Relazione “AstroParty”

Dario Berti

Alessandro Coli

Mirco Terenzi

10 Aprile 2023

Indice

Capitolo 1: Analisi

1.1 Requisiti

Il gruppo si pone l'obbiettivo di creare un multidirectional shooter multiplayer game ambientato nello spazio. L'obiettivo di questo gioco è essere l'ultima astronave in gioco, un giocatore viene eliminato nel momento in cui viene colpito da un proiettile o finisce contro un ostacolo mortale (es. laser). I giocatori potranno scegliere prima dell'inizio del gioco alcune impostazioni base quali il numero di round necessari per vincere e la presenza di powerUp e ostacoli.

Requisiti Funzionali

- Durante la partita, a differenza dei principali giochi di questo genere, l'astronave si muove costantemente in avanti ed il giocatore decide solamente quando girare e quando sparare.
- Ogni astronave può avere al massimo tre cartucce a disposizione contemporaneamente che si ricaricano nel tempo. Una volta sparato, un proiettile deve andare dritto finché non colpisce un muro, un ostacolo o un'astronave, che in tal caso deve morire.
- Durante la partita c'è la possibilità di raccogliere power-up, i quali forniscono al giocatore che li raccoglie vari bonus, in alcuni casi per un tempo limitato:
    - Scudo: uno scudo che blocca la prossima istanza di danno ricevuta dal giocatore.
    - Doppio Proiettile: al successivo comando di sparo del giocatore, la sua astronave sparerà due proiettili anziché uno.
    - Velocità aumentata: dal momento in cui questo power-up viene raccolto la velocità della navicella viene lievemente aumentata per qualche secondo.
    - Immortalità: dal momento in cui questo power-up viene raccolto la navicella sarà immortale per un breve periodo di tempo.
- Il gioco mette a disposizione varie mappe, generate casualmente ad inizio game, con elementi dannosi per il giocatore (laser) e ostacoli indistruttibili oppure distruttibili.
- La singola partita sarà conclusa nel momento in cui rimarrà una solo player in gioco, mentre l'intero game finirà nel momento in cui un giocatore raggiungerà in numero di vittorie deciso nel menù iniziale.

Requisiti Funzionali Opzionali

- Aggiunta di ulteriori power-up e ostacoli.
- Alla morte della astronave nasce un pilota che se non viene colpito entro un tempo prestabilito torna ad essere un'astronave.

Requisiti Non Funzionali

- Essendo un videogioco dovrà essere fluido in modo da garantire un'esperienza piacevole all'utente finale.
- Il gameplay e le funzionalità del videogioco devono essere ben bilanciati e intuitivi per il giocatore, al fine di permettere di esplorare il gioco e le sue meccaniche senza frustrazioni o difficoltà eccessive.

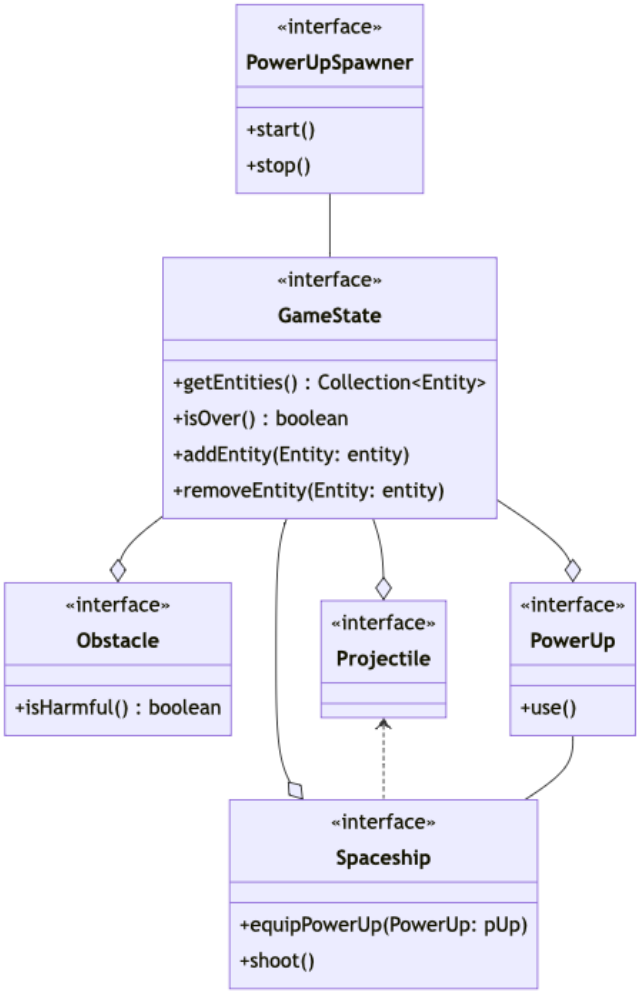
1.2 Analisi e modello del dominio

Durante una partita di AstroParty ci sono due o più astronavi che si sparano tra loro, ognuna controllata da un giocatore. Ogni giocatore ha la possibilità di sparare e raccogliere powerUp. Nel momento in cui un proiettile colpisce un bersaglio possono accadere diversi eventi a seconda dell'entità colpita:

- in caso si tratti di un'astronave, questa deve morire
- se è un muro non succede nulla
- in caso sia un ostacolo dipende se questo è distruttibile o meno

Essendo la mappa occupata da ostacoli e altre astronavi il giocatore non potrà muoversi ovunque e in caso vada a colpire un ostacolo la conseguenza dipenderà dalle caratteristiche di questo ostacolo.

Durante una partita di AstroParty vi sono vari tipi di interazioni e collisioni tra entità. Nella mappa è presente più di un'astronave, ognuna comandata da un giocatore diverso e ognuna ha la possibilità di interagire con le altre, con i power-up, con gli ostacoli e con i proiettili. Le interazioni vanno gestite ognuna in un modo diverso, se un proiettile colpisce un'astronave entrambi dovranno essere eliminati dalla mappa, se un'astronave colpisce un muro dovrà fermarsi e nessuna delle due entità dovrà essere tolta, ecc.

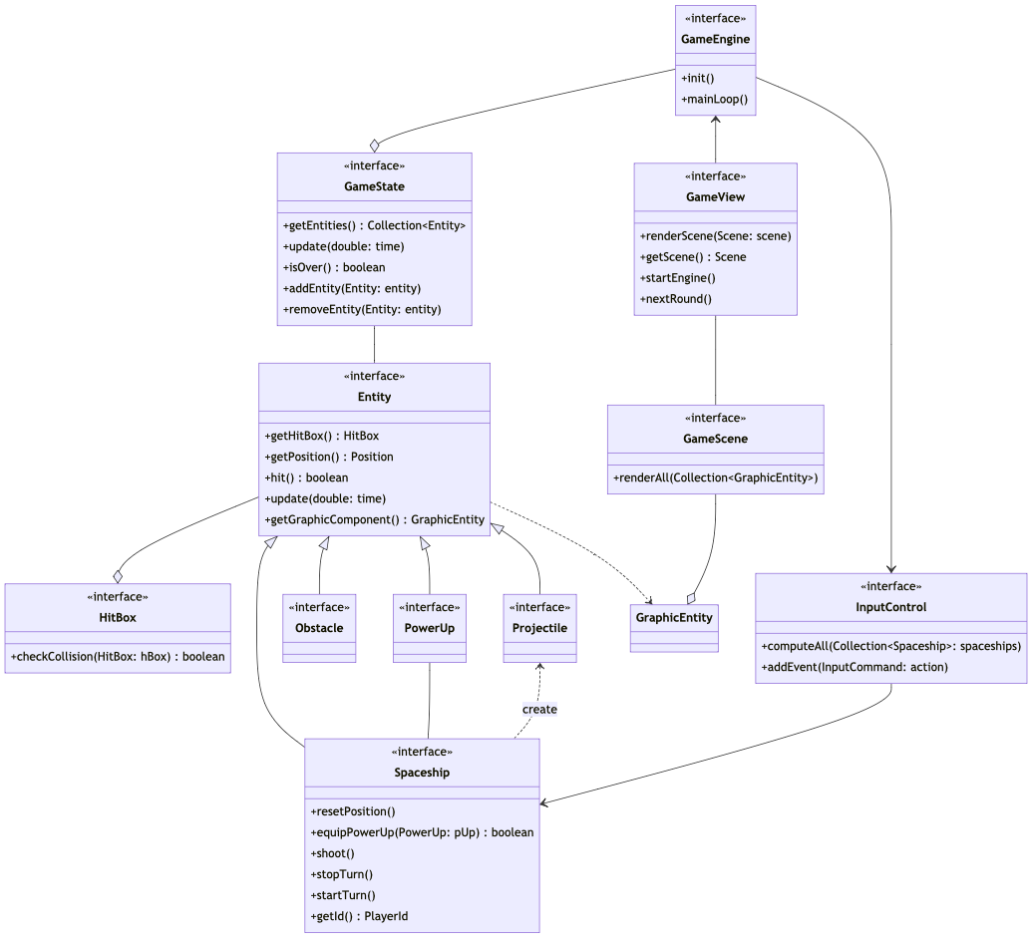


Capitolo 2: Design

2.1 Architettura

L'architettura di AstroParty segue il pattern architetturale MVC.

- **Model**: Il ruolo di model dell'Applicazione spetta ad un insieme di più interfacce, GameState che si occupa di gestire tutte le entità in gioco, PowerUpSpawner che si preoccupa della generazione dei power-up, infine ad Enitity e alle sue estensioni che rappresentano le varie entità in gioco all'interno di AstroParty.
- **Controller**: questo compito viene affidato principalmente a GameEngine, la quale ha il compito di gestire l'aggiornamento delle varie Enitity, della view e la computazione dell'Input, per la quale si appoggia ad InputControl.
- **View**: Gameview si occupa di cambiare e renderizzare le scene, mentre GameScene rappresenta la scena di gioco, che ha quindi il compito di disegnare le varie entità.

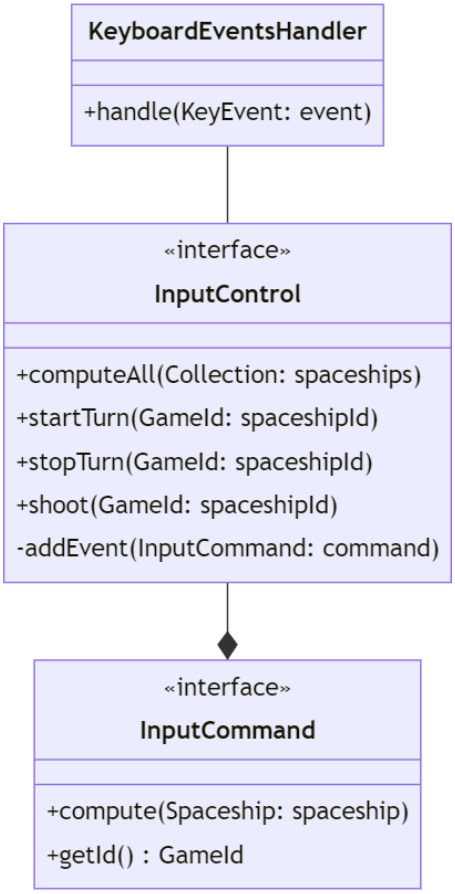


Dallo schema UML fornito sopra è abbastanza naturale notare che l'interfaccia GraphicEntity è direttamente collegata sia al Model che alla View, ma, nonostante ciò, non genera problemi in quanto contiene i dati necessari a rappresentare un' Entity a livello grafico indipendentemente dal dominio grafico scelto. Infatti, in caso in futuro si decida di passare da JavaFx a Swing o altre tecnologie, non vi sarà necessità di modificare nulla all'interno del Model, ma solamente le implementazioni di GameScene e GameView in modo da utilizzare la nuova tecnologia.

2.2 Design dettagliato

Alessandro Coli

Gestione Input



Problema

Il gioco riceve input per diversi player che potrebbero provenire da fonti diverse, quali la tastiera o un joypad o una possibile AI per I bot, ma la fonte di tale input non è di interesse dell'entità stessa. Serve quindi un modo per raccogliere tutti i comandi in un'unica coda, che poi li smisti alle entità corrette.

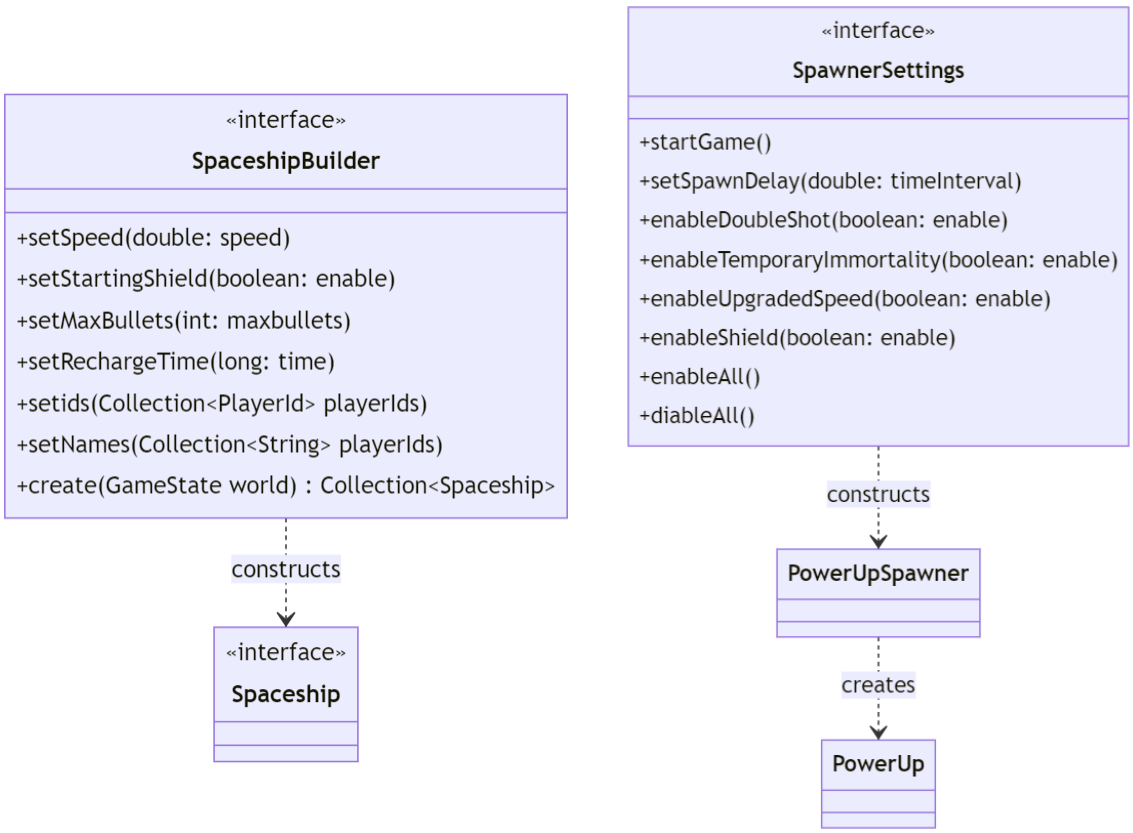
Soluzione

InputControl rappresenta una coda di InputCommand e fornisce un metodo per aggiungere ogni possibile comando di input (shoot, startTurn e stopTurn) su una qualsiasi Spaceship, la quale viene riconosciuta dal parametro GameId. Quando questi tre metodi vengono chiamati, aggiungono semplicemente un InputCommand alla coda tramite addEvent(InputCommand), che viene smaltita nel momento in cui viene chiamato computeAll() che chiama compute(Spaceship) su ogni elemento dopo aver trovato la spaceship corretta tramite il GameId.

Per esempio, nel caso dell'input da tastiera alla pressione di un qualsiasi tasto nella finestra del gioco viene chiamato il metodo handle(KeyEvent) della classe KeyboardEventsHandler, che riconosce il tipo di evento (pressione o rilascio)

e in base al tasto su cui tale evento avviene chiama il metodo corretto di InputControl, passandogli la spaceship come parametro.

Parametri di Spaceship e PowerUpSpawner



Problema

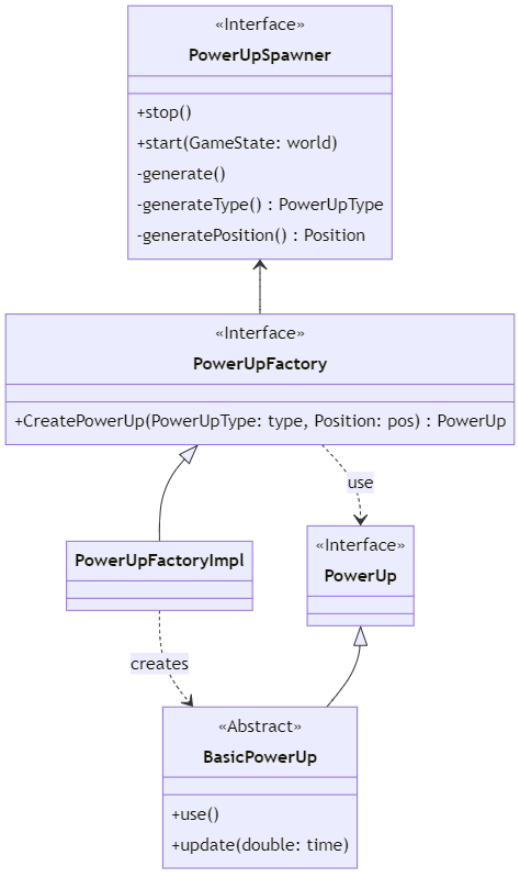
Il menù iniziale permette di selezionare quali PowerUp sono attivi e i parametri della spaceship che poi rimarranno gli stessi durante la partita e uguali per tutti i giocatori (velocità, num cartucce, …)

Soluzione

SpaceshipBuilder è un builder di Spaceship che mette a disposizione vari metodi tramite i quali si possono settare i parametri delle spaceship e un metodo per aggiungere i nomi dei player che una volta chiamato blocca tutti gli altri metodi, ad esclusione del metodo create() che può essere usato varie volte per ripetere più round con le stesse impostazioni e che ha il compito di creare effettivamente le spaceship, il numero dipende dal numero di nomi passati. In caso uno o più parametri non vengano settati direttamente tramite Iimetodi, verranno usati quelli nel file SpaceshipBuilderconfig.yml.

Allo stesso modo SpawnerSettings è un builder per PowerUpGenerator, che mette a disposizione vari metodi per attivare e disattivare i singoli tipi di PowerUp o tutti e di personalizzare il tempo tra lo spawn di un PowerUp e il successivo.

Generazione PowerUp nel tempo



Problema

I PowerUp si generano ripetutamente durante tutta la durata della partita in posizioni casuali

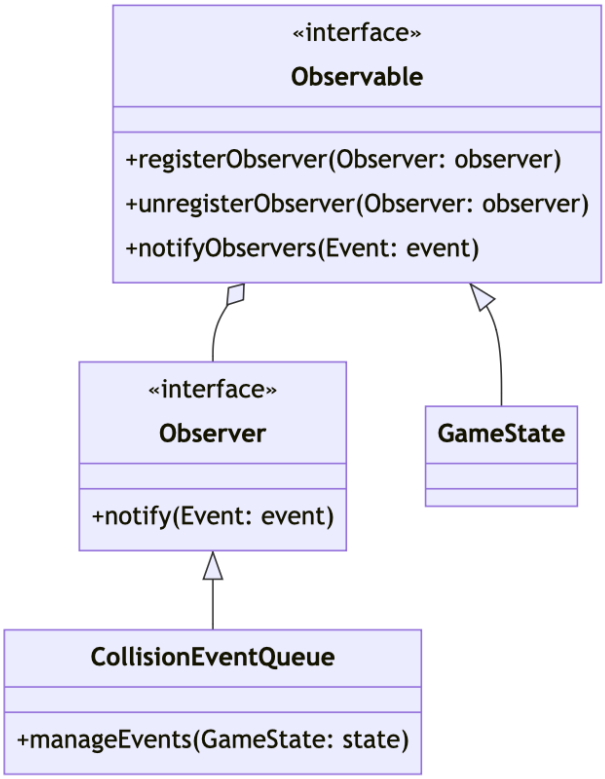
Soluzione

l'interfaccia PowerUpGenerator è responsabile di generare i PowerUp del gioco in modo casuale ad intervalli di tempo prestabiliti. Per fare ciò, usa un Timer che viene programmato affinché chiami il metodo generate() della ad ogni scadenza. Questo metodo interagisce con altre due funzioni interne, generateType() e generatePosition(), che generano casualmente rispettivamente il tipo di PowerUp e la sua posizione nella mappa di gioco, assicurandosi prima che ciò sia possibile. Una volta che questi due elementi sono stati stabiliti, la classe PowerUpGenerator passa i valori generati a PowerUpFactory, che ha il compito di creare una nuova istanza di PowerUp.

La PowerUpFactory implementa il pattern factory, che significa che è stata progettata in modo tale da poter generare qualsiasi tipo di PowerUp attraverso la combinazione di un PowerUpType e una Position tramite createPowerUp(EntityType, Position), che riceve in input questi due parametri e si occupa di creare un oggetto PowerUp corrispondente. Nel fare ciò, la PowerUpFactory chiama un metodo privato specializzato per ogni tipo di PowerUp, che a loro volta generano una classe anonima derivata dalla classe BasicPowerUp. Queste classi anonime si differenziano per la loro implementazione personalizzata dei metodi use() e update().

Mirco Terenzi

Effetti delle collisioni



Problema Gestire e cambiare (o aggiungere) effetti legati alle collisioni.

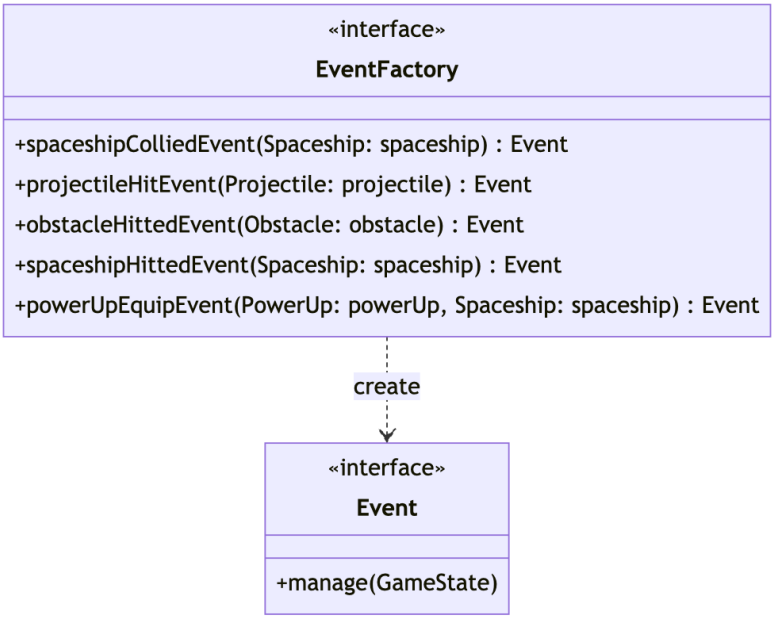
Soluzione Per favorire l'ampliamento e/o la modifica della logica legata alle collisioni tra entità è stato necessario dividere l'individuazione delle collisioni dai loro effetti.

Inizialmente, per separare questi due aspetti, è stato implementato un *observer pattern* atto a gestire gli eventi di collisione individuati nel gioco. Il pattern utilizzato rende possibile modificare la parte di fisica del gioco senza dover apportare cambiamenti alle restanti parti. Al tempo stesso, questa implementazione favorisce anche l'estensione del codice in quanto è possibile aggiungere specifici observer per ogni ulteriore conseguenza delle collisioni che si vuole implementare (ad esempio in future estensioni si potrebbe inserire un sistema di punti basato sulle navicelle colpite).

Durante la scrittura del codice è diventato evidente che, per poter gestire gli eventi di collisione, utilizzare un observer poteva generare problemi di modifiche concorrenti allo stato di gioco. Per risolvere questo problema è stato "specializzato" l'observer che prima si occupava di gestire le collisioni con un *event queue pattern* (analizzato da Robert Nystrom in "Game Programming Patterns"), pattern molto simile al precedente ma che al posto di separare solamente chi invia il messaggio da chi lo riceve, separa i due aspetti anche sotto l'aspetto temporale, soluzione perfetta per il problema riscontrato. Nel nostro caso la classe CollisionEventQueue immagazzina gli eventi e una volta chiamato il metodo manageEvents gli eventi verranno gestiti e la coda svuotata.

Vista la grande somiglianza tra i due pattern è stata lasciata l'implementazione iniziale considerando l'event queue come un'estensione di observer in modo da poter comunque sfruttare la possibilità d'aggiunta di observer per i motivi prima indicati.

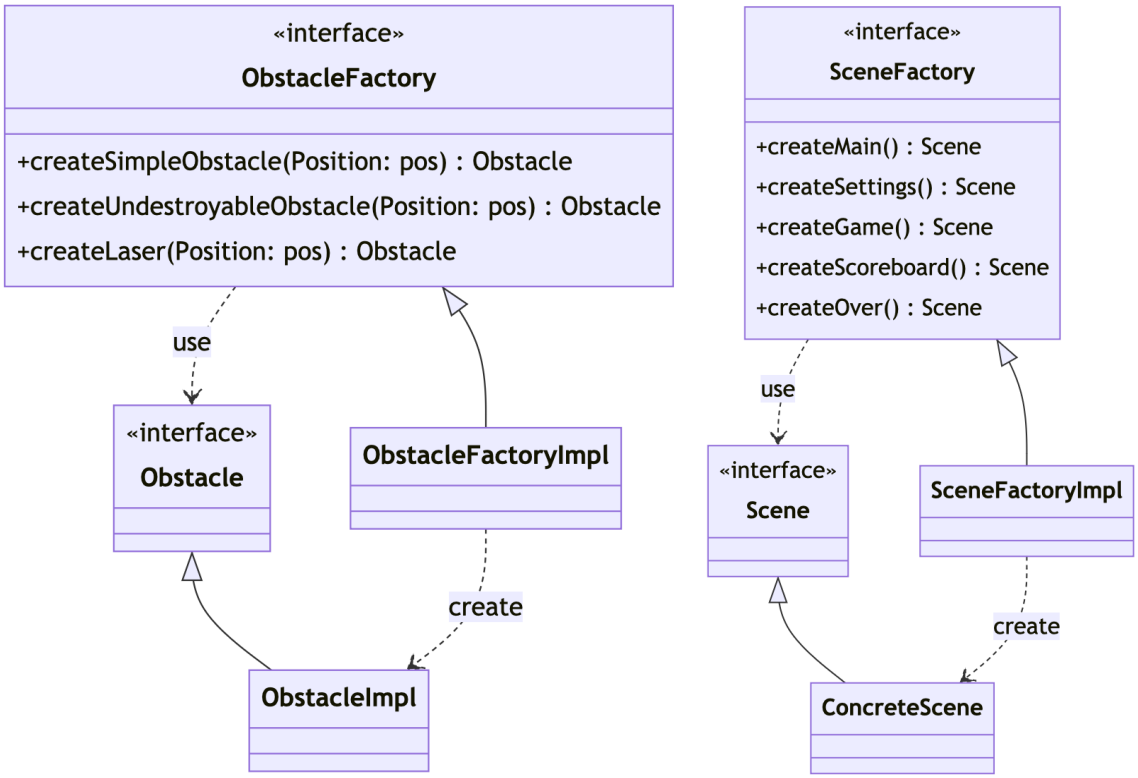
Creazione degli eventi di collisione



Problema Come anticipato nel pattern precedente, all'interno del gioco sono presenti varie tipologie di collisioni, ognuna delle quali richiede di essere gestita in maniera diversa in base al tipo di entità.

Soluzione Per facilitare la creazione di questi eventi è stato implementato un *factory method pattern*, per poter delegare la creazione degli eventi ad una classe esterna, diversa da quella che li utilizza in quanto questo aspetto non la riguarda.

Creazione degli ostacoli e delle scene



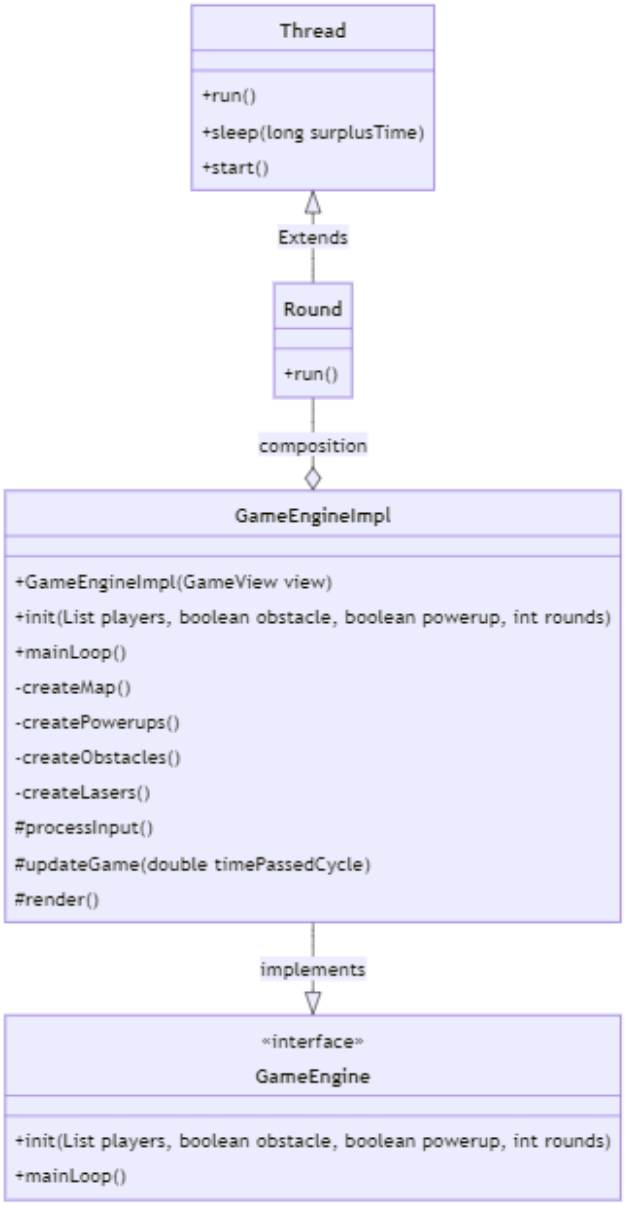
Problema All'interno del gioco sono presenti più tipologie di ostacoli che si differenziano per caratteristiche come la possibilità di essere distrutto o la sua pericolosità per la navicella del giocatore. Allo stesso modo sono presenti diverse scene per quanto riguarda l'interfaccia con l'utente finale.

Soluzione La composizione delle istanze di Obstacle non è interesse del GameEngine che invece si occupa della creazione della mappa. È stato implementato un *factory method pattern* che permettesse la richiesta di nuovi ostacoli diversi senza però preoccuparsi della loro composizione nel dettaglio. Questa divisione semplifica inoltre la modifica del codice, permette infatti di cambiare solo la parte di utilizzo che di creazione degli ostacoli senza dover alterare l'altra parte e facilita l'aggiunta di nuovi ostacoli.

Discorso analogo per quanto riguarda SceneFactory e la creazione delle scene.

Dario Berti

Creazione del Game Loop:



Problema: Creazione di un Loop che sia quasi sempre attivo durante l'esecuzione del gioco e che ne decida il ritmo. Inoltre, tale Loop deve gestire il tempo e varie componenti del gioco.

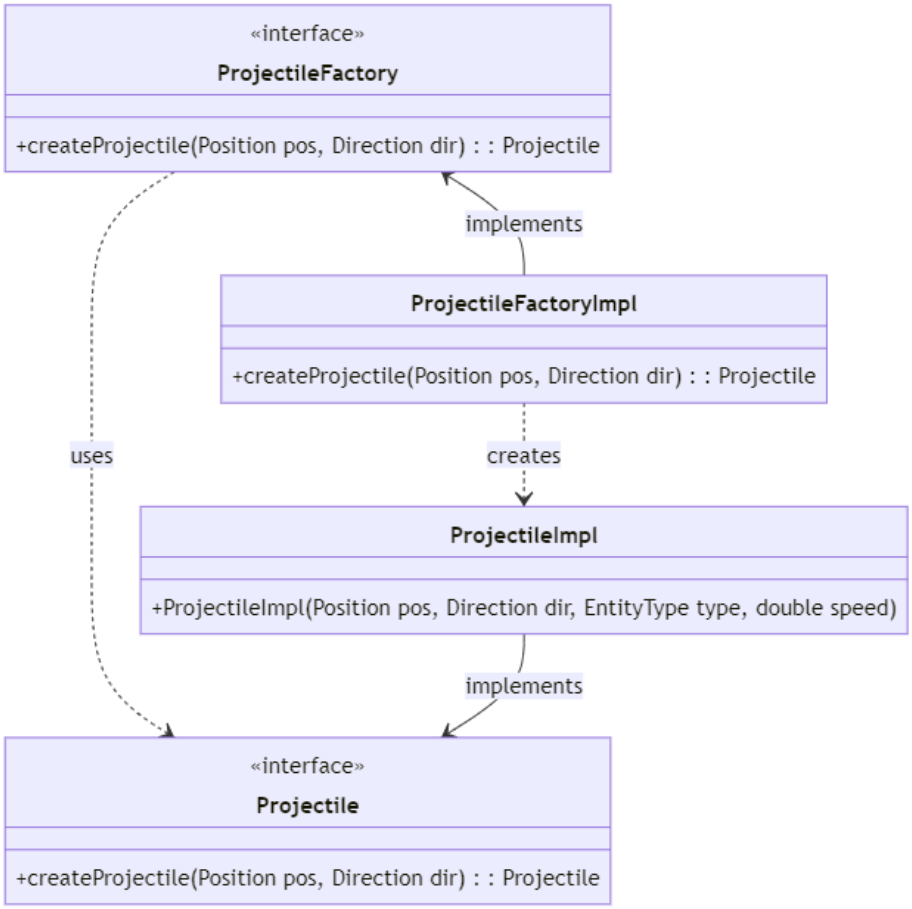
Soluzione: Creazione della classe GameEngineImpl con all'interno vari metodi, tra cui il design pattern "game loop". Tale classe è logicamente suddivisa in metodi per la creazione della mappa e per il game loop.

Il game loop gestisce il passare del tempo in modo che ogni frame sia processato a 60 fps in modo uniforme attraverso l'uso di Thread ed un intervallo di refresh fisso.

Vengono usati i Thread perché se le operazioni all'interno vengono svolte in minor tempo dell'intervallo, allora è necessario bloccare per i millisecondi restanti il loop, in modo da aspettare di processare il prossimo frame.

Bisogna controllare quando il thread viene fatto partire quindi si usa una inner class privata "Round" che estende Thread, con al suo interno il game loop.

Creazione dei proiettili:



Problema: Creazione dei proiettili.

Soluzione: Creazione dei proiettili attraverso il Factory design pattern.

La Factory permette di creare i proiettili in modo flessibile e ne nasconde l'implementazione all'utente, in quanto è all'oscuro dei dettagli di implementazione, tra cui la velocità del proiettile, il corrispondente costruttore, etc.

È necessario il raggio del proiettile che però è definito dentro l'interfaccia Projectile, in quanto non dovrebbe essere modificato.

Capitolo 3: Sviluppo

3.1 Testing automatizzato

Per effettuare i test abbiamo utilizzato JUnit 5, i test che abbiamo effettuato sono:

- **MovementTest**: si preoccupa di testare il movimento della spaceship e di conseguenza le operazioni di Position e Direction.
- **HitBoxCollisionTest**: si occupa di testare l'intercettazione delle collisioni tra diverse entità in tutte le direzioni.
- **ProjectileMovementTest**: provvede a testare il movimento dei proiettili

3.2 Metodologia di lavoro

Per prima cosa definendo l'architettura del gioco e le interfacce abbiamo posto solide basi.

Questo, insieme ad una buona suddivisione del lavoro ci ha permesso di incontrarci solamente quando necessario e lavorare parallelamente per la maggior parte dello sviluppo. Ci siamo quindi riuniti solo in caso ci fossero da valutare nuove aggiunte o modifiche architetturali e in quei momenti in cui è sorta la necessità di far collaborare al meglio le componenti di diversi membri del gruppo. Abbiamo deciso di utilizzare il DVCS con l'approccio spiegato in aula; infatti, una volta creato il repository ognuno di noi ne ha fatto un clone per lavorarci in autonomia condividendo le aggiunte mediante pull/push e lavorando quando necessario su branch specializzati.

Berti Dario:

- Proiettili (package astroparty.game.projectile)
- OverController (nel package astroparty.ui)
- Game Loop (GameEngine nel package astroparty.core)

Coli Alessandro:

- PowerUp (package astroparty.game. powerup)
- Spaceship (package astroparty.game.spaceship)
- Input (package astroparty.input)
- Grafica durante la partita (package astroparty.graphics)

Terenzi Mirco:

- Logica di gioco (package it.unibo.astroparty.game.logics)
- Ostacoli (package it.unibo.astroparty.game.obstacle)
- Collisioni (package it.unibo.astroparty.game.hitbox)
- Gestione della grafica (GameView e GameApp nel package it.unibo.astroparty.core)

In collaborazione:

- Interfaccia utente (package it.unibo.astroparty.ui) con Dario Berti

3.3 Note di sviluppo

Alessandro Coli

Utilizzo di Stream e lambda

L'uso di streams e lambda pervade la maggior parte del codice, un esempio:

https://github.com/AlessandroColi/OOP22-AstroParty/blob/805d0c30512dae1b70da51c9b28b1483840e9f23/src/main/java/it/unibo/astroparty/input/impl/InputControlImpl.java#L46-L48

https://github.com/AlessandroColi/OOP22-AstroParty/blob/805d0c30512dae1b70da51c9b28b1483840e9f23/src/main/java/it/unibo/astroparty/game/powerup/impl/PowerUpSpawnerImpl.java#L116-L118

Utilizzo di Optional

https://github.com/AlessandroColi/OOP22-AstroParty/blob/805d0c30512dae1b70da51c9b28b1483840e9f23/src/main/java/it/unibo/astroparty/graphics/impl/GraphicEntityImpl.java#L50-L53

Utilizzo di javaFx

La classe GameSceneImpl realizza una Scene di javaFx con all'interno ImageView a rappresentare le Entity di AstroParty

https://github.com/AlessandroColi/OOP22-AstroParty/blob/805d0c30512dae1b70da51c9b28b1483840e9f23/src/main/java/it/unibo/astroparty/graphics/impl/GameSceneImpl.java#L21

Mirco Terenzi

Utilizzo di Stream e lambda expressions

Utilizzato spesso in varie parti del codice, ad esempio:

https://github.com/AlessandroColi/OOP22-AstroParty/blob/3fb30bb12f139a5a58b3d686e084f4385e0db8d9/src/main/java/it/unibo/astroparty/game/logics/impl/GameStateImpl.java#L106-L113

https://github.com/AlessandroColi/OOP22-AstroParty/blob/3fb30bb12f139a5a58b3d686e084f4385e0db8d9/src/main/java/it/unibo/astroparty/ui/impl/SettingsController.java#L58-L61

https://github.com/AlessandroColi/OOP22-AstroParty/blob/3fb30bb12f139a5a58b3d686e084f4385e0db8d9/src/main/java/it/unibo/astroparty/game/logics/impl/EventFactoryImpl.java#L20

Utilizzo di JavaFX per la parte grafica

https://github.com/AlessandroColi/OOP22-AstroParty/blob/3fb30bb12f139a5a58b3d686e084f4385e0db8d9/src/main/java/it/unibo/astroparty/core/impl/GameApp.java#L14

Utilizzo di Optional

https://github.com/AlessandroColi/OOP22-AstroParty/blob/3fb30bb12f139a5a58b3d686e084f4385e0db8d9/src/main/java/it/unibo/astroparty/game/obstacle/impl/ObstacleImpl.java#L21

Utilizzo di Java Wildcards

https://github.com/AlessandroColi/OOP22-AstroParty/blob/3fb30bb12f139a5a58b3d686e084f4385e0db8d9/src/main/java/it/unibo/astroparty/game/logics/impl/GameStateImpl.java#L119

Utilizzo di algoritmi

Per quanto riguarda l'identificazione delle collisioni tra rettangoli e cerchi, ho utilizzato un algoritmo presente in "2D Game Collision Detection" di Thomas Schwarzl.

https://github.com/AlessandroColi/OOP22-AstroParty/blob/3fb30bb12f139a5a58b3d686e084f4385e0db8d9/src/main/java/it/unibo/astroparty/game/hitbox/impl/RectangleHitBoxImpl.java#L53-L70

Dario Berti

Utilizzo di stream e lambda

https://github.com/AlessandroColi/OOP22-AstroParty/blob/2a0e1ea31dc28a5faad25810654bbf58c0937fc6/src/main/java/it/unibo/astroparty/core/impl/GameEngineImpl.java#L241

Utilizzo di Thread

https://github.com/AlessandroColi/OOP22-AstroParty/blob/2a0e1ea31dc28a5faad25810654bbf58c0937fc6/src/main/java/it/unibo/astroparty/core/impl/GameEngineImpl.java#L192

Utilizzo di javaFX per la grafica di gioco

https://github.com/AlessandroColi/OOP22-AstroParty/blob/2a0e1ea31dc28a5faad25810654bbf58c0937fc6/src/main/java/it/unibo/astroparty/ui/impl/OverController.java#L15

Utilizzo di Random su una mappa di Pair

https://github.com/AlessandroColi/OOP22-AstroParty/blob/2a0e1ea31dc28a5faad25810654bbf58c0937fc6/src/main/java/it/unibo/astroparty/core/impl/GameEngineImpl.java#L145-L150

Capitolo 4: Commenti finali

4.1 Autovalutazione e lavori futuri

Coli Alessandro

Mi sento soddisfatto del risultato finale che abbiamo ottenuto con il progetto, che è stato frutto di un'efficace collaborazione all'interno del nostro gruppo. Questo successo è stato possibile grazie alla scrupolosa analisi che abbiamo condotto prima di iniziare lo sviluppo vero e proprio. Come è talvolta necessario quando si lavora ad un progetto complesso, ci siamo impegnati per capire quale fosse l'approccio migliore. Dopo aver fatto chiarezza su questo aspetto, ci siamo dedicati alla scrittura del codice vero e proprio. Durante questa fase alcune parti del codice sono state migliorate, ma l'architettura originale del progetto è rimasta sostanzialmente intatta.

Durante l'intero periodo di lavoro, ho avuto la possibilità di affrontare diverse sfide e risolvere problemi che mi hanno permesso di ampliare la mia conoscenza e di acquisire nuove competenze nel campo dell'ingegneria e della programmazione. Inoltre, ho potuto sviluppare un'ottima capacità di gestione del tempo e del lavoro in team, imparando a collaborare con i colleghi e ad organizzare al meglio il nostro lavoro per raggiungere gli obiettivi prefissati. Tutte queste esperienze, senza dubbio, si riveleranno estremamente utili per il mio futuro professionale e mi sento fortunato di essere stato parte di un progetto così stimolante e formativo.

In futuro, intendo ritornare su questo progetto per apprendere nuove nozioni mediante l'aggiunta di altre funzionalità e mediante una revisione del codice, che mi consenta di individuare parti migliorabili, quali la resa estetica di alcune parti del gioco, come la visualizzazione grafica del numero di proiettili a disposizione del giocatore.

Mirco Terenzi

Sono molto contento di questa esperienza, nonostante le mie limitate competenze nel campo del game programming, sono soddisfatto del risultato ottenuto. Questo progetto mi ha permesso di applicare gli argomenti affrontati durante il corso e soprattutto mi ha aiutato ad avere un'idea più chiara di cosa significhi lavorare in un contesto di gruppo, con tutti gli aspetti che ne derivano come il rapporto con altri programmatori e con la suddivisione del lavoro da svolgere.

Questo progetto mi ha anche dato la possibilità di approfondire, per quanto possibile, aspetti che abbiamo affrontato più superficialmente a lezione (ad esempio JavaFX o gli algoritmi di collisione tra le entità di gioco) anche se ritengo che ci sia ancora molto da analizzare, soprattutto per quanto riguarda la qualità e la pulizia del codice. In futuro mi piacerebbe migliorare questo progetto, specialmente gli aspetti che per motivi di tempo non ho sviluppato come avrei voluto, ad esempio la parte sulle hit-box di gioco, magari implementando più tipi di forme e collisioni, oppure l'aggiunta di ostacoli mobili (asteroidi) all'interno della mappa.

Dario Berti

Questo progetto è stato il lavoro di gruppo più interessante fino ad ora. Sono molto felice di come il prodotto iniziale si è evoluto.

Sono soddisfatto del mio contributo nel gruppo ma mi sento che avrei potuto approcciarlo in modo migliore. Il periodo in cui sono stato più efficiente è stato verso la seconda metà del progetto. Questo perché verso l'inizio ho fatto l'errore di analizzare troppe informazioni sparse online e troppo scrupolosamente, invece di semplicemente agire e nel peggiore dei casi, rifare.

Sono felice di essere riuscito a superare la sfida per me più difficile: la comunicazione tra varie parti del codice. Inoltre, sono fiero di come il mio codice desse raramente problemi.

Il lavoro in gruppo è stato molto fluido e con pochi intoppi. Sorprendentemente la parte più difficile da gestire in gruppo non sono state le nostre differenze, ma la suddivisione del lavoro. È stata una bella esperienza riuscire a comunicare in team e creare un intero videogioco partendo da zero. In futuro mi piacerebbe portare avanti il progetto perché mi piace molto il team ed ho varie idee applicabili e divertenti che potrebbero far trasparire meglio la mia abilità da programmatore.

Appendice A: Guida utente

Al lancio dell'applicazione ci si trova di fronte a due bottoni: uno porterà l'utente a visualizzare una pagina con una breve descrizione per ogni componente della mappa (ostacoli e power-up), l'altro che porterà al menu principale in cui si sceglie il numero di giocatori e le impostazioni della partita: esistenza di ostacoli e dei power-up. A prescindere dalla scelta vi sarà un ostacolo indistruttibile al centro della mappa, in quanto essendo che le astronavi nascono una di fronte all'altra le partite durerebbero pochi secondi.

Una volta iniziata la partita, l'astronave di ogni giocatore va costantemente dritto e alla pressione di un tasto la sua astronave ruota in senso orario, ogni astronave ha a disposizione fino a tre proiettili contemporaneamente che si ricaricano uno alla volta. Durante il gioco sarà possibile raccogliere power-up i cui effetti sono spiegati nella schermata tutorial.

Nella schermata alla fine della partita, dove si può vedere il vincitore finale, vi è un tasto che permette di ricominciare una nuova partita con le stesse identiche impostazioni (nomi, num rounds, …).

I comandi sono i seguenti:

- Giocatore 1: Q (per ruotare) W (per sparare)
- Giocatore 2: P (per ruotare) L (per sparare)
- Giocatore 3: V (per ruotare) C (per sparare)
- Giocatore 4: ↓ (per ruotare) → (per sparare)

Bibliografia

- Le Classi Position e Direction sono estensioni delle classi p2d e v2d viste a lezione con il Professor Ricci e disponibili su virtuale e git.
- La classe Pair è stata presa da un appello di esame della sessione invernale di questo corso

La maggior parte delle immagini provengono da fonti esterne:

- spaceships e proiettili
- muro indistruttibile
- muro distruttibile
- icona luna
- sfondo
- Velocità aumentata
- Doppio colpo

- Scudo
- Invincibilità