Relazione per Programmazione ad Oggetti "Polycut"

Alessandro Previati, Mathieu Maugin, Karim El Kholy

1 Giugno 2023

Indice

1 Analisi

Il software mira alla costruzione di una imitazione del gioco sviluppato da 'Halfbrick Studio' chiamato 'Fruit Ninja'. Presentato nel 2010 al pubblico ha guadagnato consensi in tutto il mondo, diventando uno dei giochi mobile più famosi e scaricati. Il nostro obiettivo è quello di replicarlo fedelmente, per quanto sia possibile, con qualche piccola modifica ideata da noi.

1.1 Requisiti

1.1.1 Requisiti funzionali

Il funzionamento dell'applicazione dovrà essere in grado di simulare il comportamento del gioco originale, ovvero dovrà essere in grado di gestire il lancio dei poligoni, gli originali frutti, e la lama, in questo caso il mouse. Il punteggio aumenta sulla base del numero di poligoni tagliati, ma bisogna fare attenzione alle bombe perchè esse faranno perdere una vita. Come le bombe, anche il non tagliare un poligono prima che esso caschi per terra farà perdere una vita, bisognerà stare attenti poiché sono solo 3. Il movimento degli oggetti dovrà essere realistico seguendo la giusta traiettoria, influenzata dalla gravità e dalla difficoltà sempre più crescente. Deve essere presente un menù iniziale di scelta, con la possibilità di giocare, leggere le regole, oppure uscire dal gioco.

1.1.2 Requisiti non funzionali

Il software dovrà essere sviluppato correttamente ed in modo efficiente, senza causare un gioco troppo lento e che consumi troppa memoria, altrimenti sarebbe quasi impossibile giocarci.

1.2 Analisi e modello del dominio

Polycut dovrà essere in grado di gestire la creazione di diversi poligoni regolari, tra i quali quadrati, pentagoni, esagoni etc, gestendo anche in maniera realistica il loro movimento. Inoltre, deve gestire correttamente la lama utilizzata dal giocatore per tagliare i diversi oggetti a schermo, muovendo il mouse su di essi. Dovrà essere in grado di gestire correttamente punteggio e vite, sulla base delle bombe tagliate e dei poligoni tagliati o caduti. Fondamentale sarà la creazione di un mondo di gioco, un'interfaccia grafica che permetterà all'utente di interagire coi poligoni e visualizzare il punteggio e le vite correnti. Il gioco inizialmente mostrerà un menù iniziale che conterrà 3 tasti: 'Play', 'Rules' ed 'Exit': alla pressione del pulsante Play la schermata corrente verrà

chiusa e sarà avviato il gioco, aprendo un'altra scheda. Alla pressione di Rules verrà aperto un pop-up contenente le regole del gioco, mentre Exit chiuderà correttamente l'applicazione.

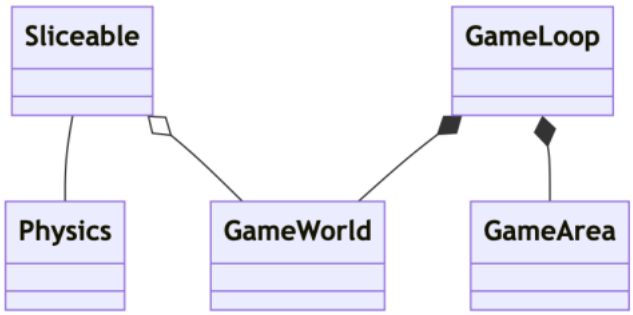


Figura 1: Schema UML del dominio dell'applicazione

2 Design

2.1 Architettura

L'architettura di Polycut segue il pattern architetturale MVC. Si è deciso di utilizzare questa tipologia di architettura per semplificare lo sviluppo. Utilizzando MVC i vantaggi sono molteplici, si passa dall'avere la possibilità di organizzare facilmente tutte le classi presenti nel software, ad avere la possibilità di aggiornare facilmente la parte di 'view', in quanto scollegata dal 'model'. Questo permette possibili aggiornamenti futuri, sulla parte visibile del software, andando a modificare, idealmente, nulla della parte logica del progetto. 'Polycut' ha diverse classi di model che devono essere gestite e poi rappresentate tramite una GUI. Sono presenti input, gestiti dal giocatore tramite il mouse, che creano la 'Blade'. Oggetti che vengono generati dal gioco, che fanno parte della logica del programma, e tutte quelle classi che collegano input del giocatore e oggetti tagliabili fanno parte del controller. La parte di 'view' non è legata a nulla, se viene modificata o cambiata interamente un buon progetto MVC non deve cambiare la propria parte di modello. Una view sviluppata in Swing, in questo caso, ci permette un design minimale ma funzionale per il menù e la grafica di gioco.

2.2 Design dettagliato

Alessandro Previati: La gestione degli oggetti Sliceable, ovvero gli oggetti tagliabili devono essere divisi in Bomb e Polygon. Bisogna trovare un modo efficiente per creare le due sottoclassi di Sliceable, ma senza ricorrere a ripetizioni all'interno del modello del progetto. Per gestire questo problema ho deciso di utilizzare il Template pattern. Mi permette di creare l'interfaccia Sliceable, che poi viene implementata in due modi differenti dalle classi Bomb e Polygon, sulla base di ciò di cui ho bisogno in quel momento di creare. Questa suddivisione servirà poi nella gestione da parte del controller per l'interazione con input dell'utente e per la GUI.

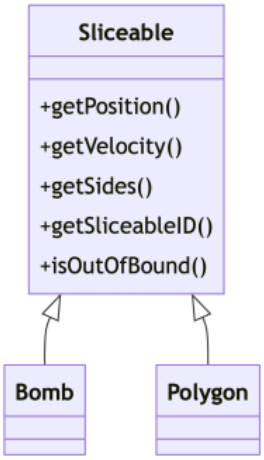


Figura 2: Schema UML del modello Sliceable secondo il pattern Template

Una seconda problematica che bisognava risolvere era il fatto di dover creare molti elementi della stessa classe, ovvero dover creare molti oggetti di tipo Sliceable, che dovranno essere bombe o poligoni. L'idea che ho deciso di mettere in atto è stata utilizzare un Factory Pattern. Questo pattern mi permette di avere una classe di controllo su tutti gli oggetti che devo creare, utilizzabile dal controller dell'applicazione. Il pattern Factory unito al Template precedente citato ed usato mi permette di avere una creazione semplice ed efficace dei diversi tipi di oggetti di cui il controller abbia bisogno, in qualsiasi momento.

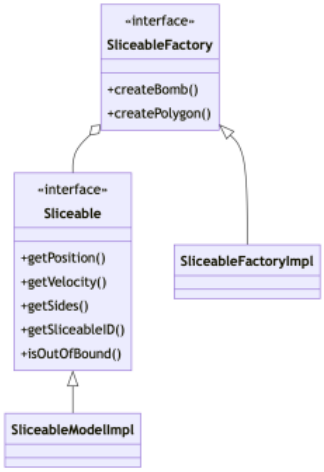


Figura 3: Schema UML del modello Sliceable secondo il pattern Factory

La gestione della fisica degli oggetti può rivelarsi più complessa del previsto. Lo Strategy Pattern permette di creare più implementazioni dell'interfaccia della fisica in modi diversi. Se nel futuro si volesse modificare il mondo di gioco, ad esempio aggiungendo l'effetto del vento, oppure una gravità diversa, mi basterebbe modificare l'implementazione della classe Physics, modificando i calcoli da effettuare. Attraverso questo procedimento si rende il codice riutilizzabile e molto più facilmente modificabile.

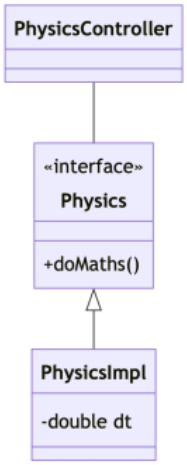


Figura 4: Schema UML della fisica

Mathieu Maugin: Appena avviato il gioco si presenta attraverso un menù basilare, provvisto di 3 pulsanti di default: Play, Rules ed Exit. Il menù è stato realizzato tramite il pattern 'Decorator', poiché grazie a questo template sarà più semplice in futuro aggiungere ulteriori pulsanti (e.g. selettore di difficoltà, di power-up, ecc..). I 3 pulsanti, ovviamente, avranno azioni diverse: Rules farà comparire un popup contenente le regole del gioco, Exit permetterà all'utente di chiudere correttamente l'applicazione, liberando la memoria associata mentre il pulsante Play chiuderà la finestra del menù e aprirà quella di gioco effettiva. Per quanto riguarda l'aggiunta di nuovi pulsanti basterà introdurre ulteriori classi che estendono la classe MenuDecorator.

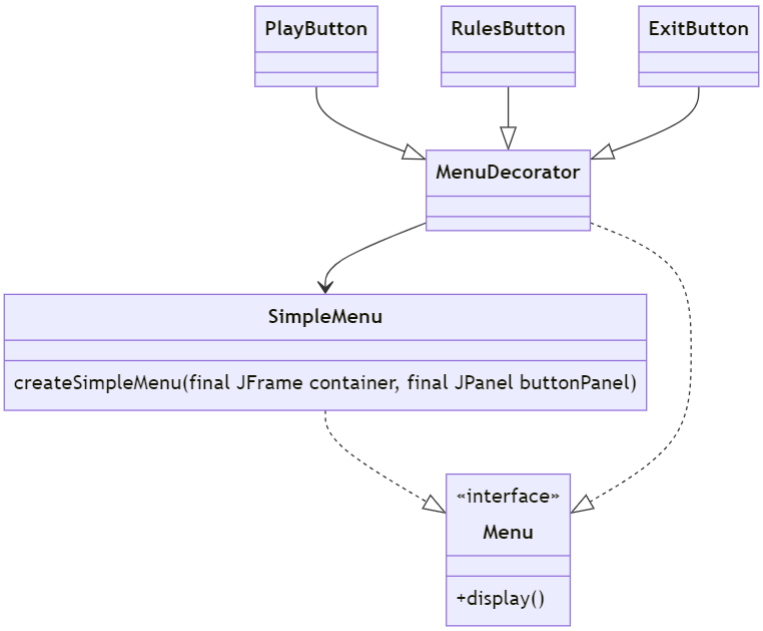


Figura 5: Schema UML del menù secondo il pattern Decorator

Il gioco prevede un metodo di input, in questo caso il mouse, da parte dell'utente in modo tale da poter "tagliare" i poligoni a schermo. Inizialmente avevo previsto una classe dedicata all'input che seguisse il pattern Observer, la quale lama (Blade) era l'elemento "Subject", mentre Sliceable gli "observer", così da poterne rendere più facile la gestione (e.g: il poligono viene tagliato e aumenta il punteggio o per effetto di un power-up tutti i poligoni sullo schermo vengono tagliati).

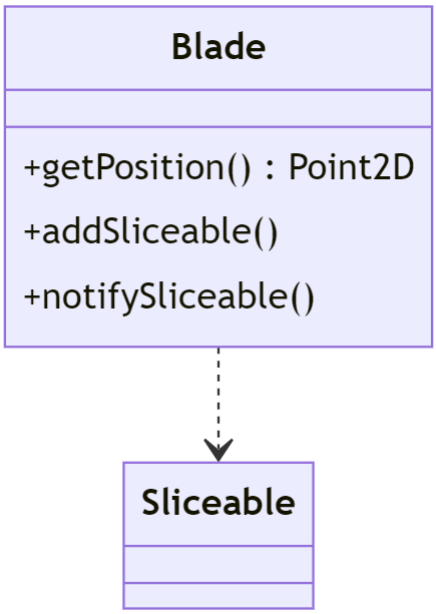


Figura 6: Schema UML di Blade secondo il pattern Observer

Un problema non indifferente è quello di rappresentare graficamente gli elementi sullo schermo secondo la fisica impostata dal modello. Come framework di sviluppo grafico abbiamo utilizzato Java Swing e AWT, mentre abbiamo implementato le classi che ci permettono di rappresentare i rispettivi oggetti creati nel modello.
N.B: questa parte è stata sviluppata in collaborazione con Karim El Kholy, poichè come stabilito nella proposta di progetto entrambi ci saremmo dovuti occupare su aspetti prettamente grafici.
Un altro problema è stato quello di dover far comunicare la parte di grafica con la parte del modello, perciò ho implementato, secondo il pattern MVC, i relativi controlli che mi permettono di gestire in modo facile e sicuro la parte relativa alla fisica e alla gestione del mondo di gioco.

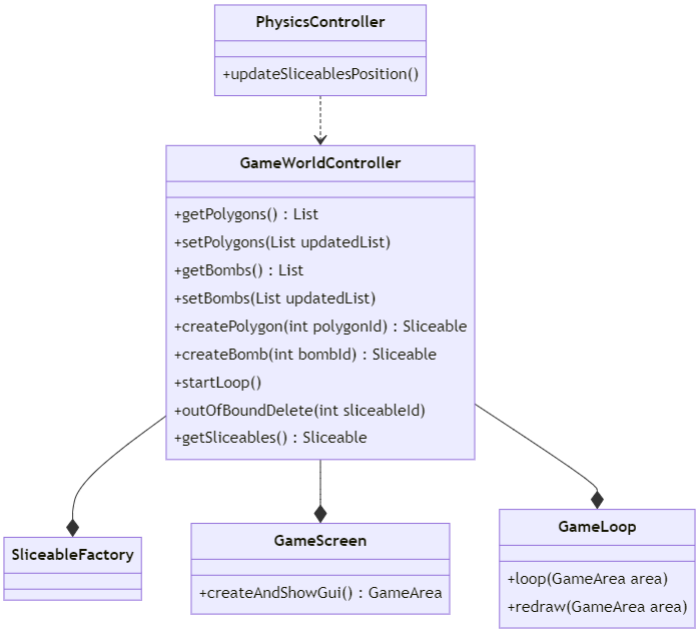


Figura 7: Schema UML di GameWorldController

Karim El Kholy: Lavorando principalmente all'espetto grafico di PolyCut ho potuto decidere come rappresentare i vari elementi del gioco assicurandomi di fornire sufficienti parametri per facilitare la comunicazione fra le componenti MVC del progetto. Problema: Gestire correttamente gli elementi nella schermata di gioco assicurandomi di dividerli in modo opportuno alle caratteristiche del gioco e mantenendo uno stile grafico funzionale al gameplay. Soluzione: Analizzando i requisiti del gioco la soluzione migliore sembrava essere quella di costruire la schermata di gioco componendo il Frame principale tramite le tre classi: GameAreaImpl (estensione di JPanel) per la zona di gameplay, LiveImpl e ScoreViewImpl (estensioni di JLabel) per mostrare lo stato di gioco.

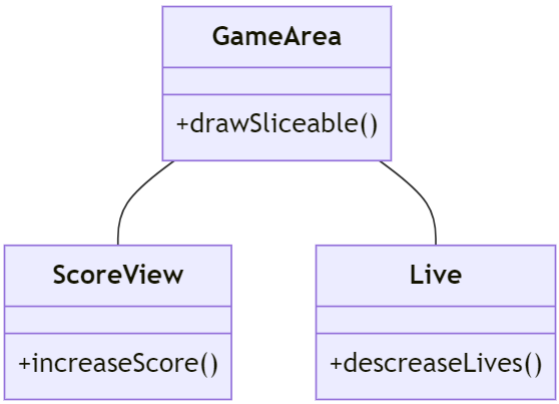


Figura 8: Schema UML di come l'area di gioco gestisce vite e punteggio

Problema: La corretta gestione degli elementi a schermo; è necessario che il frame principale comunichi con le sue componenti per assicurarsi che queste aggiornino il loro stato. Soluzione: Ho utilizzato il pattern Observer a tal proposito. La classe GameAreaImpl (subject) notifica le altre classi grafiche (observers) LiveImpl e ScoreViewImpl quando il poligono o le bombe interagiscono con il cursore e ne cambiano lo stato utilizzando i loro metodi.

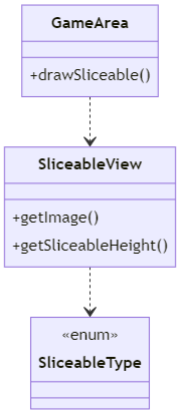
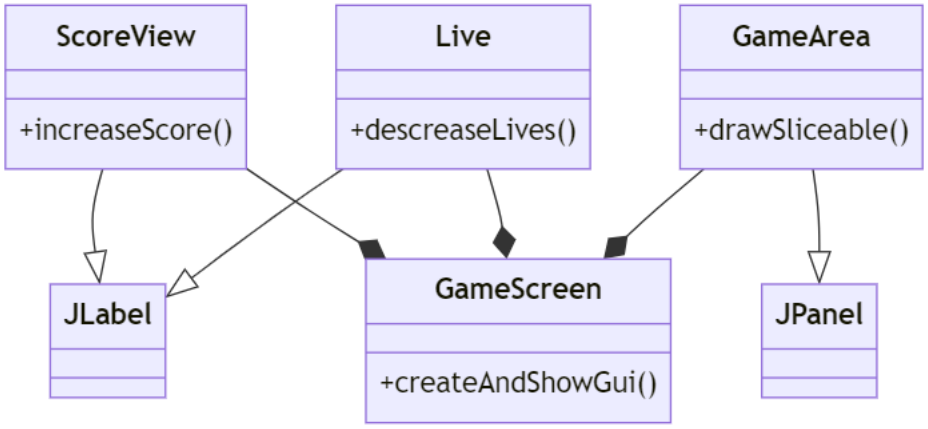


Figura 9: Schema UML di come l'area di gioco gestisce gli Sliceable

Problema: decidere come rappresentare i poligoni e le bombe nel gioco e come gestire la collisione con il cursore. Soluzione: soluzione ho deciso di semplificare il più possibile la rappresentazione dei modelli; la soluzione è stata creare dei file PNG di ogni modello di Sliceable e gestirli all'interno del Frame principale 'incapsulandoli' in dei JLabel, questa soluzione risulta inoltre molto comoda in quanto le 'hitbox' degli Sliceable sono

già disponibili essendo i bordi delle immagini stesse. Siccome ogni PNG ha caratteristiche diverse mi sono servito di una strategia che utilizza 2 pattern per disegnare l'oggetto: Factory per ottenere l'ImageIcon + le dimensioni per il JLabel e Strategy assieme a SliceableTypeEnum per eseguire la corretta gestione degli eventi basandomi sul tipo di Sliceable.



3 Sviluppo

3.1 Testing automatizzato

I componenti che abbiamo deciso di sottoporre a test sono principalmente quelli appartenenti al modello del software. Tra i quali troviamo:

- **Polygon:** Abbiamo creato la prima classe di test sui poligoni, questa dovrà testare tutti i metodi che sono presenti all'interno della suddetta classe, quali i getter ed i setter di posizione e velocità ad esempio. N.B: l'omissione di una classe speculare per la classe 'Bomb' è voluta, essendo le due classi sosia l'una dell'altra le funzioni sono le stesse.

- **SliceableFactory:** La classe di test della Sliceable Factory serve per controllare che i metodi di creazione della factory siano stati implementati correttamente, e che gli oggetti creati siano diversi e rispettino i parametri dati.

- **Physics:** I test della fisica degli oggetti sono costruiti in modo tale che ogni oggetto debba compiere gli spostamenti corretti, preso un determinato arco di tempo, e posizioni e velocità inventati.

3.2 Metodologia di lavoro

N.B: Le classi sotto elencato sono quelle progettate e scritte principalmente dallo studente possessore della sottosezione.

Alessandro Previati: Nel mio campo di gestione stabilito alla proposta di progetto avevamo deciso che sarei stato responsabile della creazione di: modello della fisica degli oggetti, game loop, e gestione della creazione degli oggetti, quali poligoni e bombe. Proprio per questo gran parte del mio lavoro è stato nelle sezioni di model e controller.

- **Model:**
  - SliceableFactory, Sliceable

    Ho creato interfaccia ed implementazione di entrambe le classi. Per gestire il problema della creazione di più oggetti di tipi diversi, come spiegato nella sezione di design ho preferito utilizzare un pattern Factory. Mi ha permesso di rendere casuale la creazione degli oggetti che verranno poi utilizzati dalla parte di controller. La casualità viene gestita semi-randomicamente allo svolgimento dei calcoli per la creazione di uno Sliceable.
  - Polygon, Bomb

    Dopo avere creato la classe Sliceable e la Factory corrispondente ho utilizzato il Template pattern per creare due diversi tipi di oggetti che implementeranno Sliceable, ovvero i poligoni e le bombe, gli oggetti che dovranno essere alla base del gioco. L'idea iniziale era quella di calcolare il centro del poligono creato ed utilizzando il raggio calcolare tutti i punti presenti all'interno del poligono, per poi disegnarlo attraverso la grafica di Swing. Questa soluzione si è rivelata poco fruttuosa, e soprattutto più complessa da disegnare. Dopo ulteriori analisi e progettazione abbiamo ripiegato su una soluzione più efficiente e meno difficile da implementare. Per disegnare uno Sliceable, che siano poligoni o bombe basta la posizione di esso in un determinato istante di gioco, a quel punto incolliamo l'immagine creata da noi nella

posizione selezionata sul JPanel principale, gestito da GameScreen. Questo ha causato un ripensamento sul codice delle due classi modello di Bomb e Polygon. Modificate a tal punto da avere solo un costruttore, in modo tale che i metodi di taglio siano gestiti da un listener posizionato sulla JLabel del singolo oggetto, e i metodi in comuni vengano gestiti dalla classe SliceableModel.

- Physics

  L'ultima classe di model è la classe riguardante la fisica degli oggetti. Lo sviluppo della fisica è stato moderatamente semplice, l'unica cosa che doveva fare era aggiornare i vettori di posizione e velocità di una lista di oggetti Sliceable, calcolando i vettori posizione e velocità successivi. Il problema principale nello sviluppo di questa classe era la gestione della posizione. Visto e considerato che Java considera come (0,0) le coordinate dell'angolo dello schermo in alto a sinistra. Questo ha complicato le cose nei calcoli, ma invertendo i segni che riguardavano calcoli lungo l'asse Y sono riuscito a sistemare il problema.

- **Controller:** Prima di iniziare con l'analisi del controller bisogna aggiungere che molte parti del controller sono state create e modificate da tutti i componenti del gruppo, più o meno brevemente. Questo perché il nostro approccio al lavoro è stato organizzato in modo tale da unire i pezzi singoli creati da ognuno di noi attraverso il controller, sul quale abbiamo lavorato parzialmente insieme.

  - GameWorldController, GameArea, GameScreen

    Ho deciso di citare questo elenco di classi perché sono tutte state modificate, sebbene brevemente, anche da me, in quanto erano necessarie per la corretta gestione del GameLoop. Nonostante tutto la maggior parte delle classi sopracitate è stata svolta dai miei compagni, e sono solo state modificate da me.

  - GameLoop

    Al contrario delle classi precedenti riguardanti controller e view la classe del GameLoop è stata principalmente utilizzata da me, con qualche modifica da parte dei miei compagni. La principale difficoltà riguardante questa classe era quella di riuscire a gestire tutte le occorrenze di creazione degli

oggetti, disegnare, e aggiornare la posizione in tempo reale. Dopo alcune prove fallimentari utilizzando cicli abbiamo deciso di utilizzare un approccio sistematico usando due Timer. Questo semplifica notevolmente la leggibilità e scorribilità del codice, in quanto abbiamo creato un timer per la creazione degli oggetti, che sulla base della difficoltà impostata sfrutterà le classi Factory per creare ed aggiungere il poligono/bomba creato allo schermo. Il secondo timer invece serve per aggiornare le posizioni dei poligoni già presenti a schermo, utilizzando PhysicsController e il GameWorldController, che salva la lista di oggetti 'logici' presenti a schermo.

Mathieu Maugin:

- **View:**
  - Menu, SimpleMenu

    Ho implementato l'interfaccia e la relativa implementazione, la quale crea una semplice schermata vuota.
  - MenuDecorator, Play, Rules, Exit

    MenuDecorator mette a disposizione un metodo per "decorare" i pulsanti inseriti, in modo che sia visibile solo la scritta e il background sottostante (non i bordi del pulsanti).
  - Play, Rules, Exit

    Sono i pulsanti visibili nel menù. Ognuno di questi ha un metodo (fornito da SimpleMenu) che permette di aggiungere il pulsante ad un pannello per poi essere visualizzato nel menù.
    In particolare il pulsante Play avrà un riferimento a GameWorldController che permetterà l'avvio del gioco.
- **Controller:** come detto precedentemente tutto il gruppo ha contribuito alla creazione/gestione dei vari controller, in particolare ho implementato:
  - GameWorldController

    Lo scopo di questo controller è quello di poter creare gli Sliceable che successivamente verranno mostrati a schermo.

- PhysicsController

  Questo controller è stato implementato per poter gestire esternamente l'aggiornamento delle posizioni degli oggetti richiamando i metodi descritti nel model. Ricava la lista di tutti gli oggetti presenti al momento e gli aggiorna la posizione, resistuendo la lista aggiornata a GameWorldController.

- GameLoop

  Ho inserito questa classe avendola modificata per quanto riguarda la parte di disegno degli oggetti e del game over. Come detto in precedenza abbiamo utilizzato due timer, uno per gestire la creazione di oggetti e l'altro per aggiornare e mostrare le posizioni.

- **Model:**

  - Blade

    Inizialmente avevo pensato di gestire l'input dell'utente tramite una classe esterna. Dopo vari tentativi e discussioni con gli altri membri del gruppo siamo arrivati alla conclusione che l'implementazione di tale classe avrebbe creato più problemi di quelli dovuti, quindi abbiamo deciso di implementare l'input semplicemente aggiungendo un Mouse Listener alla JLabel associata allo Sliceable, rimuovendo completamente la classe e l'interfaccia relativa.

Karim El Kholy: Come deciso in fase di proposta del progetto mi sono occupato prevalentemente della parte View di Poly-Cut, ho anche però collaborato con i miei colleghi contribuendo in parte minore alla componente Controller e Model del progetto. Segue una lista di elementi del progetto

Model:

- SlicebleTypeEnum

View:

- GraphicsElement: Ogni elemento grafico al suo interno è stato ricercato/disegnato e preparato dal sottoscritto

- GameArea, GameScreen, Live, ScoreView

Controller:

- GameLoop: Contribuito al GameOver
- GameWorldController: Contribuito al Loop

3.3 Note di sviluppo

Alessandro Previati:

- Utilizzo di Stream:
  - GameWorldControllerImpl
  - GameLoopImpl

Mathieu Maugin:

- Utilizzo di Stream:
  - GameLoopImpl
  - GameLoopImpl
- Utilizzo dei Generici:
  - GameWorldControllerImpl

Karim El Kholy: Feature di linguaggio avanzate utilizzate:

- Lambda expressions:
  - GameAreaImpl
- Utilizzo di Stream:
  - GameScreenImpl
  - GameScreenImpl
- Utilizzo di reflection:
  - LiveImpl
  - ScoreViewImpl
  - SliceableView

4 Commenti finali

4.1 Autovalutazione e lavori futuri

Alessandro Previati Il progetto di OOP è un lavoro molto stimolante ed interessante, ma allo stesso tempo difficile ed impegnativo. Rispettare le scadenze prese e riuscire ad organizzarsi con più persone per svolgere un programma, che non è di piccole dimensioni per persone alle prime esperienze con i progetti di gruppo, non è cosa facile. Allo stesso tempo vedere i risultati ottenuti una volta che il progetto prende forma è molto gratificante e soddisfacente. Credo di avere svolto un lavoro accettabile come primo grosso progetto della carriera non solo universitaria, di avere seguito le regole di buona programmazione abbastanza bene e soprattutto di avere scritto codice che anche una persona da fuori dovrebbe essere in grado di capire senza bisogno del mio aiuto. Migliorabile assolutamente nell'uso di tecniche di programmazione più avanzate, quali lambda, pattern o costrutti più complessi. Ma allo stesso tempo anche tenere un codice semplice e capibile alla vista ha i propri vantaggi. Nel complesso sono contento del lavoro che ho svolto, potrebbe essere una mia idea futura di sistemare e continuare questo progetto perché è interessante ma soprattutto mi sembra un esempio non troppo semplice di "OOP" che può mostrare le mie capacità di scrittura.

Mathieu Maugin: Questo progetto è stato indubbiamente molto interessante, permettendomi di mettermi in gioco e avere un primo approccio su come potrei lavorare una volta finito il mio percorso di studi. Inoltre, ho imparato cosa vuol dire lavorare in gruppo insieme ad altre persone che la pensano diversamente dal mio punto di vista e dal mio stile di programmazione. Sicuramente alcune parti del progetto (compreso il progetto stesso) le avremmo potute implementare in modi più efficienti, e personalmente parlando avrei potuto utilizzare più tecniche di programmazione come gli Stream, generici, ecc. Tutto sommato però sono soddisfatto di come sia venuto il progetto e in futuro potremmo pensare di migliorarlo per, nell'eventualità, pubblicarlo online.

Karim El Kholy: Lavorare a questo progetto è stata sicuramente la 'sfida didattica' più grossa che abbia mai affrontato, mi ha catapultato in quello che potrebbe essere benissimo un mio ambiente lavorativo futuro. Ho potuto verificare sul campo le competenze che ho acquisito nella programmazione ad oggetti, mettermi alla prova e migliorando le mie ca-

pacità per ottenere il risultato desiderato. Nonostante il progetto abbia avuto una serie di ritardi sono comunque soddisfatto di come il nostro gruppo lo abbia portato a termine. Essendo il mio primo vero lavoro in Java sono consapevole che il codice che ho scritto possa non essere di spiccata qualità, però sono comunque soddisfatto dal modo in cui ho approcciato l'obbiettivo e la metodologia con cui ho sviluppato il codice.

4.2 Difficoltà incontrate e commenti per i docenti

Alessandro Previati: Il corso di OOP è stato molto interessante e bello dal mio punto di vista, sicuramente uno dei migliori fino ad adesso. Allo stesso tempo il progetto è stato complesso da organizzare in gruppo e seguire le deadline. Forse una riorganizzazione delle date di scadenza potrebbe giocare a favore degli studenti, magari proponendo anche delle possibilità di progetti più piccoli, ma con meno peso sul voto finale.

Mathieu Maugin:

Karim El Kholy: Ad oggi penso che il corso di OOP sia probabilmente il corso più complicato nel nostro percorso di studi. Allo stesso tempo mi rendo conto che gli argomenti trattati e le competenze sviluppate durante questo percorso sono di fondamentale importanza per un programmatore che si rispetti. A ogni modo voglio lasciare un commento al fine di possibilmente aiutare futuri studenti come me che hanno avuto particolare difficolta a fare propria la materia. Probabilmente la mia maggiore difficoltà che ho riscontrato durante corso sono stati i turni di laboratorio a mio parere troppo discordanti dalle nozioni e metodologia spiegati in aula; spesso con consegne poco chiare e che danno per scontato alcuni aspetti di programmazione che lo studente potrebbe non aver ancora padroneggiato

5 Guida Utente

Questo gioco è un'imitazione del famoso gioco 'Fruit Ninja'. Una volta aperto verrà mostrato il menù. Puoi scegliere tra 3 pulsanti: gioca, regole o esci.

- **Gioca:** avvia il gioco.
- **Regole:** Mostra una finestra pop-up che spiega in italiano cosa vedrai nel gioco.
- **Esci:** chiude l'applicazione.

Una volta giocato, devi solo spostare il cursore per iniziare a tagliare gli oggetti. Se non muovi il mouse, non taglierai l'oggetto, anche se passa sopra il cursore.
Mentre ti muovi, non toccare le bombe e non lasciare cadere i poligoni, altrimenti perderai una vita. Taglia i poligoni per ottenere punti e prova a battere il record.
Durante il gioco, puoi premere il pulsante ESC per chiudere completamente l'applicazione.
Buona fortuna!