come parametro.

Il metodo **update_velocity** che viene utilizzato quando la navicella fa uno sprint in una nuova direzione, quindi inserisce il valore del nuovo angolo (che veniva tenuto salvato per dare la direzione ai proiettili e ai laser) nell'attributo che contiene l'angolo e successivamente viene aggiornato anche il vettore velocity per far cambiare direzione all'oggetto.

Il metodo **move** aggiorna semplicemente l'attributo della posizione dell'oggetto sommandoli il vettore velocity.

Il metodo **collides_with** controlla la distanza tra i due oggetti e ritorna True se la distanza è minore dei due raggi degli oggetti controllati.

3.3 Classe Spaceship

La classe Spaceship estende la classe Entity quindi avrà i comportamenti di quella classe, ma oltre a questo contiene alcuni attributi in più:
life: contiene il numero di vite della navicella.
Energy: contiene l'energia attuale della navicella.
Energy_missle: contiene il valore del costo in energia di un missile.
Energy_laser: contiene il valore del costo in energia di un laser.

```
class Spaceship(Entity):
    def __init__(self, position, angle, player):
        logging.debug(f'init start player {player}')
        self.life = 3
        self.energy = 50
        self.energy_missle = 100
        self.energy_laser = 400
        super().__init__(
            position, _get_sprite("player" + str(player)), 3, angle
        )

        logging.debug(f'init done player {player}')

    def change_angle(self, sign=True):
        direction = 1 if sign else -1
        self.new_angle += 4 * direction
```

Il metodo **change_angle** che modifica il valore dell'attributo new_angle di 4 gradi in positivo o in negativo a dipendenza del valore sign passato come parametro.

3.4 Classe Missle

Questa classe estende la classe Entity e ha i comportamenti di essa. Contiene l'attributo **damage** che definisce i danni del missile.

```
class Missle(Entity):
    def __init__(self, position, angle):
        self.damage = 1
        super().__init__(
            position, _get_sprite("bullet"), 8, angle
        )
```

3.5 Classe Laser

Questa classe estende la classe Entity e ha i comportamenti di essa. Contiene l'attributo **damage** che definisce i danni del laser come nella classe Missile.

```
class Laser(Entity):
    def __init__(self, position, angle):
        self.damage = 2
        super().__init__(
            position, _get_sprite("laser"), 11, angle
        )
```

3.6 Classe Planet

Questa classe estende la classe Entity e ha i comportamenti di essa. Contiene l'attributo **damage** che definisce i danni del pianeta che in questo caso sono come la quantità di vita di una navicella e quindi la distruggerà istantaneamente quando collidono.

```
class Planet(Entity):
    def __init__(self, position):
        self.damage = 3
        super().__init__(
            position, _get_sprite("planet"), 0, 0
        )
```

3.7 Classe Spacewar

Questa è la classe principale del gioco, in questa classe vengono gestiti gli input da tastiera, la generazione dei diversi oggetti in gioco e gestisce quale interfaccia mostrare in base allo stato del gioco.
Gli attributi presenti in questa classe sono i seguenti:
screen: contiene le dimensioni dello schermo.
Game: contiene il valore dello stato del gioco, in base a questo valore viene scelto quale interfaccia mostrare.
Width: contiene il valore della larghezza dello schermo.
Height: contiene il valore dell'altezza dello schermo.
Singleplayer: definisce se il gioco deve essere avviato in modalità singleplayer oppure no.
Gravity: definisce se il gioco deve contenere la gravità oppure no.
Planet: definisce se il gioco deve contenere il pianeta oppure no.
Spaceships: è un array di Spaceship che contiene le navicelle.
Missle1: contiene i missili che vengono sparati dalla prima navicella.
Missle2: contiene i missili che vengono sparati dalla seconda navicella.
Planetgame: contiene il pianeta.
Player1: è una variabile che permette di selezionare quale navicella appartiene al player 1.
Player2: è una variabile che permette di selezionare quale navicella appartiene al player 2.
Laser1: contiene il laser che viene sparato dal player 1.
Laser2: contiene il laser che viene sparato dal player 2.

3.7.1 Main_loop

Questo metodo fa partire un loop infinito del codice, ogni ciclo verrà chiamato il metodo che legge gli input della tastiera e controlla il valore dell'attributo game per vedere quale interfaccia mostrare all'utente.

```
def main_loop(self):
    while True:
        self.clock.tick(60)
        self._keyboard_input()
        if self.game == 0:
            self._show_home()
        if self.game == 1:
            self._show_info()
        if self.game == 2:
            self._show_game()
        if self.game == 3:
            self._show_end()
```

3.7.2 _keyboard_input

```
def _keyboard_input(self):
    for event in pygame.event.get():
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE) or event.type == pygame.QUIT:
            quit()
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_m) and self.game == 0:
            self.singleplayer = not self.singleplayer
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_g) and self.game == 0:
            self.gravity = not self.gravity
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_p) and self.game == 0:
            self.planet = not self.planet
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_s) and self.game == 0:
            if self.player1 == 1:
                self.player1 = 2
                self.player2 = 1
            else:
                self.player1 = 1
                self.player2 = 2
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_i) and (self.game == 0 or self.game == 1):
            if self.game == 0:
                self.game = 1
            else:
                self.game = 0
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_RETURN) and self.game == 0:


            ship1 = Spaceship((self.width*1/4, self.height/2), 0, self.player1)
            self.spaceships.append(ship1)

            ship2 = Spaceship((self.width*3/4, self.height/2), 180, self.player2)
            self.spaceships.append(ship2)

            if self.planet:
                self.planetgame = Planet((self.width/2, self.height/2))

            self.game = 2
```

Questo pezzo di codice fa i controlli dell'input della tastiera della schermata home e info cambiando i valori delle diverse modalità di gioco e creando le due navicelle e il pianeta quando viene premuto ENTER.

```
keys = pygame.key.get_pressed()
if (event.type == pygame.KEYDOWN and event.key == pygame.K_d) and self.game == 2:
    self.spaceships[0].change_angle(False)
if (keys[pygame.K_d]) and self.game == 2:
    self.spaceships[0].change_angle(False)
if (event.type == pygame.KEYDOWN and event.key == pygame.K_a) and self.game == 2:
    self.spaceships[0].change_angle(True)
if (keys[pygame.K_a]) and self.game == 2:
    self.spaceships[0].change_angle(True)

if (event.type == pygame.KEYDOWN and event.key == pygame.K_l) and self.game == 2 and self.singleplayer == False:
    self.spaceships[1].change_angle(False)
if (keys[pygame.K_l]) and self.game == 2 and self.singleplayer == False:
    self.spaceships[1].change_angle(False)
if (event.type == pygame.KEYDOWN and event.key == pygame.K_j) and self.game == 2 and self.singleplayer == False:
    self.spaceships[1].change_angle(True)
if (keys[pygame.K_j]) and self.game == 2 and self.singleplayer == False:
    self.spaceships[1].change_angle(True)

if (event.type == pygame.KEYDOWN and event.key == pygame.K_w) and self.game == 2:
    self.spaceships[0].start = True
    self.spaceships[0].update_velocity()

if (event.type == pygame.KEYDOWN and event.key == pygame.K_i) and self.game == 2 and self.singleplayer == False:
    self.spaceships[1].start = True
    self.spaceships[1].update_velocity()
```

Questa parte del metodo invece si occupa del movimento e della rotazione della navicella, la seconda navicella potrà essere mossa quando non si avvierà il gioco in singleplayer. Per ruotare la navicella verrà richiamato il metodo change_angle, mentre per far cambiare direzione allo spostamento della navicella verrà richiamato il metodo update_velocity.

```
if (event.type == pygame.KEYDOWN and event.key == pygame.K_e) and self.game == 2:
    if self.spaceships[0].energy > self.spaceships[0].energy_missle:
        missle = Missle(self.spaceships[0].position, self.spaceships[0].new_angle)
        self.missles1.append(missle)
        self.spaceships[0].energy = self.spaceships[0].energy - self.spaceships[0].energy_missle

if (event.type == pygame.KEYDOWN and event.key == pygame.K_o) and self.game == 2 and self.singleplayer == False:
    if self.spaceships[1].energy > self.spaceships[1].energy_missle:
        missle = Missle(self.spaceships[1].position, self.spaceships[1].new_angle)
        self.missles2.append(missle)
        self.spaceships[1].energy = self.spaceships[1].energy - self.spaceships[1].energy_missle

if (event.type == pygame.KEYDOWN and event.key == pygame.K_q) and self.game == 2:
    if self.spaceships[0].energy >= self.spaceships[0].energy_laser:
        self.laser1 = Laser(self.spaceships[0].position, self.spaceships[0].new_angle)
        self.spaceships[0].energy = self.spaceships[0].energy - self.spaceships[0].energy_laser

if (event.type == pygame.KEYDOWN and event.key == pygame.K_u) and self.game == 2 and self.singleplayer == False:
    if self.spaceships[1].energy >= self.spaceships[1].energy_laser:
        self.laser2 = Laser(self.spaceships[1].position, self.spaceships[1].new_angle)
        self.spaceships[1].energy = self.spaceships[1].energy - self.spaceships[1].energy_laser

if (event.type == pygame.KEYDOWN and event.key == pygame.K_h) and self.game == 3:
    self.spaceships = []
    self.missles1 = []
    self.missles2 = []
    self.planetgame = None
    self.laser1 = None
    self.laser2 = None
    self.game = 0
```

Questa ultima parte del metodo invece si occupa di quello che è lo sparare, verranno creati i laser o i missili in base al tasto premuto e verrà sottratta l'energia in base all'oggetto creato.
L'ultimo controllo si occupa della schermata di fine del gioco in cui se verrà premuta la "h" verranno svuotati tutti gli array e messo lo stato del gioco a 0 per tornare alla schermata home.

3.7.3 _write_text

Questo metodo si occupa di creare il testo, impostarli il colore, la posizione,la grandezza e il font grazie ai valori passati come parametro.

```
def _write_text(self, string, fontSize, color, x, y):
    font = pygame.font.Font('./font/BungeeShade-Regular.ttf', fontSize)
    text = font.render(string, True, color)
    rect = text.get_rect()
    rect.center = (x, y)
    self.screen.blit(text, rect)
```

3.7.4 _show_home

Questo metodo si occupa di rappresentare a schermo le scritte necessarie per rappresentare la schermata home, per ogni scritta verrà utilizzato il metodo _write_text.

```
def _show_home(self):
    self.screen.fill((0,0,0))
    self.write = self._write_text("SPACEWAR", 90, (255,255,255), self.width / 2, self.height / 3.5)
    self.write = self._write_text("Press enter to play", 30, (255,255,255), self.width / 5, self.height / 2)

    self.write = self._write_text("Press M to change mode", 30, (255,255,255), self.width / 3 * 2.5, self.height / 2)
    self.write = self._write_text("singleplayer: " + str(self.singleplayer), 30, (255,255,255), self.width / 3 * 2.5, self.height / 2 + 40)

    self.write = self._write_text("Press s to change spaceship", 30, (255,255,255), self.width / 5, self.height / 2 + 200)

    self.write = self._write_text("Press g to set gravity", 30, (255,255,255), self.width / 3 * 2.5, self.height / 2 + 200)
    self.write = self._write_text("gravity: " + str(self.gravity), 30, (255,255,255), self.width / 3 * 2.5, self.height / 2 + 240)

    self.write = self._write_text("Press p for the planet", 30, (255,255,255), self.width / 5, self.height / 2 + 360)
    self.write = self._write_text("planet: " + str(self.planet), 30, (255,255,255), self.width / 5, self.height / 2 + 400)

    self.write = self._write_text("Press i for info", 30, (255,255,255), self.width / 3 * 2.5, self.height / 2 + 400)

    pygame.display.flip()
```

3.7.5 _show_info

Questo metodo si occupa di rappresentare a schermo le scritte necessarie per rappresentare la schermata delle informazioni, per ogni scritta verrà utilizzato il metodo _write_text.

```
def _show_info(self):
    self.screen.fill((0,0,0))
    self.write = self._write_text("Information", 90, (255,255,255), self.width / 2, self.height / 3.5)

    self.write = self._write_text("player 1: movement with wad, missle with e and laser with q", 30, (255,255,255), self.width / 2, self.height / 2)
    self.write = self._write_text("player 2: movement with ijl, missle with o and laser with u", 30, (255,255,255), self.width / 2, self.height / 2 + 200)

    self.write = self._write_text("Press i to back home", 30, (255,255,255), self.width / 3 * 2.5, self.height / 2 + 400)
    pygame.display.flip()
```

3.7.6 _show_game

Questo metodo rappresenta la schermata home e si occupa di richiamare i metodi che spostano e disegnano i vari oggetti del gioco nella schemata.

```
def _show_game(self):
    self.screen.fill((0,0,0))
    self.write = self._write_text("Player 1", 30, (255,255,255), 130, self.height * 1/15)
    self.write = self._write_text("Life: " + str(self.spaceships[0].life), 25, (255,255,255), 130, self.height * 1/9)
    self.write = self._write_text("Player 2", 30, (255,255,255), self.width - 130, self.height * 1/15)
    self.write = self._write_text("Life: " + str(self.spaceships[1].life), 25, (255,255,255), self.width - 130, self.height * 1/9)
    try:
        for (i, spaceship) in enumerate(self.spaceships):
            spaceship.draw(self.screen)
        for (i, missle) in enumerate(self.missles1):
            missle.draw(self.screen)
        for (i, missle) in enumerate(self.missles2):
            missle.draw(self.screen)

        if self.planet:
            self.planetgame.draw(self.screen)


        for missle in self.missles2:
            missle.start = True
            missle.move()
            if missle.collides_with(self.spaceships[0]):
                if self.spaceships[0].life > 1:
                    self.spaceships[0].life = self.spaceships[0].life - missle.damage
                    self.missles2.remove(missle)
                    del missle
                else:
                    self.spaceships[0].life = 0
                    self.game = 3
            if self.planet and missle.collides_with(self.planetgame):
                self.missles2.remove(missle)
                del missle

        for missle in self.missles1:
            missle.start = True
            missle.move()
            if missle.collides_with(self.spaceships[1]):
                if self.spaceships[1].life > 1:
                    self.spaceships[1].life = self.spaceships[1].life - missle.damage
                    self.missles1.remove(missle)
                    del missle
                else:
                    self.spaceships[1].life = 0
                    self.game = 3
            if self.planet and missle.collides_with(self.planetgame):
                self.missles1.remove(missle)
                del missle
```

Questa prima parte del metodo si occupa di mostrare come prima cosa le vite dei due player, successivamente si occupa di mostrare a schermo le navicelle, i missili e il pianeta tramite il metodo draw. I due grossi cicli invece si occupano di aggiornare la posizione dei missili e di controllare le collisioni sia con la navicella che con il pianeta. Ad ogni collisione i missili verranno rimossi e se la collisione è avvenuta con la navicella verrà ridotta la sua vita.

```
    for spaceship in self.spaceships:
        spaceship.move()
        if spaceship.energy < 401:
            spaceship.energy = spaceship.energy + 1
        if self.planet and spaceship.collides_with(self.planetgame):
            spaceship.life = spaceship.life  - self.planetgame.damage
            self.game = 3

    if self.spaceships[0].energy >= 400:
        self.write = self._write_text("Laser ball available", 20, (255,0,0), 160, self.height * 1/6)

    if self.spaceships[1].energy >= 400:
        self.write = self._write_text("Laser ball available", 20, (255,0,0), self.width - 160, self.height * 1/6)

    if self.laser1 != None:
        self.laser1.start = True
        self.laser1.move()
        self.laser1.draw(self.screen)
        if self.laser1.collides_with(self.spaceships[1]):
            if self.spaceships[1].life > 2:
                self.spaceships[1].life = self.spaceships[1].life - self.laser1.damage
                self.laser1 = None
            else:
                self.spaceships[1].life = 0
                self.game = 3
        if self.planet and self.laser1.collides_with(self.planetgame):
            self.laser1 = None

    if self.laser2 != None:
        self.laser2.start = True
        self.laser2.move()
        self.laser2.draw(self.screen)
        if self.laser2.collides_with(self.spaceships[0]):
            if self.spaceships[0].life > 2:
                self.spaceships[0].life = self.spaceships[0].life - self.laser2.damage
                self.laser2 = None
            else:
                self.spaceships[0].life = 0
                self.game = 3
        if self.planet and self.laser2.collides_with(self.planetgame):
            self.laser2 = None

except Exception as e:
    logging.exception('error loading ships')
```

In questa parte del metodo invece viene aumentata l'energia della navicella e controllate le collisioni con il pianeta.
Successivamente viene fatto un controllo per vedere se l'energia ha raggiunto il valore di 400 per poter attivare la possibilità di sparare i laser.
Come ultima cosa vengono fatti muovere e stampare i laser sull'interfaccia di gioco e inoltre vengono controllate le loro collisioni con le navicelle e con il pianeta.

```
self._destroy_missles()

self._destroy_lasers()

self.check_spaceship_position()

pygame.display.flip()
```

Come ultima cosa in questo metodo vengono richiamati i vari metodi per controllare gli oggetti che escono dalla schermata di gioco.

3.7.7 _show_end

Questo metodo mostra semplicemente la schermata di fine del gioco con mostrato il player che ha vinto la partita e il tasto da premere per tornare alla home.

```
def _show_end(self):
    self.screen.fill((0,0,0))
    self.write = self._write_text("GAME OVER", 90, (255,255,255), self.width / 2, self.height / 3.5)
    if self.spaceships[0].life > 0 and self.spaceships[1].life < 1:
        self.write = self._write_text("Player 1 Win", 90, (255,255,255), self.width / 2, self.height / 2)
    else:
        self.write = self._write_text("Player 2 Win", 90, (255,255,255), self.width / 2, self.height / 2)

    self.write = self._write_text("Press h to home", 30, (255,255,255), self.width / 3 * 2.5, self.height / 2 + 400)

    pygame.display.flip()
```

3.7.8 _destroy_missles

Questo metodo si occupa di controllare la posizione dei missili e se finiscono fuori dalla schermata di gioco verranno distrutti.

```
def _destroy_missles(self):
    for missle in self.missles1:
        if (missle.psx > self.width + 60 or missle.psx < 0 - 60) or (missle.psy > self.height + 60 or missle.psy < 0 - 60):
            self.missles1.remove(missle)
            del missle
    for missle in self.missles2:
        if (missle.psx > self.width + 60 or missle.psx < 0 - 60) or (missle.psy > self.height + 60 or missle.psy < 0 - 60):
            self.missles2.remove(missle)
            del missle
```

3.7.9 _destroy_lasers

Come per il metodo della distruzione dei missili questo metodo si occupa di distruggere i laser che escono dalla schermata di gioco

```
def _destroy_lasers(self):
    if self.laser1 != None:
        if (self.laser1.psx > self.width + 100 or self.laser1.psx < 0 - 100) or (self.laser1.psy > self.height + 100 or self.laser1.psy < 0 - 100):
            self.laser1 = None
    if self.laser2 != None:
        if (self.laser2.psx > self.width + 100 or self.laser2.psx < 0 - 100) or (self.laser2.psy > self.height + 100 or self.laser2.psy < 0 - 100):
            self.laser2 = None
```

3.7.10 _check_spaceship_position

Questo metodo si occupa di controllare le posizioni in x e in y della navicella per farla riapparire dalla parte opposta della schermata quando escono dalla visuale così da evitare che le navicelle vaghino nel nulla ma restino sull'interfaccia di gioco.

```
def check_spaceship_position(self):
    for spaceship in self.spaceships:
        if spaceship.psx > self.width + 30:
            spaceship.position = (0 - 30, spaceship.psy)
        if spaceship.psx < 0 - 30:
            spaceship.position = (self.width + 30, spaceship.psy)
        if spaceship.psy > self.height + 30:
            spaceship.position = (spaceship.psx, 0 - 30)
        if spaceship.psy < 0 - 30:
            spaceship.position = (spaceship.psx, self.height + 30)
```

4 Test

4.1 Protocollo di test

Test Case: **Riferimento**:	TC-001 REQ-001 REQ-002	**Nome:**	Test tastiera
Descrizione:	I 2 giocatori devono poter muovere e sparare tramite la tastiera		
Procedura:	Avviare il gioco e controllare che i 2 giocatori possono muoversi e sparare con I tasti definiti nei Requisiti 001 e 002		
Risultati attesi:	I tasti permettono di muovere la navicella e sparare correttamente		

Test Case: **Riferimento**:	TC-002 REQ-003	**Nome:**	Modalità singleplayer o 2 player
Descrizione:	Gioco funzionante sia in single player che con 2 giocatori		
Procedura:	Avviare il gioco in entrambe le modalità e testare se il secondo player funziona autonomamente in single player e se con 2 giocatori la seconda navicella funziona con i comandi.		
Risultati attesi:	In sigleplayer la seconda navicella si muove e spara da sola, mentre con 2 giocatori la seconda navicella si muove e spara con l'utilizzo della tastiera		

Test Case: **Riferimento**:	TC-003 REQ-005	**Nome:**	Sparare missile e laser
Descrizione:	La navicella deve poter sparare missili o laser		
Procedura:	Avviare il gioco e testare che con un tasto la navicella spara dei missili e con l'altro invece un laser		
Risultati attesi:	La navicella può sparare si laser che missili		

Test Case: **Riferimento**:	TC-004 REQ-004	**Nome:**	Test gravità
Descrizione:	Il gioco deve avere la modalità con gravità e quella senza		
Procedura:	Avviare il gioco con la gravità attivata e vedere se la navicella e I missile vengono attratti dal pianeta al centro della schermata di gioco		
Risultati attesi:	La naviccella e I missile sono soggetti alla gravità del pianeta		

Test Case:	TC-005	**Nome:**	Laser con determinata dimensione
Riferimento:	REQ-006		
Descrizione:	Il laser sparato dalla navicella deve avere una determinate dimensione		
Procedura:	Provare a sparare il laser con la navicella e controllare che il laser abbia una dimensione ne troppo grande, ne troppo piccola		
Risultati attesi:	Il laser ha una dimensione accettabile (ne troppo grande, ne toppo piccolo)		

Test Case:	TC-006	**Nome:**	Energia navicella
Riferimento:	REQ-007		
Descrizione:	La navicella ha un limite di energia che si scarica sparando		
Procedura:	Provare a sparare con la navicella e controllare che non possa sparare troppi colpi di fila e aspettare che l'energia si ricarichi per poter sparare di nuovo		
Risultati attesi:	La navicella non può sparare troppi colpi di fila e l'energia si ricarica quando non spari		

Test Case:	TC-007	**Nome:**	Scudi navicella
Riferimento:	REQ-008		
Descrizione:	La navicella non esplode quando viene colpita ma ha degli scudi		
Procedura:	Sparare ad una navicella e vedere se quando viene colpita scende lo scudo		
Risultati attesi:	Quando viene colpita la navicella scende lo scudo		

Test Case:	TC-008	**Nome:**	Possibilità di avere il pianeta
Riferimento:	REQ-009		
Descrizione:	Al centro della schermata di gioco deve esserci la possibilità di avere un pianeta che distruggerà le navicelle che ci vanno contro		
Procedura:	Avviare il gioco con il pianeta attivo e controllare che la navicella si distrugga quando ci va contro		
Risultati attesi:	Il pianeta è presente al centro della schermata		

4.2 Risultati test

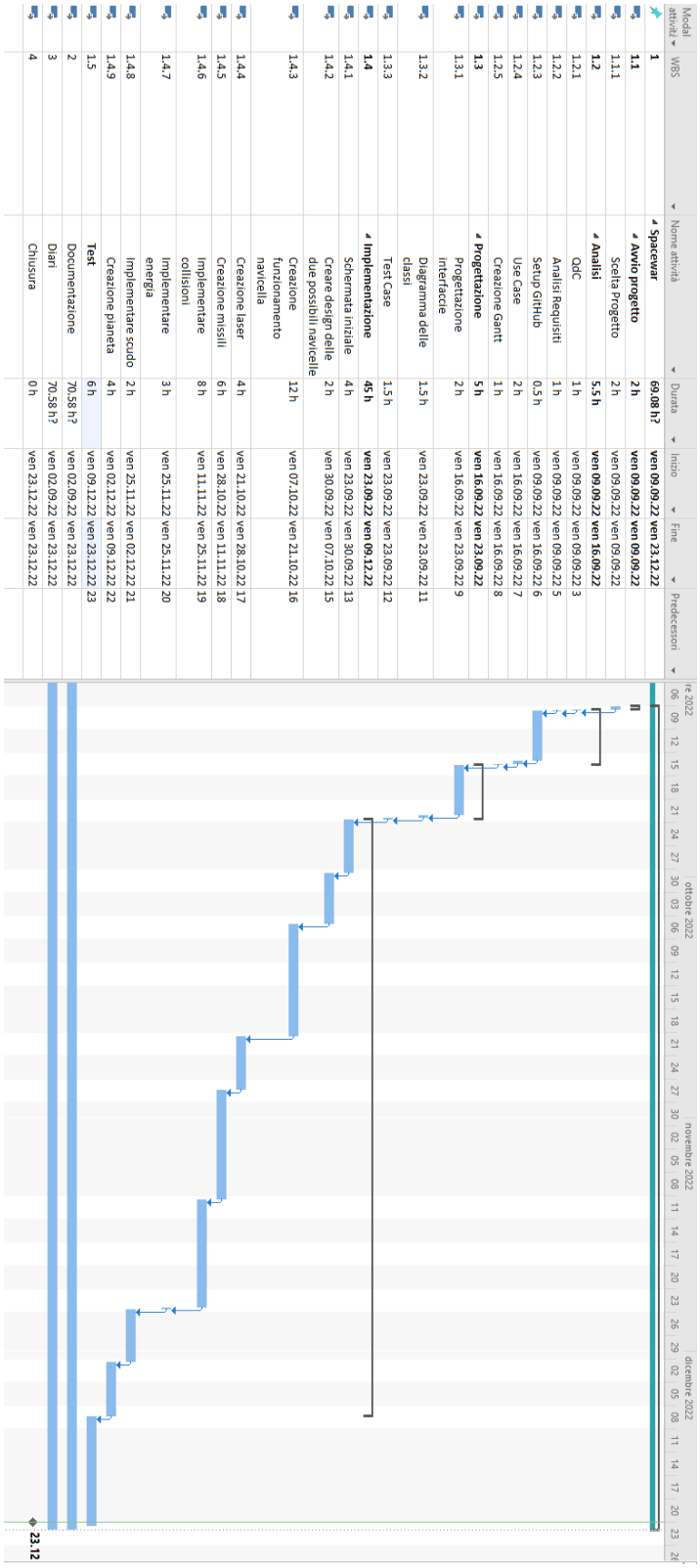
Test	Risultato
TC-001	Passato
TC-002	Fallito
TC-003	Passato
TC-004	Fallito
TC-005	Passato
TC-006	Passato
TC-007	Passato
TC-008	Passato

4.3 Mancanze/limitazioni conosciute

Purtroppo, non sono riuscito ad implementare la modalità in singleplayer e la gravità.
Avendo perso 12 ore di lezione e avendo riscontrato qualche problema nel realizzare altre cose non sono riuscito ad implementare queste due funzionalità.
Per la gravità mi ero preso del tempo per fare ricerche su come farla ma non ho trovato nulla di veramente utile per il mio caso e essendomi rimasto troppo poco alla fine del progetto non sono riuscito ad implementarla.
Anche per la modalità in singleplayer per questioni di tempo non sono riuscito ad implementarla.

5 Consuntivo

Questo è il mio gantt consuntivo:



6 Conclusioni

Non mi aspettavo di bloccarmi su alcune cose come invece è successo e quindi non sono riuscito a completare tutte le funzionalità del gioco, purtroppo avendo perso tempo su degli errori e avendo perso due lezioni non sono riuscito a concludere tutto. Nonostante questo sono contento di aver lavorato ad un videogioco che è molto interessante sia per la programmazione ad oggetti che per il linguaggio che ho utilizzato per realizzarlo. Nonostante le funzionalità non implementate il gioco funzione abbastanza bene quindi mi ritengo in parte soddisfatto.

6.1 Sviluppi futuri

Sicuramente bisognerebbe implementare il singleplayer e la gravità che non sono riuscito a sviluppare e come sviluppi futuri ci sarebbe di interessante il multigiocatore online e aggiungere modalità di gioco con difficoltà in più.

6.2 Considerazioni personali

Grazie a questo progetto ho migliorato le mie capacità nella programmazione ad oggetti e all'utilizzo del linguaggio di programmazione python.

7 Bibliografia

7.1 Sitografia

https://www.pixilart.com/draw

https://app.moqups.com/

https://app.diagrams.net/

https://deep-fold.itch.io/pixel-planet-generator

8 Allegati

Elenco degli allegati, esempio:

- Diari di lavoro
- Codici sorgente/documentazione macchine virtuali
- Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
- Documentazione di prodotti di terzi
- Eventuali guide utente / Manuali di utilizzo
- Mandato e/o QdC
- Prodotto