SUPER MARIO BROS

DOCUMENTAZIONE TECNICA

4AI, A.S. 2024/2025

ALESSANDRO REGONESI

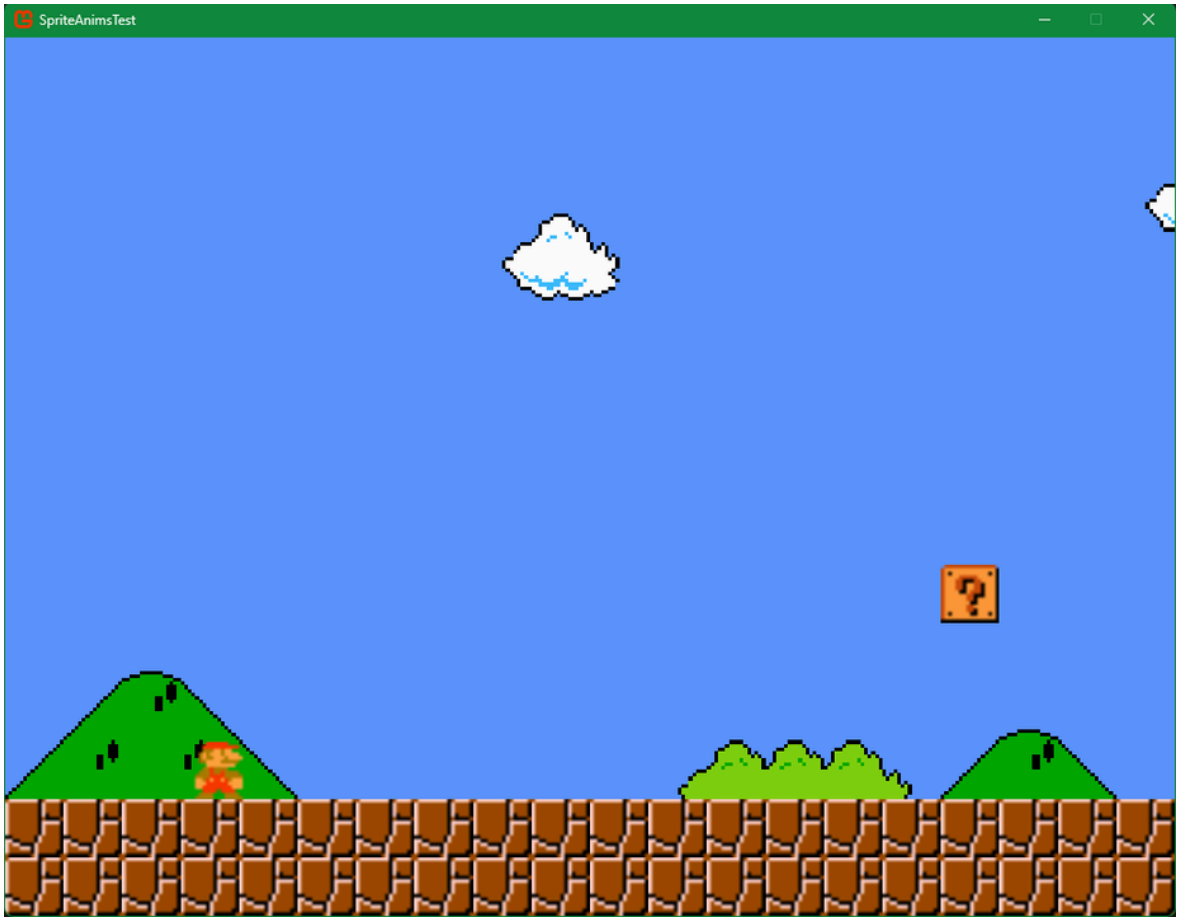
MATTEO BUIZZA

Quale è il problema?

Ricreazione del videogame “Super Mario Bros”, più specificatamente il primo livello del gioco (1-1) usando la programmazione a oggetti, ereditarietà e polimorfismo.

Cosa abbiamo a disposizione?

Per realizzare questo progetto, abbiamo a disposizione un linguaggio di programmazione orientato agli oggetti (Object Oriented Programming - O.O.P.) e un’ambiente di sviluppo integrato (IDE - Integrated Development Environment), nonché il programma Visual Studio di Microsoft, equipaggiato con un framework adatto allo sviluppo di videogiochi 2D, MonoGame, di cui maggiori informazioni si posso trovare qui.



Classi principali del programma

Classe Game1

Questa classe gestisce tutta la logica principale del gioco, collegando tutte le classi per permetterne la corretta visione e l'utilizzo a runtime, cioè durante l'esecuzione del programma.

La classe definisce un gioco basato su XNA e gestisce la logica di rendering, aggiornamento e interazioni del gameplay. Ecco i punti chiave:

- **Setup grafico e finestra di gioco:**
  Viene configurato il GraphicsDeviceManager per impostare la larghezza e l'altezza della finestra (1024×768) e si utilizza lo SpriteBatch per disegnare gli elementi grafici.
- **Gestione del livello e dello scrolling:**
  Viene caricata una texture di livello (3376×240) e si calcola un fattore di scala in base all'altezza dello schermo. La variabile cameraX tiene traccia della posizione orizzontale della visuale, permettendo lo scrolling mentre il giocatore si muove.
- **Player e animazioni:**
  Il gioco utilizza una classe Player (non mostrata nel codice) per gestire fisica, animazioni e aggiornamenti del personaggio (Mario). Viene caricata la sprite sheet di Mario per rappresentarlo graficamente.
- **Musica e pausa:**
  Viene caricata e riprodotta una musica di sottofondo con MediaPlayer. È presente anche la gestione della pausa: premendo il tasto P il gioco si interrompe, mettendo in pausa la musica e visualizzando la scritta "PAUSE" con un font personalizzato.
- **Ostacoli e nemici:**
  Il livello è arricchito da numerosi ostacoli (piastrelle, mattoni, blocchi, lucky block e tubi) aggiunti a una lista. Inoltre, vengono creati nemici (Goomba) che si muovono e interagiscono con il giocatore.
  Durante l'update, il gioco controlla le collisioni tra il giocatore e i Goomba, determinando se il giocatore li schiaccia o subisce un danno.
- **Rendering:**
  La fase di disegno separa il rendering del livello (con point clamp per preservare lo stile pixel art) e quello del player, degli ostacoli e dei nemici (utilizzando linear filtering per un aspetto più morbido). Se il gioco è in pausa, viene sovrapposta la scritta "PAUSE".

Questi sono gli elementi principali che compongono la classe, offrendo una panoramica delle funzionalità implementate nel gioco.

Classe Entity

La classe astratta **Entity** rappresenta la base per ogni oggetto del gioco che possiede fisica e animazioni. Ecco i punti principali:

- **Proprietà e Costruttore:**
  Definisce proprietà come la posizione, la velocità, le dimensioni e il rettangolo di collisione (BoundingBox). Il costruttore inizializza queste proprietà in base ai parametri ricevuti.

- **Fisica e Animazione:**
  Il metodo Update applica una forza di gravità alla velocità, aggiorna la posizione e la BoundingBox, e aggiorna l'animazione corrente (gestita tramite currentAnimation).
- **Rendering:**
  Il metodo Draw si occupa di disegnare l'entità sullo schermo, adattando la posizione in base alla camera e applicando un flip orizzontale se la velocità indica un movimento verso sinistra.
- **Collisioni:**
  È presente un metodo astratto HandleCollisions che le classi derivate (ad esempio, Player o Goomba) devono implementare per gestire le collisioni con gli ostacoli.

Questi elementi costituiscono la struttura di base per le entità del gioco, fornendo funzionalità comuni di fisica, animazione e gestione delle collisioni.

Le classi derivate da essa sono: **Player e Goomba.**

Classe Obstacle

La classe **Obstacle** rappresenta un ostacolo nel gioco e contiene:

- **Proprietà:**
  - ***Position***: la posizione dell'ostacolo.
  - ***Texture***: la grafica utilizzata per l'ostacolo.
  - ***BoundingBox***: il rettangolo di collisione.
- **Costruttore:**
  Inizializza la texture e la posizione.
- **Metodo Draw:**
  Disegna l'ostacolo sullo schermo tenendo conto dello scale e della posizione della camera, e aggiorna la BoundingBox in base alla posizione e alle dimensioni della texture.

Classe Player

La classe **Player** estende **Entity** e gestisce il comportamento del personaggio nel gioco (ad es. Mario). I punti principali sono:

- **Parametri di movimento:**
  Gestisce la fisica del movimento con accelerazione, attrito e velocità massima, sia per camminare che per correre. Include inoltre la logica per il salto (con altezza variabile) e il rilevamento del "skid" in caso di cambio brusco di direzione.
- **Stati del Player:**
  Utilizza un enum con tre stati: *Alive*, *Dying* e *DeadPause*. Questi stati determinano il comportamento del personaggio, passando dalla normale gestione dei movimenti alla fase di morte e successiva pausa prima del reset.
- **Animazioni:**
  Il costruttore inizializza diverse animazioni (idle, jump, skid, walking, running e death) e il metodo Update sceglie quella appropriata in base allo stato e alle azioni del giocatore.
- **Aggiornamento e collisioni:**
  Il metodo Update elabora gli input della tastiera per muovere il player, applica la fisica

(inclusa la gravità) e gestisce i limiti di movimento (confini orizzontali e collisione con il pavimento).
Il metodo HandleCollisions controlla le collisioni con gli ostacoli, correggendo la posizione in caso di impatto verticale e orizzontale.

- **Gestione della morte e reset:**
  Il metodo Die cambia lo stato a Dying, attivando l'animazione di morte e modificando la velocità, mentre il metodo Reset ripristina la posizione e i parametri iniziali per riprendere il gioco.

Questi elementi costituiscono la logica centrale per il movimento, le animazioni e le interazioni del personaggio nel gioco.

La classe astratta **Animation** gestisce le animazioni di un'entità. I punti principali sono:

- **Proprietà e campi:**
  - **Texture e frames:** Tiene una texture e un array di rettangoli che definiscono i singoli frame dell'animazione.
  - **Tempi e frame correnti:** Utilizza un timer per determinare quando passare al frame successivo, in base al valore di *frameTime*, e mantiene l'indice del frame corrente.
- **Metodo Update:**
  Aggiorna il timer in base al tempo trascorso e, una volta superato il tempo del frame, incrementa il frame corrente. Se si raggiunge l'ultimo frame, ricomincia dall'inizio.
- **Metodo Draw:**
  Disegna il frame corrente della texture sullo schermo, applicando la scala e gli effetti grafici (ad esempio il flip orizzontale) se necessario.

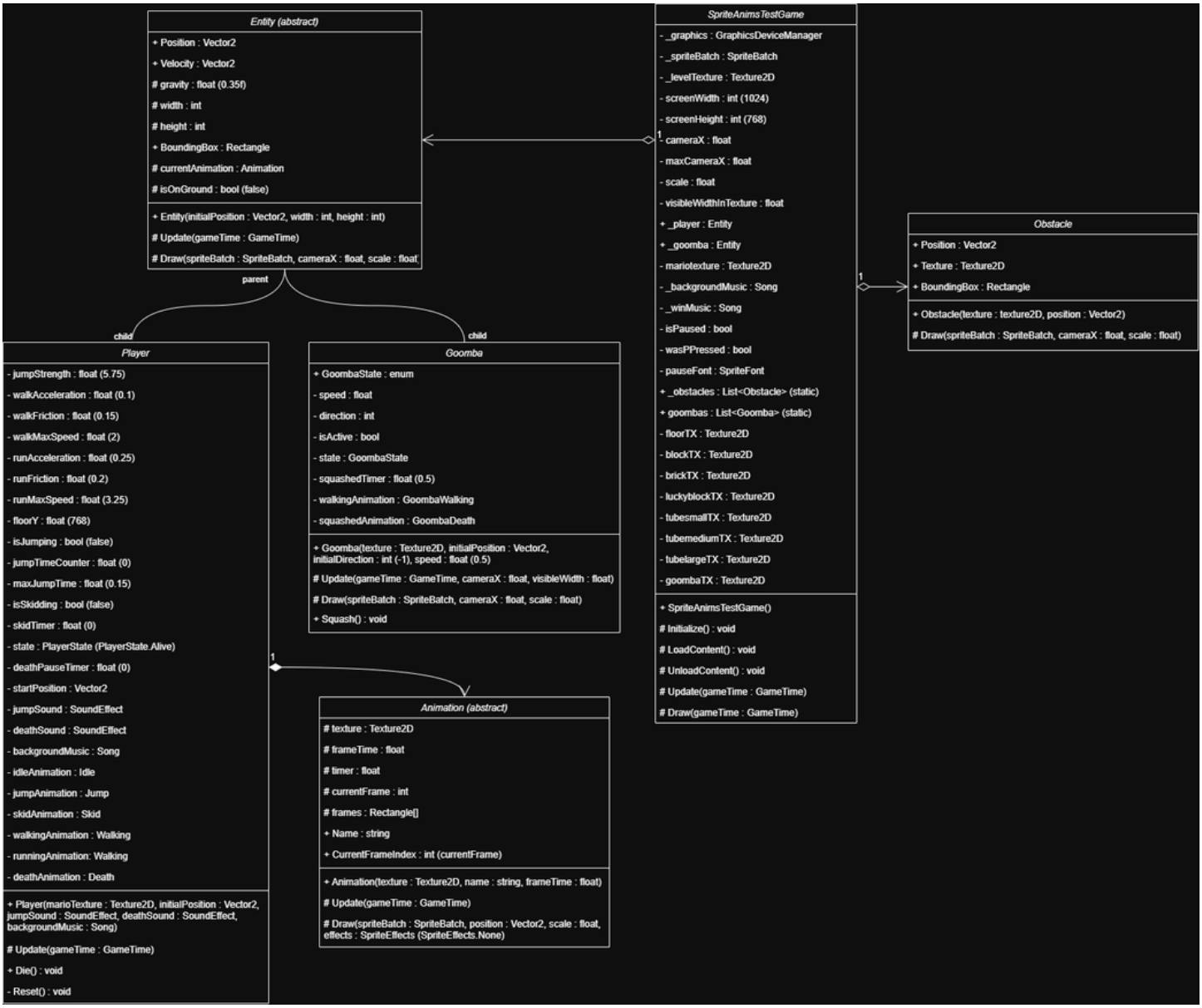
Questa struttura fornisce la base per animazioni cicliche e temporizzate nelle entità del gioco.

Dalla classe **Animation** derivano: Idle, Walking, Jump, Death, Skid, GoombaWalking e GoombaDeath

Diagramma delle classi

Progetto in Unified Modeling Language (UML)

File editabile: SMB.drawio



Focus: chi ha fatto cosa?

Inizializzazioni, Creazione Progetto	**Buizza**
Creazione Classi e prima Beta.	**Buizza**
Caricamento Texture e Contenuti	**Buizza**
Collisioni Player e Goomba con ostacoli	**Regonesi**
Metodo Vincita	**Regonesi**
Metodo Morte	**Buizza**
Animazioni Goomba e Visualizzazione Ostacoli	**Regonesi**
Fix Varie di Errori nel codice	**Regonesi**
Manuale Utente e Documentazione Tecnica	**Buizza**
Diagramma UML	**Regonesi**