

# Progetto Metodologie di Programmazione 2023/2024

## JBubbleBobble

### Panoramica

---

Composizione del gruppo : 1 persona

Nome : Alessandro Ciccarone

Matricola : 1700581

Corso : Teledidattica

#### Specifiche di base implementate

- gestione del profilo utente ( nickname, avatar, partite giocate, vinte e perse, livello ...)
- gestione di una partita completa con almeno 8 livelli giocabili
- 3 tipi di nemici con grafica e comportamento di gioco differenti
- gestione del punteggio e delle vite
- 10 power up
- schermata hai vinto, game over, continua, classifica
- 2 diversi tipi di bolle
- uso appropriato di MVC , Observer Observable e altri Design Pattern
- adozione di Java Swing per la GUI
- utilizzo appropriato di stream
- riproduzione di audio sample
- animazioni ed effetti speciali

#### Specifiche extra implementate

- 4 livelli in più, di cui uno con un Boss (livello 11) ed un livello bonus (livello 12)
- un totale di 13 power up (di cui 10 oggetti che aumentano il punteggio, e 3 oggetti che conferiscono poteri temporanei, od un punteggio nel caso i poteri siano già attivi)
- pannello OPZIONI
  - Permette di abilitare e disabilitare separatamente la musica e gli effetti sonori.
  - Permette di abilitare e disabilitare la modalità FUN, che permette di ascoltare brani musicali selezionati al posto della classica canzone di Bubble Bobble. (abilitato di default)
  - Permette di scegliere lo sfondo generale dell'applicazione tra 3 diverse proposte.
  - Permette di selezionare la difficoltà del gioco, che influenza il punteggio ed il numero di vite alla rinascita.
- pannello PAUSA
  - Mette in pausa il gioco mostrandolo come sfondo sotto un pannello semitrasparente.
  - Permette di tornare al menu, di ricominciare il livello, di mostrare il tutorial, di tornare alla partita.

- Permette di utilizzare il music player (vedi dopo)
- Permette di disabilitare o abilitare musica e suoni.
- pannello CREDITS
  - Mostra la fonte di alcuni elementi utilizzati nel gioco.
  - Ringraziamento a chi ha fatto da beta tester.
  - Easter egg (cliccando l'uovo 50 volte si apre il pannello SURPRISE).
- pannello EASTER EGG
  - Mostra una foto del mio bellissimo cane dopo aver trovato l'easter egg.
- pannello TUTORIAL
  - Spiega brevemente come giocare e mostra i punteggi dei vari oggetti
- supporto a Mouse e Tastiera
  - I menù sono navigabili sia con il mouse che con la tastiera, i bottoni reagiscono graficamente alla selezione corrente ed al passaggio del mouse.
- Music player che permette di cambiare canzone (precedente o successiva) e di abilitare o disabilitare separatamente musica e suoni.
- Eseguibile .exe per diffonderlo ai miei amici e farli diventare Beta Tester.

## Decisioni di progettazione

---

### Logica del gioco generale

Principalmente il gioco si basa su un Thread della classe Game nel pacchetto main. Questo thread calcola il tempo passato e decide di aggiornare separatamente i dati del gioco e la grafica, sulla base dei parametri FPS (frame per second, per la grafica) ed UPS (update per second, per i dati).

Aggiornare i dati più velocemente di quanto si aggiorni la grafica permette di avere un'esperienza fluida.

Un FPS maggiore di 60 è inutile perchè il gioco ha già raggiunto il massimo della fluidità.

Un UPS maggiore del valore di default (180) farebbe percepire il gioco come velocizzato, mentre un valore inferiore lo farebbe percepire rallentato, ritengo che 180 sia ottimale per un'esperienza piacevole.

In un ciclo while infinito all'interno del metodo run() della classe Game (implementa runnable) viene costantemente calcolato un valore "delta" per l'update dei dati e della grafica.

Nel momento in cui questo valore supera 1, vuol dire che è stato superato il numero di frame o update impostati per ogni secondo, quindi vengono rispettivamente aggiornati grafica o dati e viene sottratto 1 al delta, in questo modo eventuali avanzi temporali non vengono persi.

Altro elemento fondamentale della logica del gioco è un Enumerazione chiamata Context, in cui sono enumerate tutte le varie situazioni di gioco (partita, pausa, menu, opzioni etc...)

Nel momento in cui è ora di aggiornare i dati, il thread chiama il metodo di update del controllore del contesto corrente, attraverso uno switch.

Nel momento in cui è ora di aggiornare la grafica, il thread chiama il metodo repaint del JPanel, che a sua volta chiama il metodo draw del contesto specifico (sempre attraverso uno switch nella classe Game).

In questo modo update di dati e grafica sono centralizzati nella classe Game ed è semplice aggiungere o rimuovere uno specifico contesto.

## Logica dei livelli

Un livello è strutturato principalmente in un array bidimensionale di interi, che formano una matrice di "caselle" in cui il giocatore sarà libero o meno di muoversi.

Questa matrice viene caricata da un file di testo, che oltre a varie informazioni sul livello, come numero, posizione di partenza e posizione di partenza dei nemici, offre una rappresentazione del livello sotto forma di lettere che vanno da A ad L e zeri.

Nel momento in cui una casella è 0, vuol dire che è spazio vuoto dove le entità si potranno muovere liberamente, nel caso in cui una casella sia una lettera, questa lettera rappresenta il tipo di casella che verrà mostrata nella grafica.

Questo approccio permette di usare uno svariato numero di tipi di caselle, io ho scelto di usare un tipo di casella diversa per ogni livello.

Un file di un livello è strutturato come segue:

- riga 0 = numero livello
  - riga 1 = coordinate di partenza giocatore (x,y)
  - riga 2 = coordinate di partenza dei nemici di tipo ZEN
  - riga 3 = coordinate di partenza dei nemici di tipo INVADER
  - riga 4 = coordinate di partenza dei nemici di tipo MIGHTA
  - riga 5 = coordinate di partenza dei nemici di tipo BOSS
  - da riga 6 a riga 27 = tipo di caselle utilizzate nella riga

Qui un esempio del livello 10.

## Logica delle entità

Un'entità è un oggetto dotato di movimento ( giocatore, nemici, bolle ).

La logica dell'entità è che hanno un indicatore di posizione x ed y che li rappresentano all'interno dello spazio di gioco, formato dalla grandezza di una casella (32) moltiplicata per le caselle in altezza e larghezza.

Dei controlli di collisione mostrano:

- se un'entità si trova in una casella dove è permesso muoversi ( o meno ) creando così dei confini.
- se un entità entra in collisione con un'altra entità ( utile per sapere se una bolla colpisce un nemico, se il giocatore viene toccato da un nemico, se il giocatore tocca un collezionabile... ) Il giocatore si muove sulla base dei tasti della tastiera premuti, mentre le altre entità si muovono in modo autonomo in base alla progettazione delle singole entità.

## Logica della grafica

Ho scelto di usare Java Swing ed utilizzare un unico JPanel che viene disegnato in base al contesto corrente.

### **GRANDE CRITICITA' DI PROGETTAZIONE**

In origine ho scelto di non utilizzare un Layout, bensì di disegnare manualmente qualsiasi elemento grafico del gioco, progettando classi personalizzate per i bottoni e posizionando manualmente qualsiasi cosa. Ho scelto di fare così perchè mi sembrava di avere il controllo assoluto sulla grafica, quello che non ho pensato è che stavo basando qualsiasi posizione sulla base della risoluzione del mio schermo ( ad esempio la schermata di gioco è di 732 x 768 pixel, che sul mio schermo è perfetta ).

Quello che non avevo considerato, è che cambiando la risoluzione, tutti gli elementi grafici rimangono della stessa grandezza, non adeguandosi allo schermo.

Nel caso di una risoluzione minore, c'è il pericolo che la finestra non entri nello schermo e non c'è modo di ridimensionarla, nel caso di una risoluzione maggiore, c'è il pericolo che la finestra risulti molto piccola.

Per risolvere questo problema, notato a pochi giorni dalla consegna del progetto, ho deciso di adottare un meccanismo di scala, che calcola la differenza tra le dimensioni dello schermo corrente, ottenute con ToolKit, e le dimensioni dello schermo su cui ho progettato il gioco.

Questa divisione da una scala per cui dividere gli elementi grafici in modo che risultino sempre grandi come sul mio schermo.

Ad esempio, giocando sul mio schermo, questa scala è 1 quindi gli elementi rimangono come sono, giocando su una risoluzione minore, la scala diventa ad esempio 1.2, quindi un elemento che sul mio schermo prenderebbe 100 pixel, in uno schermo con questa scala prenderebbe 83 pixel, risultando delle stesse dimensioni con cui è stato progettato.

Questo meccanismo mi sembra funzionare perfettamente per ogni aspetto grafico, tranne nel caso della rappresentazione delle caselle di gioco. Essendo queste caselle progettate graficamente per essere continue una a fianco all'altra, nel caso in cui la grandezza della finestra non sia un numero divisibile senza resto con il numero delle caselle di una riga, risultano delle linee nere di divisione tra alcune caselle. Nulla che impedisca di giocare ma non è esteticamente bello.

Di seguito un esempio di risoluzione adeguata in cui non si verificano problemi, e di risoluzione in cui si presenta questo problema grafico.

Ottimale



Problematica



Una soluzione forse più adeguata sarebbe l'utilizzo di un layout, ma ho il timore che problemi grafici si presentino a prescindere in determinate risoluzioni per via della natura delle caselle. Un'altro svantaggio di questa mia scelta progettuale è che un eventuale svista nello scalare un elemento non è visibile finché non viene testata su una risoluzione differente.

Data l'impellenza di consegnare il progetto, per ora ritengo che la soluzione della scala sia adeguata in quanto non crea problemi in una qualsiasi risoluzione che abbia un rapporto standard di 16:9 (come quasi tutti gli schermi).

## Logica dei profili

Così come i livelli, i profili si basano su file di testo che vengono creati quando si crea un profilo, e caricati ogni volta che il gioco parte o che viene aperta una sezione che li riguarda (carica, cancella salvataggi, nuova partita, statistiche..).

I profili vengono aggiornati in svariate situazioni, ad esempio quando si passa un livello, quando si muore, quando si creano, quando si cancellano.

Nella creazione di un profilo nel pannello NEW GAME, è obbligatorio scegliere un nickname ed un avatar (che ha solo una valenza estetica per la rappresentazione del profilo, il giocatore è a prescindere il draghetto verde).

Un file di un profilo è strutturato come segue:

- riga 0 = nickname
- riga 1 = tipo di avatar
- riga 2 = ultima vita del giocatore
- riga 3 = numero di partite vinte
- riga 4 = numero di partite perse
- riga 6 = ultimo punteggio
- riga 7 = miglior punteggio
- da riga 8 a riga 12 = migliori 5 punteggi

Ecco un esempio.

```
1 ALESSANDRO
2 1
3 1
4 5
5 3
6 28
7 446600
8 446600
9 446600
10 375400
11 330800
12 243400
13 148000
```

Per l'ordinamento dei profili e dei loro dati nella sezione statistiche, vengono utilizzati gli stream che manipolano la lista di profili per adattarla alla specifica richiesta di ordinamento.

## Logica dei nemici

I nemici si muovono autonomamente grazie all'utilizzo di una logica specifica per tipo di nemico ed un thread specifico per tipo che aggiunge un controllo sul tempo di alcune azioni (ad esempio cambia direzione o salta ogni tot secondi). Questo thread serve anche per permettere un'animazione nel momento in cui i nemici muoiono (che altrimenti scomparirebbero all'istante) e per liberare i nemici dalle bolle se rimangono intrappolati troppo tempo.

Ogni nemico è capace di seguire i movimenti del giocatore, cercando di raggiungere la sua posizione ogni tanto (un numero intero casuale costantemente aggiornato è usato come condizione per abilitare l'inseguimento).

Un nemico muore se viene toccato mentre è intrappolato sul tetto all'interno di una bolla.

I nemici di tipo Zen hanno un comportamento di base, seguono il nemico, cambiano direzione se sbattono contro un muro o casualmente, saltano casualmente.

I nemici di tipo Invader hanno una probabilità più alta di seguire il nemico e mostrano un'animazione arrabbiata quando lo fanno, inoltre non cambiano direzione casualmente ma solo se sbattono contro il muro.

I nemici di tipo Mighta non camminano ma fluttuano diagonalmente finché non sbattono contro un muro del bordo, cambiando direzione, possono attraversare tutte le altre caselle che non sono del bordo esterno.

I Boss seguono il comportamento dei nemici Mighta ma non possono essere intrappolati nelle bolle, piuttosto hanno una quantità di vita che diminuisce ad ogni collisione con una bolla, e muoiono nel momento in cui la loro vita arriva a 0. Quando vengono colpiti mostrano un'animazione differente.

## Logica delle bolle

Le bolle vengono generate in un numero limitato, tenendo conto delle bolle già presenti nel gioco (sia quelle sparate sia quelle che hanno un nemico all'interno).

Se è possibile sparare una bolla, essa viene creata in base alla direzione del giocatore quando ha sparato, viaggiando per un tot in orizzontale e poi iniziano a salire fino al tetto.

Se durante la loro vita vengono a contatto con un nemico (tranne un nemico Boss) lo intrappolano e lo portano in cima al tetto. Una volta arrivate sul tetto, anche se vuote, esplodono automaticamente grazie ad un timer associato ad ogni bolla al momento della creazione.

Una volta che una bolla tocca un nemico, in realtà cessa di esistere, ed è il nemico a cambiare animazione per mostrare di essere intrappolato e a fare conteggio nel conto delle bolle massime, insieme alle bolle ancora esistenti.

Le bolle hanno un controllo di collisione per restare all'interno del bordo esteriore del gioco.

Esistono due tipi di bolle, quella normale (azzurra) e quella speciale (giallo/verde) che ha la particolarità di lasciare un collezionabile (oggetto o potere) quando esplode (sia vuota che con un nemico).

## Gestione dell'audio

Suoni specifici vengono fatti partire quando si verificano azioni specifiche, come saltare, sparare una bolla, colpire un nemico, subire un danno, morire (diverso per giocatore e nemici), superare un livello.

Per quanto riguarda l'audio, come dicevo prima ho integrato un rudimentale riproduttore musicale che permette di poter cambiare canzone da una libreria ristretta di canzoni, o di riprodurre in loop la canzone base del gioco se non si desidera attivare la modalità divertimento.

## **PICCOLA CRICITA' DI PROGETTAZIONE**

L'utilizzo di un gestore audio che riconosce solo file di tipo WAV, fa sì che il 90% del peso del gioco sia dato dalle canzoni. Dato che i file WAV (a bassa risoluzione) sono 10 volti più grandi rispetto ad un mp3 (ad alta risoluzione), ho dovuto limitare in quantità e qualità la libreria musicale.

Sarebbe utile utilizzare un motore audio che legga i file MP3 in modo da evitare la conversione ed alleggerire enormemente il programma.

Questa criticità nasce dalla scelta di riprodurre varie canzoni, altrimenti per una singola canzone ed una manciata di effetti audio non avrei riscontrato un problema di spazio nonostante l'utilizzo di file WAV.

## IDEA DI PROGETTAZIONE

Sarebbe interessante dare un'opzione per associare una canzone ad un particolare livello, in modo da apprezzare tutta la libreria musicale, o scegliere di ascoltare liberamente le canzoni.

# Design Patterns utilizzati

---

## Observable Observer

Ho utilizzato il pattern OO principalmente per quanto riguarda la HeadBar, che visualizza livello, difficoltà, punteggio, poteri attivi e vite durante una partita.

A differenza della logica del gioco che ha bisogno di essere aggiornata costantemente, ho ritenuto più opportuno usare questo pattern per la headbar che viene aggiornata saltuariamente (solo quando si perde una vita, si guadagnano dei punti, si acquisisce un potere o si cambia livello o difficoltà).

La classe HeadBar è quindi osservatore, ed osserva la classe Player (che estende Entity che è Osservabile) e la classe LevelController (che racchiude il livello corrente con l'informazione sul numero del livello).

Quando Player imposta il numero di vite, o acquisisce punti o poteri, invia una notifica agli osservatori.

Quando LevelController imposta un livello, invia una notifica agli osservatori.

Un altro caso in cui ho usato questo pattern è stato per ProfilesController, anche qui per la natura saltuaria dell'aggiornamento dei dati. ProfilesController è osservatore ed osserva anche lui Player e LevelController, in modo da poter salvare la partita ogni volta che un livello viene applicato o che il giocatore muore.

Entrambi gli osservatori ricevono come notifica l'istanza dell'oggetto Player e un oggetto di classe Level.

## Singleton

Il singleton è stato generosamente usato per lo scambio di informazioni tra classi diverse. E' applicato a qualsiasi controller (controllers dei contesti, delle entità, dei profili ecc.) ed in generale a qualsiasi entità che non richieda più di un'istanza.

Nel modello è applicato anche ai contenitori, speciali classi che si occupano di immagazzinare oggetti di specifici tipi (ad esempio BubbleContainer contiene una lista di oggetti Bubble e si occupa della loro creazione).

Ho ritenuto che fosse una buona scelta quella di usare singleton perché mi permette di chiamare con chiarezza specifici metodi di altre classi, ad esempio è utilissima per l'Audio Manager perché permette di riprodurre uno specifico suono da qualsiasi parte del codice. Ovviamente nel modello non vengono chiamati singleton presenti al di fuori del modello.

## Model-view-controller

Ho cercato il più possibile di distinguere le classi in base alla loro appartenenza rispetto al modello MVC, e credo di essere riuscito ad ottenere un modello riutilizzabile al 100%. Diverso il discorso per view e controller, perchè in un MVC ideale il controller doveva aggiornare i dati del modello, e la view doveva aggiornare la grafica, invece ho scelto di delegare al controller il controllo della grafica, la view semplicemente chiama dei metodi del controller per sapere cosa disegnare. Questo mi ha permesso di semplificare la gestione delle entità e delle componenti grafiche a discapito della riusabilità del codice.

Nel modello si trovano le entità di base, i livelli, i contenitori delle entità, l'enumerazione Context, i profili e la logica generale.

Nel controller ci sono i controllori di tutti gli oggetti del modello, il main, ed il motore del gioco (classe Game) che si occupa di aggiornare sia il modello sia la grafica.

Inoltre si trovano i controllori di file e audio che permettono rispettivamente il caricamento e salvataggio dei file ed il caricamento e la riproduzione dell'audio.

Nella view sono presenti il JFrame ed il JPanel su cui viene disegnato il gioco, la classe HeadBar (che disegna informazioni sulla partita in corso), e le varie componenti grafiche (Bottoni di vario genere, MusicPlayer e altro).

## Presentazione dei vari contesti

---

Di seguito verranno illustrati i vari contesti del gioco, mostrando le principali funzioni e delle immagini esplicative.

Ogni contesto è associato ad un controller che si occupa di gestire l'aggiornamento dei suoi dati e la rappresentazione grafica dello stesso.

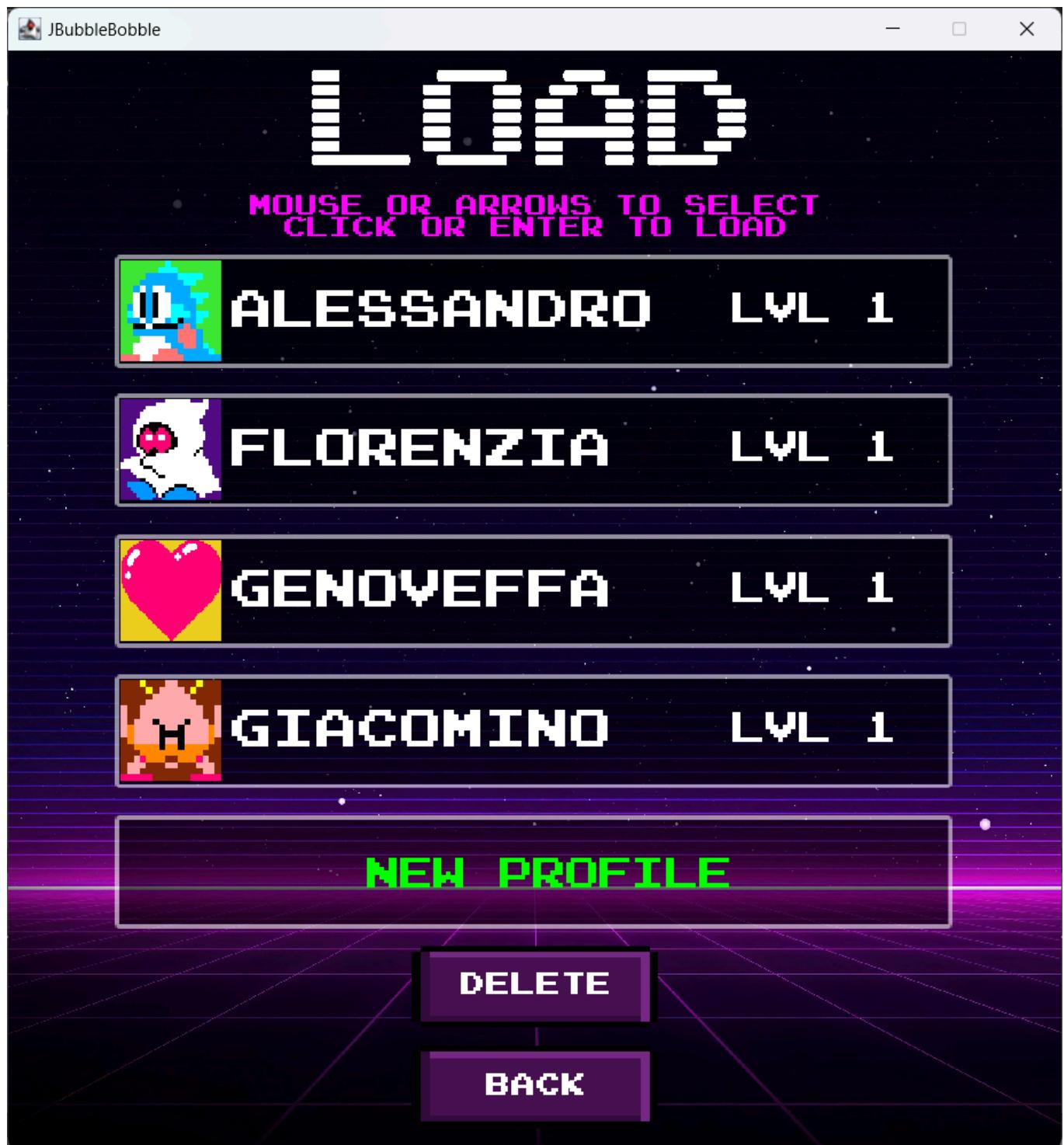
## MENU

Il menù è il primo contesto ad essere applicato all'avvio del gioco. Permette di navigare tra le varie sezioni del gioco, ed il suo controller si occupa di far partire la musica all'avvio. Da qui è possibile accedere alle sezioni per caricare una partita, avviare una nuova partita, cambiare le impostazioni, visualizzare le statistiche di gioco, i crediti, ed uscire dal gioco.



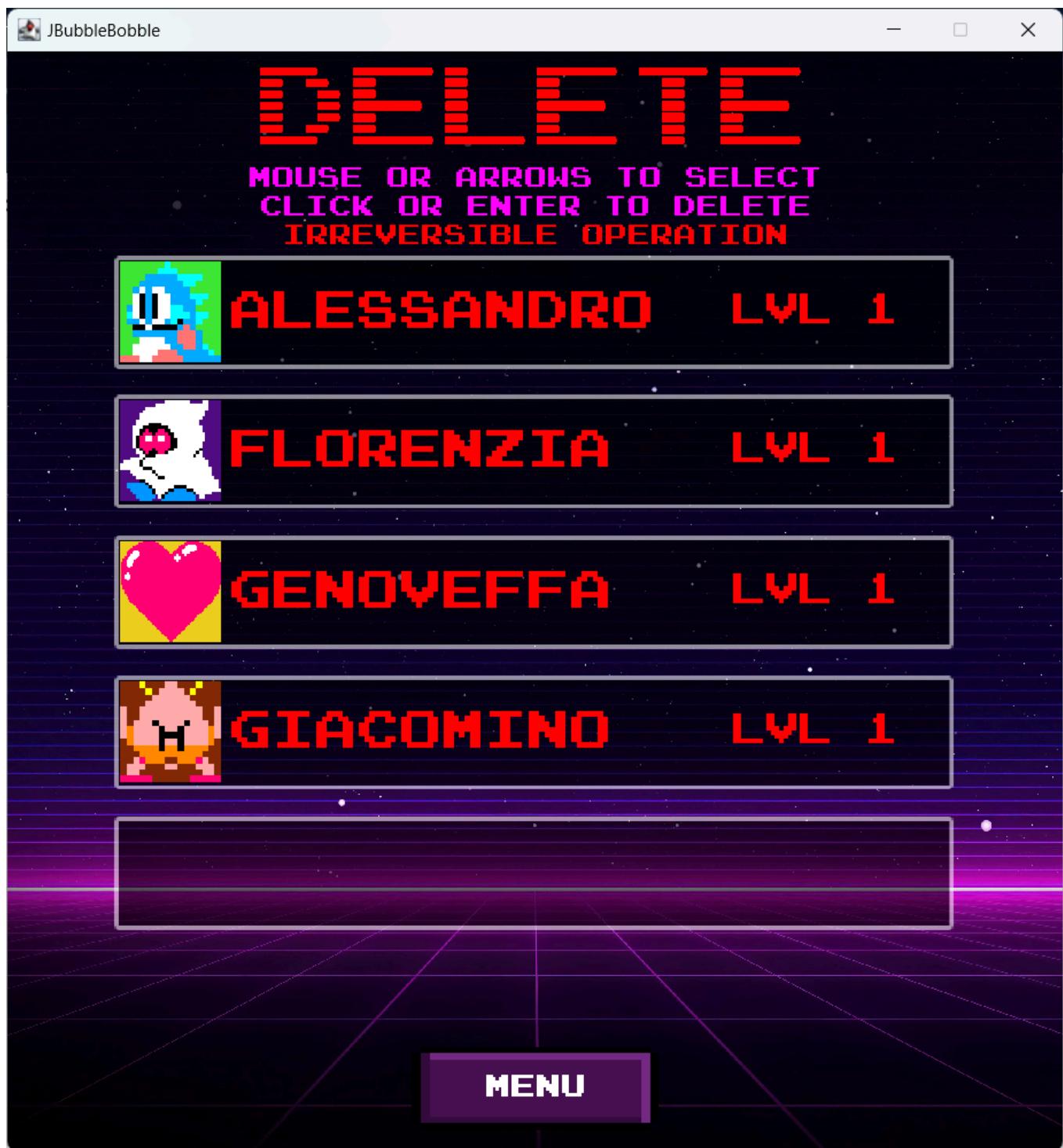
## LOAD

Da qui è possibile visualizzare i profili salvati e caricare le rispettive partite. Nel caso in cui non ci siano profili salvati, si viene automaticamente reindirizzati alla schermata per creare un nuovo profilo.  
E' presente un tasto per accedere al contesto in cui si possono cancellare i profili.  
Ho imposto un limite di 5 profili per mostrarli in modo piacevole in questa schermata, sarebbe utile utilizzare un menu a scorrimento per superare questo limite.  
Se viene selezionato uno slot vuoto, si passa al contesto di creazione di un nuovo profilo.



## DELETE

Molto simile alla sezione Load, permette di eliminare i profili salvati. Viene aperta automaticamente se si prova a creare un nuovo profilo ma ce ne sono già cinque, in tal caso viene mostrato un avviso che invita a cancellare almeno un profilo, e quando lo si fa il tasto “new” viene mostrato in modo da poterne creare un altro.

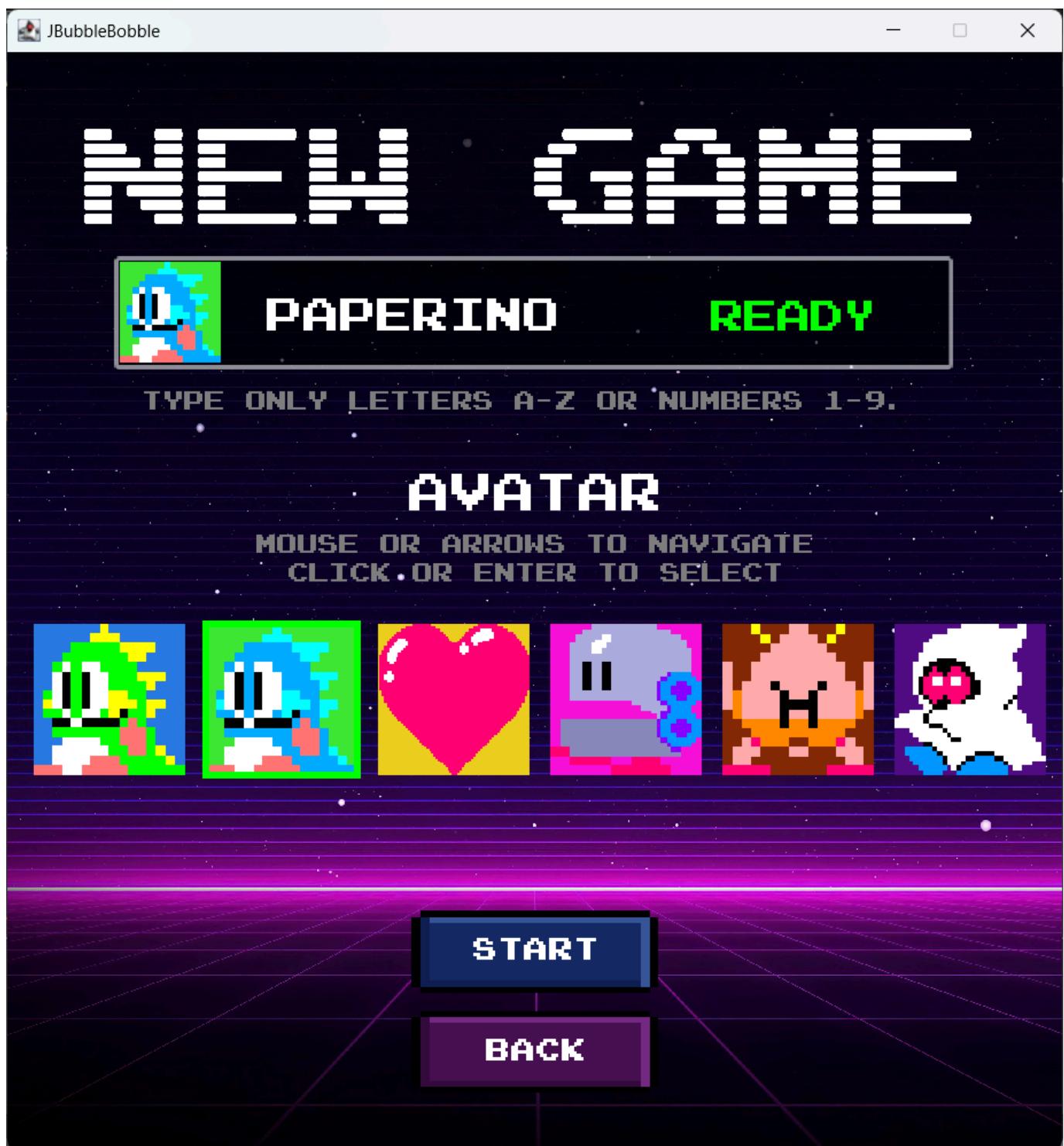


## NEW GAME

Contesto per creare i nuovi profili. Degli avvertimenti fanno presente se manca il nickname, se è selezionato un nickname già utilizzato da qualcun'altro o se non è stato selezionato un avatar. Nel momento in cui è tutto pronto, compare il tasto START che crea il profilo ed inizia la partita (portando al contesto PLAYING). Uscire senza premere start non crea il profilo.



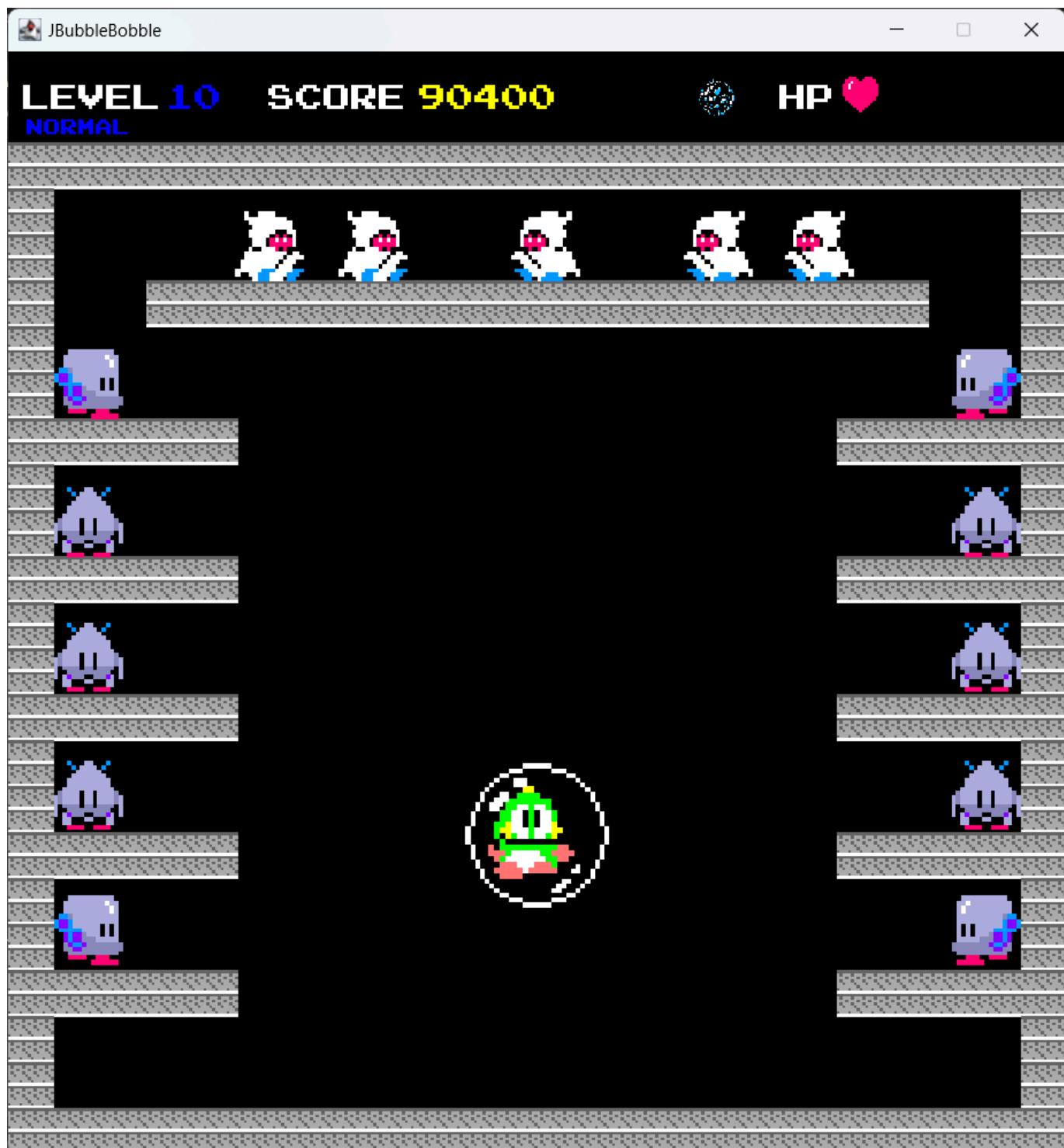
ALESSANDRO **NICKNAME TAKEN**



## PLAYING

Contesto in cui si svolge il gioco vero e proprio. Mostra la schermata di gioco, la headbar con le informazioni correnti, e permette di mettere pausa con il tasto Esc.

Porta automaticamente alla schermata NEXT LEVEL o GAME OVER o WIN a seconda della situazione.

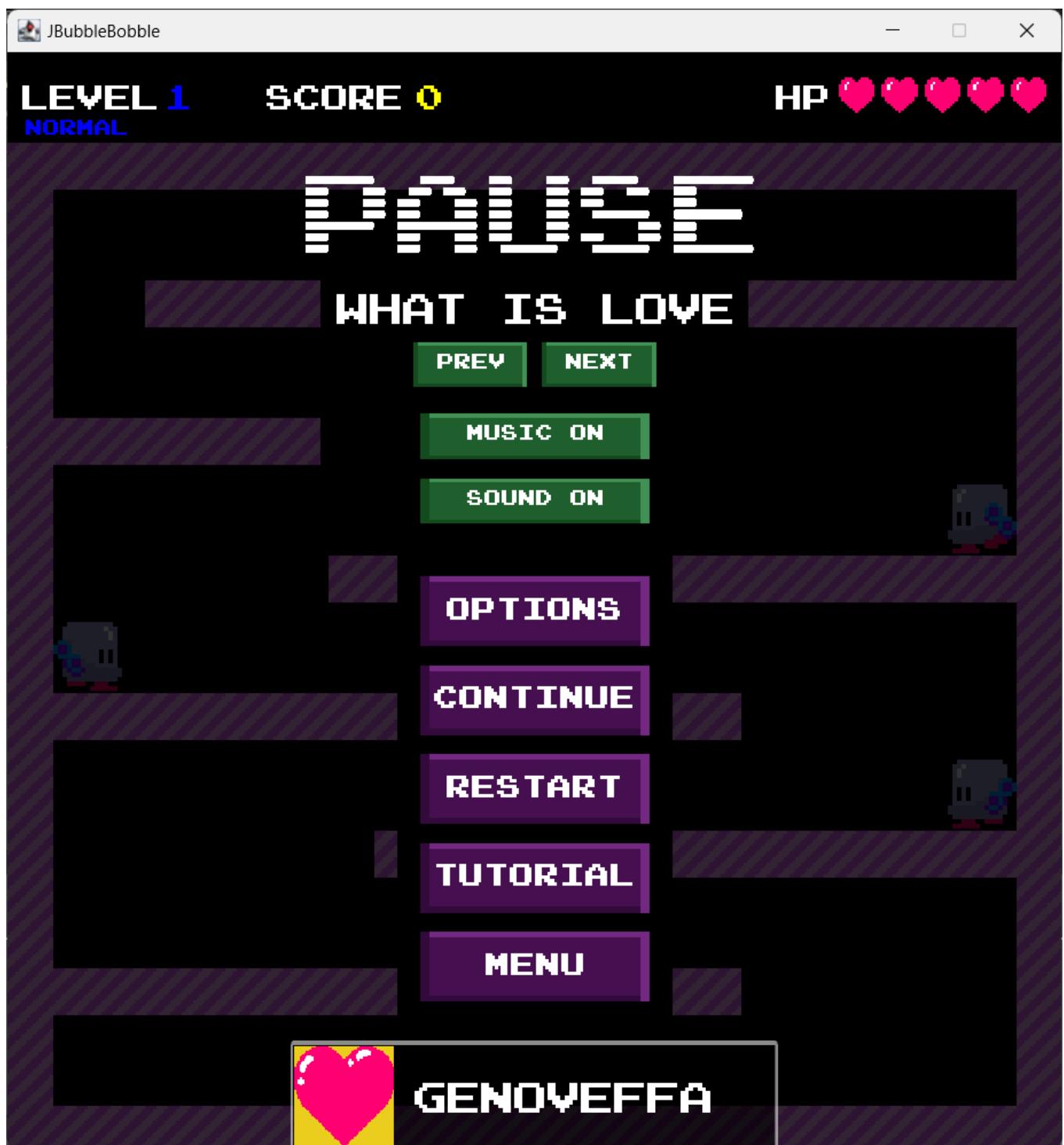


## PAUSA

Attivata dal tasto Esc durante una partita, mette in pausa il gioco (che non viene più aggiornato) e disegna dietro uno sfondo semitrasparente l'ultima situazione del gioco.

Presenta un lettore musicale per cambiare canzone e gestire l'audio, e tasti per continuare, resettare il livello, uscire alle impostazioni o al menu principale, o guardare il tutorial.

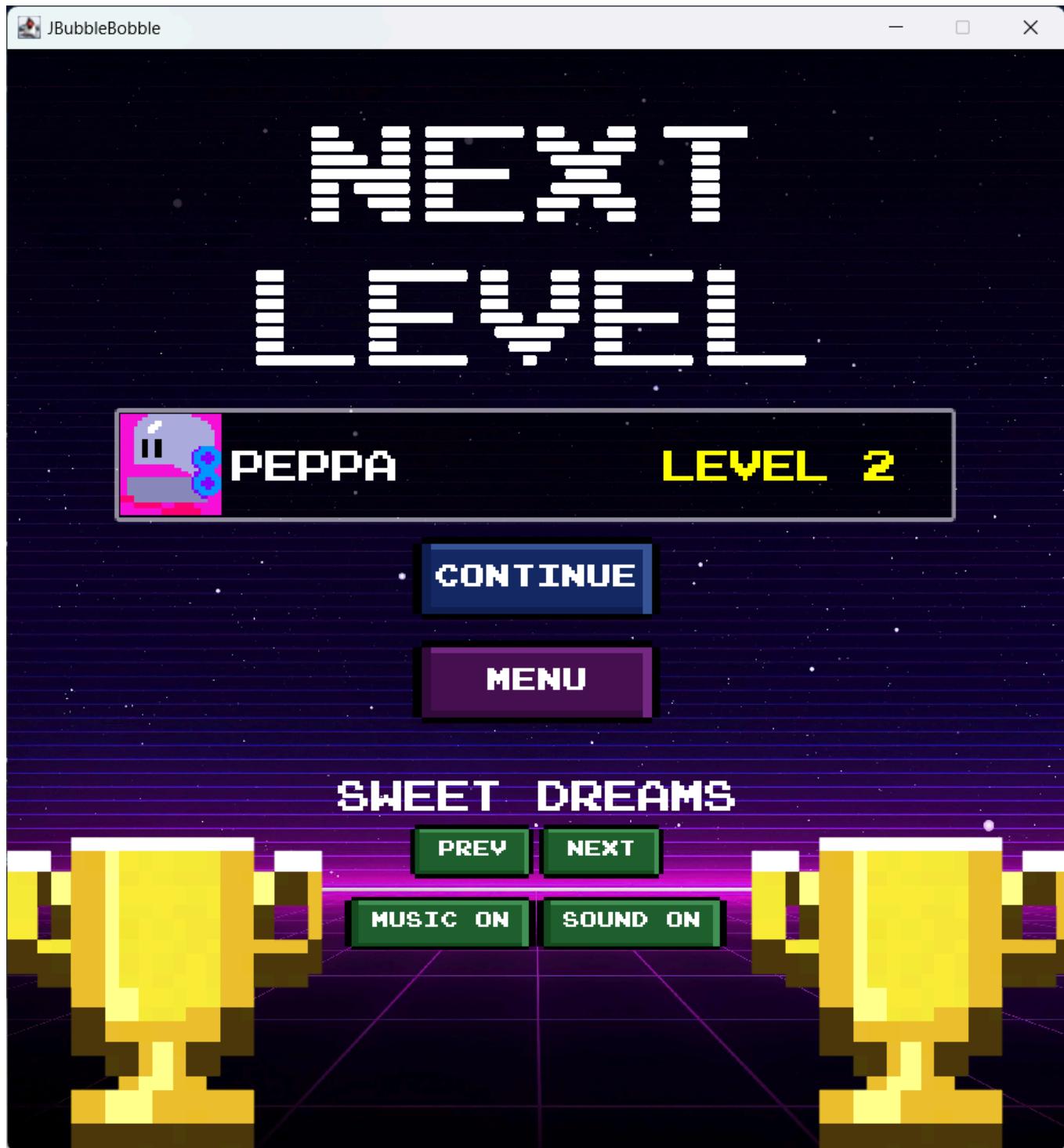
Viene mostrato anche il profilo attuale.



## NEXT LEVEL

Viene mostrato quando si supera un livello, mostra il profilo corrente, il livello raggiunto, una graziosa coppa e dei tasti per andare avanti o tornare al menu principale. Un livello si ritiene superato se tutti i nemici sono morti e tutti i collezionabili sono raccolti.

Presenta un lettore musicale per cambiare canzone od impostare il suono.

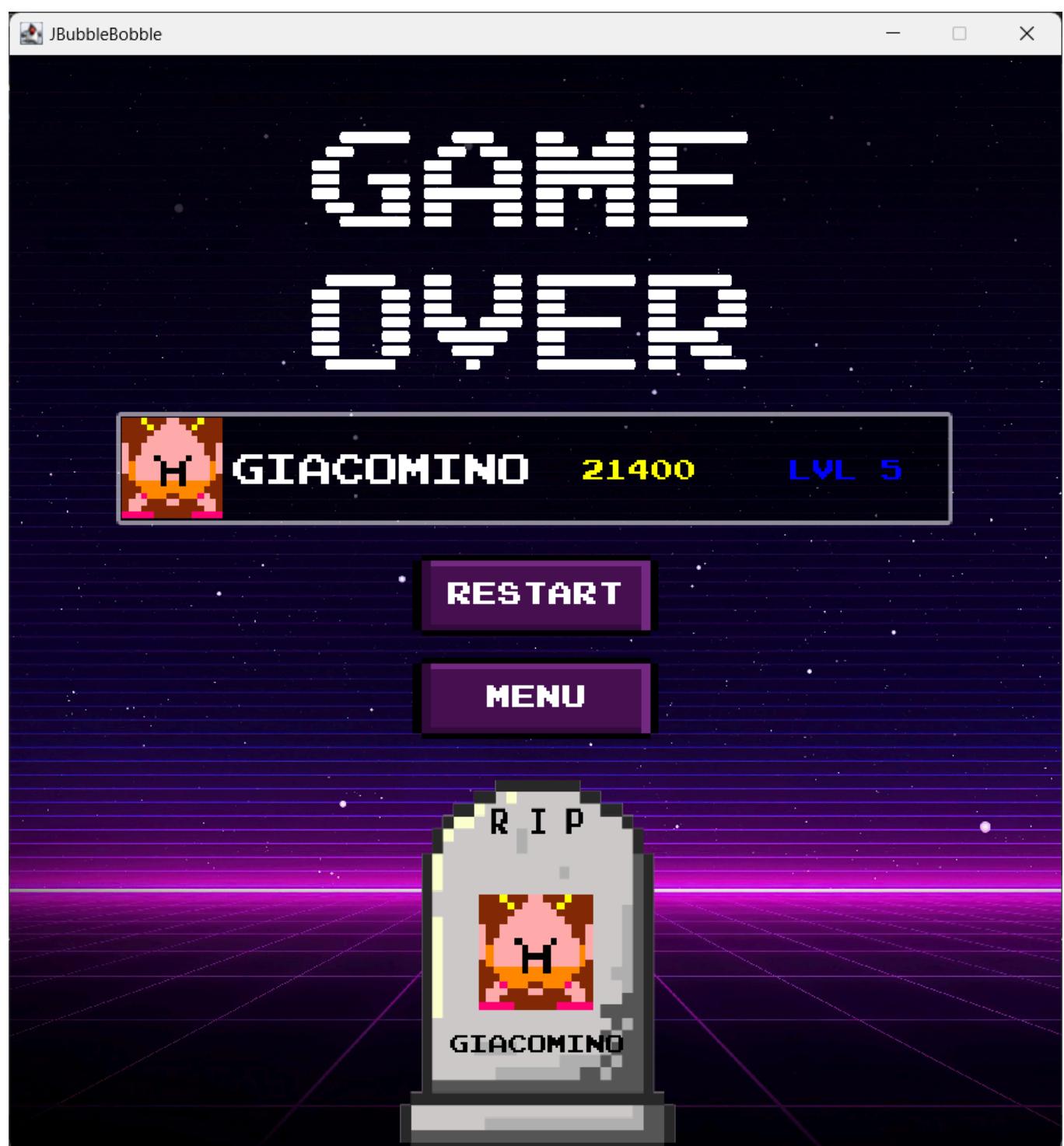


## GAME OVER

Viene mostrato se la vita del giocatore scende a 0. E' possibile tornare al menu o ricominciare, con numero di vite che dipende dalla difficoltà.

In modalità facile si ricomincia col massimo delle vite, 5, in modalità normale si ricomincia con la metà delle vite massime, 2 ed in entrambi i casi si mantiene il punteggio che si aveva quando è iniziato il livello. In modalità difficile si ricomincia con 5 vite ma si torna al primo livello e si azzera il punteggio.

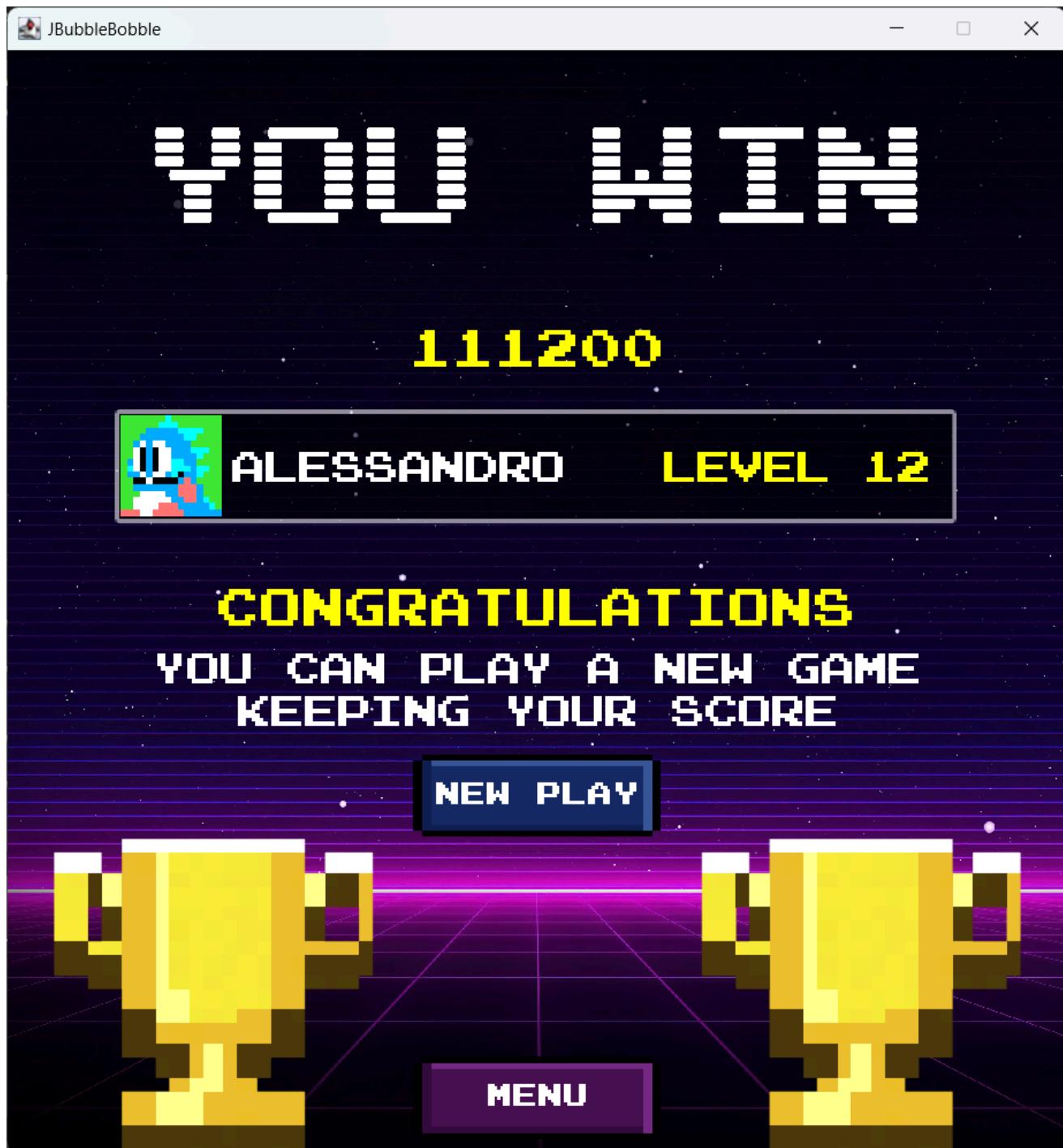
Viene mostrato il profilo con punteggio e livello raggiunto ed una graziosa tomba.



## WIN

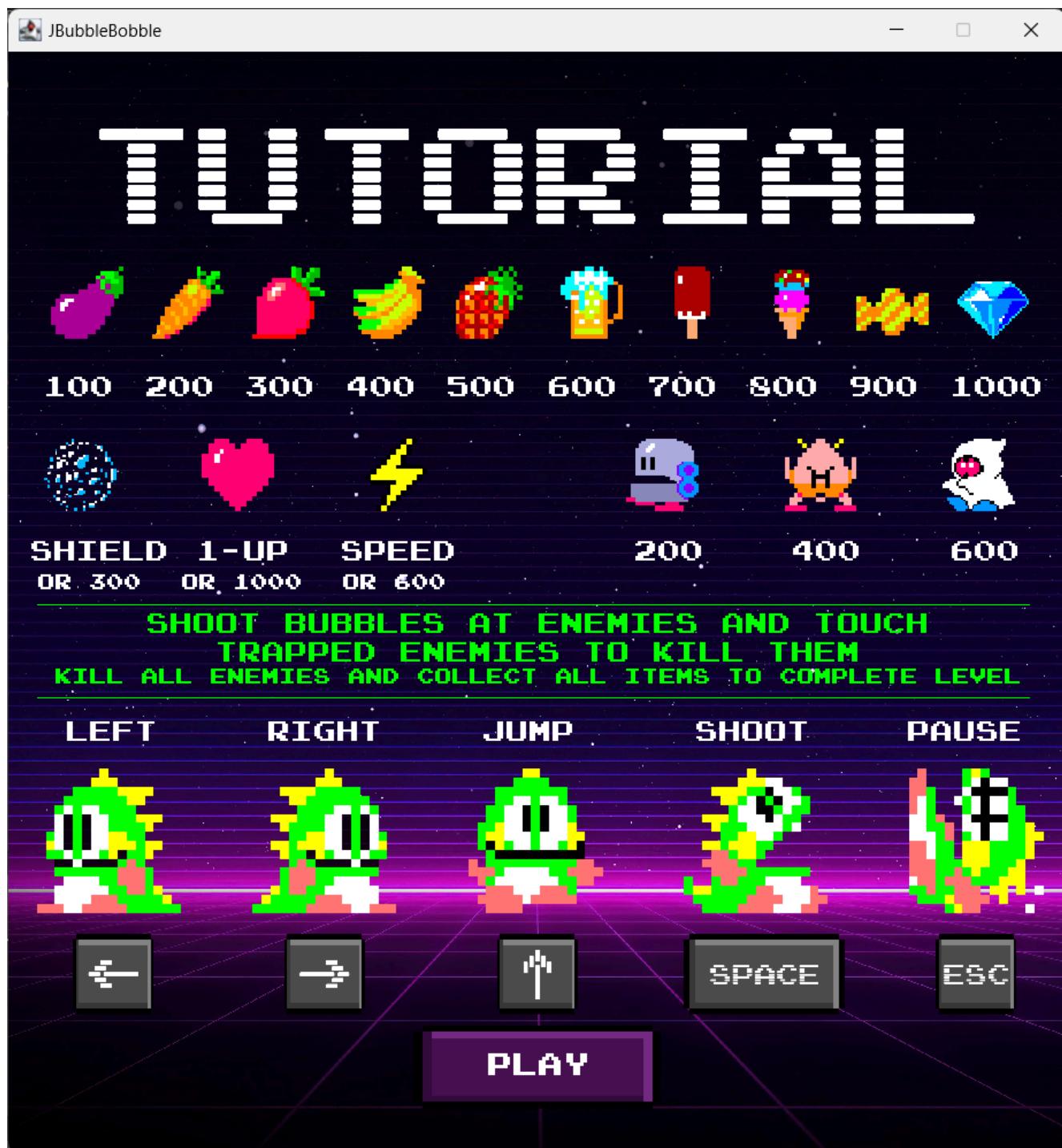
Viene mostrato quando si completano tutti i livelli. E' possibile ricominciare il gioco mantenendo il punteggio attuale o tornare al menù principale.

Viene mostrato il profilo con il punteggio ed il livello, insieme a delle graziose coppe.



## TUTORIAL

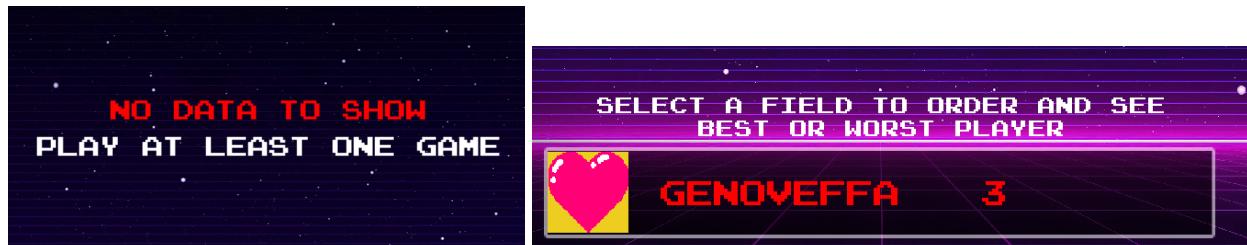
Viene mostrato all'inizio di ogni nuova partita o spingendo il tasto apposito nel menu pausa.  
Mostra i punteggi degli oggetti o dei nemici uccisi, lo scopo del gioco, i tasti per muovere il giocatore.



## STATISTICS

Visualizza i profili, l'ultimo livello giocato, il miglior punteggio, il numero di vittorie, sconfitte, e di partite giocate (vittorie + sconfitte). Si adatta automaticamente al numero di profili e mostra un avviso se non ci sono dati da mostrare. E' possibile selezionare un campo per avere un ordinamento crescente o decrescente su quel determinato campo, e mostrare in basso il giocatore con il dato migliore (verde) o peggiore (rosso) della categoria. Di default vengono mostrati i punteggi migliori, el caso di un ordinamento alfabetico viene mostrato solamente il nome.

Qui un esempio di schermata senza dati, ordinamento per livello più basso, ed in grande ordinamento per miglior punteggio.



NICKNAME	LEVEL	SCORE+	WIN	LOST	TOT
ALESSANDRO	8	863200	4	26	30
GENOVEFFA	3	104800	1	3	4
GIACOMINO	5	23400	0	5	5

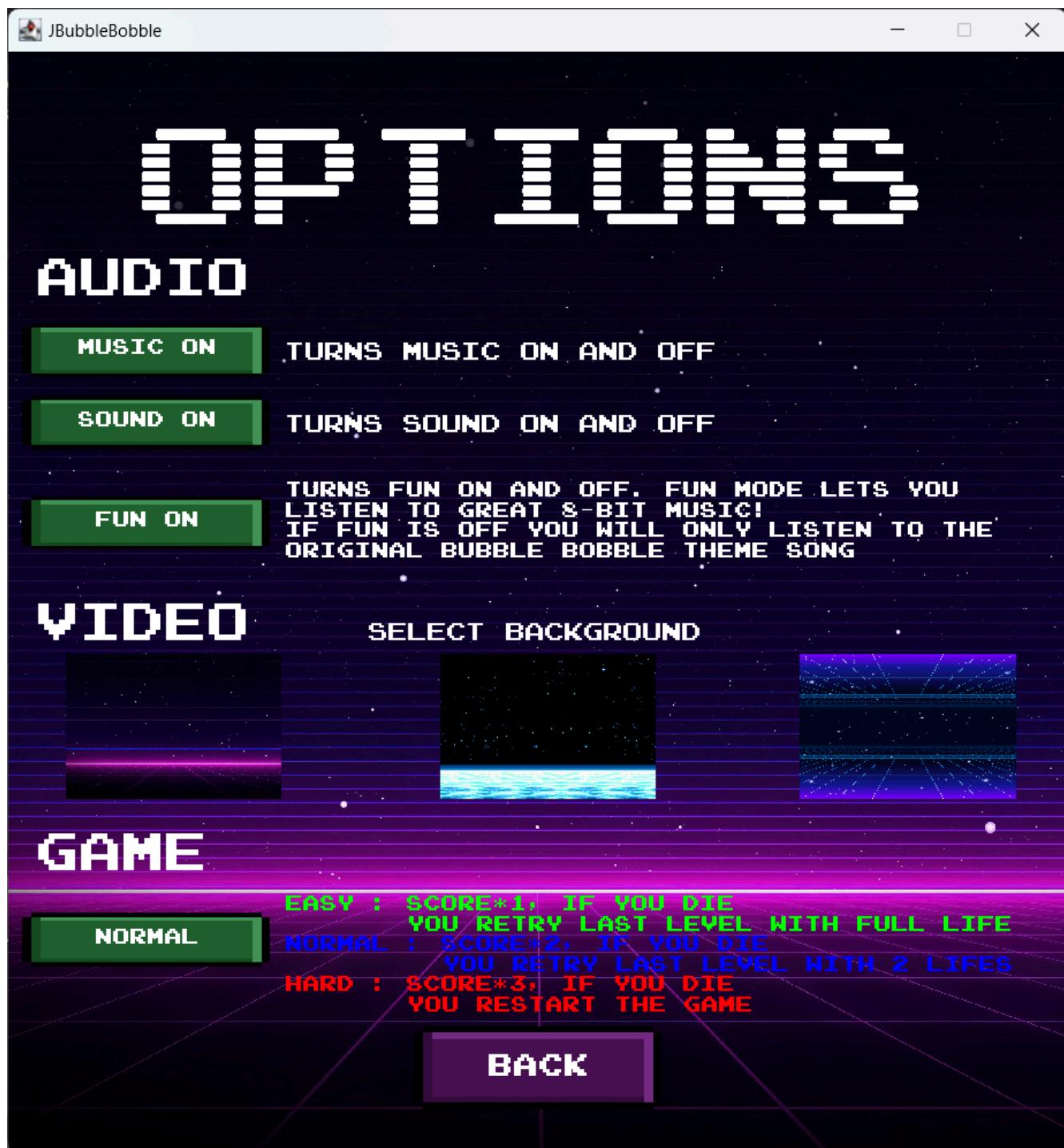
SELECT A FIELD TO ORDER AND SEE  
BEST OR WORST PLAYER

ALESSANDRO 863200

BACK

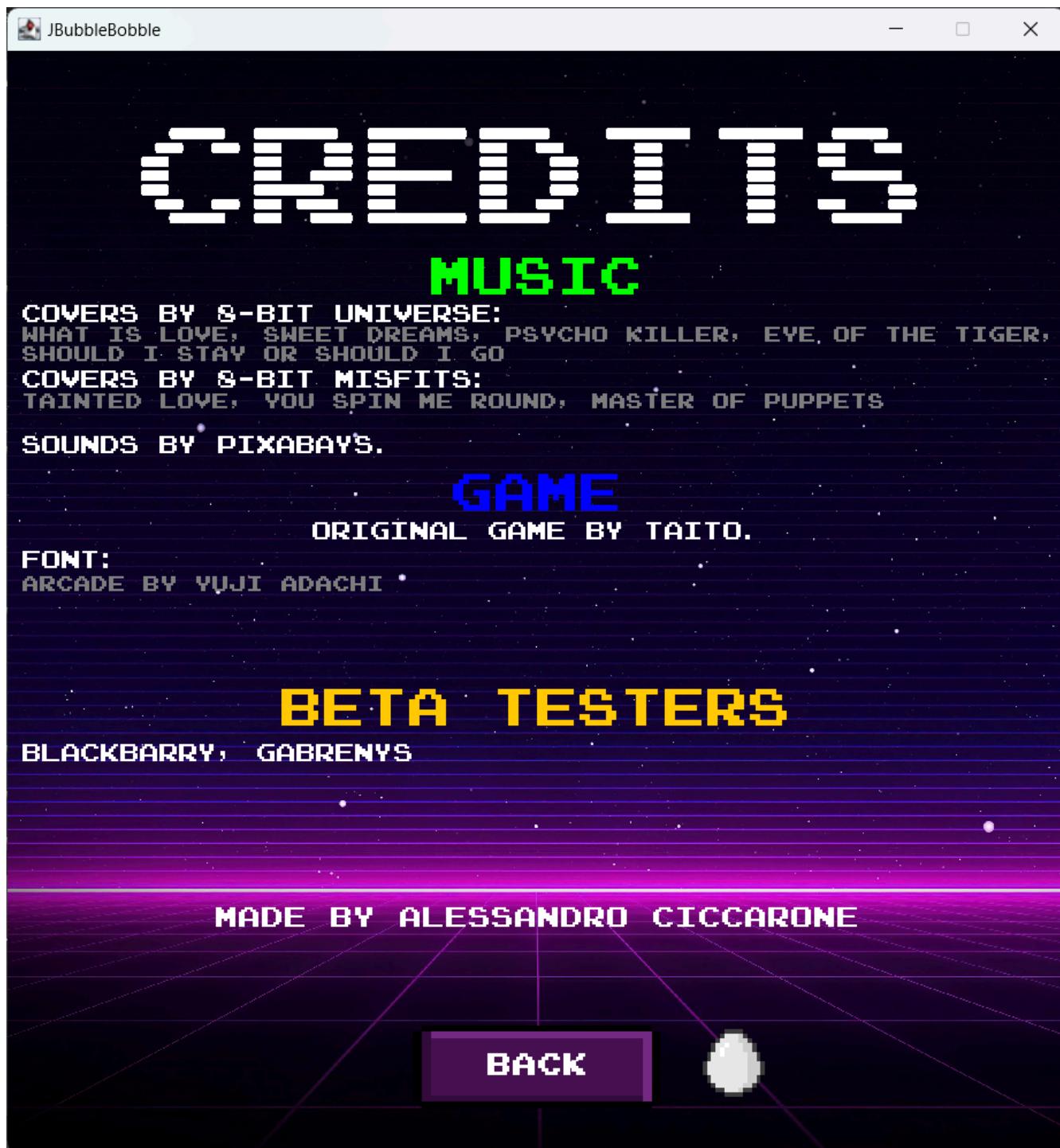
## OPTIONS

Permette di gestire l'audio, lo sfondo dell'applicazione visualizzato in tutti i contesti tranne Playing e Pause, e la difficoltà di gioco. La difficoltà è impostata a Normale all'avvio dell'applicazione, originariamente era anche all'interno del menu di pausa ma questo rendeva facilissimo imbrogliare, quindi l'ho confinata nelle opzioni in modo che uscire dalla partita reimposti il livello.



## CREDITS

Mostra la fonte di alcune componenti del gioco, un ringraziamento a chi mi ha fatto da Beta Tester, ed un piccolo easter egg, che se cliccato dice DAJE, se cliccato 50 volte di seguito porta alla schermata SURPRISE.

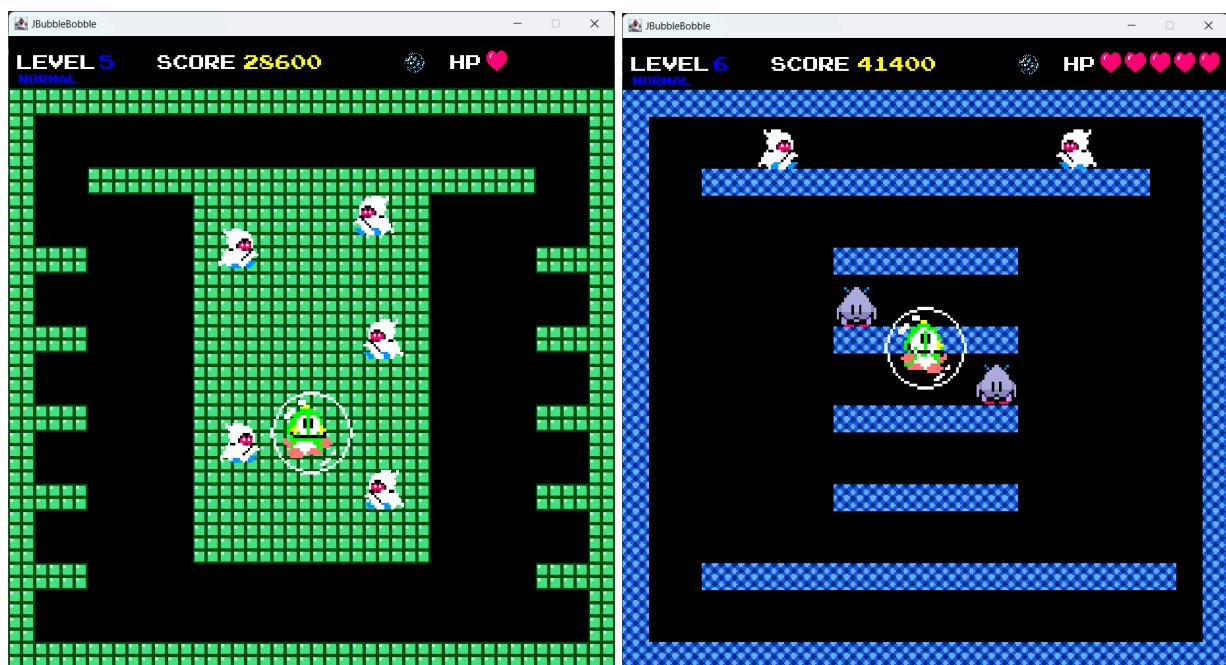
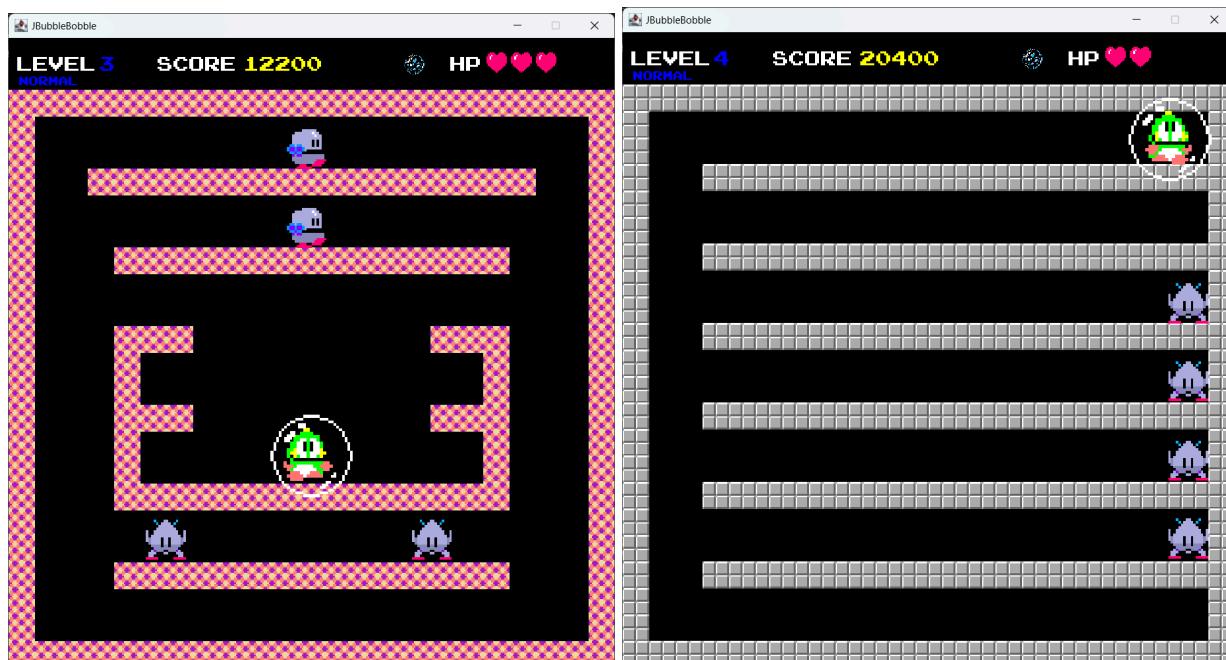
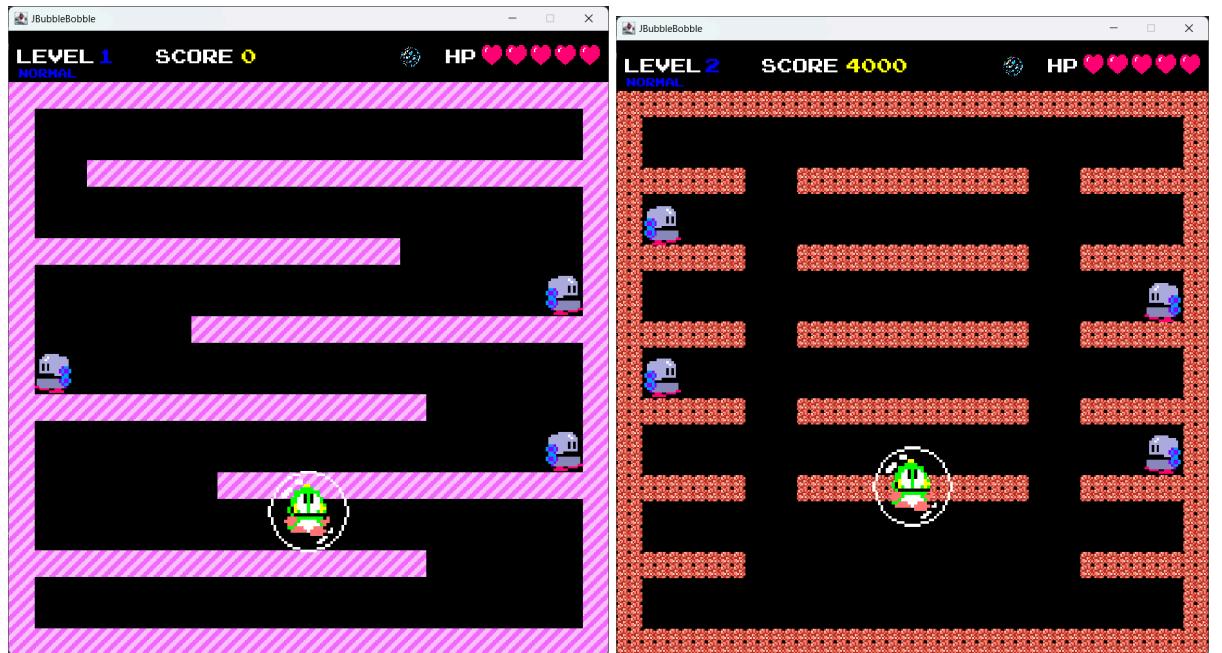


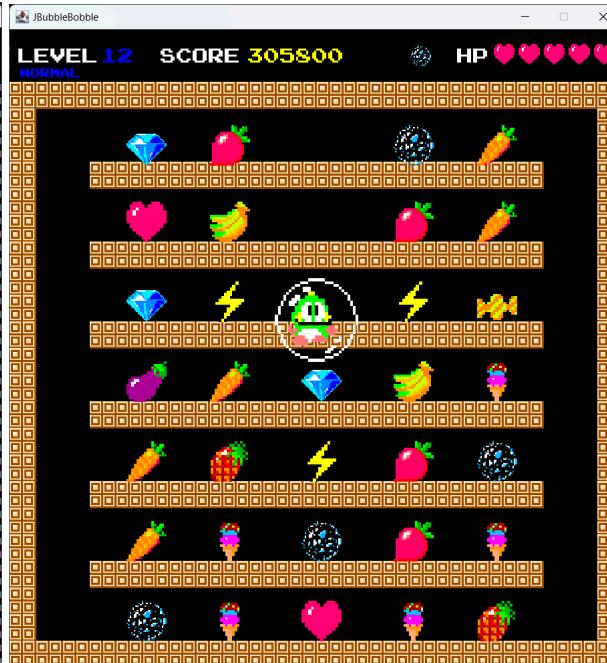
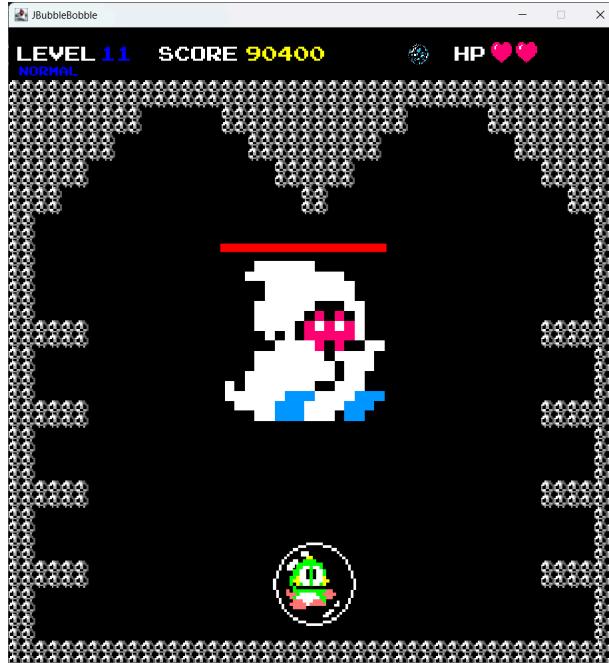
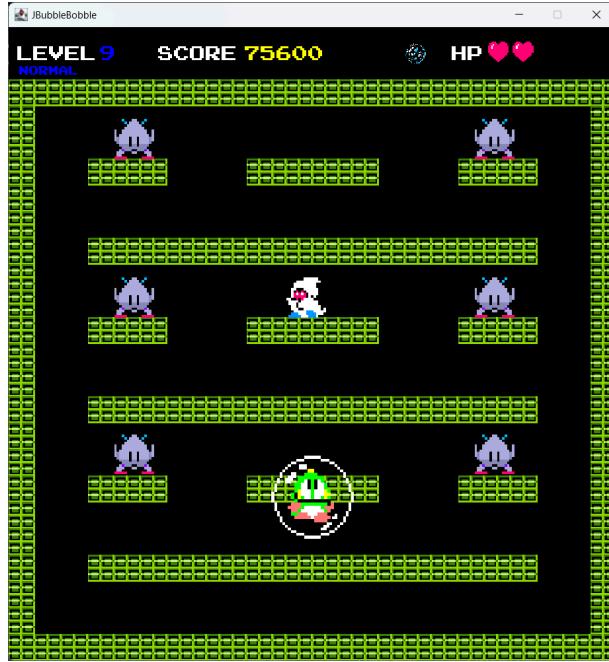
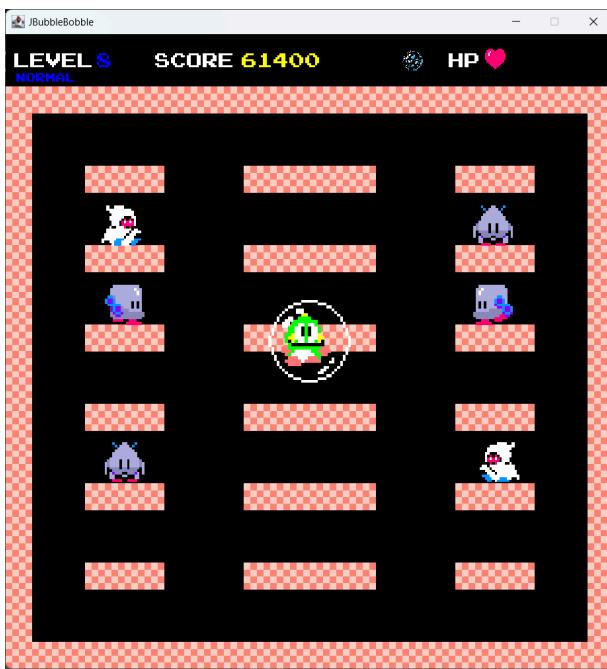
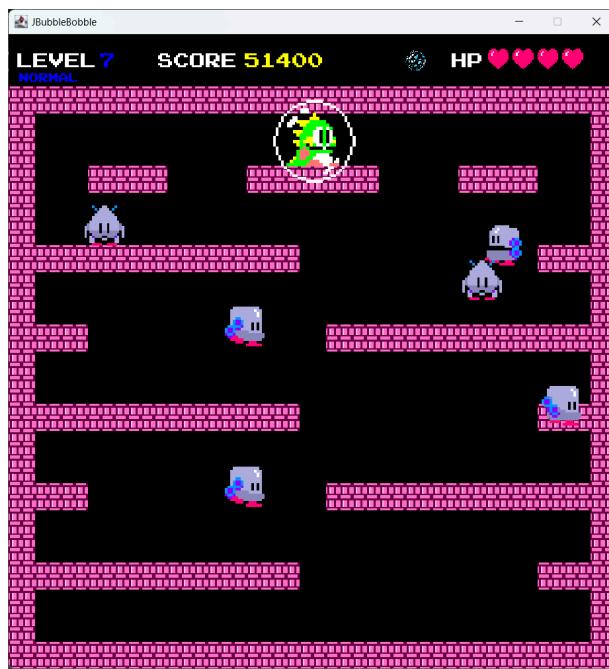
# EASTER EGG

Nulla da aggiungere alla bellezza estrema di questo cagnolone.



# PANORAMICA DEI LIVELLI





# Considerazioni finali

---

## Cose che avrei voluto implementare :

- Modalità fullscreen
- Avatar non solo come immagine profilo ma anche come giocatore
- Salvataggio della difficoltà di gioco nel profilo
- Slot di salvataggio vuoti nascosti nella modalità Delete.
- Gestione del focus perso cambiando finestra
- Migliore gestione della collisione
- Migliore gestione del comportamento dei nemici, che al momento sono molto rudimentali ed un po' ripetitivi, specialmente il boss.
- Gestore musicale che supporta gli mp3
- Codici segreti per attivare i trucchi (vita infinita, cambia livello..)
- Criptaggio dei file di livello e salvataggi, ora facilmente modificabili per imbrogliare (e probabilmente far fioccare le eccezioni)
- Controllo sull'integrità dei file
- Utilizzo appropriato di un layout per evitare di dover scalare a mano ogni elemento grafico.
- Miglior gestione delle statistiche. Per esempio mostrare i migliori punteggi di un singolo profilo, anche perché ho creato una lista dei migliori punteggi ma non viene mai usata.
- Riproduzione di una specifica canzone per ogni livello, in modo da apprezzare la libreria musicale e poter mettere una canzone Metal per il boss.

## Problemi noti :

- Entrare in pausa non ferma lo scorrere del tempo dello scudo attivo.
- L'animazione del giocatore ferito non avviene se il giocatore viene ferito in posizione di partenza (questo perché l'animazione inizia quando si viene colpiti e finisce quando si torna al punto di partenza). **Possibile soluzione** spostare il controllo del tempo di animazione nel thread del giocatore invece che basarsi sulla posizione.
- Le entità saltano su qualsiasi piattaforma sia alla portata del salto, anche se si tratta di una piattaforma alta più di una casella, incastrandosi. Ho ovviato al problema non creando situazioni del genere nei livelli, ma può comunque capitare in quei livelli in cui ho disegnato delle strutture verticali. Di solito basta saltare per uscire dal blocco, una **possibile soluzione** è migliorare il controllo della collisione e tenere conto non solo della prossima casella ma anche di quella dopo.
- Se due nemici vengono uccisi a distanza ravvicinata, viene riprodotto solo un suono, oppure il secondo viene riprodotto in ritardo. Ho provato semplicemente a fermare il suono corrente e farlo ripartire però si crea un effetto loop sul suono. Bisognerebbe indagare ulteriormente. Questo problema si applica a tutti i suoni (tranne le canzoni che ho gestito in modo diverso).