Relazione per "Programmazione ad Oggetti"

Nicolò Guerra Emma Leonardi Filippo Casadei
Lorenzo Tagliani

28 aprile 2022

Indice

Capitolo 1

Analisi

Bubble Blaster è un gioco della categoria puzzle games, clone del gioco arcade Puzzle Bubble. Il gioco è formato da una schermata rettangolare in cui vengono create file di bolle colorate che scendono dall'alto. Obiettivo del gioco è utilizzare il cannone posto in fondo alla schermata per formare gruppi di bolle dello stesso colore e farle così esplodere, cercando di ottenere il maggior punteggio possibile. La partita è persa se le bolle arrivano in fondo allo schermo.

1.1 Requisiti

Requisiti funzionali

- All'apertura del gioco verrà mostrata una schermata per la scelta delle modalità di gioco e delle opzioni.
- Il gioco dovrà generare una griglia di bolle colorate in maniera casuale.
- Il cannone in basso dovrà permettere di sparare bolle generate casualmente, muovendosi angolarmente a destra e sinistra.
- Le bolle sparate dal cannone potranno rimbalzare sui muri del campo di gioco se colpiti.
- Le bolle dovranno scoppiare alla formazione di gruppi di almeno 4 bolle dello stesso colore adiacenti, facendo cadere eventuali altre bolle sottostanti senza ulteriori sostegni.
- Il gioco dovrà dare una schermata di game over se le palline raggiungono il fondo dello schermo.

- Il gioco dovrà gestire un punteggio.

Requisiti non funzionali

- Il gioco dovrà fornire una esperienza fluida.
- Il gioco dovrà avere musica di sottofondo e effetti sonori per lo scoppio delle bolle e il game over.

1.1.1 Requisiti facoltativi

Questi requisiti non sono necessari al funzionamento di base del gioco e per questioni di tempistica potrebbero non essere implementati.

- Salvataggio e caricamento della partita
- Gestione di una leaderboard con i punteggi migliori
- Effetti sonori e musica
- Preview della prossima pallina che verrà caricata sul cannone e possibilità di scambiarla con quella caricata
- Diversi livelli di difficoltà
- Animazioni fluide
- Modalità 2 giocatori

1.2 Analisi e modello del dominio

L'entità principale in gioco è la Bubble, un insieme di Bubble formano una Grid. Una Grid dovrà poter essere generata in maniera casuale. Una bolla potrà essere sparata da un cannone, il quale potrà anche decidere di scambiarla con la successiva. Lo scoppio delle bolle dovrà inoltre far incrementare il punteggio.

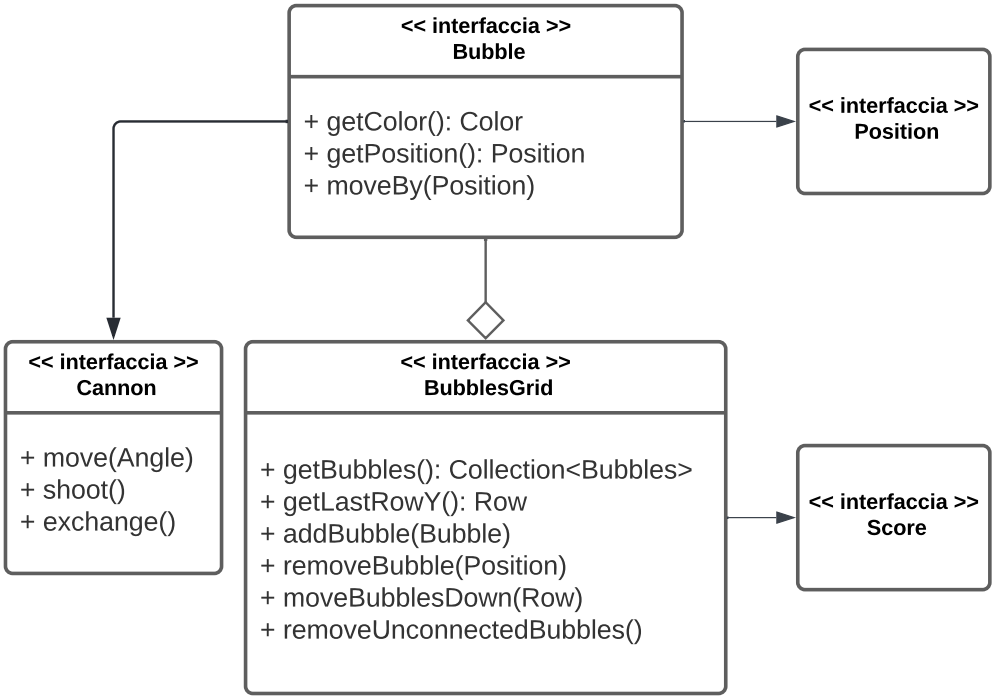

Figura 1.1: UML delle entità che rappresentano il dominio del problema.

Capitolo 2

Design

2.1 Architettura

Per questo progetto è stato scelto di fare uso del pattern MVC, che consente di separare in maniera efficace la logica del dominio da quella di visualizzazione e interazione con l'utente. L'interfaccia principale del Model è rappresentata dal Level che fornisce le informazioni necessarie alla rappresentazione di una partita. La view è nascosta da un interfaccia che non dipende dalla sua implementazione, in questo modo una sua sostituzione in blocco non dovrebbe comportare modifiche al resto dell'architettura dell'applicazione, questo perché il modello non è in alcun modo a conoscenza di come venga rappresentato all'esterno. Il punto d'ingresso del Model è il Cannon, che permette al Controller di modificare lo stato del gioco muovendo il cannone e sparando le bolle. Il tempo del gioco è scandito dal Controller. Il movimento del cannone è invece un evento generato dalla View, che viene passato al Model attraverso il controller e gestito in maniera asincrona.

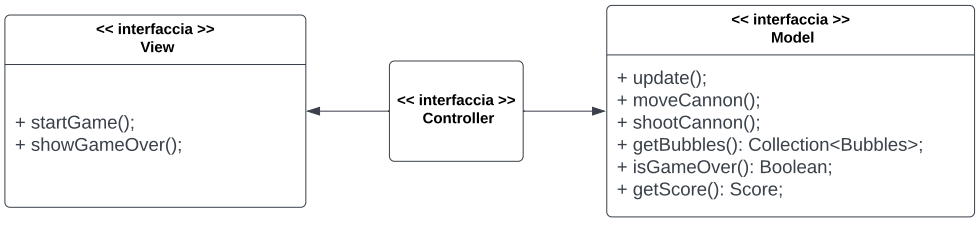

Figura 2.1: Schema UML architetturale del gioco. L'interfaccia `Controller` gestisce il flusso dell'applicazione, mentre la `View` gestisce l'interazione con l'utente e il `Model` fornisce le informazioni sul `Level`.

2.2 Design dettagliato

2.2.1 Nicolò Guerra

Caratteristiche della griglia di bolle

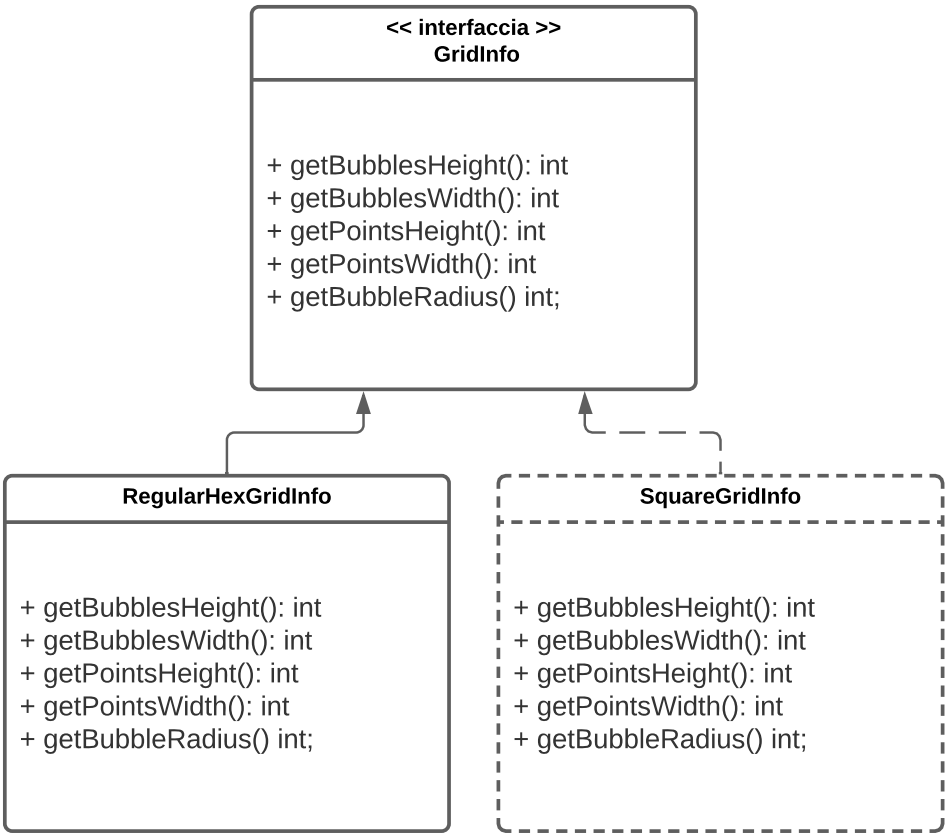

Figura 2.2: Rappresentazione UML dell'oggetto GridInfo

Problema Potenzialmente il gioco può gestire più tipi di griglie diverse per quanto riguarda dimensioni, forme delle caselle e altre caratteristiche, il che può portare a disallineamenti e incoerenze tra parti diverse del modello, oltre a una difficoltà di rappresentazione delle bolle nello spazio da parte della View, che non conosce griglie o altre entità specifiche del Model.

Soluzione Le caratteristiche della griglia vengono fornite da una interfaccia di tipo GridInfo implementata come *Strategy*, in questo caso ad esempio

la RegularHexGridInfo viene inizializzata con le dimensioni in bolle della griglia e si occupa autonomamente di effettuare i calcoli necessari per avere una griglia di esagoni regolari. Un'ulteriore implementazione potrebbe essere ad'esempio una SquareGridInfo che fornisce informazioni per una griglia a celle quadrate.

Gestione del GameOver

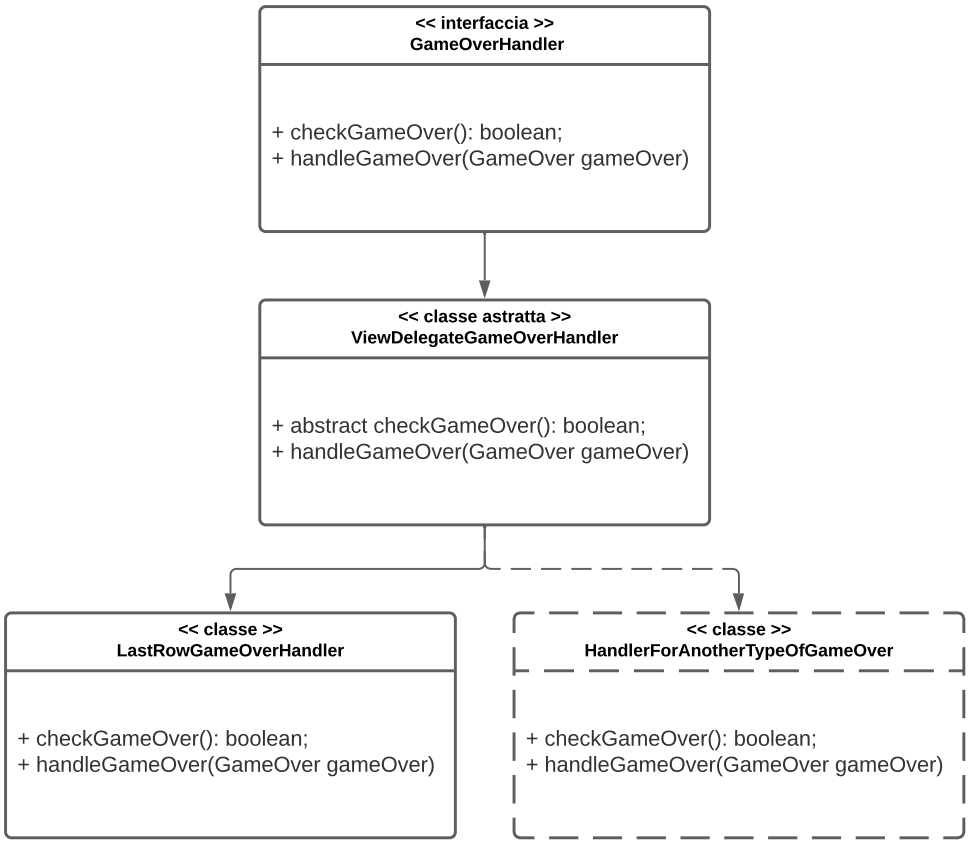

Figura 2.3: Rappresentazione UML del gestore dei GameOver

Problema I GameOver potrebbero essere generati in modi diversi (tempo scaduto, raggiungimento dell'ultima riga...) e voler essere gestiti in maniera diversa.

Soluzione Un interfaccia GameOverHandler con 2 metodi per controllare se è avvenuto un GameOver e per gestirlo, implementando questa interfac-

cia è possibile definire quando avviene un GameOver e come questo venga gestito. Per separare ulteriormente controllo e gestione una classe astratta ViewDelegateGameOverHandler che implementa la gestione passando l'evento di gameover alla View, e con un metodo astratto per controllare se è accaduto un gameover. Quest'ultimo viene implementato dalla sottoclasse LastRowGameOverHandler, la quale non fa altro che controllare se le bolle nella griglia hanno raggiunto l'ultima riga.

Salvataggio su file

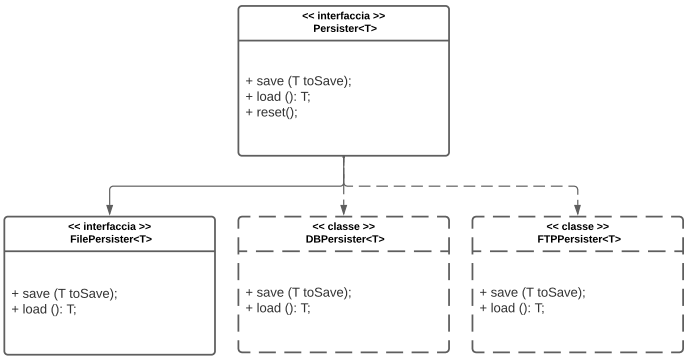

Figura 2.4: Strategy per un oggetto che salva oggetti

Problema Spesso l'applicazione ha necessita di salvare lo stato di alcuni suoi oggetti per poi recuperarlo in un secondo momento.

Soluzione Per poter salvare lo stato di oggetti diversi su disco ho creato un interfaccia generica persister che fa da strategy, con un metodo per salvare un oggetto e un metodo per leggerlo. L'implementazione di questa nasconde all'utilizzatore il modo in cui vengono persistiti. Si possono quindi creare ad esempio diverse implementazioni che salvano su file, su database o in cloud implementando i metodi read e write. Rispettando la stessa interfaccia le sottoclassi diventano intercambiabili tra di loro.

Caricamento degli assets della View

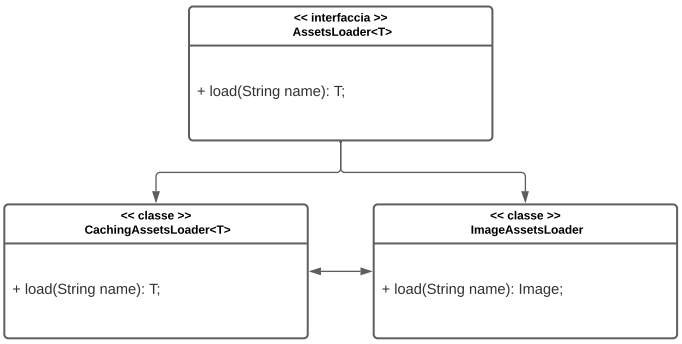

Figura 2.5: Decorator per cache di un assets loader

Problema L'implementazione grafica della View ha bisogno che le vengano forniti gli assets del gioco come le figure delle bolle e del cannone. Deve essere un caricamento efficiente perché gli assets possono anche essere richiesti più volte al secondo.

Soluzione Per delegare il caricamento degli assets a un altro componente è stata creata l'interfaccia AssetsLoader che ha un metodo in grado di caricare un assets la cui implementazione non è specificata. Una classe StandardAssetsLoader carica gli assets dalla cartella resources su richiesta e un CachedAssetsLoader la decora implementando un meccanismo di cache per ridurre gli accessi a disco e aumentarne l'efficienza. In questo modo è facile riutilizzare il meccanismo di cache per un AssetsLoader che carica altri tipi di assets.

Gestione del tempo di gioco

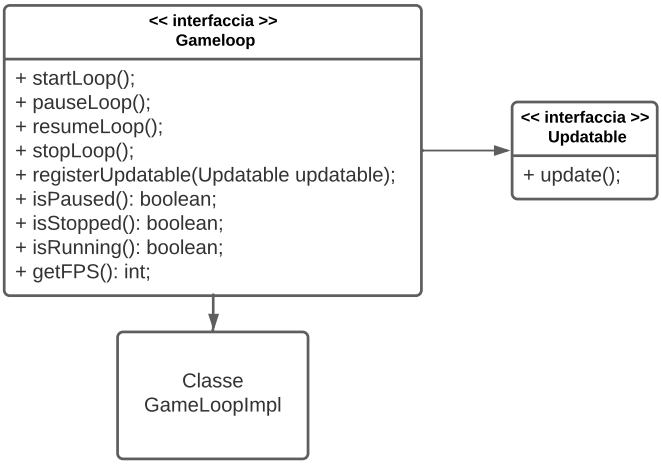

Figura 2.6: UML del Gameloop

Problema Il gioco ha bisogno di essere aggiornato a intervalli regolari, ad esempio per gestire il movimento delle palline o per far ridisegnare la View.

Soluzione È stato utilizzato il pattern Gameloop, creando un interfaccia che ne definisce le interazioni con il Controller, presso cui altri componenti che implementano Updatable possono registrarsi per essere aggiornati ad ogni ciclo.

2.2.2 Emma Leonardi

Gestione del campo di Bolle

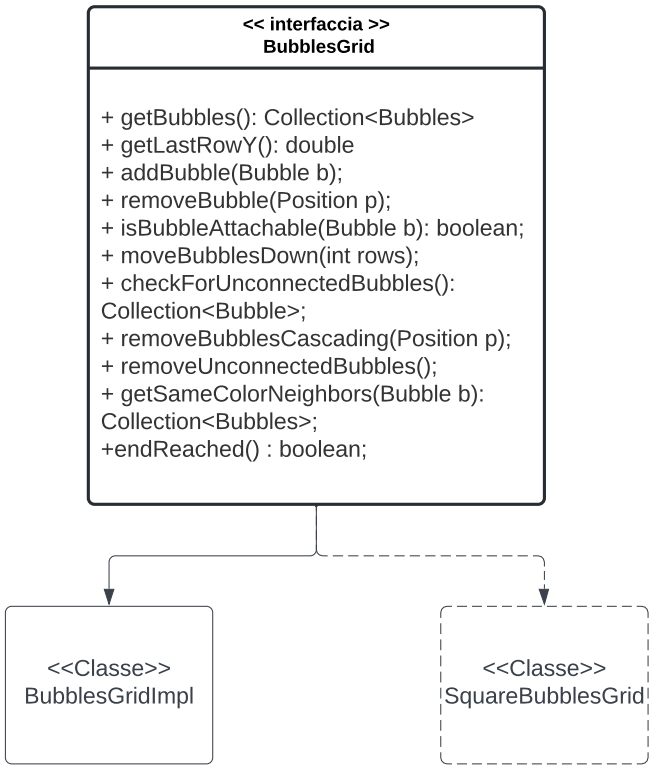

Figura 2.7: UML del BubblesGrid

Problema La griglia di Bolle deve essere indipendente dalla forma delle bolle

Soluzione Un interfaccia BubblesGrid con metodi che astraggono la forma delle bolle. In particolare l'implementazione di BubblesGridImpl è per griglie di esagoni con la punta verso l'alto, ma implementando l'interfaccia si possono creare griglie di qualsiasi forma di bolle.

Gestione di tre valori appaiati

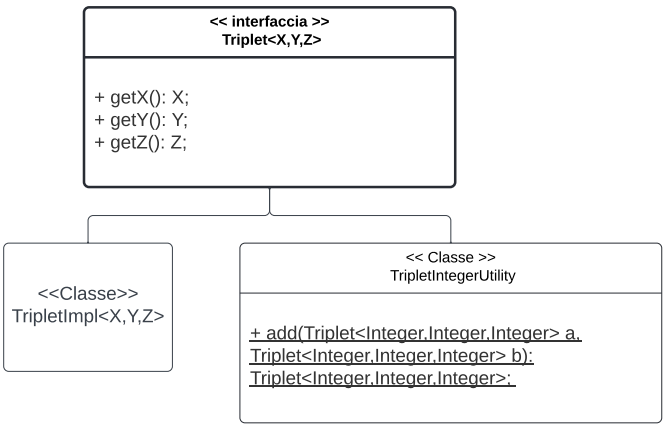

Figura 2.8: UML del Triplet

Problema Nella griglia ho usato coordinate con tre valori e avevo necessità di un oggetto per memorizzarle

Soluzione Ho creato un'interfaccia generica che supporta qualsiasi tipo di tripletta di valori, poi implementata in una classe generica. Per poter sommare i valori ho poi creato una classe con metodi statici che fa la somma delle triplette.

2.2.3 Filippo Casadei

Movimento e collisioni delle bolle

Problema Le bolle normalmente sono identificate da un colore e una posizione, ma hanno bisogno di una velocità per potersi spostare all'interno della griglia di gioco. Perciò è necessaria un tipo di bolla che può avere una sua velocità.

Soluzione Per risolvere questo punto è stata modellata una MovingBubble, che estende una bolla normale, che però è caratterizzata da una velocità e da varie operazioni su di essa.

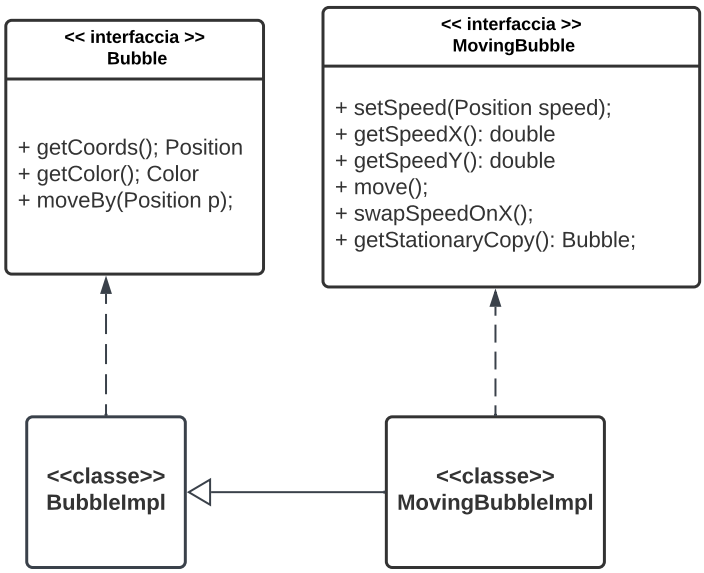

Figura 2.9: UML della MovingBubble

Problema L'applicazione si comporrà di una griglia di gioco, sulla quale saranno presenti a loro volta una griglia di bolle e la bolla che verrà sparata dal cannone, quest'ultima potrà muoversi liberamente finchè non verrà a contatto con le bolle già presenti, perciò bisogna controllare che gli spostamenti della bolla in movimento siano "legali" e che quando questa collida con la griglia di bolle si annetta ad essa. Inoltre la bolla dovrà poter rimbalzare sui muri laterali facendole cambiare direzione di movimento.

Soluzione L'entità che gestirà i movimenti della bolla sarà il MovementHandler, quest'ultimo avrà il compito di muovere la bolla solo in posizioni "legali" e nel caso stia per uscire dalla griglia gestisca il movimento della bolla riposizionandola coerentemente a seconda della sua velocità, facendola rimbalzare se necessario. Il MovementHandler deve sapere che bolla dovrà controllare, di base non c'è nessuna bolla da controllare.

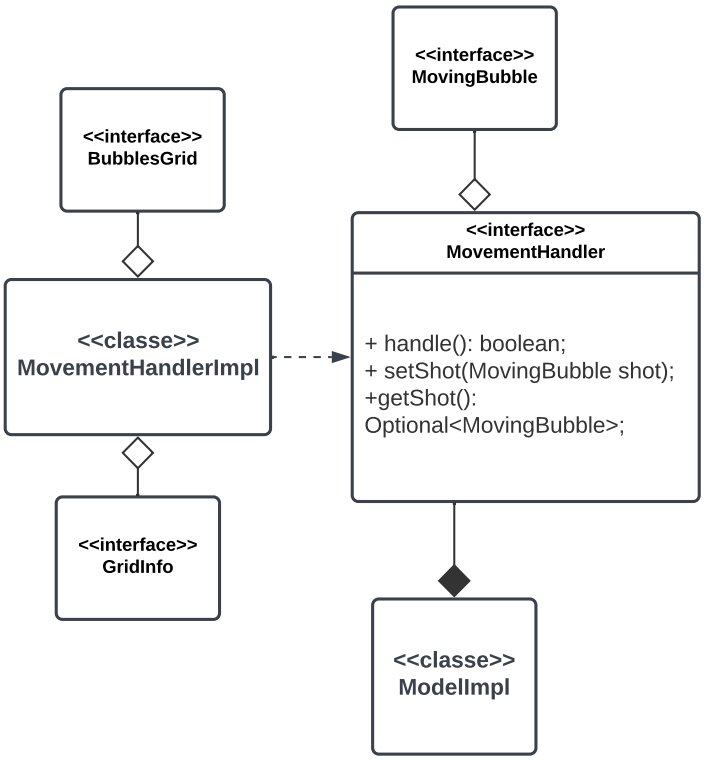

Figura 2.10: UML del MovementHandler

Gestione dei livelli

Problema L'applicazione dovrà creare ogni volta un nuovo livello che il giocatore potrà giocare, questo livello deve poter essere modificabile in base al livello di sfida che vorrà affrontare il giocatore.

Soluzione Il Level è stato modellato per contenere gli elementi fondamentali di una partita, ovvero è suo compito generare la nuova griglia e il cannone con cui il giocatore interagirà, inoltre contiene anche il valore numerico del punteggio attuale. Dato che implementa l'interfaccia Serializable è anche possibile salvare lo stato del livello in qualsiasi momento per poi riprendere da dove si era rimasti.

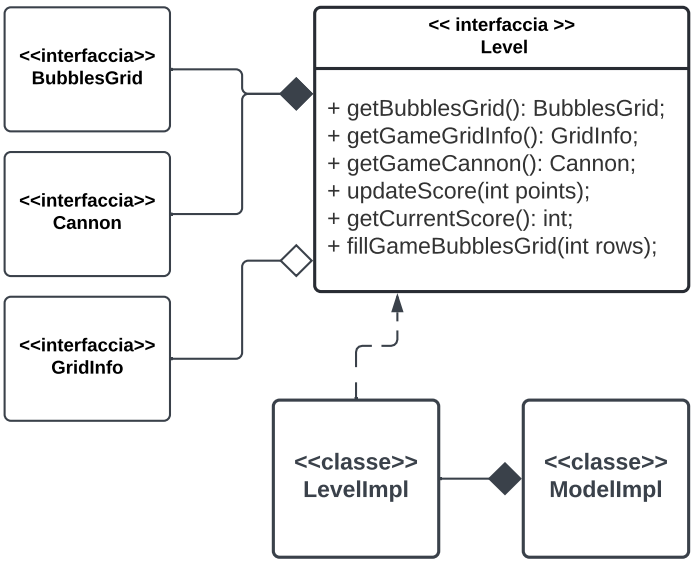

Figura 2.11: UML del Level

Rappresentazione grafica delle bolle e del campo di gioco

Problema L'utente per interagire correttamente con l'applicazione necessita di avere una corretta rappresentazione grafica delle posizioni delle bolle e del cannone, questa dovrà inoltre esser aggiornata coerentemente durante lo svolgimento della partita.

Soluzione Per aggiornare continuamente l'intera griglia la view principale farà utilizzo del CanvasDrawer, il quale ha il compito di ridisegnare ad ogni tick ogni elemento di gioco, per fare ciò delega il compito di disegnare le bolle, sia quelle della griglia sia quella in movimento, ad un BubbleDrawer, mentre il compito di disegnare il cannone alla giusta angolazione è delegato ad un CannonDrawer.

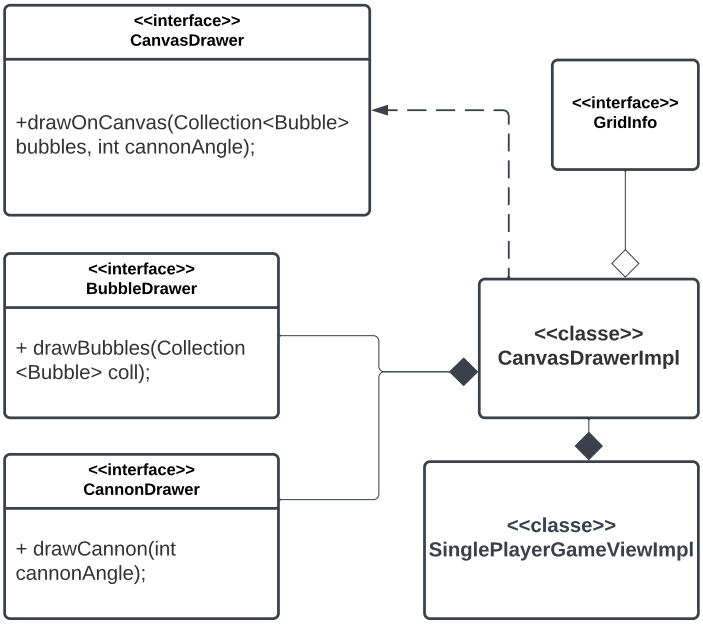

Figura 2.12: UML del CanvasDrawer

2.2.4 Lorenzo Tagliani

Generazione delle bolle

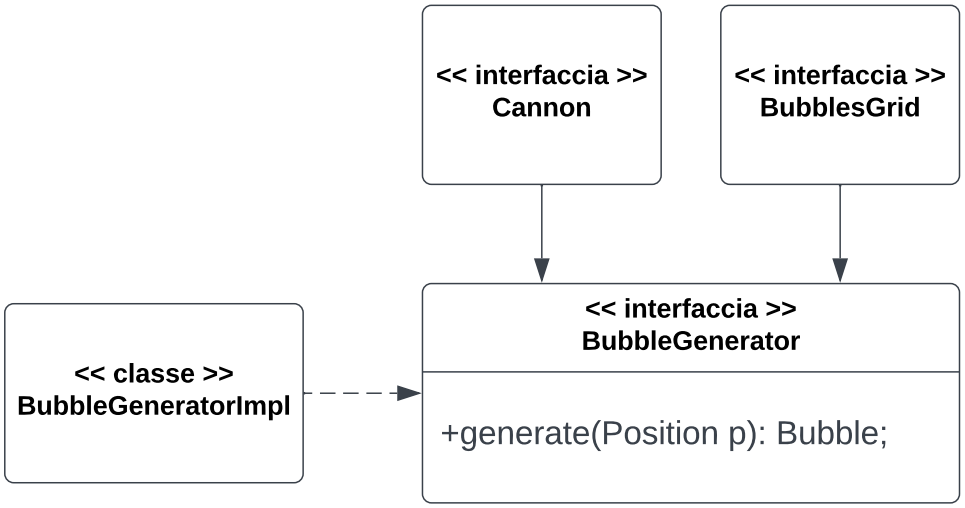

Figura 2.13: Rappresentazione UML dell'oggetto BubbleGenerator

Problema Nel progetto è stata necessaria, molteplici volte, una generazione casuale delle bolle.

Soluzione È stato creato un generatore che, basandosi sulla posizione della bolla da posizionare sulla griglia, sfrutta una lista di colori dichiarata per generare una bolla di colore casuale. Si è preferito fare un generatore come oggetto a sé stante invece di implementarlo nel cannone o nella griglia stessa.

Gestione dei punteggi

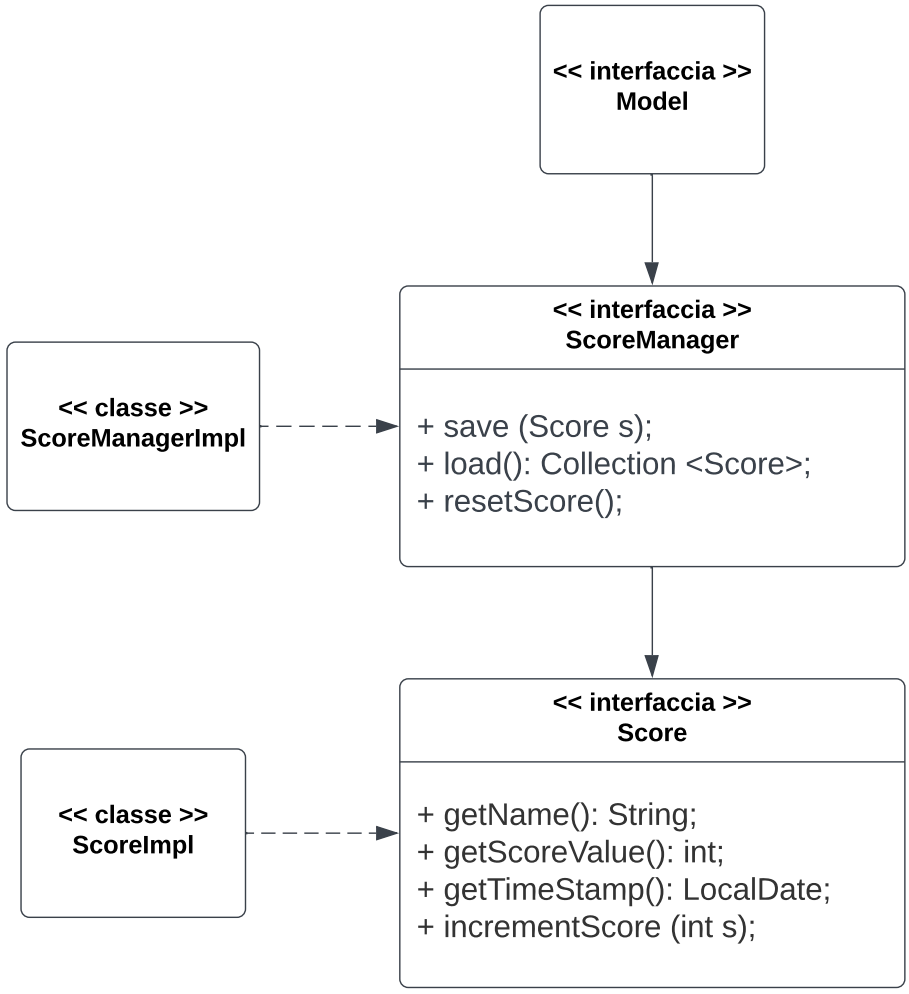

Figura 2.14: Rappresentazione UML della gestione dei punteggi

Problema Uno degli obiettivi pricipali del progetto era il salvataggio dei punteggi su file, ed al caricamento di essi a richiesta del sistema.

Soluzione È stato deciso che il punteggio ottenuto dall'utente dovesse essere composto di nome dell'utente, punteggio e data, per questo è stato creato l'oggetto Score, usato poi nello ScoreManager per poter salvare i punteggi su un file, tramite il File Persister creato da Nicolò Guerra. La lista viene

creata e gestita dalla classe ScoreTable, creata per semplificare e organizzare meglio il tutto.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Abbiamo usato JUnit 5 per fare test automatici delle classi. Abbiamo testato le seguenti classi:

- GameLoop
- RegularHexGrid
- Level
- BubbleGenerator
- BubblesGrid
- Bubble
- Cannon
- COLOR
- MovementHandler
- MovingBubble
- Persister
- Position
- Score
- ScoreManager

- Triplet
- TripletIntegerUtility
- VectorConverter

3.2 Metodologia di lavoro

Per lavorare in parallelo su parti diverse del software abbiamo iniziato definendo le interfacce principali dell'applicazione, per poi procedere a sviluppare ognuno a valle delle interfacce che costituivano la sua parte di progetto. Abbiamo utilizzato branch separati per le feature effettuando il merge sul main alla conclusione di ogni parte di progetto.

3.2.1 Nicolò Guerra

In questo progetto i miei compiti sono stati:

- Gestire l'avvio dell'applicazione e la connessione tra i componenti MVC (package bbblast.application)
- Gameover (package bbblast.controller.gameover)
- Tempo di gioco (package bbblast.controller.gameover)
- Informazioni sulla griglia (GridInfo e RegularHexGridInfo nel package bbblast.model)
- Persistenza (package bbblast.utils.persister)
- Menu principale (package bbblast.utils.menu)
- View per le opzioni (package bbblast.view.options)
- Caricamento degli assets (package bbblast.view.singleplayer.assetsloader)

3.2.2 Emma Leonardi

In questo progetto i miei compiti sono stati:

- Gestire il comportamento di Griglie di bolle (package bbblast.model BubblesGrid e BubblesGridImpl)
- Bolle statiche (package bbblast.model Bubble e BubbleImpl)

- Comportamento del cannone (package bbblast.model Cannon e CannonImpl)
- Grafica del cannone (package bblast.view.singleplayer CannonDraweImpl e CannonDrawer)
- Triplette di valori (package bbblast.utils Triplet, TripletImpl)
- Somma di Triplette di valori interi (package bbblast.utils TripletIntegerUtility)
- Conversione velocità vettoriale in componenti (package bbblast.utils VectorConverter e VectorConverterImpl)

3.2.3 Filippo Casadei

In questo progetto i miei compiti sono stati:

- Bolle dinamiche (MovingBubble e MovingBubbleImpl, package bbblast.model)
- Gestione del movimento e delle collisioni (MovementHandler e MovementHandlerImpl, package bbblast.model)
- Divisione in punti (Position e PositionImpl, package bbblast.utils)
- Grafica delle bolle (BubbleDrawer e BubbleDrawerImpl, package bbblast.view.singleplayer)
- Disegnare la grafica di gioco (CanvasDrawer e CanvasDrawerImpl, package bbblast.view.singleplayer)
- Creazione del livello di gioco (Level e LevelImpl, package bbblast.model.level)

3.2.4 Lorenzo Tagliani

In questo progetto i miei compiti sono stati:

- Generazione delle bolle sul campo (BubbleGenerator e BubbleGeneratorImpl, package bbblast.model)
- Scrittura, salvataggio e caricamento dei punteggi (ScoreManager e ScoreManagerImpl, package bbblast.utils)
- Struttura dello score (Score e ScoreImpl, package bbblast.utils)

- Strutturazione della lista di score (ScoreTable, package bbblast.utils)
- Gestione dell'aggiornamento in tempo reale dello score

3.3 Note di sviluppo

3.3.1 Nicolò Guerra

- Progettazione di classi generiche
- Lambda expressions
- Stream
- Optional
- JavaFX
- Thread

3.3.2 Emma Leonardi

- Stream
- Progettazione di classi generiche
- Lambda expressions
- JavaFX

Per la gestione delle coordinate su tre assi ho usato lo pseudocodice di Amit Patel `https://www.redblobgames.com/grids/hexagons/` e ho usato sempre il suo codice per cercare gli esagoni vicini.

3.3.3 Filippo Casadei

- JavaFX
- Optional

3.3.4 Lorenzo Tagliani

- JavaFX
- Lambda expressions

Capitolo 4

Commenti finali

In quest'ultimo capitolo si tirano le somme del lavoro svolto e si delineano eventuali sviluppi futuri.

4.1 Autovalutazione e lavori futuri

4.1.1 Nicolò Guerra

Non sono soddisfatto del progetto che abbiamo svolto. Purtroppo abbiamo avuto diversi problemi già a partire dalla parte di design generale dell'applicazione, complice anche l'inesperienza essendo la prima volta che ognuno di noi portava avanti un progetto che tutto sommato comincia ad avere dimensioni importanti. Ci sono stati alcuni problemi anche nella coordinazione interna del team, e sicuramente questo non ha influito positivamente sul risultato finale. Probabilmente avremmo dovuto iniziare scegliendo un progetto diverso da sviluppare e una divisone di compiti diversa, cosa che purtroppo non potevamo sapere prima di iniziare il progetto. Avremmo dovuto inoltre soffermarci meglio sulle interazioni tra le diverse parti del software. Sono aspetti a cui starò molto più attento una prossima volta.

4.1.2 Emma Leonardi

Sono soddisfatta del mio lavoro individuale, ma non del progetto in totale. Ci sono stati parecchi problemi sulle parti del codice in comune, perchè ci sono stati dei problemi di comunicazione e incomprensioni, spesso dovute a una prima fase di design poco precisa. Questo mi ha rallentato nello sviluppo della mia parte di progetto e a tratti ha costretto a riscrivere parti di codice frutto di aspettative diverse di implementazione. Non ha aiutato la coordinazione del progetto anche l'inesperienza, che ci ha fatto sottovalutare alcuni aspetti

che si sono rivelati problemi implementativi. In futuro cercherò di spiegarmi meglio sulle caratteristiche del software prodotto da me e comunicare meglio con i miei compagni di progetto.

4.1.3 Filippo Casadei

Non sono soddisfatto di come è stato svolto il progetto. Sin dall'inizio della progettazione ci sono state numerose incomprensioni, che hanno portato a dover discutere più volte punti già trattati e capire come risolvere i problemi generati, rallentando di molto il lavoro di tutti i componenti. A causa di questi problemi, a cui si aggiunge anche l'inesperienza di lavorare in progetti più complessi con un team di sviluppo, la qualità del risultato finale è indubbiamente calato. Sicuramente però è stata un'esperienza formativa considerando che da ogni sbaglio commesso si è potuto comprendere come sarebbe stato meglio agire, ad esempio una miglior comunicazione iniziale tra i membri definidendo più nello specifico gli aspetti dell'applicazione ed una miglior suddivisione dello sviluppo del software.

4.1.4 Lorenzo Tagliani

Non sono soddisfatto sia dello svolgimento del progetto, sia del mio lavoro individuale. Abbiamo riscontrato molti problemi nella condivisione del codice comune, dovuti alla nostra poca esperienza e poca organizzazione. Il grande vantaggio di questo progetto è l'esperienza che ne abbiamo guadagnato, la consapevolezza delle difficoltà organizzative e tecnice che abbiamo riscontrato non può che aiutarci per i futuri lavori. Devo, comunque, dirmi soddisfatto, nel finale, nel vedere il lavoro complessivo, per quanto disorganizzato e complesso, funzionare.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Nicolò Guerra

Sicuramente la caratteristica che ho apprezzato del corso è la disponibilità dei docenti, le correzioni in laboratorio allo svolgimento di ogni esercizio risultano molto utili per migliorare la tecnica. Sarebbe interessante approfondire meglio la parte di analisi di dominio di un problema e di design di una soluzione, al di la di quella che è la programmazione Java, che forse è più semplice da capire.

4.2.2 Emma Leonardi

Mi è piaciuta la struttura del corso, i laboratori e l'attenzione dei docenti verso gli studenti. Ho trovato di difficile comprensione gli aspetti di Programmazione ad Oggetti avanzati, come Stream e Pattern e avrei voluto gli si fosse dedicato più tempo.

4.2.3 Filippo Casadei

Il corso è stato molto interessante, i docenti hanno avuto la capacità di motivare gli studenti allo studio della materia e i laboratori si son rivelati esser di grande aiuto per i temi trattati nelle lezioni teoriche. L'unico punto da me non ben compreso sono gli aspetti riguardanti i pattern, il loro utilizzo e come riconoscere la necessità di utilizzarli.

4.2.4 Lorenzo Tagliani

Ho trovato molto interessante il corso, sia per la struttura che per l'esposizione degli argomenti da parte dei professori, i quali riuscivano a motivare, più di altri, allo studio. Personalmente ho avuto alcune difficoltà negli ultimi argomenti trattati quali Lambda e Stream.

Appendice A

Guida utente

Il gioco parte con una schermata inizale, navigabile con il mouse.

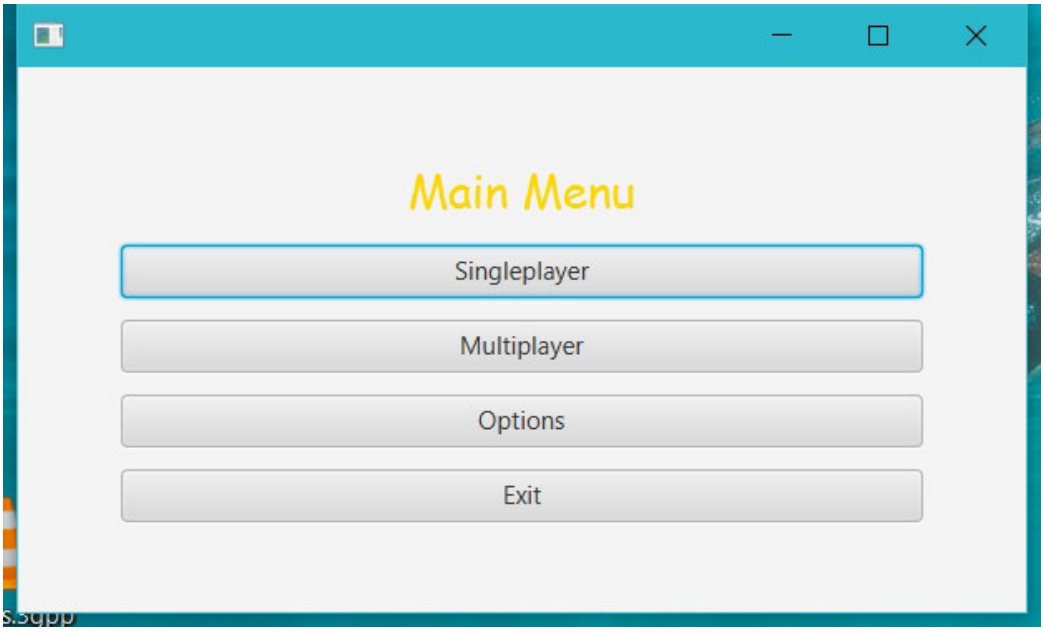

Figura A.1: La schermata iniziale

Il tasto singleplayer fa partire il gioco vero e proprio. Il tasto multiplayer porta ad una schermata di errore, perchè per motivi di tempo non è stato implementato.

Figura A.2: La schermata di opzioni

Il tasto opzioni porta alla schermata di opzioni, con due slider per musica e effetti sonori. Le opzioni sono salvabili su file, ma per problemi implementativi non essendoci nel gioco ne' musica ne' effetti sonori, non cambiano nulla nel gioco. Il tasto exit fa uscire dall'applicazione.

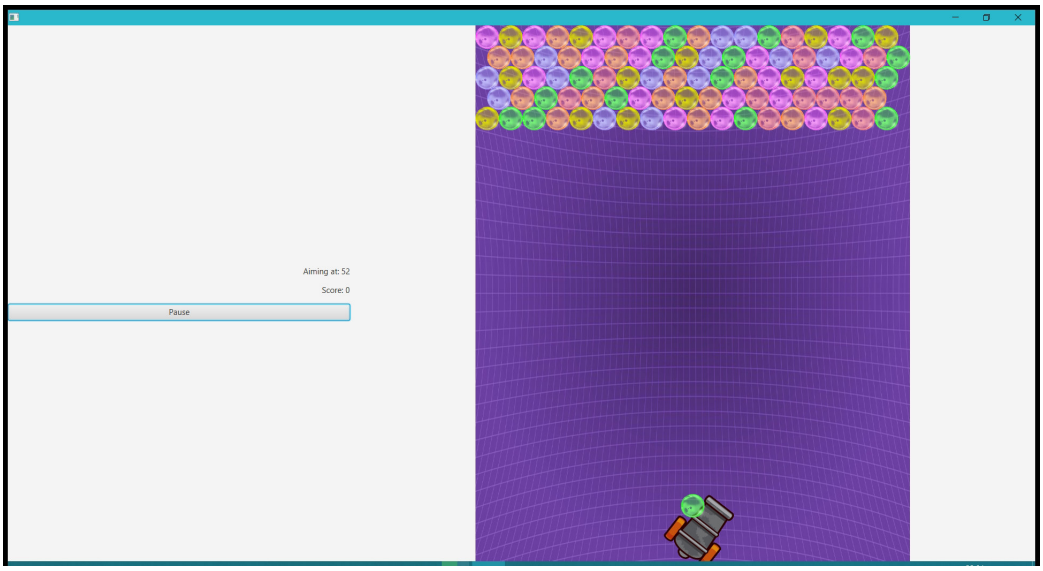

Figura A.3: La schermata di gioco

Il cannone, posto in basso e nel centro della schermata, si comanda con i tasti 'A' e 'D' che rispettivamente lo fanno girare verso destra e sinistra. La bolla viene fatta sparare con il tasto 'W'. La bolla una volta sparata viaggia in linea retta e rimbalza se colpisce le pareti del campo di gioco. Le bolle si uniscono alla griglia e al raggiungimento di 4 o più bolle dello stesso colore, scoppiano aumentando il punteggio. Se si formano gruppi di bolle "volanti" ovvero senza nessuna bolla che li collega alla cima dello schermo, quando si elimina l'ultima bolla di sostegno anche le bolle volanti scoppiano. Il gioco finisce quando le bolle raggiungono i 4/5 dello schermo. A game over, viene presentata una schermata per salvare il punteggio. Si viene poi riportati nella schermata iniziale

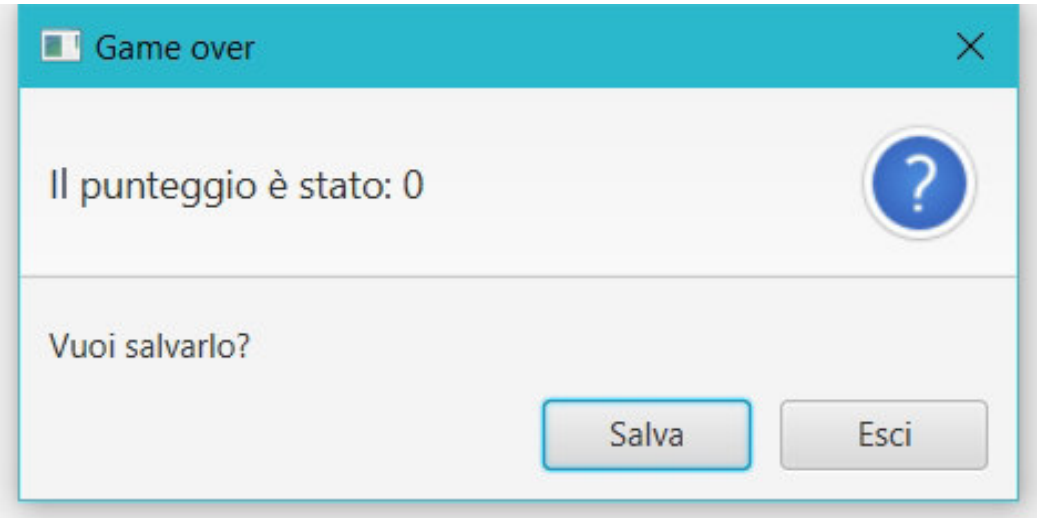

Figura A.4: Il messaggio di GameOver

Appendice B

Esercitazioni di laboratorio

In questo capitolo ciascuno studente elenca gli esercizi di laboratorio che ha svolto (se ne ha svolti), elencando i permalink dei post sul forum dove è avvenuta la consegna.

B.0.1 Nicolò Guerra

- Laboratorio 07: `https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p137443`
- Laboratorio 08: `https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p138741`

B.0.2 Emma Leonardi

- Laboratorio 05: `https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p135211`
- Laboratorio 06: `https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p135213`
- Laboratorio 07: `https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p136445`
- Laboratorio 08: `https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p138799`
- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p138800`

- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p139973