Relazione per programmazione ad oggetti

“LXR4”

Gianluca Bianchi, Francesco Candoli, Federica Guiducci, Chiara De Nardi

10 Agosto 2023

INDICE

CAPITOLO 1

ANALISI

1.1 REQUISITI

Il progetto si pone l'obiettivo di creare un videogioco puzzle game con più livelli ambientato nell'antico Egitto. L'obiettivo di questo gioco è di eliminare tutta la coda di palline prima che arrivi alla piramide finale tramite un cannone spara-palline. Le palline devono seguire un percorso specifico segnato nell'immagine dello sfondo. Queste devono formare una sequenza di 3 o più colori uguali adiacenti per essere eliminate. Il colore della pallina che verrà sparata sarà visualizzato all'interno del cannone, in modo da saperlo in anticipo.

REQUISITI FUNZIONALI

- Il gioco si apre con la schermata iniziale dove l'utente può scegliere da quale livello iniziare e visualizzare i comandi di gioco nell'help menu.
- Durante la partita, il giocatore può muovere il cannone solo a destra e sinistra all'interno del campo da gioco e può anche sparare le palline, sempre tramite input da tastiera.
- Il giocatore può percorrere vari livelli di difficoltà crescente che contengono anche informazioni riguardanti il punteggio, il quale aumenta man mano che le palline vengono distrutte. Invece, diminuisce ogni qualvolta il giocatore lancia una pallina a vuoto.
- Corretta collisione delle palline, sia con i bordi del gioco che con altri oggetti presenti.
- Mettere in pausa il gioco, riprendendolo successivamente.
- Implementazione dei suoni di background e di sparo delle palline.
- Estendibilità del codice per futura aggiunta di mappe e dettagli più articolati in modo da poter modificare il gioco.

REQUISITI NON FUNZIONALI

- Interfaccia di gioco semplice e intuitiva da rendere immediato l'avvio dei livelli.
- Prestazioni in grado di offrire un'esperienza di gioco fluida e gradevole.
- Creazione di livelli randomici con difficoltà diverse.

1.2 ANALISI E MODELLO DEL DOMINIO

Il gioco è strutturato a livelli di difficoltà crescente, ognuno dei quali presenta una diversa disposizione della coda delle palline e una stanza con una diversa ambientazione. Ogni livello contiene informazioni riguardo il punteggio che aumenta distruggendo le varie palline.
Quando le palline entrano nella piramide finale avviene il game over, dove poi il giocatore può scegliere se riprovare quel livello o cambiarlo. Stessa cosa avviene quando invece il giocatore vince e quindi riesce ad eliminare tutta la coda delle palline.
Il giocatore ha a disposizione un cannone che si può muovere solo orizzontalmente per sparare le palline, tramite input da tastiera.

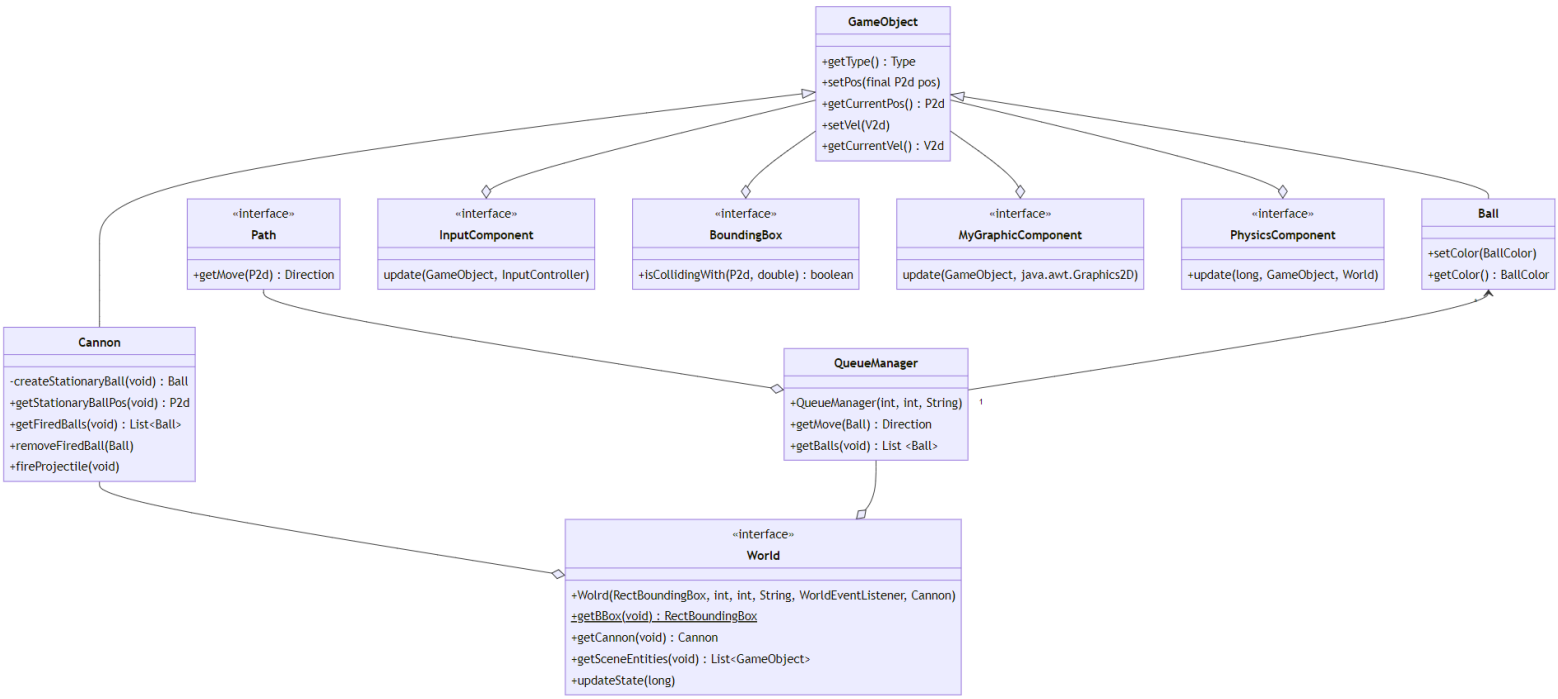


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

CAPITOLO 2

DESIGN

2.1 ARCHITETTURA

Luxor è stato sviluppato adottando un pattern architetturale con struttura MVC sfruttando i suoi vantaggi, ma implementandola con l'uso del pattern Component, permettendo a una singola entità di estendersi su altri domini senza unirli in un'unica classe.
Applicando questo al dominio dell'applicazione, ovvero il model, otteniamo che ogni elemento del gioco è un'entità. Questa entità sarà decorata con componenti, ciascuno dei quali appartiene a un dominio specifico (input, fisica, grafica...) e questi componenti saranno responsabili dell'aggiornamento del dominio all'interno dell'entità.

Dal punto di vista del pattern architetturale Model-View-Controller, i rispettivi moduli sono rappresentati da:

- Model: il ruolo di model è affidato alla classe GameState. Essa rappresenta lo stato del gioco, includendo il punteggio, il mondo, la possibilità di pausa e i vari livelli.
- View: il ruolo della view è affidato all'interfaccia Scene e alla sua implementazione della classe SceneImpl. Esse hanno il compito di rappresentare le varie entità in modo grafico.
- Controller: il ruolo del controller viene affidato al GameEngine che ad ogni ciclo di main loop aggiorna il model e renderizza tutto nella view.

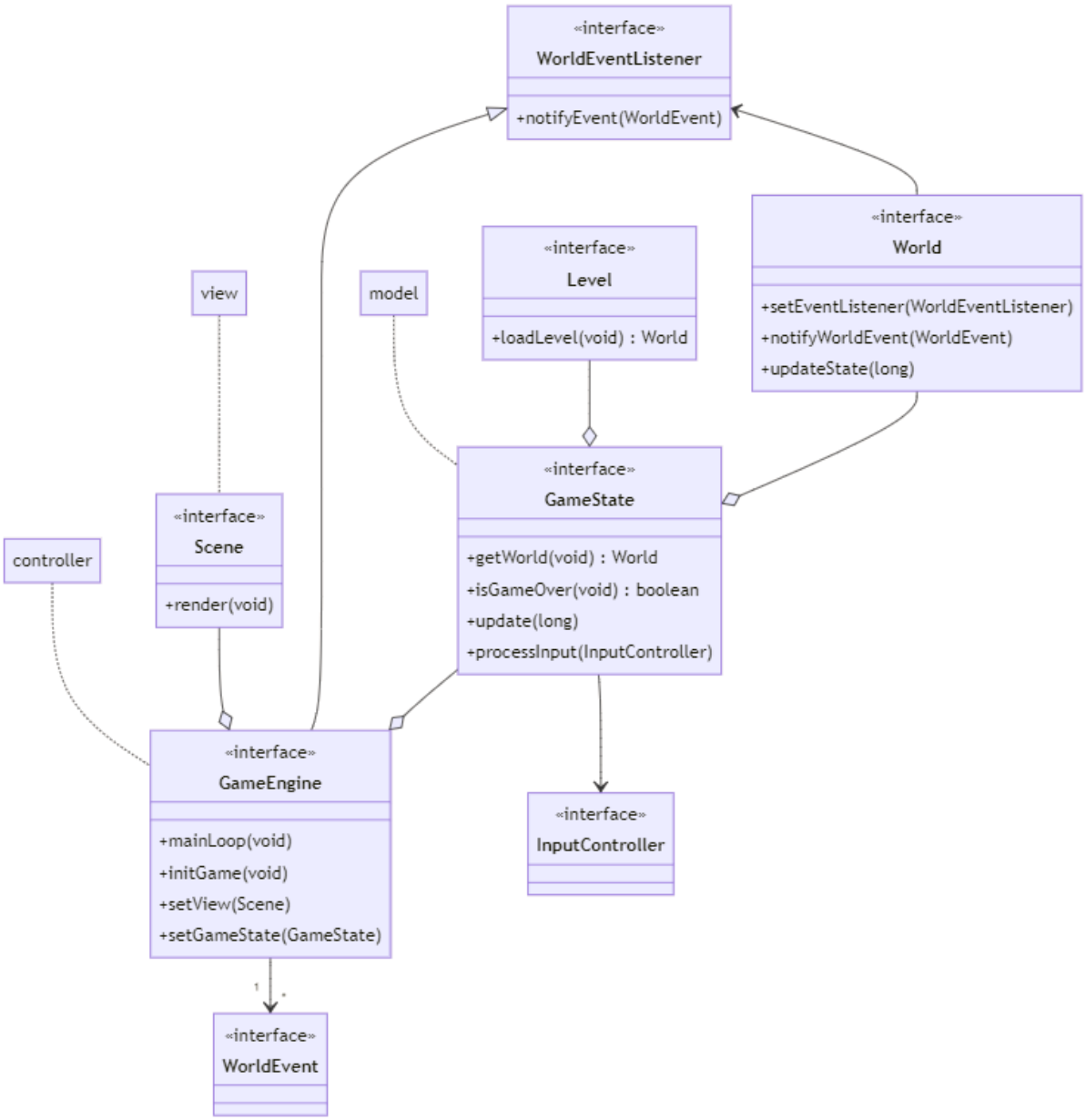


Figura 2.1: Schema UML architetturale di Luxor.

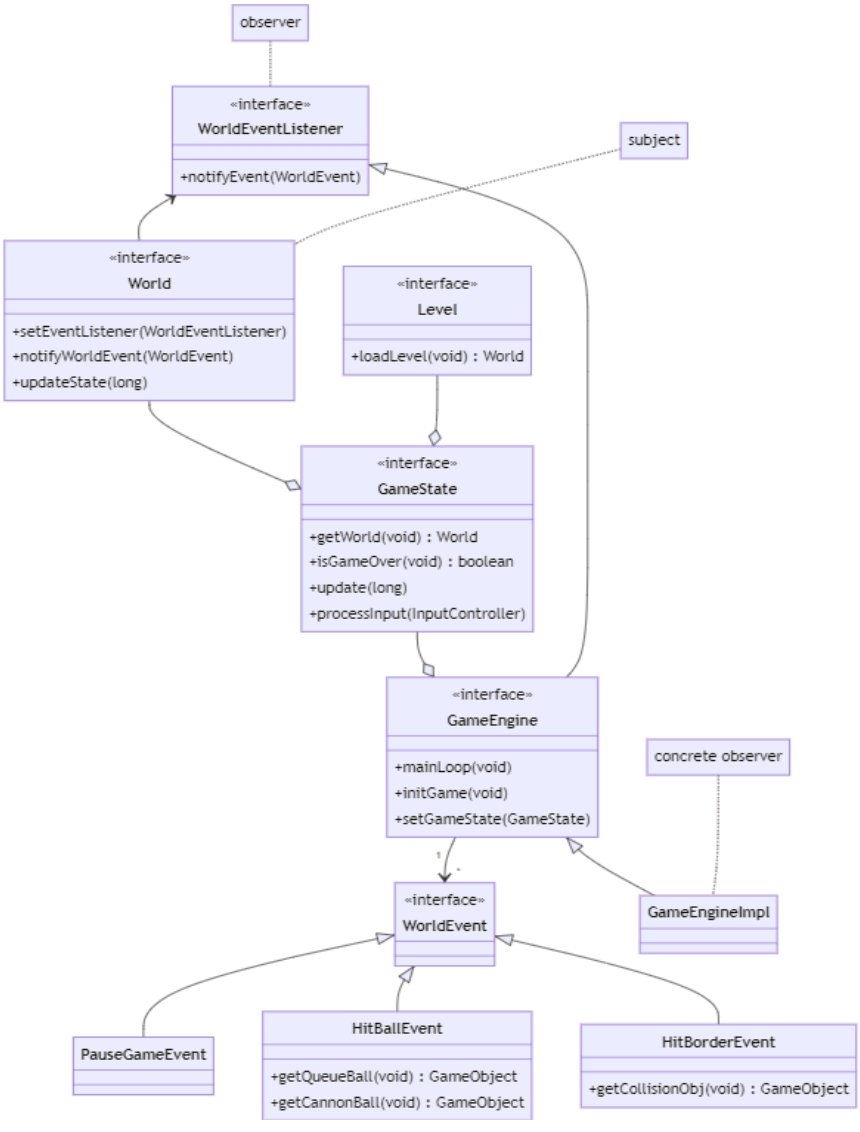
Dal diagramma UML fornito sopra si nota la presenza di una differenza dalla classica architettura MVC, ovvero la GraphicsComponent, la quale ha sia riferimenti alla view che al model però rimanendo sempre separati tra di loro.

2.2 DESIGN DETTAGLIATO

GIANLUCA BIANCHI

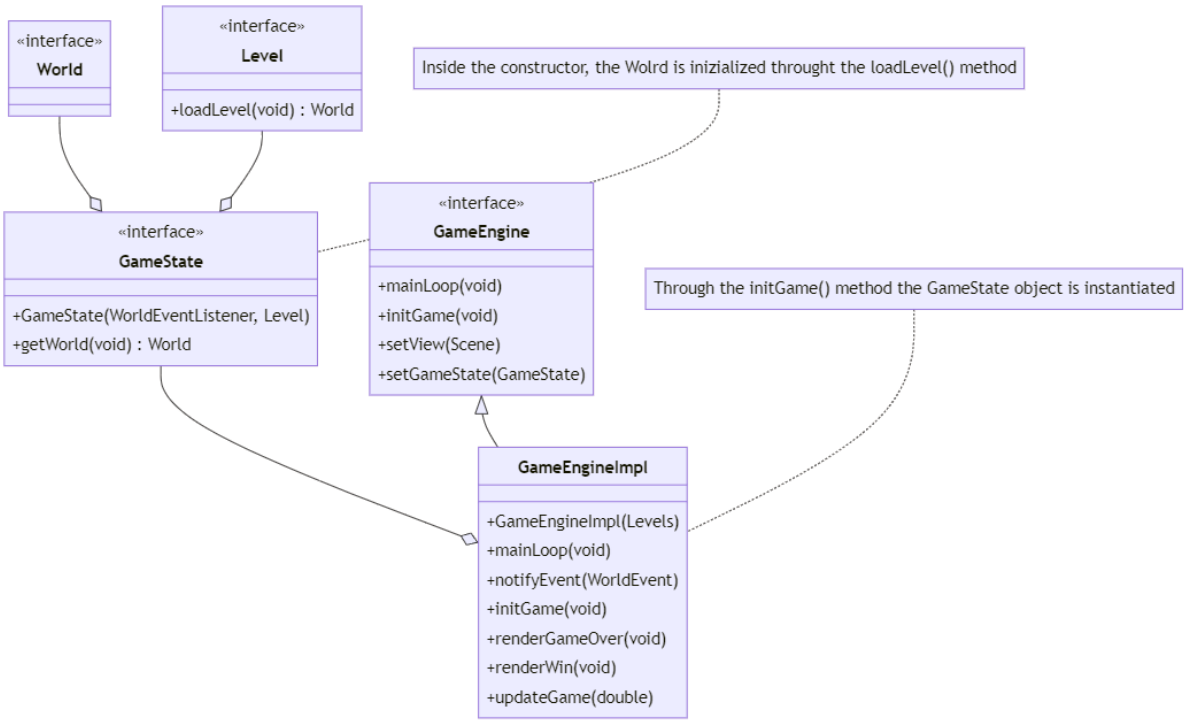
- Gestione Game Loop:
    - **Problema**: Gestire i vari eventi generati durante l'esecuzione del gioco con conseguente aggiornamento dello stato del world e game state ad ogni ciclo di game loop

    - **Soluzione**: Si è optato per l'utilizzo del pattern Observer con qualche modifica strutturale.
      La classe che gestisce il game loop implementa anche l'interfaccia WorldEventListener, che corrisponderà al ruolo di observer e l'interfaccia World svolgerà il ruolo di Subject (con la differenza che non si avranno tanti observer ma solamente il GameEngine).
      Inoltre, il metodo per notificare la generazione di un evento all'observer accetterà come parametro un WorldEvent, interfaccia che verrà implementata con delle classi che ne estenderanno il comportamento in base al tipo di evento generato.
      Ad ogni notifyEvent il GameEngineImpl inserirà l'evento in una apposita coda degli eventi e ad ogni ciclo di game loop svuoterà la coda e li elaborerà singolarmente.



- Gestione del GameState:
    - **Problema**: gestire l'istanziazione dell'oggetto World con i corretti parametri in base al livello selezionato passato nel GameState.

    - **Soluzione**: per risolvere il problema è stata creata un'interfaccia funzionale Level che possiede un metodo "loadLevel()" che restituisce un oggetto di tipo World.
      In questo modo, tramite pattern Strategy e l'utilizzo di un'interfaccia funzionale, il GameEngineImpl nel metodo "initGame()" passerà un'implementazione specifica di Level in base al livello selezionato che è stato passato come parametro del costruttore.



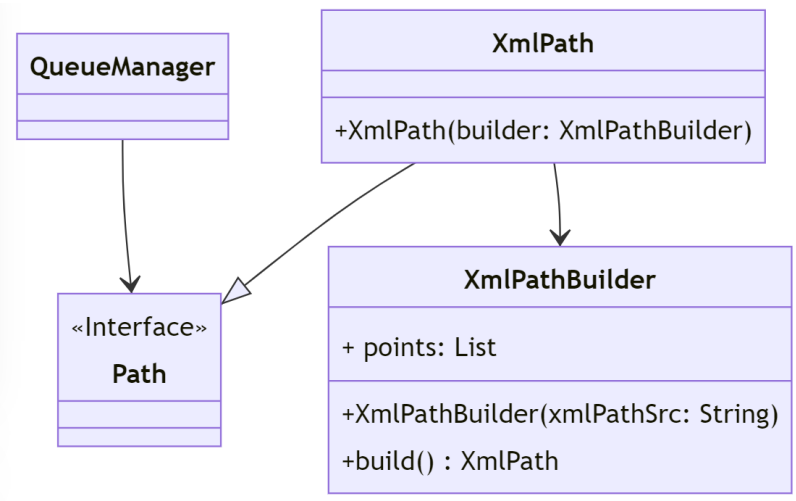
- Gestione collisioni con la coda delle palline:
    - **Problema**: ad ogni ciclo di game loop il game state dovrà aggiornare lo stato della coda delle palline in base alle collisioni avvenute

    - **Soluzione**: ad ogni ciclo il GameState chiamerà un apposito metodo del World per verificare se ci sono collisioni tra una pallina sparata dal cannone e la coda.
      Se vi è una collisione il World notificherà un HitBallEvent al GameEngineImpl il quale andrà a fetchare l'evento e a chiamare il metodo insertCollisionBall() del World tramite il GameState e la coda sarà già aggiornata per il prossimo render() nella View.

FRANCESCO CANDOLI

- Gestione del tracciato delle palline:
    - **Problema**: Gestire la modalità con la quale le palline si muoveranno all'interno della mappa di gioco gestendo quindi le palline stesse ed il tracciato su cui si muovono.

    - **Soluzione**: Prima di tutto si è creata la classe Ball in quanto essere l'elemento che verrà in seguito implementato all'interno della gestione.
      Viene gestito prima di tutto il percorso che le palline percorreranno, viene fatto tramite il pattern strategy utilizzando un'interfaccia Path, la quale definisce un contratto che rappresenta il tracciato sul quale le palline si muoveranno. Successivamente, ho implementato una classe chiamata XmlPath che permette di definire Path personalizzati e complessi tramite l'utilizzo di un file XML.
      Il vantaggio di utilizzare il pattern strategy in questo contesto è che si possono facilmente aggiungere nuove implementazioni di Path per gestire diversi tipi di tracciati. Inoltre, il pattern strategy favorisce la separazione delle responsabilità, rendendo il codice più pulito, mantenibile e facilmente testabile.
      Il file XmlPath presenta inoltre il pattern builder grazie al quale si riesce a separare la creazione dell'oggetto dalla sua rappresentazione.
      Infine vi è una classe QueueManager che tramite l'utilizzo di un Path gestisce varie operazioni riguardanti la coda di Ball.



Schema UML per i pattern Strategy e Builder

- Gestione della pausa della partita:
    - **Problema**: Implementare la possibilità di mettere in pausa il gioco.

    - **Soluzione**: Ho scelto di tenere traccia dello stato di gioco all'interno del gameStateImpl tramite una semplice variabile booleana, è stato creato inoltre un evento (PauseGameEvent) che indica la volontà dell'utente di cambiare tale stato. Per mettere fisicamente in pausa il gioco si è notato che basta fermare l'update delle varie componenti del gioco per fare in modo che nessuna entità presente possa essere aggiornata.

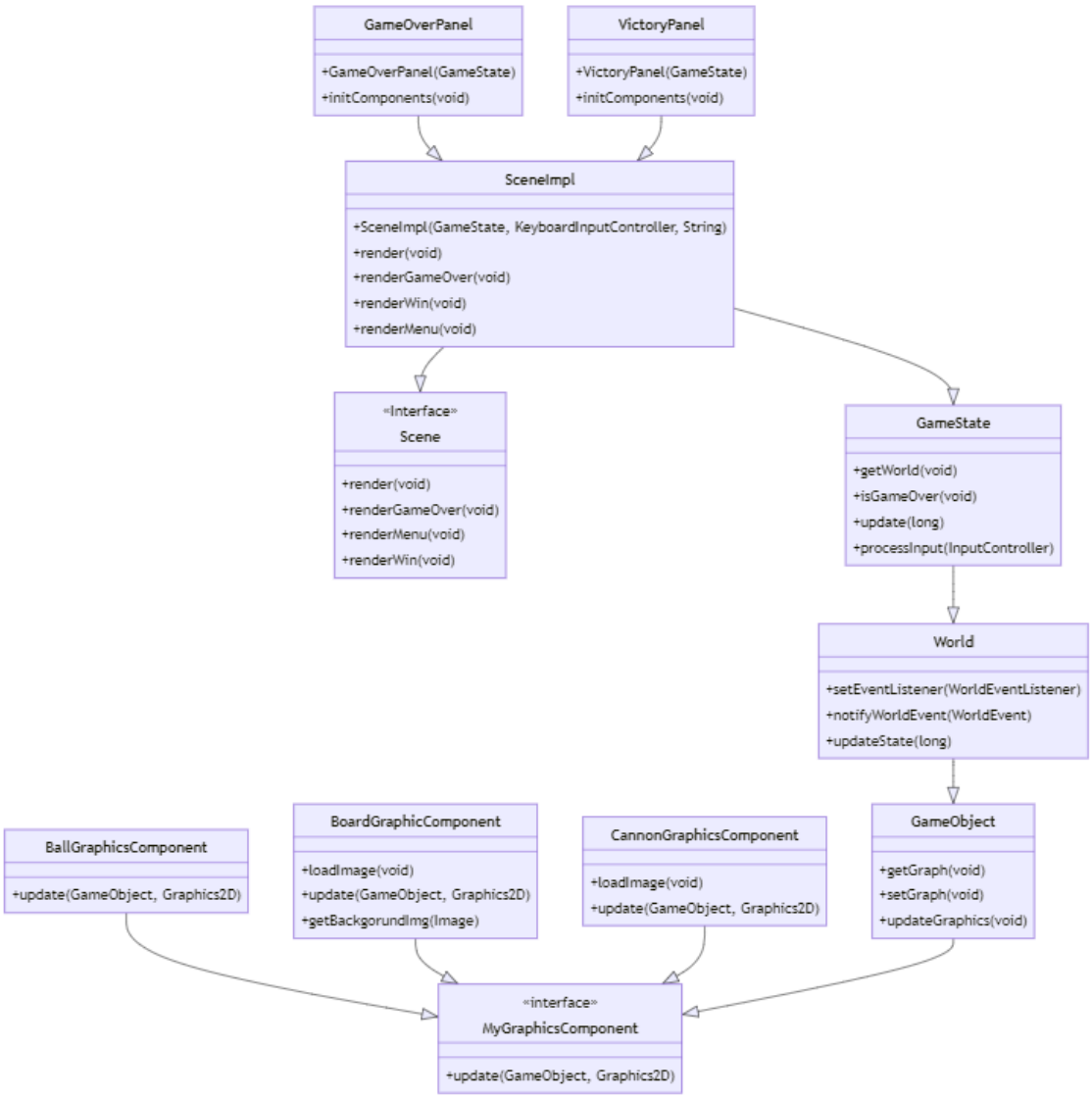
- Gestione degli effetti sonori:
    - **Problema**: Implementare la presenza di effetti sonori all'interno del gioco automatizzati in base all'accadere di eventi specifici.

    - **Soluzione**: È stato scelto innanzitutto di implementare due file audio, una musica di sottofondo per rendere più piacevole e tranquilla l'esperienza di gioco e il suono della collisione tra una pallina sparata e le palline della coda principale.
      Viene creata appositamente una classe SoundPlayer che ha lo scopo di gestire la riproduzione dei file audio permettendo di riprodurli e fermarli garantendo varie possibilità come quella di mettere anche in loop. I metodi del SoundPlayer verranno poi utilizzati nel World per poter personalizzare l'utilizzo di essi in momenti specifici del gioco.

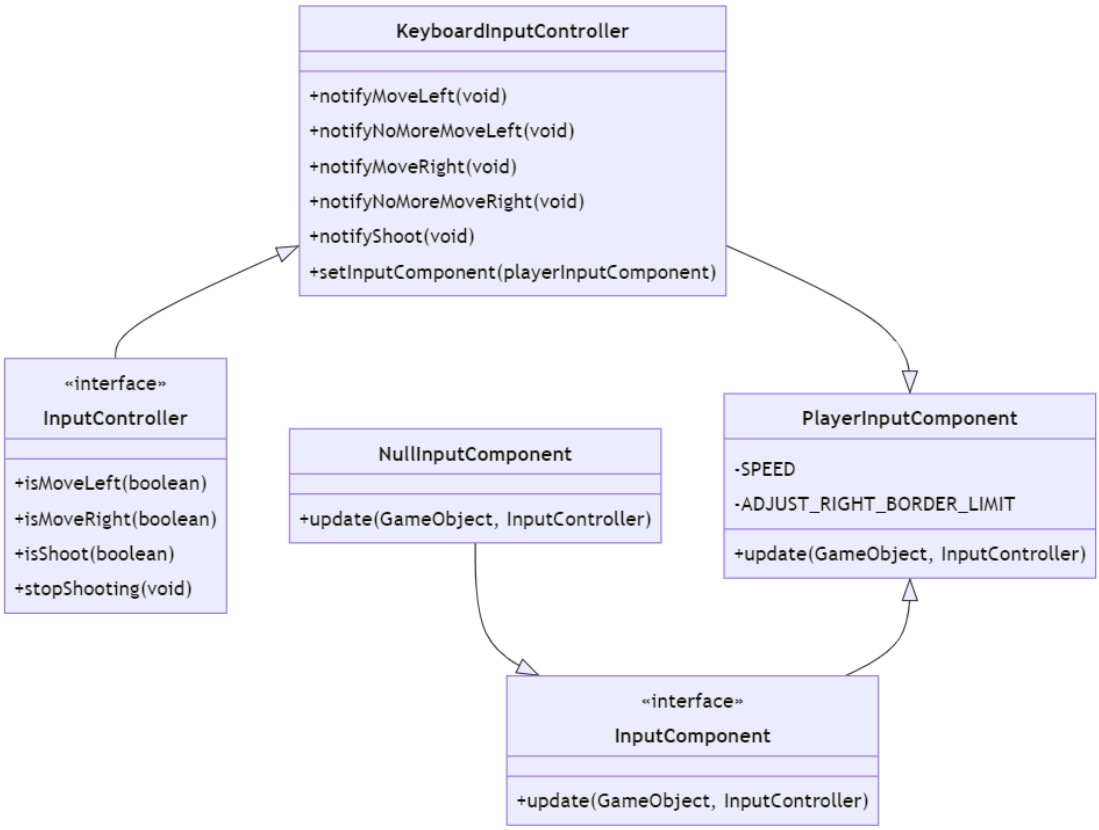
FEDERICA GUIDUCCI

- Sviluppo della grafica del gioco:
  - **Problema:** creare e implementare gli elementi visivi e interattivi che compongono l'interfaccia utente del gioco.

  - **Soluzione:** Si è scelto di creare ed implementare gli elementi visivi e interattivi dell'interfaccia utente, seguendo un'organizzazione modulare e gerarchica. è presente anche un'interfaccia chiamata GraphicsComponent, che definisce il contratto per tutte le componenti grafiche del gioco. Sono state poi create diverse classi che implementano l'interfaccia, uno per ogni componente grafico.
    Inoltre sono state definite altre classi per tutti gli altri "elementi mancanti", quali i pannelli di gioco (vittoria e gameover), scene, menu di gioco e sceneImpl, classe che ha il compito di implementare tutti i dettagli specifici di una determinata scena di gioco.



- Gestione input del giocatore:
    - **Problema:** acquisire, interpretare e rispondere ai comandi impartiti dall'utente durante il gioco. Gli input del giocatore sono dati attraverso tastiera.
    - **Soluzione** per gestire l'acquisizione degli input si è scelto di adottare un approccio modulare e gerarchico. è stata creata l'interfaccia InputComponent che definisce il contratto per tutti i componenti di input nel gioco. è presente la classe PlayerInputComponent che implementa l'interfaccia InputComponent, che ha lo scopo di gestire gli input del giocatore per un oggetto di gioco specifico (movimento del cannone e sparo). è stata creata anche una classe per gestire tutti gli input non specifici (NullInputComponent). con l'interfaccia InputController invece si definisce il contratto per per un controller di input, ovvero fornisce i dati di input per il gioco (es rilevazione dei tasti premuti, posizione del mouse). infine la classe KeyBoardInputController che implementa l'interfaccia InputController, gestisce gli input della tastiera, come il rilevamento dei tasti premuti e fornisce i dati di input rilevanti per il gioco.



- Condizione di fine gioco:
    - **Problema:** definire le circostanze che determinano quando la partita termina, sia in caso di vittoria che in caso di sconfitta. Queste condizioni stabiliscono i criteri che determinano se il giocatore ha vinto completando con successo il livello o se ha perso a causa di una situazione di gioco specifica.

    - **Soluzione** dentro la classe GameStateImpl ho implementato il metodo per controllare le circostanze che determinano quando la partita termina. inizialmente controllo se c'è almeno una pallina nella conda delle palline. verifico poi che la palla possa continuare a muoversi nella sua direzione, se la pallina non può continuare a muoversi nella sua direzione corrente, significa che è arrivata a un punto di blocco o un ostacolo e la partita è finita.

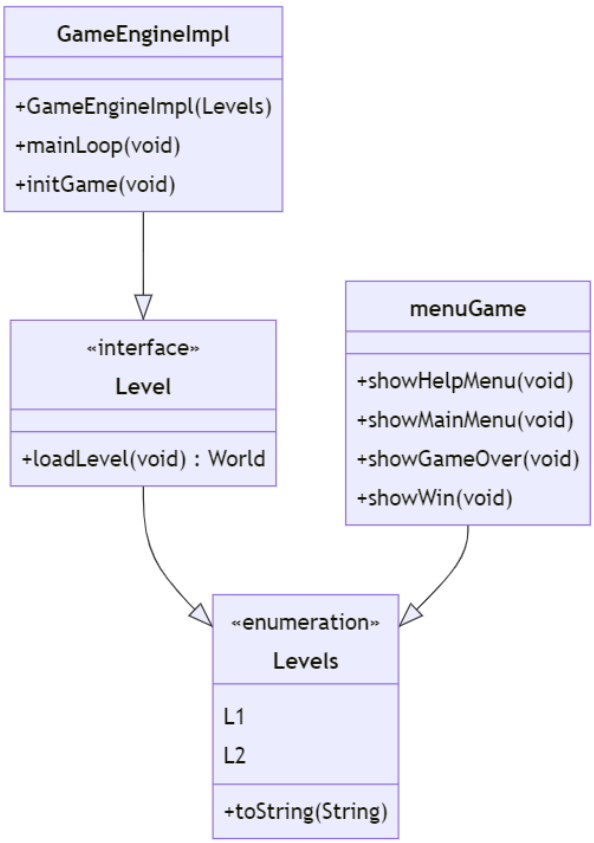
- Implementazione del worldObject e GameObject:
    - **Problema**: Creazione di classi "GameObject" e "WorldObject" per rappresentare oggetti e elementi nel mondo di gioco.

    - **Soluzione:** per risolvere questo problema, creeremo due classi separate: GameObject e WorldObject.
      La classe GameObject rappresenta un'entità giocabile nel mondo di gioco. Contiene attributi e metodi che definiscono le caratteristiche e i comportamenti interattivi dell'oggetto.
      La classe WorldObject rappresenta elementi dell'ambiente, oggetti statici o altre entità nel mondo di gioco che non hanno un comportamento interattivo complesso come i "GameObject".
      è stata poi definita la classe GameObjectFactory che ha il compito di creare dei gameObjects

CHIARA DE NARDI

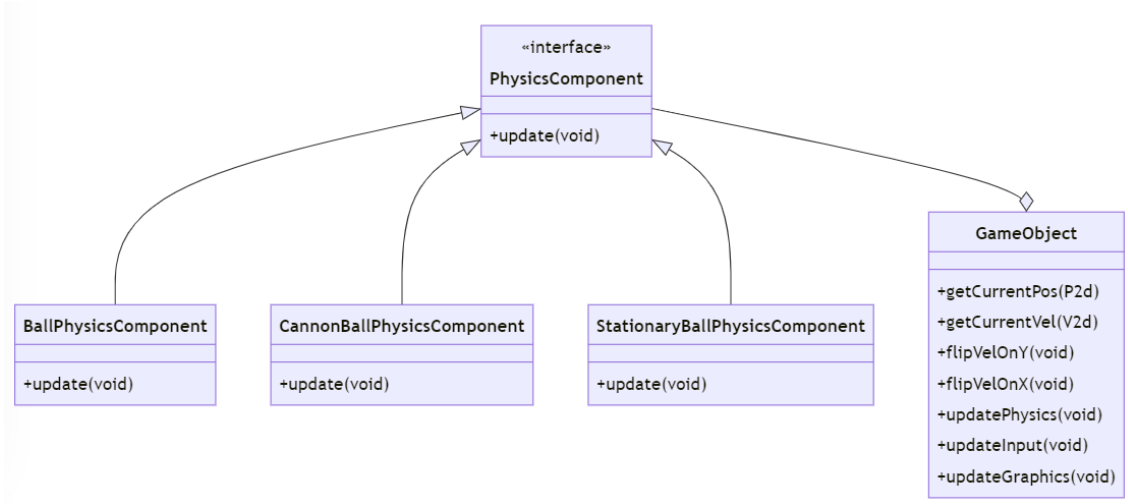
- Gestione del caricamento dei livelli:
  - **Problema**: Gestire il caricamento di un livello a scelta del giocatore prima di iniziare la partita.

  - **Soluzione:** Viene gestito usando il pattern Factory Method attraverso l'interfaccia Level, la quale tramite il metodo astratto loadLevel() restituisce un oggetto World che rappresenta il livello del gioco. La classe GameEngineImpl poi indica il livello selezionato nel menugame e istanzia la corrispondente implementazione di World.
    E' stato scelto questo pattern per fornire un modo flessibile e scalabile per creare istanze di oggetti "world" che rappresentano i diversi livelli del gioco.



- Gestione della fisica degli oggetti:
    - **Problema:** Rendere riutilizzabile le caratteristiche della fisica degli oggetti.

    - **Soluzione:** Ho usato il design pattern Strategy attraverso l'interfaccia PhysicsComponent per separare l'algoritmo di aggiornamento fisico dei GameObject dal proprio contenuto specifico. Questo design pattern consente di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili, promuovendo la modularità, la flessibilità e la riutilizzabilità del codice.
      Infatti, PhysicsComponent definisce un contratto per tutti i componenti che si occupano dell'aggiornamento fisico dei GameObject nel mondo di gioco, fornendo un metodo di default "update" che definisce l'algoritmo di base. Le altre classi per l'aggiornamento della fisica della pallina implementano quest'interfaccia.



- Gestione delle collisioni delle palline sparate dal cannone:
    - **Problema:** Gestire le collisioni delle palline sparate dal cannone che colpiscono i bordi del gioco.

    - **Soluzione**: Viene usato il pattern Strategy nella classe BoundaryCollision attraverso l'enumerazione "collisionEdge". Esso rappresenta diverse strategie per il tipo specifico di collisione con un bordo specifico del mondo di gioco in modo flessibile e riutilizzabile. Infatti, se in futuro volessimo aggiungere nuovi tipi di bordi o gestire collisioni più complesse con i bordi potremmo estendere l'enum e aggiungere nuove costanti, senza modificare il resto del codice. Inoltre, altre parti del sistema possono interagire con la classe BoundaryCollision senza preoccuparsi dei dettagli specifici di come sono gestite le collisioni, poiché queste sono incapsulate all'interno della strategia rappresentata dall'enum.

CAPITOLO 3

SVILUPPO

3.1 TESTING AUTOMATIZZATO

Per effettuare i test abbiamo usato JUnit 5.
I test che abbiamo effettuato sono:

- **PathTest:** controlla che la creazione di un Path e lo scorrimento di una posizione lungo esso avvengano senza problemi senza lanciare eccezioni di alcun tipo.
- **QueueManagerTest:** controlla che il QueueManager venga istanziato correttamente e che i suoi metodi funzionino sulla coda di Ball che contiene.
- **GameRenderingTest**: controlla che quando vengono lanciate le varie schermate viene visualizzato tutto senza problemi, anche se su diversi sistemi operativi
- **KeyboardInputControllerTest:** si verifica che la classe KeyboardInputController sia in grado di gestire correttamente gli input di spostamento e sparo, assicurando che le variabili di stato corrispondano correttamente agli input forniti.
- **GameStateTest:** permette di testare il funzionamento degli aspetti principali del GameState come l'inizializzazione, il raggiungimento delle condizioni di fine gioco e la gestione del punteggio.
- **GameEngineTest:** verifica che non ci siano errori all'avvio del game loop e che la ricezione degli eventi avvenga in maniera corretta.
- **WorldTest:** verifica la corretta istanziazione dell'oggetto e il corretto aggiornamento della coda incluso l'inserimento di palline dopo una collisione

3.2 METODOLOGIA DI LAVORO

Prima di cominciare con lo sviluppo abbiamo impostato l'architettura del gioco definendo delle interfacce.
La suddivisione iniziale del lavoro è stata fatta in maniera equilibrata, in modo da poter lavorare parallelamente e poi incontrarci solo quando necessario.
Abbiamo usato il DVCS con l'approccio spiegato in aula, infatti dopo la creazione del repository ognuno di noi ne ha fatto un clone per lavorare in autonomia condividendo le aggiunte mediante pull e push.

GIANLUCA BIANCHI
In autonomia mi sono occupato delle seguenti classi e della loro implementazione:

- definizione di GameEngine (package it.unibo.core.api)
- implementazione di GameEngineImpl (package it.unibo.core.impl)
- definizione di GameState (package it.unibo.model.api)
- implementazione di GameState (package it.unibo.model.impl)
- definizione di World (package it.unibo.model.api)
- implementazione di World (package it.unibo.model.impl)
- definizione di WorldEventListener (package it.unibo.events.api)
- definizione di WorldEVent (package it.unibo.events.api)
- implementazione di HitBallEvent (package it.unibo.events.impl)
- implementazione di HitBorderEvent (package it.unibo.events.impl)
- definizione di BoundingBox (package it.unibo.model.collisions.api)
- implementazione di CircleBoundingBox (package it.unibo.model.collisions.impl)
- implementazione di RectBoundingBox (package it.unibo.model.collisions.impl)

FRANCESCO CANDOLI
In autonomia mi sono occupato delle seguenti classi e della loro implementazione durante lo sviluppo del gioco considerando il grande utilizzo che ne viene fatto:

- Implementazione di V2d (package it.unibo.utils)
- Implementazione di P2d (package it.unibo.utils)
- Implementazione di Ball (package it.unibo.model.impl)
- Implementazione di BallColor (package it.unibo.enums)
- Implementazione di Direction (package it.unibo.enums)
- Implementazione di SoundPlayer (package it.unibo.utils)
- Implementazione di PauseGameEvent (package it.unibo.events.impl)
- Implementazione di QueueManager (package it.unibo.utils)

Con la collaborazione di Gianluca Bianchi:

- Implementazione di XmlPath (package it.unibo.utils.impl)

FEDERICA GUIDUCCI

In autonomia mi sono occupata di:

- implementazione di Scene (package it.unibo.graphics.api)
- implementazione BallGraphicsComponent (package it.unibo.graphics.impl)
- implementazione BoardGraphicComponent (package it.unibo.graphics.impl)
- implementazione CannonGraphicsComponent (package it.unibo.graphics.impl)
- implementazione GameOverPanel (package it.unibo.graphics.impl)
- implementazione Graphics (package it.unibo.graphics.impl)
- implementazione MyGraphicsComponent (package it.unibo.graphics.impl)
- implementazione SceneImpl (package it.unibo.graphics.impl)
- implementazione VictoryPanel (package it.unibo.graphics.impl)
- implementazione InputComponent (package it.unibo.input.api)
- implementazione InputController (package it.unibo.input.api)
- implementazione KeyboardInputController (package it.unibo.input.impl)
- implementazione NullInputController (package it.unibo.input.impl)
- implementazione PlayerInputController (package it.unibo.input.impl)
- implementazione GameObject (package it.unibo.model.impl)
- implementazione GameObjectFactory (package it.unibo.core.impl)

Con la collaborazione di Chiara De Nardi:

- gestire le collisioni del cannon con i bordi del campo. (package it.unibo.input.impl.PlayerInputComponent)

CHIARA DE NARDI

In autonomia mi sono occupata di:

- Implementazione del cannone e delle sue palline sparate (package it.unibo.model.Cannon)
- Implementazione della fisica del gioco (package it.unibo.physics.api.PhysicsComponent)
- Implementazioni della fisica della pallina sparata dal cannone (package it.unibo.physics.impl.CannonBallPhysicsComponent)
- Implementazioni della fisica della pallina stazionaria (package it.unibo.physics.impl.StationaryBallPhysicsComponent)
- Implementazioni della fisica delle palline (package it.unibo.physics.impl.BallPhysicsComponent)
- Implementazione della gestione dei livelli (package it.unibo.enums.Levels)
- Implementazione del menù di gioco (package it.unibo.graphics.impl.MenuGame)
- Implementazione della fisica delle collisioni con i bordi del campo (package it.unibo.physics.impl.BoundaryCollision)

Con la collaborazione di Federica Guiduccii:

- gestire le collisioni del cannon con i bordi del campo. (package it.unibo.input.impl.PlayerInputComponent)

3.3 NOTE DI SVILUPPO

GIANLUCA BIANCHI

- **Utilizzo di un'interfaccia funzionale**
  Utilizzata in Level per permettere al GameEngineImpl di generare implementazioni di essa tramite Lambda expressions.

- **Utilizzo di lambda expression**
  Utilizzate nel metodo initGame() del GameEngineImpl per fornire implementazioni di Level dinamicamente in base al tipo di livello selezionato nel menu.
  Permalink:
  https://github.com/giangian2/OOP22-luxor4/blob/f6a8e18099f17efb389eccdb13ba2a715df5cf0d/src/main/java/it/unibo/core/impl/GameEngineImpl.java#L111

- **Utilizzo di Optional**
  Utilizzato come return type nei metodi checkCollisionWithBalls() e checkCollisionWithBoundaries() all'interno della classe World.
  Permalinks:
  https://github.com/giangian2/OOP22-luxor4/blob/f6a8e18099f17efb389eccdb13ba2a715df5cf0d/src/main/java/it/unibo/model/impl/World.java#L323
  https://github.com/giangian2/OOP22-luxor4/blob/f6a8e18099f17efb389eccdb13ba2a715df5cf0d/src/main/java/it/unibo/model/impl/World.java#L342

- **Utilizzo della libreria java.util.logging**
  Permalink: https://github.com/giangian2/OOP22-luxor4/blob/e7b071e39ca919ba9aded8507df1cd027274916a/src/main/java/it/unibo/App.java#L15

- **Utilizzo della libreria org.w3c.dom e javax.xml.parsers**
  https://github.com/giangian2/OOP22-luxor4/blob/336e58f66d1e1f5c5dbdd0c4f529522c4ce57099/src/main/java/it/unibo/utils/impl/XmlPath.java#L126

FRANCESCO CANDOLI

- **Utilizzo della libreria Java Sound API**
  Utilizzata all'interno di SoundPlayer.java, per comprendere come utilizzarla per lo scopo è stato utilizzata la seguente pagina web:
  https://www.baeldung.com/java-play-sound
  Permalink: https://github.com/giangian2/OOP22-luxor4/blob/master/src/main/java/it/unibo/utils/SoundPlayer.java

- **Utilizzo della libreria java.util.logging**
  Permalink: https://github.com/giangian2/OOP22-luxor4/blob/master/src/main/java/it/unibo/utils/SoundPlayer.java#L57

- **Utilizzo di Optional**
  Permalink: https://github.com/giangian2/OOP22-luxor4/blob/master/src/main/java/it/unibo/utils/QueueManager.java#L165-L189

- **Utilizzo di Stream**
  Permalink: https://github.com/giangian2/OOP22-luxor4/blob/master/src/main/java/it/unibo/utils/SoundPlayer.java#L123-L128

FEDERICA GUIDUCCI

- **Utilizzo della libreria java.util.logging**
  Permalinks:
  https://github.com/giangian2/OOP22-luxor4/blob/f6a8e18099f17efb389eccdb13ba2a715df5cf0d/src/main/java/it/unibo/graphics/impl/BoardGraphicComponent.java#L49
  https://github.com/giangian2/OOP22-luxor4/blob/f6a8e18099f17efb389eccdb13ba2a715df5cf0d/src/main/java/it/unibo/graphics/impl/CannonGraphicsComponent.java#L42
  https://github.com/giangian2/OOP22-luxor4/blob/f6a8e18099f17efb389eccdb13ba2a715df5cf0d/src/main/java/it/unibo/graphics/impl/SceneImpl.java#L147

- **Utilizzo di Pattern (Component Pattern/ECS pattern):**
  https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/input/api/InputComponent.java#L1-L19
  https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/input/api/InputController.java#L1-L33
  https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/input/impl/KeyboardInputController.java#L1-L102

https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/input/impl/NullInputComponent.java#L1-L23
https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/input/impl/PlayerInputComponent.java#L1-L61

- **Utilizzo di Pattern (MVC):**
  https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/graphics/api/Scene.java#L1-L35
  https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/graphics/api/MyGraphicsComponent.java#L1-L19
  https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/graphics/impl/SceneImpl.java#L1-L316
  https://github.com/giangian2/OOP22-luxor4/blob/c0610089036a687eda757ac3816d04d443ef4b9f/src/main/java/it/unibo/input/impl/KeyboardInputController.java#L1-L102

CHIARA DE NARDI

- **Utilizzo di lambda expressions:**
  Nella classe MenuGame vengono usate per definire gli action listener dei pulsanti, rendendo il codice più conciso ed eliminando la necessità di dichiarare classi anonime separate per ciascun ascoltatore.
  Permalink:https://github.com/giangian2/OOP22-luxor4/blob/e81e2c7e594f94199cca2a6178d252c554998240/src/main/java/it/unibo/graphics/impl/MenuGame.java#L166

- **Utilizzo della libreria Java Util Logging:**
  Viene usata nella classe MenuGame
  Permalink:https://github.com/giangian2/OOP22-luxor4/blob/e81e2c7e594f94199cca2a6178d252c554998240/src/main/java/it/unibo/graphics/impl/MenuGame.java#L19

- **Utilizzo di thread:**
  Nella classe MenuGame viene usato un thread per l'avvio del gioco tramite il pulsante "Start Game".
  Permalink:https://github.com/giangian2/OOP22-luxor4/blob/e81e2c7e594f94199cca2a6178d252c554998240/src/main/java/it/unibo/graphics/impl/MenuGame.java#L260-L270

- **Utilizzo di interfaccia funzionale:**
  Nella classe MenuGame viene usata per definire gli ActionListener per i pulsanti. In questo caso l'interfaccia funzionale è "ActionListener", che è un'interfaccia predefinita in Java ed è usata per definire il comportamento da eseguire quando un pulsante viene premuto.
  Permalink:https://github.com/giangian2/OOP22-luxor4/blob/e81e2c7e594f94199cca2a6178d252c554998240/src/main/java/it/unibo/graphics/impl/MenuGame.java#L249

- **Utilizzo di risorse da un file nel classpath del progetto:**
  Nella classe MenuGame vengono usate risorse da file nel classpath.
  Permalink:https://github.com/giangian2/OOP22-luxor4/blob/e81e2c7e594f94199cca2a6178d252c554998240/src/main/java/it/unibo/graphics/impl/MenuGame.java#L137-L161

- **Utilizzo di Optional:**
  Nella classe CannonBallPhysicsComponent vengono usati degli Optional per rappresentare la possibilità che un oggetto possa essere presente o assente.
  Permalink:https://github.com/giangian2/OOP22-luxor4/blob/e81e2c7e594f94199cca2a6178d252c554998240/src/main/java/it/unibo/physics/impl/CannonBallPhysicsComponent.java#L42-L51

CAPITOLO 4

COMMENTI FINALI

4.1 AUTOVALUTAZIONE E LAVORI FUTURI

GIANLUCA BIANCHI
Sebbene per lavoro io sia abituato a seguire progetti di gruppo ritengo che nonostante qualche leggera difficoltà sia andato tutto liscio e sono contento del risultato finale ottenuto. Lo svolgimento del progetto è stato istruttivo e coinvolgente considerando anche la poca esperienza che mi ritrovavo ad avere nello specifico nello sviluppo di videogiochi.
Ritengo il mio lavoro ben svolto e se mai dovesse esserci la possibilità e l'occasione sarebbe anche interessante portare avanti il progetto in un'ottica futura.

FRANCESCO CANDOLI
Per quanto riguarda il lato collaborazione, essendo stata la mia prima esperienza di lavoro insieme ad un team di persone, mi è servito molto per capire bene cosa significa lavorare in gruppo e ciò mi ha sicuramente aiutato e formato per quando in futuro mi ritroverò a dover fare di nuovo un lavoro in team.
Anche per quanto riguarda il lato programmazione sono molto contento di com'è andata perché nonostante le tante difficoltà incontrate nel mentre sono stato in grado di adattarmi alla situazione e di risolvere i problemi; inoltre è stato molto motivante vedere il progetto pian piano prendere forma e aspetto.
Sarebbe troppo bello dire che non sono state riscontrate problematiche durante lo svolgimento del progetto ma alla fine siamo stati in grado di superarle e credo che quest'esperienza sarà senz'altro utile in futuro per altri progetti che dovrò realizzare.

FEDERICA GUIDUCCI
Il progetto è stato un'esperienza impegnativa e complessa, ma purtroppo non mi ritengo soddisfatta dei miei risultati ottenuti. Ha richiesto molto tempo e sforzo, che ho dovuto togliere dallo studio di altre materie. Questo mi ha causato stress e problemi a livello psicologico, che hanno influenzato anche il mio rapporto con gli altri componenti del gruppo.
Nonostante abbia raggiunto gli obiettivi che mi ero prefissata, ritengo che lo sviluppo della mia parte del progetto sia piuttosto semplice e poco ottimizzato.
Nonostante ciò, devo riconoscere che questa esperienza è stata utile per arricchire il mio bagaglio culturale e per comprendere meglio dove posso migliorarmi in futuro. Mi ha dato l'opportunità di mettermi alla prova e di capire quali sono le mie aree di miglioramento per progetti futuri.
In conclusione, sebbene il progetto non abbia soddisfatto completamente le mie aspettative, l'ho vissuto come un'opportunità di crescita personale e professionale.

CHIARA DE NARDI

Sono contenta di questa esperienza, nonostante le mie limitate competenze nel campo del game programming. Questo progetto mi ha permesso di applicare gli argomenti studiati durante il corso e di aver provato cosa significa lavorare in gruppo, specialmente nel settore informatico.

Avrei voluto però approcciarmi al progetto in modo migliore, approfondendo di più le mie conoscenze e organizzando il tempo a mia disposizione in modo più saggio.

Anche se questo progetto mi ha permesso di acquisire nuove conoscenze e di mettermi molto alla prova, personalmente ritengo che non sia il progetto giusto per me da portare avanti in futuro.

APPENDICE A

GUIDA UTENTE

Quando viene fatto partire l'applicativo ci si ritroverà in una scena con 3 bottoni:

- Help: schermata dove ci sono le istruzioni per giocare.
- Livelli: menù a tendina dove si può scegliere su quale livello giocare.
- Start Game: fa partire il livello selezionato.

I comandi di gioco sono i seguenti:

- P: per fermare il gioco e poi per riprenderlo.
- SPACE: per sparare le palline dal cannone.
- ← (Left Arrow): per muovere il cannone a sinistra.
- → (Right Arrow): per muovere il cannone a destra.

BIBLIOGRAFIA

- Nella fase di analisi abbiamo usato come spunto il git mostrato durante la lezione svolta dal professore A. Ricci. https://github.com/pslab-unibo/oop-game-prog-patterns-2022