Progetto di Programmazione ad oggetti e Ingegneria del software

Università degli studi di Urbino
Denny Bini

Figura 1. WordsGame

Indice

1 Specifica del software

Il progetto consiste nella realizzazione di un gioco di parole composto da tre livelli con difficoltà crescente, in linguaggio c# windows-based in considerazione anche di una portabilità futura verso il sistema operativo Android.

Ogni livello deve contenere:

- Un titolo con l'indicazione del livello;
- Una descrizione che mostri la modalità di gioco;
- Un punteggio aggiornato in tempo reale in base alla logica di gioco;
- Una griglia di lettere estratte casualmente dall'alfabeto italiano. Ad ogni lettera è assegnato un valore che si basa sulla difficoltà di composizione nella lingua italiana;
- Un bottone per richiedere il ricaricamento del livello;
- Un bottone per richiedere il ricaricamento del gioco.

Requisiti di gioco dei singoli livelli:

- Il 1° livello è composto da una griglia di lettere 5x5 (25 lettere), sarà considerato superato se sono stati totalizzati almeno 30 punti;
- Il 2° livello è composto da una griglia di lettere 6x6 (36 lettere), sarà considerato superato se sono stati totalizzati almeno 60 punti;
- Il 3° livello è composto da una griglia di lettere 7x7 (49 lettere), sarà considerato superato se sono stati totalizzati almeno 120 punti: essendo l'ultimo livello, se superato il gioco termina;
- Sono ammesse concorrenti al punteggio solo parole di minimo 4 lettere;
- Non possono essere selezionate nello stesso livello parole precedentemente composte;
- La composizione delle parole, sarà effettuata con le seguenti indicazioni: una volta selezionata la prima lettera sarà possibile selezionare solo lettere adiacenti all'intera selezione, anche con scelta diagonale. Eccezion fatta se la lettera rimane nei bordi della griglia, in tal caso sarà possibile selezionare altre lettere anch'esse appartenenti al bordo esterno.

2 Studio del problema

Dalla specifica del problema nascono diverse domande:

1. Quale Framework usare per implementare un gioco di parole che possa essere installato su più piattaforme?
2. Come costruire i livelli?
3. Come caricare le immagini e i testi da visualizzare nell'interfaccia?
4. Come intercettare l'interazione del mouse con gli oggetti della scena?
5. Se viene eseguito un click come invocare i metodi dell'oggetto cliccato?
6. Come caricare un dizionario per poter validare le parole?

2.1 Analisi del problema

Di seguito l'analisi delle soluzioni alle domande che ci siamo posti:

1. Scegliamo Monogame in quanto si tratta di un Framework, che ci permetterà qualora lo vogliamo, di fare il porting su più piattaforme, tra cui Android. Inoltre supporta come da specifica richiesta il linguaggio C#.
2. Come costruire i livelli: uso del pattern Builder in quanto il più adatto per la costruzione grafica del livello. Lo scopo è quello di avere un'interfaccia per la realizzazione e l'assemblaggio delle parti, che tramite incapsulamento e information hiding renda di più alto livello il modo in cui esso viene costruito. Inoltre tale scelta potrebbe agevolare notevolmente la modifica degli elementi posizionati nello spazio grafico del gioco, con la possibilità anche di poter definire i testi ed effettuare future personalizzazioni.
3. Come caricare le immagini e i testi da visualizzare nell'interfaccia. La classe Microsoft.Xna.Framework.Game mette a disposizione un metodo, chiamato draw, che a seconda del frame rate video può variare la chiamata tra i 30 e 60 fps (frame per second). Dunque è possibile gestire la visualizzazione a video, inserendo gli oggetti grafici direttamente all'interno del suddetto metodo. Per cercare di rendere più leggero il metodo draw usiamo il pattern Proxy.
4. Come intercettare l'interazione del mouse con gli oggetti della scena: in XNA su cui MonoGame si basa non abbiamo un sistema come in windows form, doppio click sul componente bottone e generazione di un evento. Dunque va valutata una soluzione più a basso livello. E' possibile acquisire la posizione del mouse nel momento del click e calcolare se è avvenuto all'interno dell'area di un oggetto. Quindi come intercettare il click? Il click si intercetta al rilascio del pulsante sinistro del mouse, anche se non basta considerare solo questo caso, anche lo stesso stato di pressione del mouse è infatti importante in quanto consente di selezionare le lettere consecutivamente l'una accanto all'altro e al rilascio del click deselezionarle.
5. Come registrarsi ad un evento: chi si registra al click? Saranno i singoli oggetti della scena che avranno la necessità di cambiare il loro stato in caso di pressione del click del mouse, questo cambiamento di stato potrà permettere al livello di indicare quali lettere sono state selezionate. Dunque in caso di pressione del click la lettera viene selezionata, in caso di rilascio tutte le lettere saranno deselezionate e sarà conteggiato il punteggio. Per effettuare la registrazione si pensa di usare la classe EventHandler di C# che ci permette di registrare e de-registrare i metodi da chiamare, un po' come farebbe un pattern observer con il detach degli oggetti che si sono registrati al subject.
6. Come caricare il dizionario, in modo che resti sempre accessibile e abbia un caricamento di tipo lazy? Il pattern singleton può aiutarci al nostro scopo in quanto ha l'inizializzazione Lazy. Inoltre una volta instanziato, non effettua nuove richieste di apertura / caricamento del file.
7. Come cercare di rendere la logica separata dalla classe che si occupa dell'interfaccia utente? Con la creazione di una classe che si occupa di richiamare i livelli, gli eventi e altre funzionalità legate al gioco.

3 Scelte architetturali

Abbiamo scelto come base di sviluppo il framework di monogame. Di seguito una breve panoramica:

3.1 Breve panoramica su Monogame

Nella sezione precedente, abbiamo visto delle possibili soluzioni per la realizzazione del progetto. Andremo ora a vedere come sono state utilizzate in pratica per risolvere le problematiche sopracitate.

Prima di poter affrontare l'implementazione vera e propria è necessario dare una breve panoramica di Monogame (la spiegazione che segue è estratta direttamente da Wikipedia) Monogame è un'implementazione open source del framework Microsoft XNA 4. Quest'ultimo permette di realizzare videogiochi per diverse piattaforme quali Xbox 360, PC e smartphone con sistemi operativi Windows Phone. MonoGame offre la possibilità di effettuare il porting dei videogiochi realizzati per tali dispositivi al fine di renderli disponibili anche su altre piattaforme quali Linux, iOS, Mac OS X, Android e Windows 8 Metro
Game: è il core dell'applicazione, gestisce internamente il game loop (input, update, render) ed espone le proprietà e i metodi da utilizzare.
Content.ContentManager: consente la gestione delle risorse fisiche del gioco.
Graphics.Texture2D: contiene la matrice di colori di uno sprite bidimensionale, caricati in memoria dall'oggetto ContentManager a partire da un file immagine, tipicamente di estensione PNG per la gestione della trasparenza.
- Graphics.SpriteBatch: consente il rendering degli oggetti Texture2D.
- Vector2: rappresenta una coppia di coordinate bidimensionali per il posizionamento degli oggetti Texture2D.
- Rectangle: rappresenta un'area con quattro lati per identificare posizione, larghezza e altezza degli oggetti Texture2D.

3.2 Schema UML delle classi principali del progetto

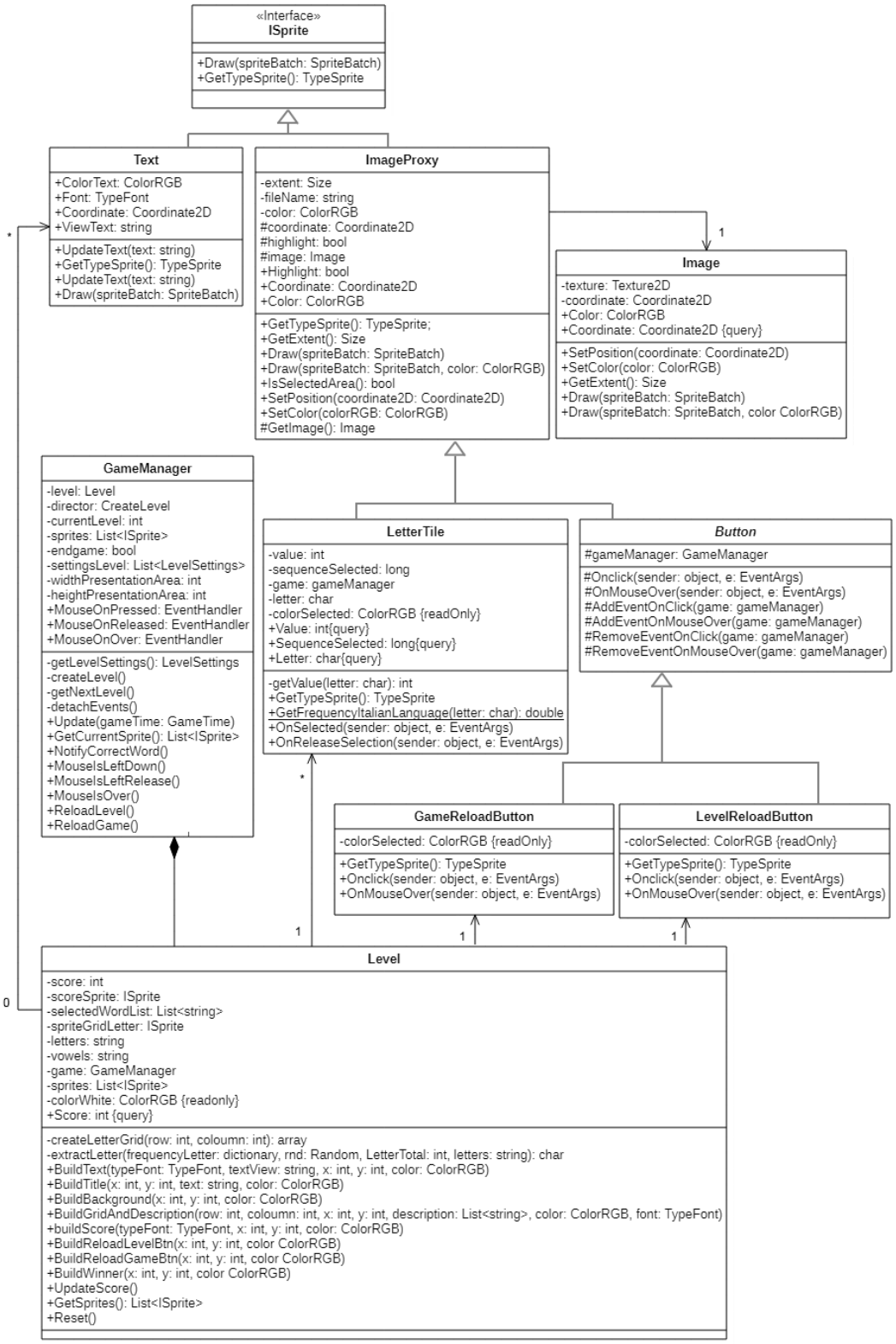

Figura 2. UML - Diagramma delle classi

Ora che abbiamo dato un'occhiata allo schema UML del Diagramma delle classi principali. Vediamo gli oggetti grafici che compongono la scena. Immagini Testi Dunque abbiamo creato un interfaccia chiamata ISprite.

Questa interfaccia viene restituita come prodotto della costruzione del livello. In una prima fase abbiamo il GameManager: classe creata per gestire le interazioni tra gli oggetti del gioco: si occupa di gestire gli eventi, di inizializzare i livelli e terminare il gioco.

Il GameManager costruisce i livelli creando un istanza CreateLevel, questa classe la vedremo più avanti fa parte del Pattern Builder. CreateLevel richiama i metodi di Level con cui vuole costruire il livello, ma il modo in cui quest'ultimo lo fa è nascosto a CreateLevel grazie all'incapsulamento del codice che genera i singoli sprite. Infatti solo Level sa come istanziare i singoli elementi che andranno a comporre la scena cioè i tipi che ereditano da ISprite.

Come primitive dunque ci sono gli oggetti grafici di tipo immagini e testi, ma per specializzarli e permettere di eseguire delle azioni, abbiamo bisogno di alcuni sottotipi per questo abbiamo creato: LetterTile: Il tassello lettera eredita da ImageProxy (pattern Proxy); Button: è una classe astratta dunque non può essere direttamente istanziata, infatti da essa ereditano GameReloadButton e LevelReloadButton.

Il cuore della logica del gioco, validazione delle parole e composizione della griglia con le lettere e conteggio del punteggio è gestita dalla classe Level. Level dunque crea gli oggetti grafici, ma non solo avendo il pieno controllo degli sprite presenti, può validare le lettere selezionate e conteggiare il valore acquisito aggiornando lo score.

Per evitare un uso delle librerie di Xna in tutto il progetto, abbiamo creato una serie di utility, invece che usare il Vector2D ci siamo creati una struttura coordinate2D. Lo stesso vale anche per altre strutture come il colore per cui è stata creata ColorRGB, e TypeFont come tipo enum per i font. Ora vedremo la struttura dei Pattern usati nel progetto.

3.3 Implementazione dei pattern

Partendo dal presupposto che si è deciso di procedere con 3 pattern, elenchiamo di seguito prima i pattern e una breve spiegazione degli stessi:

3.4 Pattern Builder

Pattern Creazionale il pattern Builder è un pattern creazionale, è consigliato quando la creazione di un oggetto complesso deve essere mantenuta separata e indipendente dalle parti di cui l'oggetto è composto e dal modo in cui queste sono assemblate.

Classe/Interfaccia	Ruolo	Descrizione
CreateLevel	Director	CreateLevel con il suo ruolo di director si occupa di implementare un metodo BuildLevel e AddWin e di richiamare i metodi di Level per comporre il livello.
ILevelBuilder	Builder	Definisce l'interfaccia per l'implementazione dei seguenti metodi: BuildBackground BuildTitle BuildText BuildGridAndDescription BuildScore BuildWinner BuildReloadLevelBtn BuildReloadGameBtn Reset; GetSprites;
Level	Concrete Builder	Implementa l'interfaccia IlevelBuilder. Nel dettaglio si occupa di creare le singole componenti del livello, tra cui la creazione degli oggetti sprite del gioco: i tasselli lettere, i testi e i bottoni per eseguire le azioni di reload livello e di inizializzazione di una nuova partita. Inoltre il suo scopo è anche quello di validare le lettere selezionate dall'interfaccia utente per verificare che appartengono ad una parola di senso compiuto. Ogni lettera infatti tiene il suo stato di selezione e la classe Level tiene memoria di tutti gli sprite istanziati nel livello. scorrendo quelli di tipo TileLetter selezionati effettua un controllo di parola attraverso un dizionario caricato nel sistema e in caso di esito positivo chiama il metodo NotifyCorrectWord()del GameManager.

3.5 Pattern Singleton

Pattern Creazionale Assicurare che una classe abbia una sola istanza e fornire un punto di accesso globale a tale istanza

Classe/Interfaccia	Ruolo	Descrizione
GlobalWordsContainer	Singleton	Ritorna un'unica istanza del dizionario richiesto

3.6 Pattern Proxy

Pattern Strutturale Fornire un surrogato o un segnaposto di un altro oggetto per controllare l'accesso a tale oggetto.

Classe/Interfaccia	Ruolo	Descrizione
Image	Real Subject	Immagine da visualizzare a video.
ImageProxy	Proxy	Si comporta come un surrogato dell'immagine, ritardandone il caricamento effettivo solo al suo metodo draw.
ISubject	Subject	Definisce l'interfaccia comune di ImageProxy e Image, infatti definisce i seguenti metodi: Draw e getExtent che sono comuni ad entrambi

Quindi riepilogando il pattern Builder, ci ha permesso di risolvere il punto 2 dell'analisi del problema. Il pattern Singleton il punto 6 e il pattern Proxy il punto 3.

3.7 Gestione degli eventi

La classe GameManager definisce le proprietà EventHandler. Gli EventHandler che abbiamo deciso di realizzare sono tre: MouseOnReleased, MouseOnPressed e MouseOnOver questo perché come anticipato nella precedente sezione nello studio del problema abbiamo bisogno di due eventi per le lettere: uno per sapere se l'elemento è stato pressato e l'altro se l'elemento è stato rilasciato, mentre per i bottoni è importante sapere se il mouse è sopra o è stato fatto click con rilascio del mouse. Gli sprite di tipo lettera si registrano a MouseOnReleased e MouseOnPressed, mentre quelli di tipo bottone a MouseOnReleased e MouseOnOver. L'effettiva verifica se eseguire qualcosa nel metodo avviene solo se è avvenuta un'intersezione tra la posizione del mouse e quella dell'oggetto proprietario del metodo registrato. Per controllare l'intersezione viene verificata che la coordinata x e y della posizione del mouse rientri all'interno di un'area rettangolare circoscritta all'oggetto richiamato. Se l'oggetto richiamato è all'interno dell'area eseguirà il metodo. A seconda del sotto-tipo di sprite eseguirà qualcosa: nel caso della lettera (classe LetterTile) una proprietà verrà settata per sapere se è stata selezionata la lettera e in un'altra proprietà verrà assegnato l'orario con un DateTime.Now.Ticks. Successivamente quando il click del mouse verrà rilasciato le proprietà sono riportate ai loro valori di default. Oltre le lettere abbiamo i bottoni. I bottoni ereditano da ImageProxy e sono dei sotto-tipi di ISprite. Abbiamo il bottone di tipo "GameReloadButton" e "LevelReloadButton". Essi hanno lo scopo di registrarsi al solo evento MouseOnReleased e MouseOnOver, perché necessitiamo di far partire il metodo da richiamare solo dopo che il bottone è stato rilasciato o il mouse è sopra. Gli oggetti di tipo text attualmente non sono agganciati agli eventi, ma se ne avessimo necessità basterà aggiungere la sottoscrizione dell'evento direttamente al loro costruttore, come fatto per gli altri tipi.

Per permettere a qualunque oggetto di poter registrare un suo metodo al click del mouse e di poter ricevere eventuali chiamate, non agganciati ad eventi, il GameManager viene passato alla classe Level ma anche ai sotto-tipi sprite. Ad esempio Level richiama il metodo NotifyCorrectWord del GameManager, per generare un suono. A tale scopo è stato inserito il MoviePlayer. Il gameManager inoltre si occupa di rimuovere gli eventi registrati quando si passa ad un nuovo livello o si termina il gioco. Si è inoltre optato per una creazione dei livelli attraverso dei LevelSettings, GameManager prende in ingresso questi settings e chiede a createLevel di costruire in base al settaggio del corrente livello da visualizzare.

3.8 Gestione delle eccezioni

Abbiamo considerato importante sollevare le eccezioni in caso di impostazioni livello mancanti, file dizionario non trovato e altre eventuali eccezioni, presenti nell'acquisizione di informazioni fondamentali per il proseguimento del gioco.
Non solleviamo eccezioni, nel caso in cui per qualche ragione il Movieplayer non sta funzionando. Dalla documentazione di Momogame si evince che il Movieplayer smetta di funzionare e tenendo conto che un suono non pregiudica la giocabilità (al massimo la arricchisce), si è ritenuto di non procedere con il blocco, ma con un eccezione gestita da try and catch.

4 Documentazione sull'utilizzo

I sorgenti sono scaricabili da Github:

`https://github.com/denny2682/WordsGame`

Requisiti: Visual Studio 2019.
Effettuare restore nuget per lo scaricamento dei pacchetti di Monogame/Xna.
Il progetto è sviluppato per netcore 3.1.
Monogame mette a disposizione un editor per gestire le risorse di progetto, chiamato mgcb.
Disponibile al seguente link:

`https://docs.monogame.net/articles/tools/mgcb.html`

5 Use cases

Gli use cases che abbiamo definito coinvolgono un solo attore: il giocatore chiamato P1 (player).

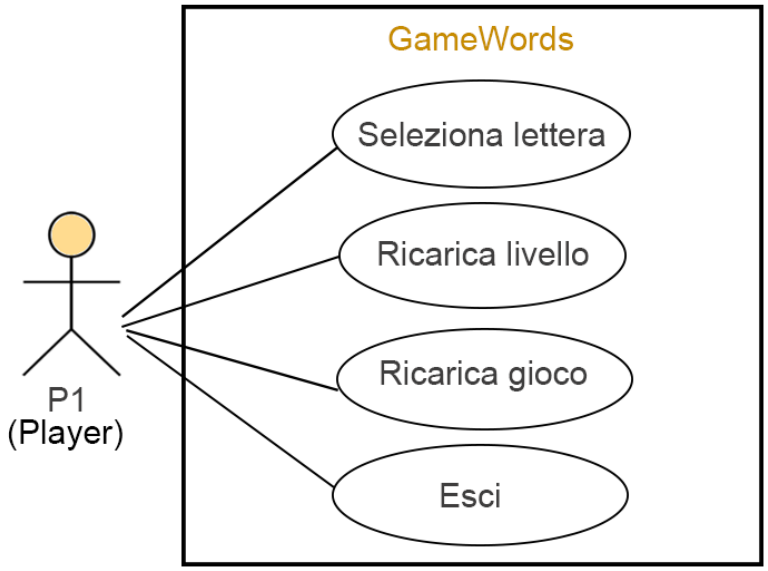

Figura 3. UML-Schema Use cases

Use Case: SELEZIONA LETTERA
ID: UC1
Actor: P1 (Player)
Preconditions: P1 non ha terminato tutti i livelli
Basic course of events: 1) P1: Clicca sopra la tessera lettera senza rilasciare il pulsante sinistro del mouse. 2) La tessera viene evidenziata.
Postconditions: P1 rilascia il pulsante del mouse, la parola che si compone con la selezione delle lettere effettuata precedentemente viene verificata, se corretta aumenta il punteggio.
Alternative paths: P1 non rilascia il pulsante sinistro del mouse, può continuare a selezionare altre lettere presenti nella griglia.

Use Case: RICARICA LIVELLO
ID: UC2
Actor: P1 (Player)
Preconditions: P1 non ha terminato tutti i livelli
Basic course of events: P1 clicca il bottone “Ricarica livello”.
Postconditions: P1 si ritrova nello stesso livello, la griglia delle lettere è completamente rinnovata e il punteggio azzerato.
Alternative paths: -

Use Case: RICARICA GIOCO
ID: UC3
Actor: P1 (Player)
Preconditions: Non esiste una precondizione, l’utente può sempre ricaricare il gioco.
Basic course of events: P1 clicca il bottone “Ricarica gioco”
Postconditions: P1 inizia nuovamente il gioco dal primo livello
Alternative paths: -

Use Case: ESCI
ID: UC4
Actor: P1 (Player)
Preconditions: P1 può uscire dal gioco in ogni momento.
Basic course of events: P1 preme il tasto ESC della tastiera.
Postconditions: P1 termina la partita.
Alternative paths: -

6 Screenshot dell'applicazione

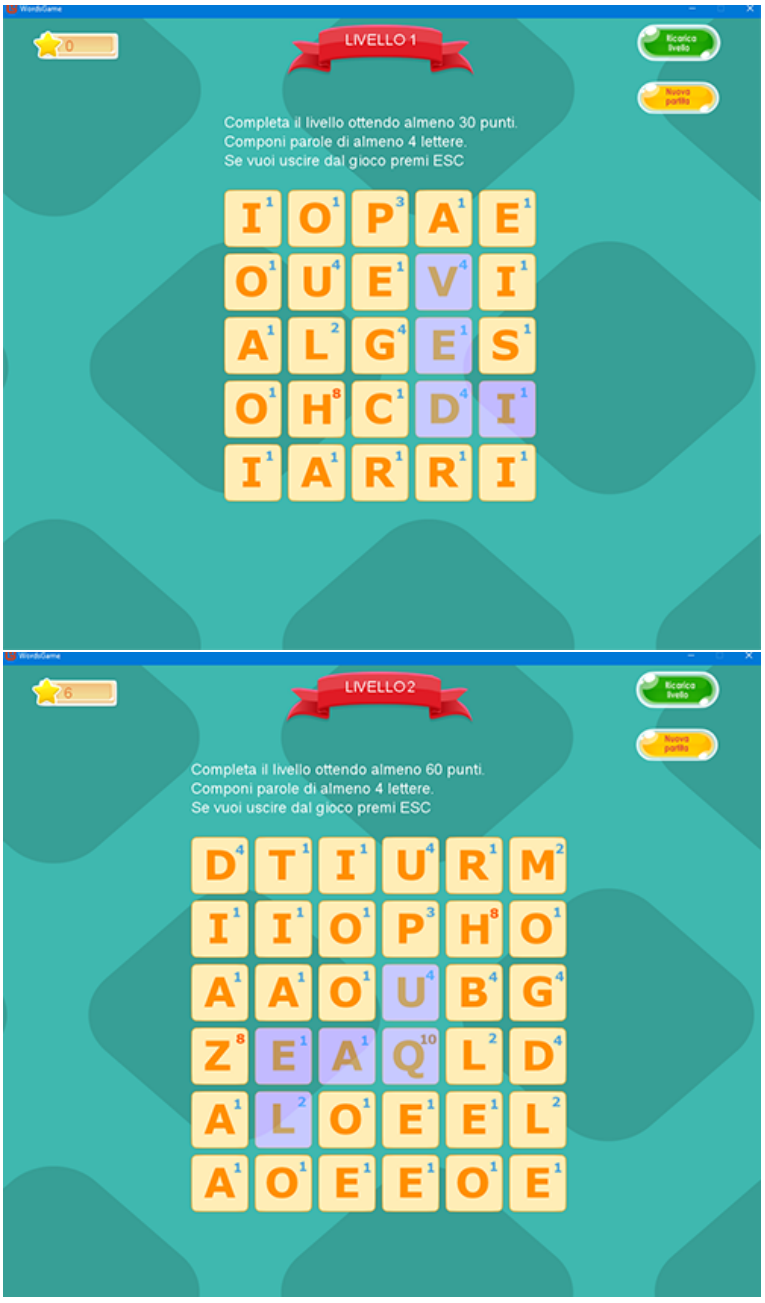

Figura 4. Screenshot WordsGame - Livello 1 e Livello 2

Figura 5. Screenshot WordsGame - Partita vinta