



UNIVERSITÀ DEGLI STUDI DI NAPOLI
“PARTHENOPE”

DIPARTIMENTO DI SCIENZE E TECNOLOGIE
CORSO DI LAUREA MAGISTRALE IN
MACHINE LEARNING & BIG DATA

CORSO DI COMPUTER GRAPHICS

Arkanoid 3D

Dario Musella
0120000240

Anno Accademico 2020/2021

Indice

1	Introduzione	3
1.1	Il videogioco Arkanoid	3
2	Arkanoid 3D	6
2.1	Componenti del gioco	7
2.2	Illustrazione gioco	8
2.3	Comandi di gioco	9
3	Dettagli implementativi	10
3.1	Interfaccia GLUT	11
3.2	La classe facciata	12
3.3	Menu di gioco	13
3.3.1	Gestione dei punteggi	14
3.4	Elementi di gioco	16
3.4.1	Gestione delle collisioni	17
3.4.2	Mattoni in Arkanoid 3D	22
3.5	Livelli di gioco	24
3.6	Motore di gioco	27

Capitolo 1

Introduzione

In questo elaborato viene illustrata l'implementazione di un videogioco basato sul modello di Arkanoid, tramite l'utilizzo delle librerie OpenGL.

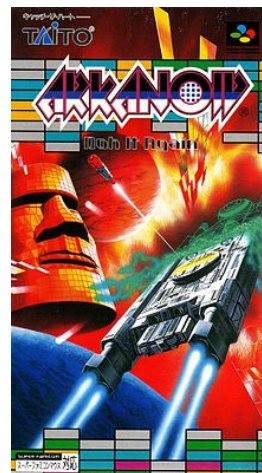
Arkanoid 3D è stato scritto in linguaggio C++, mediante le librerie FreeGLUT e GLEW che permettono rispettivamente di gestire le finestre di contesti OpenGL (con annessi eventi di input) mediante chiamate alle API [1] e di verificare quali funzionalità OpenGL possono essere usate sulla macchina che esegue il software [2]. Il gioco presenta texture che sono state applicate mediante la libreria SOIL. Il progetto è stato sviluppato su una piattaforma hardware dotata di CPU *Intel Core i3 2310M*, con GPU integrata *Intel HD Graphics 3000* che supporta OpenGL 3.1.

1.1 Il videogioco Arkanoid











Arkanoid è un videogioco arcade sviluppato dalla software house Taito nel 1986, rilasciato inizialmente su cabinati e successivamente portato su Commodore 64, Amiga e altri sistemi hardware. Lo scopo del gioco consiste nel superare 33 livelli, dove bisogna abbattere una serie di mattoncini colorati con una pallina che viene fatta rimbalzare da una pedana (comandata dal videogiocatore) contro i mattoncini stessi. Il giocatore ha a disposizione un numero limitato di vite, che si esauriscono quando la pallina cade nella parte inferiore dello schermo [3].

La trama ruota intorno all'astronave madre "Arkanoid", che viene distrutta in seguito ad un attacco da forze aliene. L'unica superstite è l'astronave "Vaus", che riesce a sfuggire all'imboscata ma resta successivamente intrappolata in una deformazione spazio temporale causata da colui che si scopre essere l'artefice di tutto: "Doh", un monolite a forma di moai¹.

Il videogioco è una riedizione del classico *Breakout*, rilasciato nel 1976 e si presenta con numerose migliorie dal punto di vista grafico e con aggiunte al gameplay che ne aumentano le funzionalità e la giocabilità. Rispetto al videogioco su cui si basa, Arkanoid presenta ad ogni livello uno schema di mattoni differente e i mattoni differiscono per colore e comportamento: i mattoni colorati costituiscono la maggior parte dei livelli, vengono distrutti ad un solo colpo e in base al colore rilasciano un differente punteggio, i mattoni argento rilasciano un punteggio variabile e necessitano di più colpi per essere distrutti, ed i mattoni oro che non possono essere distrutti. La tabella seguente descrive in maniera esaustiva tutti i mattoni presenti nel videogioco.



¹Statue di tufo a forma di testa cilindroidica, presenti in gran numero sull'Isola di Pasqua.

Mattone	Colore	Colpi	Valore
	Bianco	Un colpo	50 punti
	Arancione	Un colpo	60 punti
	Azzurro	Un colpo	70 punti
	Verde	Un colpo	80 punti
	Rosso	Un colpo	90 punti
	Blu	Un colpo	100 punti
	Viola	Un colpo	110 punti
	Giallo	Un colpo	120 punti
	Argento	Due colpi + $\frac{\text{livello}}{8}$	50 punti \times livello
	Oro	Indistruttibile	50 punti

Altro particolare che contraddistingue Arkanoid è la presenza di bonus nascosti che cadono verso la pedana quando vengono demoliti alcuni mattoncini, oltre alla presenza di alieni che “disturbano” il giocatore deviando la pallina quando colpiti.

Capitolo 2

Arkanoid 3D

L'applicativo che è stato sviluppato emula le meccaniche del videogioco Arkanoid, riprendendo alcune delle più importanti caratteristiche presenti, con l'aggiunta di alcune completamente originali.

L'aggiunta della dicitura *3D* al titolo denota la prima variazione rispetto al videogioco originale, siccome il motore di gioco è realizzato in tre dimensioni. Sebbene il gameplay sia pressoché identico all'originale, dove le animazioni avvengono solo in orizzontale e verticale, gli oggetti che compongono il gioco sono a tutti gli effetti dei solidi. Questa particolare tecnica è stata applicata nel corso degli anni da numerose software house, e prendono il nome di videogiochi 2.5D: l'ambiente è generato con grafica 3D ma l'azione di gioco si svolge su un solo piano bidimensionale.









Figura 2.1: Il videogioco 2.5D *LittleBigPlanet* presenta oggetti solidi in diverse posizioni in prospettiva, ma l'azione di gioco è svolta solo su un piano

2.1 Componenti del gioco

Il gioco è composto da tre oggetti fondamentali: **palla**, **pedana** e **mattoni**.

I mattoni presentano diverse peculiarità e hanno un tasso di valore e salute differenti. Le quattro macro categorie consistono nei mattoni **semplici** che necessitano di un colpo per essere demoliti, mattoni colore **oro** più resistenti che richiedono due colpi per essere distrutti, mattoni **grigi** che non possono essere distrutti in alcun modo (non rientrano nel conteggio dei mattoni del livello) e mattoni **invisibili** che in primo luogo risultano tali, ma diventano visibili quando colpiti la prima volta e assumono un colore rosso. In particolare, i mattoni invisibili non rientrano nel conteggio dei mattoni da distruggere fin quando sono effettivamente trasparenti, ma quando vengono colpiti sarà necessario distruggerli per completare il livello. La tabella di seguito racchiude tutte le caratteristiche dei mattoni [4] presenti nel gioco.

Mattone	Colore	Colpi	Valore
	Blu	Un colpo	50 punti
	Verde	Un colpo	75 punti
	Viola	Un colpo	100 punti
	Oro	Due colpi	150 punti
	Grigio	Indistruttibile	0 punti
	Invisibile Rosso (quando colpito)	Due colpi	120 punti

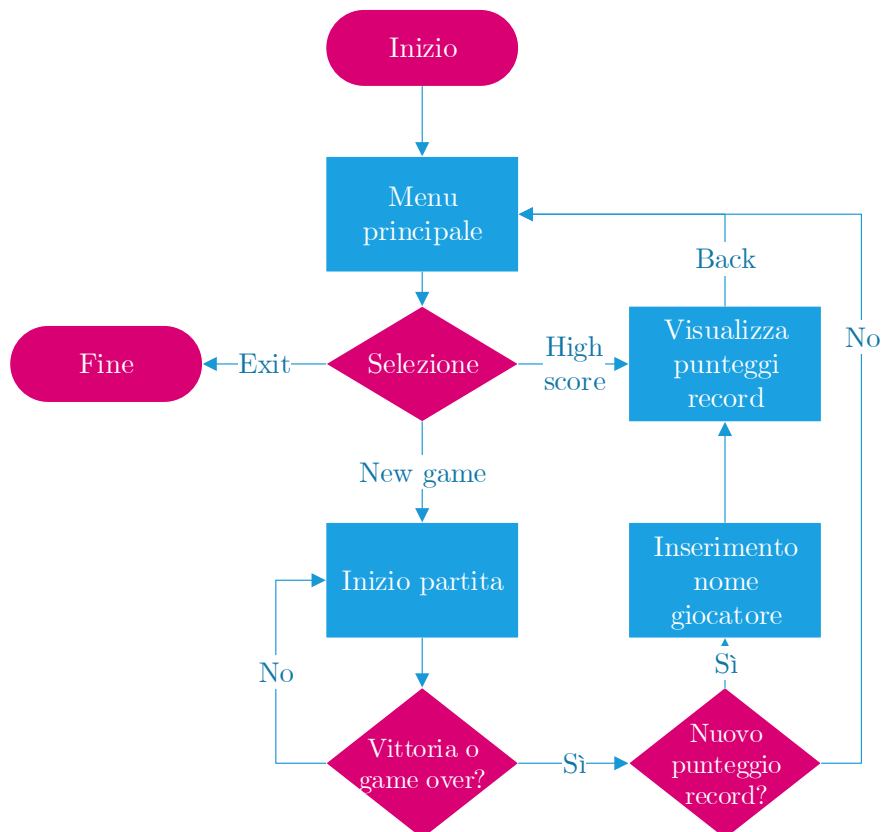
Questa implementazione viene fornita con sei livelli di gioco, di crescente difficoltà. I primi livelli presentano una quantità di mattoni speciali ridotta, mentre in quelli successivi costituiscono una buona parte della quantità di mattoni totali. In particolare gli ultimi livelli presentano una ulteriore difficoltà dovuta al movimento dei mattoni sullo schermo.

2.2 Illustrazione gioco

Il videogioco si presenta all'avvio con un menù che fornisce delle opzioni, come l'avvio di una nuova partita o la visualizzazione dei punteggi record totalizzati.

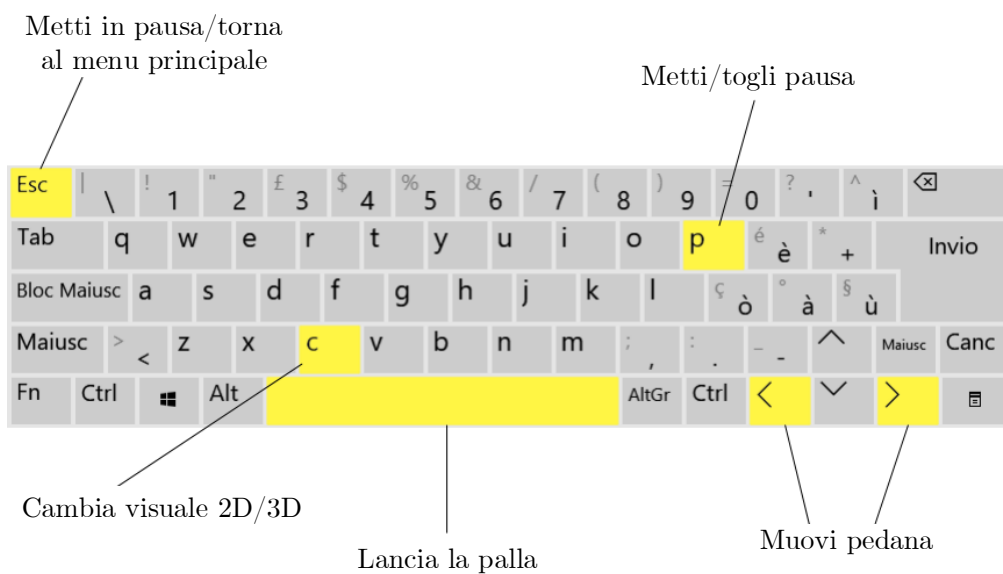


Il diagramma riportato qui di seguito illustra il funzionamento del gioco attraverso il passaggio tra i vari contesti.



2.3 Comandi di gioco

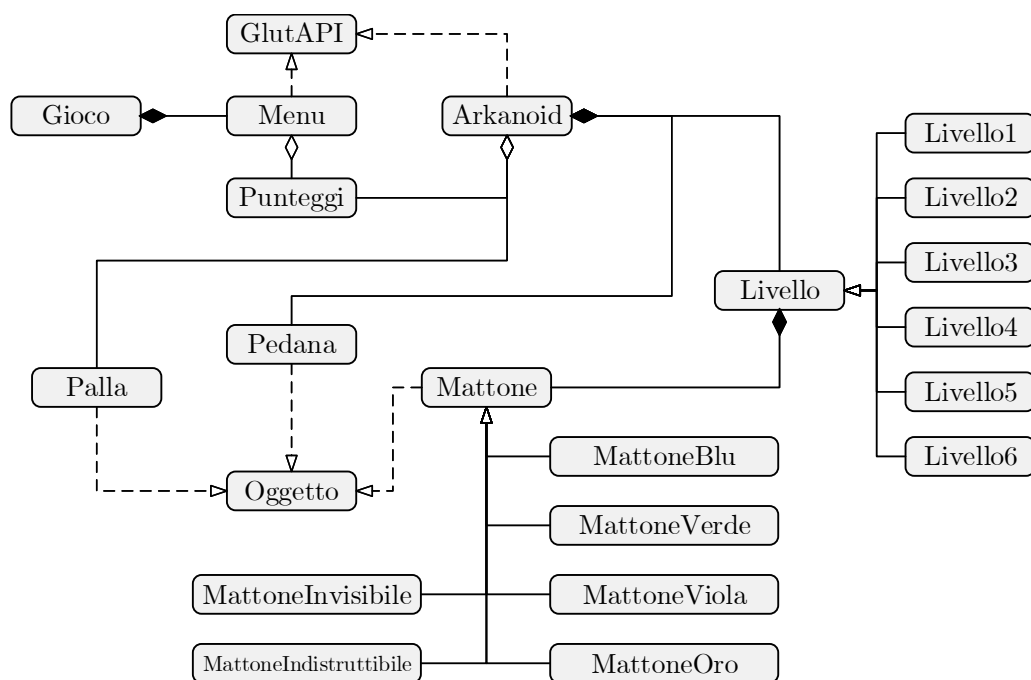
I comandi del videogioco consistono in poche e semplici operazioni, come il movimento della pedana con le **freccie direzionali sinistra** e **destra** ed il lancio della palla con la **barra spaziatrice** per iniziare a giocare. Il gioco permette anche di essere messo in pausa e di tornare al menù principale.



Capitolo 3

Dettagli implementativi

Di seguito vengono approfonditi i dettagli implementativi di maggiore rilievo del videogioco. Viene illustrato in primo luogo un diagramma UML semplificato al fine di agevolarne la lettura e successivamente approfondito.



Nell'applicativo si riconoscono due contesti corrispondenti al **menù di gioco** e alla **partita** che si gioca.

3.1 Interfaccia GLUT

GlutAPI

```
-instance: GlutAPI*
-----
#setInstance(): void
#clearInstance(): void
-display(): void
-reshape(w: int,h: int): void
-mousePress(button: int, state: int, x: int, y: int): void
-mouseMotion(x: int, y: int): void
-mousePassiveMotion(x: int, y: int): void
-keyboard(key: uchar, x: int, y: int): void
-keyboardDown(key: uchar, x: int, y: int): void
-keyboardUp(key: uchar, x: int, y: int): void
-specialKeyboardDown(key: int, x: int, y: int): void
-specialKeyboardUp(key: int, x: int, y: int): void
#displayWrapper(): void
#reshapeWrapper(w: int,h: int): void
#mousePressWrapper(button: int, state: int, x: int, y: int): void
#mouseMotionWrapper(x: int, y: int): void
#mousePassiveMotionWrapper(x: int, y: int): void
#keyboardWrapper(key: uchar, x: int, y: int): void
#keyboardDownWrapper(key: uchar, x: int, y: int): void
#keyboardUpWrapper(key: uchar, x: int, y: int): void
#specialKeyboardDownWrapper(key: int, x: int, y: int): void
#specialKeyboardUpWrapper(key: int, x: int, y: int): void
```

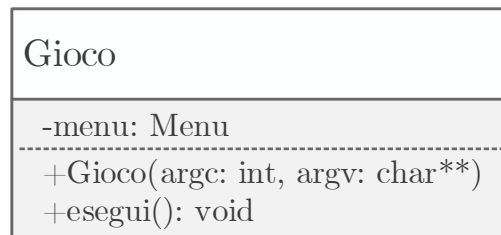
La classe astratta **GlutAPI** costituisce un'interfaccia che permette di sfruttare le funzionalità di GLUT all'interno di un contesto orientato agli oggetti.

I metodi privati presenti all'interno vengono implementate dalle due classi **Menu** e **Arkanoid**, in quanto le funzioni callback passate alle funzioni GLUT (`glut...Func()`) non sono propriamente compatibili con la sintassi delle classi C++.¹ Il *workaround* è applicato mediante i corrispettivi metodi statici protetti che presentano la parola **Wrapper** al termine del nome del metodo:

¹Le funzioni callback del C prevedono il tipo `void *`, che è diverso dal tipo del metodo di una classe.

in questo modo i dettagli funzionali di GLUT sono nascosti alle classi che ne sfruttano le funzionalità.

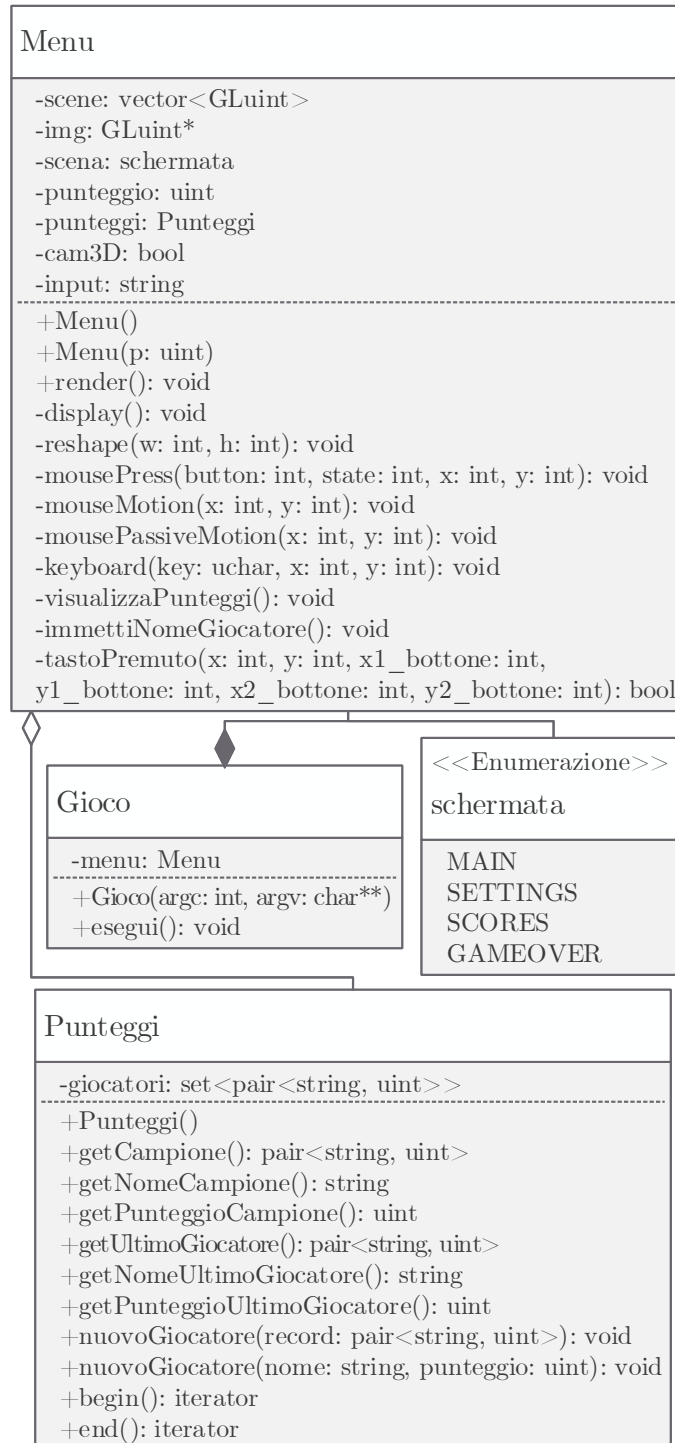
3.2 La classe facciata



La classe `Gioco` non è altro che un'interfaccia semplificata di tutto il sistema di gioco, utilizzato dall'entry point del programma per eseguire il videogioco. Come citato prima, `Arkanoid 3D` suddivide il menù principale e la partita in due contesti distinti e separati tra loro, ma al di fuori si osserva un unico nucleo di gioco.

Questa classe inizializza le impostazioni di GLUT, settando le dimensioni della finestra e il doppio buffer, e fornisce un unico metodo semplificato `esegui()` che nasconde il modo in cui esegue `Arkanoid 3D` [5], cioè istanziando un oggetto di tipo `Menu`.

3.3 Menu di gioco



La classe **Menu** implementa il menù principale di Arkanoid 3D. Permette di avviare una nuova partita, di visualizzare i migliori punteggi totalizzati, di scegliere dal menù delle impostazioni l'inquadratura di gioco (effetto 2D o effetto 3D) e di chiudere l'eseguibile.

L'oggetto corrispondente viene istanziato mediante la classe **Gioco** che funge da facciata per il **main** del progetto.

La grafica del menù è stata realizzata mediante un semplice texture mapping di un rettangolo, con una telecamera ortogonale impostata con le stesse dimensioni della finestra GLUT. Attraverso le callback che osservano i movimenti ed i clic del mouse, il menù visualizza una texture corrispondente al contesto corrente e attende un clic: se avviene in un'area rettangolare (interna al rettangolo/finestra) carica una nuova texture collegata all'opzione scelta o istanzia un oggetto **Arkanoid** che dà inizio ad una partita. In tal caso **Menu** passa l'informazione relativa alle impostazioni della telecamera al nuovo oggetto corrispondente al gioco effettivo.

3.3.1 Gestione dei punteggi

Il menù di gioco permette, attraverso il pulsante "HIGH SCORE", di visualizzare i 12 migliori punteggi totalizzati ad Arkanoid 3D. La gestione dei punteggi è delegata alla classe apposita **Punteggi**, che attraverso delle funzioni di I/O legge e salva su file i nomi dei giocatori ed i punteggi totalizzati.

Al termine di una partita, prima di passare al contesto del menù principale, la classe **Arkanoid** verifica se il punteggio totalizzato corrisponde ad un record ed in tal caso passerà un parametro ad un nuovo oggetto **Menu**, che carica l'interfaccia relativa all'inserimento del nome del giocatore. Alla pressione del tasto invio, il punteggio viene salvato nella top 12 e viene aggiornato il file.

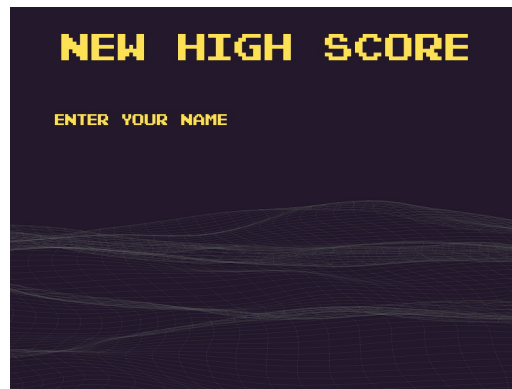
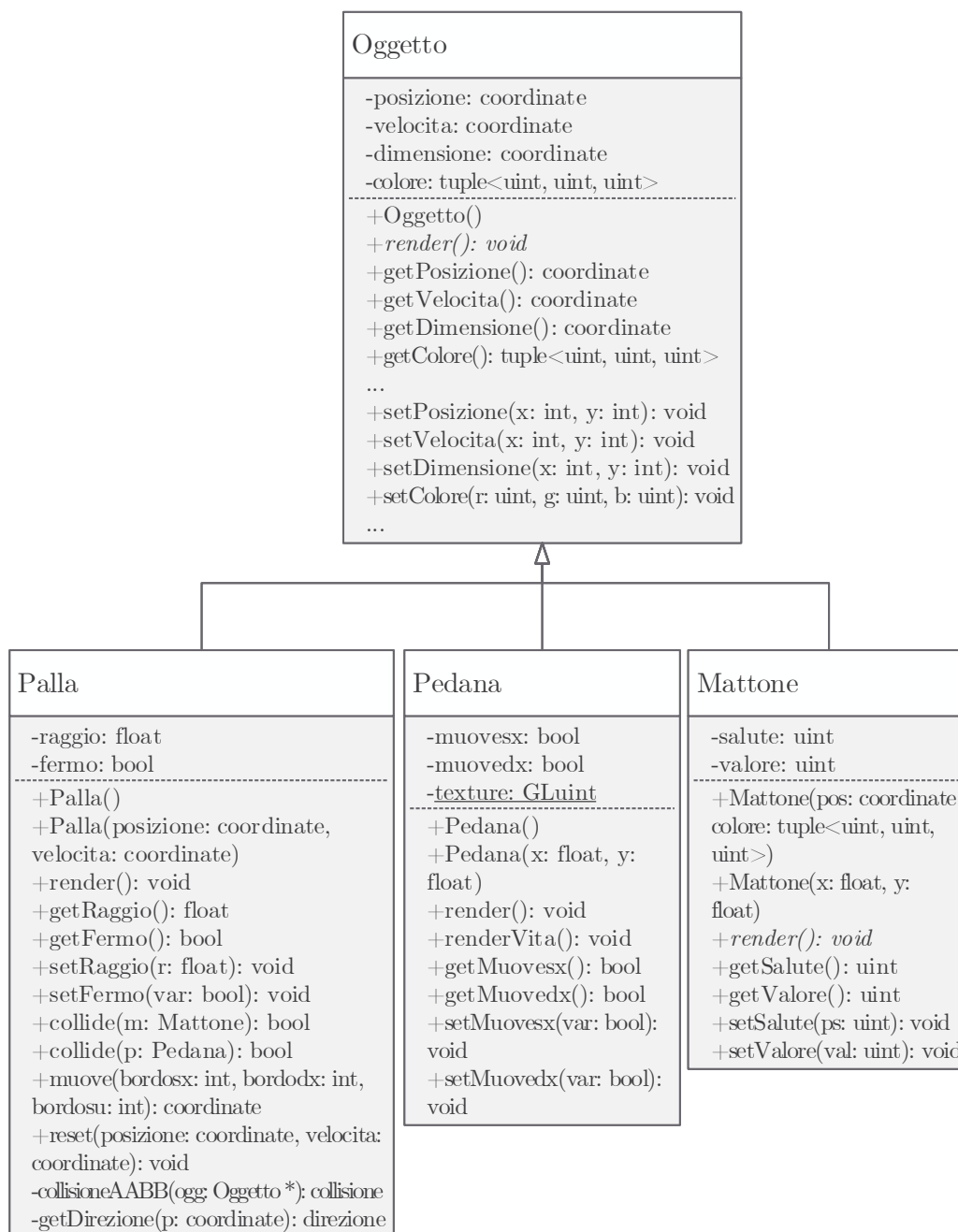


Figura 3.1: Schermata di inserimento del nome del giocatore

3.4 Elementi di gioco



La classe astratta **Oggetto** fornisce delle operazioni di base per gli elementi fondamentali che animano il videogioco. Dispone di informazioni sotto forma di coordinate 2D (implementate come dei `pair<float, float>`) riguardo la posizione, dimensione ed eventuale velocità di movimento del generico oggetto, che è data da una quantità che ad ogni istante viene aggiunta o sottratta alla posizione.

Le tre tipologie di oggetti fondamentali ereditano le proprietà della generica classe **Oggetto** e le estendono specializzando per i singoli tipi. La classe **Pedana** possiede delle informazioni relative al movimento comandato dal giocatore, tramite delle variabili booleane che informano di un movimento verso sinistra o destra. Inoltre fornisce due possibili tipi di rendering, dove `render()` visualizza la pedana comandata dal giocatore, mentre `renderVita()` mostra una miniatura della pedana, utilizzato per visualizzare il numero di vite a disposizione.

La classe **Palla** aggiunge informazioni relative al raggio della sfera, all'eventuale movimento (se il giocatore preme la barra spaziatrice, la palla si muove, se la palla cade, si ferma) e gestisce le collisioni con gli altri oggetti presenti nel gioco.

3.4.1 Gestione delle collisioni

Per ottenere un risultato accettabile è stata implementata la tecnica AABB (axis-aligned bounding box) cerchio-rettangolo per la *collision detection* e le operazioni di riposizionamento e cambio direzione per la *collision resolution* [6].

Il calcolo delle collisioni non avviene elaborando i dati dei vertici degli oggetti presi in considerazione, poiché ciò renderebbe le operazioni di calcolo piuttosto complicate. Per tale motivo si sfrutta un'approssimazione degli oggetti presi in esame, e siccome il videogioco Arkanoid (che sia il gioco originale o l'implementazione riportata in questo elaborato) presenta solo figure solide elementari, risulta più semplice ricorrere a formule matematiche per il calcolo delle collisioni.

Importante evidenziare che nonostante gli oggetti di Arkanoid 3D siano tridimensionali, gli algoritmi lavorano con un'approssimazione bidimensionale degli oggetti considerati in quanto il gameplay si svolge solo lungo gli assi x e y .

Rilevamento delle collisioni

In Arkanoid si ha a che fare con collisioni tra un rettangolo ed un cerchio. Per il rettangolo si definisce un AABB con una posizione in alto a sinistra data dalla posizione, e in basso a destra data dalla posizione più la dimensione del poligono, mentre per il cerchio si considera la posizione più il raggio.

Una collisione avviene quando i due oggetti entrano uno nell'area dell'altro: si controlla se per ogni asse i bordi degli oggetti si sovrappongono.

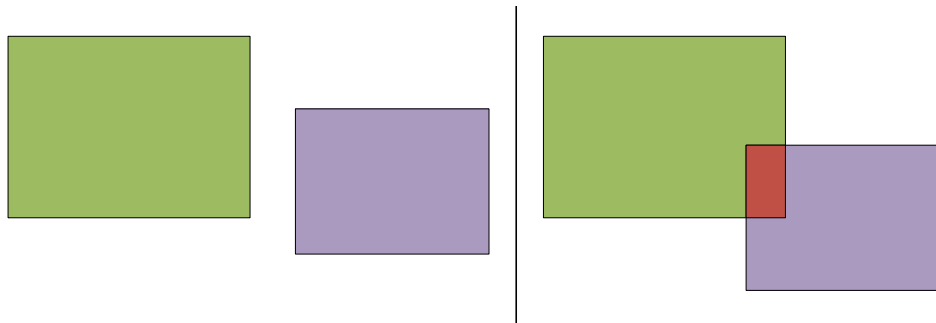


Figura 3.2: Collisione AABB tra due rettangoli

Per verificare se effettivamente si presenta una collisione tra un cerchio e un rettangolo si calcola il punto del rettangolo che risulta più vicino al cerchio, e se la distanza del punto dal cerchio è minore del raggio r , allora si è verificata una collisione.

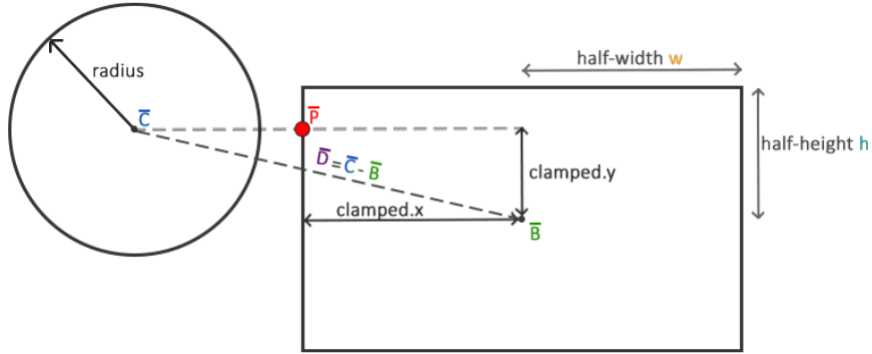
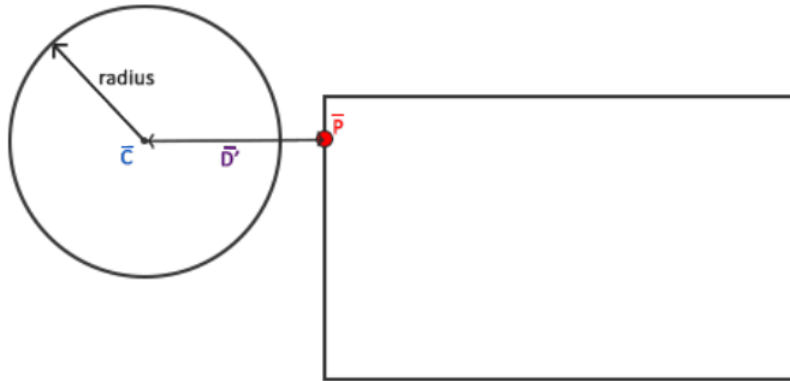


Figura 3.3: Calcolo della collisione AABT tra un cerchio e un rettangolo

Si calcola la differenza tra il centro della circonferenza \bar{C} e il centro del rettangolo \bar{B} per ottenere \bar{D} . Successivamente si effettua il *clamp*² del vettore \bar{D} alla *half-extents* del rettangolo di larghezza w e altezza h , cioè alle distanze tra il centro del rettangolo e i suoi bordi, che corrisponde alla metà della dimensione del rettangolo. Questo calcolo genera un vettore *clamped* \bar{P} , che è il punto più vicino dal rettangolo al cerchio, e si trova sul bordo del rettangolo (a meno che il centro della circonferenza non sia all'interno del rettangolo).



Successivamente si calcola il vettore differenza $\bar{D}' = \bar{C} - \bar{P}$, se \bar{D}' è minore del raggio della circonferenza, allora si è verificata una collisione.

²Operazione di “troncamento” dei valori in un intervallo.

Risoluzione delle collisioni

Quando viene rilevata una collisione, è necessario innanzitutto riposizionare il cerchio in modo che non sia più all'interno del rettangolo, e cambiarne la direzione di movimento.

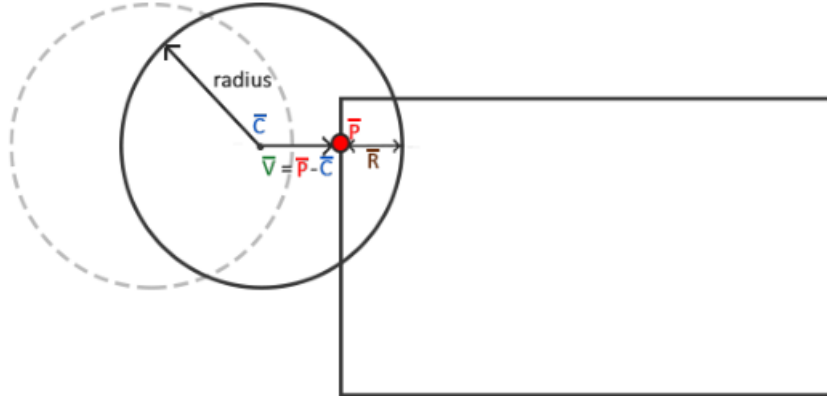


Figura 3.4: Risoluzione della collisione

Per riposizionare il cerchio al di fuori va calcolato il vettore \bar{R} , che stabilisce il livello della penetrazione della palla nel rettangolo. Per calcolare la penetrazione \bar{R} va sottratto alla lunghezza del raggio r la differenza $\bar{V} = \bar{P} - \bar{C}$, cioè tra il punto più vicino \bar{P} e il centro della circonferenza \bar{C} , pertanto $\bar{R} = r - \bar{B}$. In questo modo si sposta il cerchio verso l'esterno, annullando il vettore \bar{R} .

Una volta riposizionato il cerchio, va determinata la direzione del cerchio in modo da simulare un rimbalzo. Questo avviene modificando opportunamente la velocità, facendo affidamento alla seguente logica nel caso di collisione con un mattone:

- $vel(x) = -vel(x)$ se la collisione avviene ai lati sinistra o destra
- $vel(y) = -vel(y)$ se la collisione avviene ai lati superiore o inferiore

L'approccio che è stato utilizzato consiste nell'utilizzare quattro AABB per ogni mattone anziché uno solo, posizionati sui bordi del rettangoli. In questo modo bisogna calcolare quale dei quattro AABB, e quindi quale bordo è stato colpito.

Definendo quattro vettori che puntano a nord, sud, ovest ed est, si procede a calcolare il prodotto scalare di ognuno di essi con il vettore differenza \bar{V} e si determina quale tra i quattro vettori restituisce il prodotto scalare maggiore.

Nel caso di collisione tra il cerchio e la pedana, la risoluzione avviene in maniera differente. Una volta riposizionato il cerchio al di fuori della pedana, la sua nuova velocità sull'asse x non sarà più semplicemente l'opposto della sua precedente velocità, ma varia in base a quanto la palla si trova distante dal centro orizzontale della pedana: più la palla è distante dal centro, più la palla diviene veloce e più grande risulta la variazione sull'asse x . Il calcolo avviene determinando la percentuale di quanto la palla dista dal centro orizzontale della pedana, quindi la velocità viene influenzata da tale valore percentuale.

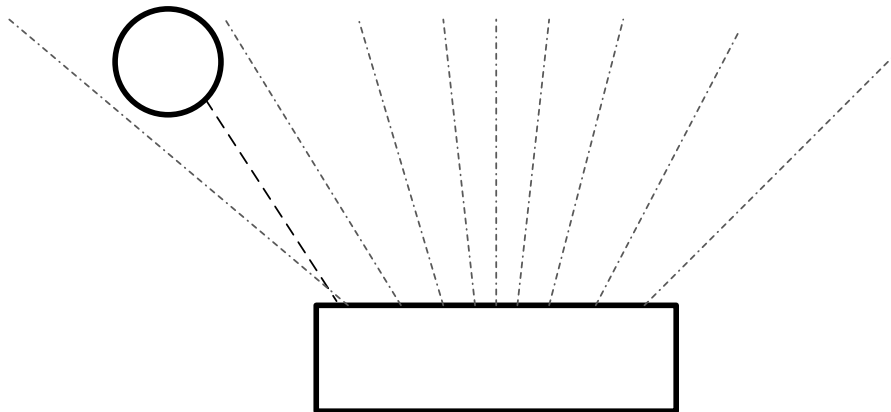
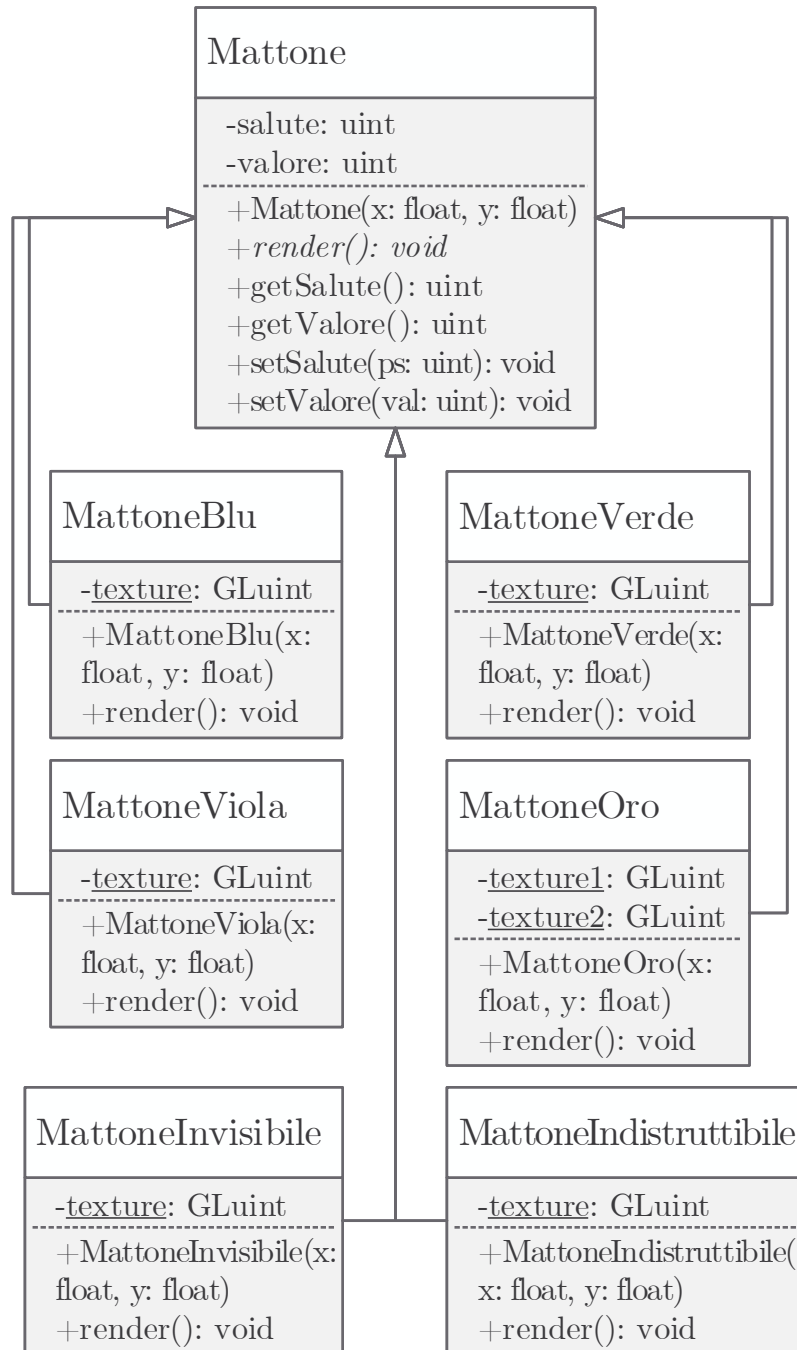


Figura 3.5: Le linee doppiamente tratteggiate determinano la direzione della palla in seguito alla collisione con la pedana

3.4.2 Mattoni in Arkanoid 3D



La classe `Mattone` implementa le operazioni comuni a tutti i mattoni, ma delega alle classi derivate il compito di gestire il rendering su OpenGL in quanto differisce in base al tipo di mattone. Oltre ai differenti valori di punteggio e di salute, infatti, va ridefinito il comportamento di rendering, che nel caso della classe `MattoneInvisibile` il mattone deve essere fisicamente presente ma non visibile a meno che non venga colpito, mentre nel caso della classe `MattoneOro`, essa deve mantenere due texture che sono applicate in base alla salute: se il mattone oro ha salute massima allora viene visualizzato un mattone intatto, altrimenti un mattone danneggiato.

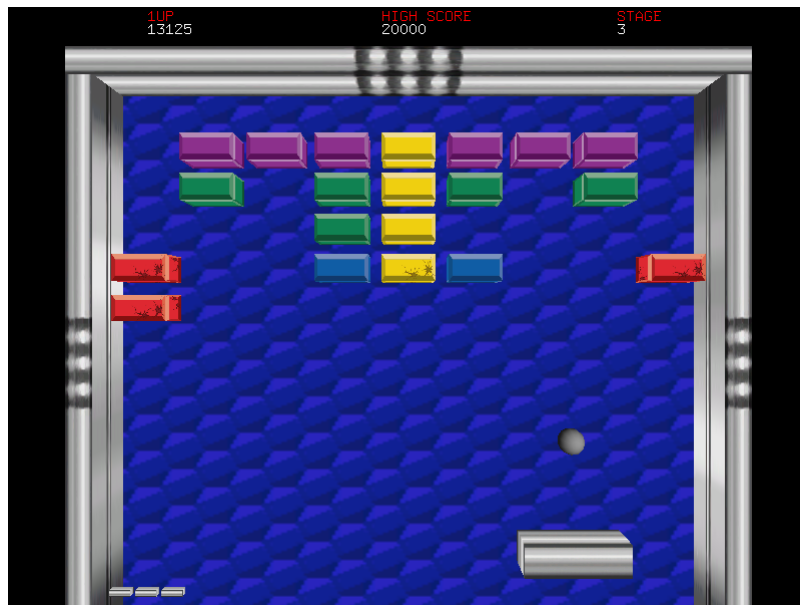
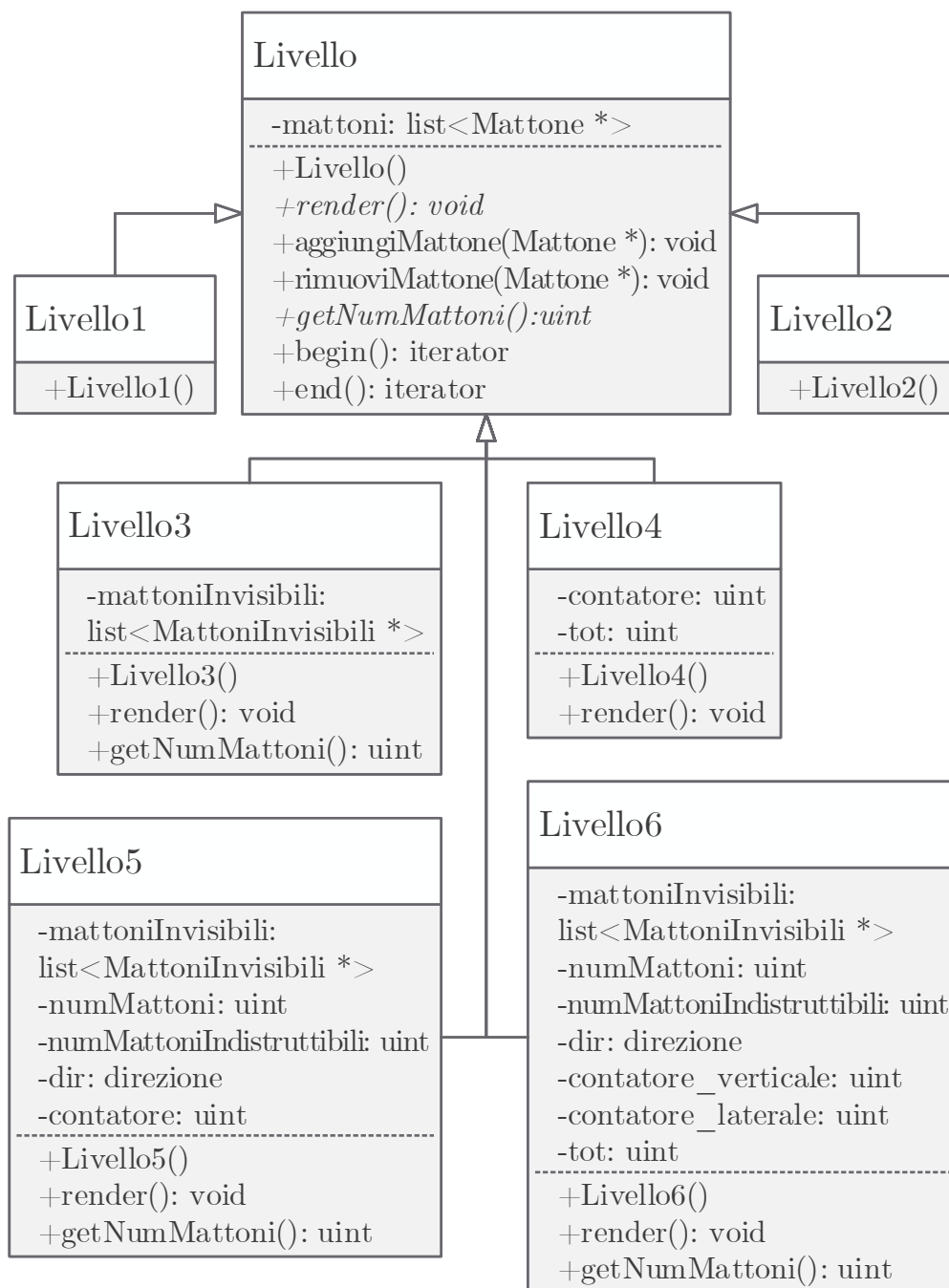


Figura 3.6: La palla ha scoperto alcuni mattoni invisibili e ha colpito dei mattoni dorati

3.5 Livelli di gioco



Concettualmente, un livello non è altro che un insieme di mattoni posizionati sullo schermo, infatti la classe `Livello` contiene una lista di mattoni, più dei metodi per accedervi e modificarla. Ogni livello di gioco costituisce una classe a sé stante, siccome la creazione del livello è delegata al costruttore che lo assembla posizionando i mattoni in base ad uno schema prefissato.

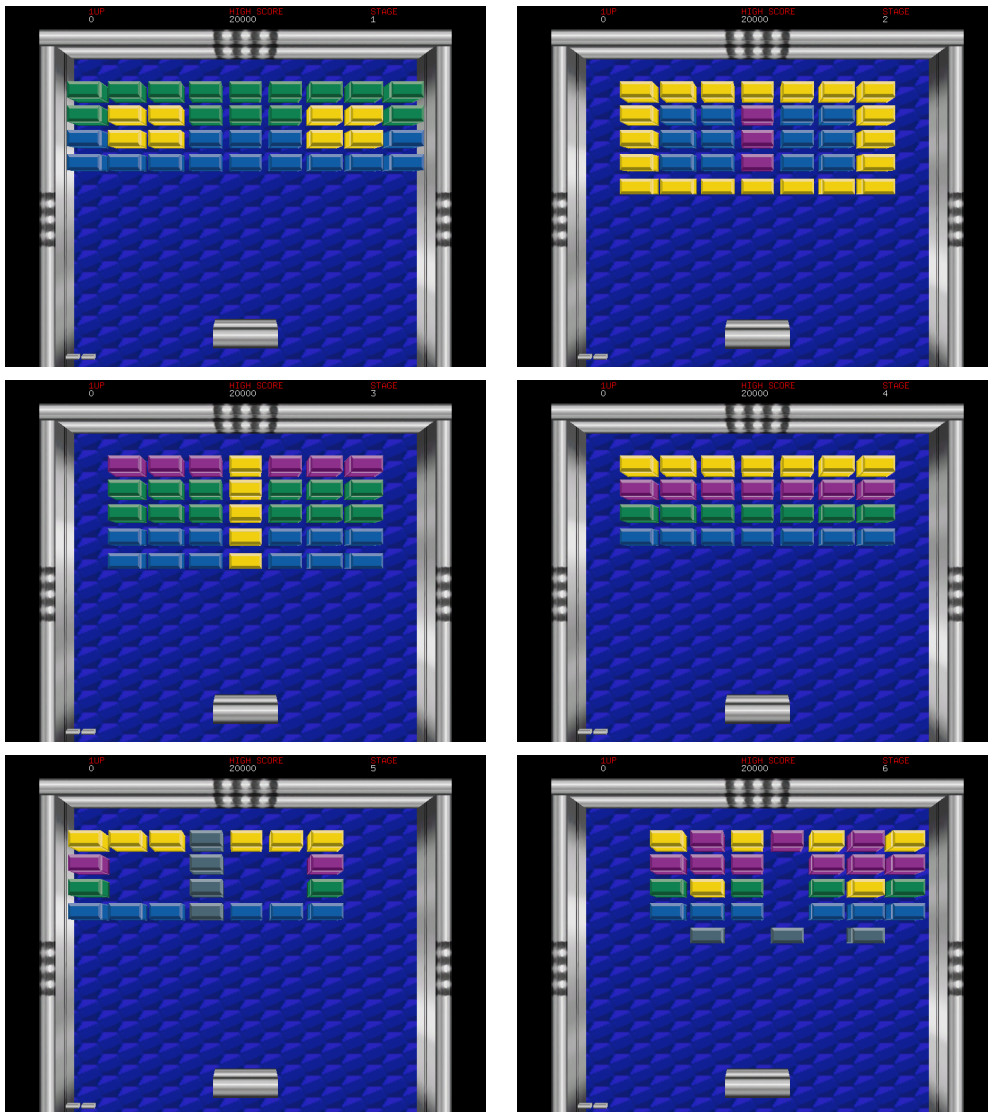


Figura 3.7: I sei livelli di gioco

I due metodi virtuali `render()` e `getNumMattoni()` forniscono un'implementazione basica per i livelli più semplici, come `Livello1` e `Livello2`, mentre i livelli successivi oltre ad estenderne le funzionalità fanno un override di questi due metodi. Quando un livello contiene dei mattoni invisibili o indistruttibili, questi devono essere tenuti traccia attraverso ulteriori informazioni, siccome i mattoni indistruttibili e i mattoni effettivamente (mai colpiti) invisibili sono esclusi dal conteggio del numero totale di mattoni da distruggere. Il metodo `getNumMattoni()` restituisce, in questo caso, il numero di mattoni totali meno quelli indistruttibili e quelli effettivamente invisibili. Quando un livello contiene dei mattoni invisibili, questi sono tracciati attraverso una lista di `MattoniInvisibili` che contiene quelli effettivamente invisibili, e nel caso in cui uno di questi dovesse essere colpito e di conseguenza visualizzato a schermo, allora viene rimosso dalla lista, diminuendo di fatto il numero di mattoni non considerati e di conseguenza aumentando il numero totale di mattoni da distruggere per completare il livello.

Fare override al metodo `render()`, invece, permette di implementare nuove caratteristiche al singolo livello, come ad esempio il movimento dei mattoni verso il basso della classe `Livello4` ad ogni verificarsi di un predicato, oppure il movimento laterale ad ogni istante di tempo della classe `Livello5`.

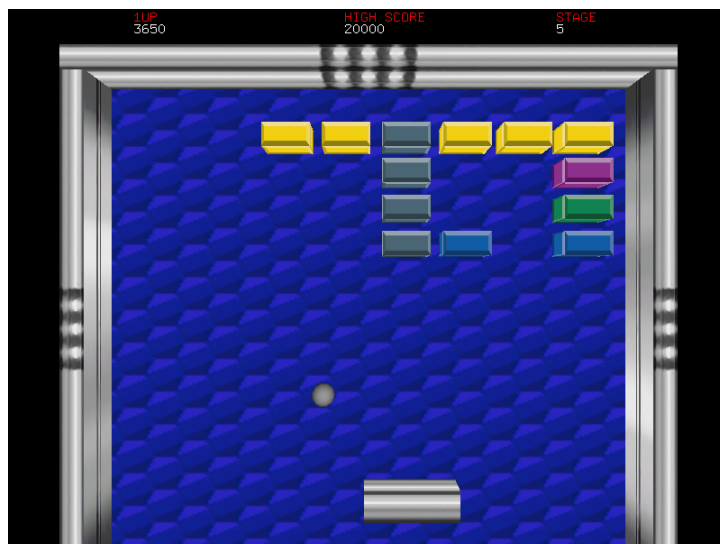


Figura 3.8: Livello 5, i mattoni sono spostati verso destra

3.6 Motore di gioco

Arkanoid

```
-livelli: vector<Livello *>
-palla: Palla
-pedana: Pedana
-livello: Livello *
-vite: uint
-i_livello: uint
-punteggio: uint
-cont_vite_punteggio: uint
-viteHUD: vector<Pedana>
-topPunteggi: Punteggi
-x: int
-y: int
-z: int
-sfondo: GLuint
-bordo_o: GLuint
-bordo_v: GLuint
-cam3D: bool
-pausa: bool
-----
+Arkanoid()
+Arkanoid(c: bool)
+render(): void
-renderHUD(): void
-cambiaCamera(): void
-renderScene(): void
-display(): void
-reshape(w: int, h: int): void
-keyboard(key: uint, x: int, y: int): void
-specialKeyboardDown(key: int, x: int, y: int): void
-specialKeyboardUp(key: int, x: int, y: int): void
```

La classe `Arkanoid` costituisce il motore che mette in relazione gli oggetti di gioco illustrati precedentemente. Gestisce il rendering dell'HUD di gioco, il punteggio della partita e fornisce al giocatore l'input tramite tastiera³. In questo contesto di gioco viene impostata una camera prospettica che, in base al valore dell'attributo `cam3D` viene posizionata in zone differenti: se tale attributo è falso, allora la camera si trova di fronte all'area di gioco, dando un effetto di falso 3D, altrimenti viene posizionata lateralmente.

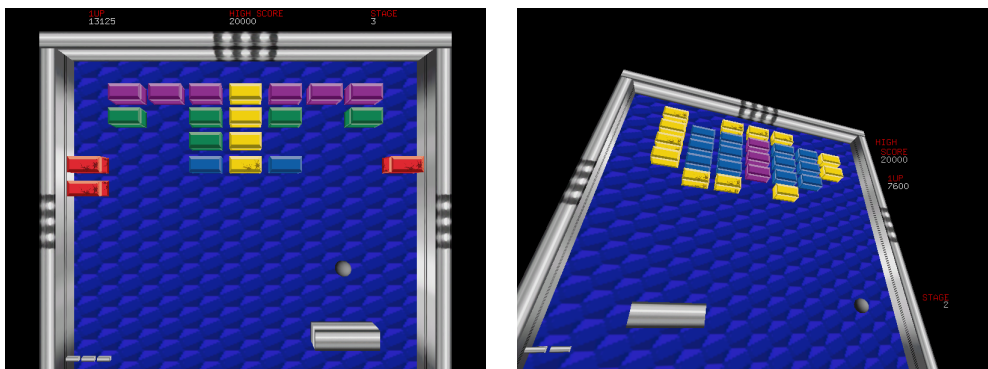


Figura 3.9: Le due posizioni della camera

La camera può essere cambiata durante la partita e dalle impostazioni nel menù principale, il costruttore parametrico della classe riceve come parametro l'informazione riguardo a tale impostazione permettendo di iniziare la partita con il proprio settaggio.

I livelli di gioco vengono renderizzati dal motore caricando il vettore `livelli` con le sei classi inerenti ai rispettivi livelli, mentre i campi `livello` e `i_livello`, che sono rispettivamente un puntatore di tipo `Livello` e un indice intero, determinano il livello corrente che deve essere affrontato dal giocatore. All'inizio della partita, ad esempio, `livello` punta al primo elemento (determinato da `i_livello`) del vettore dei livelli di gioco, e quando l'oggetto puntato, che è di tipo `Livello1`, restituisce l'informazione `getNumMattoni() == 0` allora `i_livello` verrà incrementato e `livello` punterà all'elemento susseguente del vettore, caricando il livello successivo.

³I comandi possibili sono illustrati al paragrafo 2.3

La classe **Arkanoid** gestisce lo svolgimento del livello occupandosi di eliminare i mattoni che presentano una salute nulla e di gestire il punteggio della partita, memorizzato nell'attributo **punteggio**. All'inizio della partita, il gioco inizia con tre vite a disposizione e ogni qualvolta che vengono totalizzati 10000 punti viene aggiunta una vita. Questa funzionalità è stata implementata memorizzando un'ulteriore informazione nel campo **cont_vite_punteggio**, che funge da accumulatore di punti parallelamente a **punteggio**, ma quando **cont_vite_punteggio** è maggiore o uguale a 10000, viene incrementato il dato **vite** e **cont_vite_punteggio** viene resettato a 0.

Quando **ilivello** è maggiore⁴ di 5 o **vite** è uguale a 0, la partita termina. Viene visualizzata a schermo una scritta *GAME OVER* e alla pressione di un tasto viene chiuso il contesto di gioco per essere aperto il contesto del menù principale, che varia in base al punteggio totalizzato. La classe **Arkanoid** contiene informazioni sui migliori punteggi, e al termine della partita verifica se **punteggio** risulta maggiore dell'ultimo miglior punteggio. Se è stato fatto un punteggio record, l'oggetto **Menu** istanziato viene inizializzato da un costruttore parametrico che legge il punteggio totalizzato nell'ultima partita, e visualizza a schermo una schermata che chiede al giocatore di inserire il proprio nome da assegnare al punteggio record realizzato. Se invece il punteggio non risulta essere un record, allora alla chiusura della partita viene visualizzata la schermata principale del menù.

⁴Gli indici vanno da 0 a 5

Bibliografia

- [1] *The freeglut Project*, consultato il 31/12/2020. indirizzo: <http://freeglut.sourceforge.net/>.
- [2] *GLEW: The OpenGL Extension Wrangler Library*, consultato il 31/12/2020. indirizzo: <http://glew.sourceforge.net/>.
- [3] «Arkanoid,» *Il dizionario dei Videogiochi*, consultato il 31/12/2020. indirizzo: <http://www.dizionariovideogiochi.it/doku.php?id=arkanoid>.
- [4] imaginelabs, *Breakout (Brick Breaker) Tile Set - Free*, OpenGameArt.org, 2018. indirizzo: <https://opengameart.org/content/breakout-brick-breaker-tile-set-free>.
- [5] A. Shvets, *Dive Into Design Patterns*. Refactoring.Guru, 2019, pp. 209–218.
- [6] J. de Vries, *Learn OpenGL - Graphics Programming*, Kendall e Welling, cur. 2020, pp. 481–490, ISBN: 978-90-90-33256-7.