Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica

A.a. 2024/2025

Sessione di laurea Dicembre 2025

Versionamento Adattivo nel Creative Coding: Strumento per l'Esplorazione Artistica basato su AST

Relatori:

Juan Pablo Sáenz Moreno

Luigi De Russis

Candidato:

Stefano Di Leo

Ringraziamenti

Giunto al termine di questo lungo e sofferto percorso universitario, vorrei ringraziare alcune persone senza le quali non sarei riuscito ad arrivare al traguardo.

Innanzitutto ringrazio i miei relatori Juan Pablo Sáenz Morenoe Luigi De Russis per avermi guidato durante questa ultima prova.

Ringrazio poi i miei genitori Antonella e Marcello per avermi sostenuto e permesso di studiare, oltre che per la loro pazienza e disponibilità. Ringrazio anche mia sorella Chiara, per avermi aiutato quando ne avevo bisogno e per essermi stata vicina, anche da lontano.

Infine grazie ai miei amici vecchi e nuovi, con i quali ho passato il tempo libero, lavorato a progetti universitari e non e che sono stati una preziosa lezione di vita.

Indice

Elenco delle tabelle

Elenco delle figure

Glossario

CC

Creative Coding

VCS

Version Control System

AST

Abstract Syntax Tree

DOM

Document Object Model

CRDT

Conflict-free Replicated Data Type

API

Application Programming Interface

UI

User Interface

UX

User Experience

IDE

Integrated Development Environment

Capitolo 1

Introduzione

Fin dalla loro comparsa, i computer sono stati utilizzati anche per la realizzazione di opere artistiche e creative nelle maniere più disparate: si pensi agli strumenti per il disegno digitale molto utilizzati dagli artisti, ma come il Creative Coding.

Di particolare interesse per questa tesi è la pratica del Creative Coding, nella quale gli artisti utilizzano codice in linguaggi di programmazione per realizzare le proprie opere che in questo contesto prendono il nome di *sketch*. In tale pratica, gli artisti procedono in maniera fortemente iterativa, scrivendo, modificando e talvolta scartando il proprio codice, osservandone la resa e raffinando il loro lavoro poco alla volta. Gli artisti non procedono necessariamente in maniera deterministica o con uno scopo preciso, bensì fanno numerosi tentativi finché non si imbattono in un'opera che li soddisfi. In un certo senso essi "esplorano" e "giocano" con il codice in maniera molto diversa da quanto avviene nella programmazione nella sua accezione più comunemente intesa. Per questo motivo il Creative Coding si può considerare come una branca dell'"Exploratory Programming".

Nel corso del tempo, attorno alla pratica si sono formate numerose comunità di appassionati e sono stati sviluppati numerosi strumenti – open-source e non – per agevolare il lavoro dei programmatori creativi, strumenti che hanno migliorato l'ergonomia del processo creativo, rendendo possibili miglioramenti come feedback essenzialmente in tempo reale e permettendo quindi iterazioni ancora più rapide e numerose.

Ultimamente, infatti, sempre più artisti si sono avvicinati al mondo della programmazione e del Creative Coding proprio per sfruttarne le nuove potenzialità che essi offrono per la creazione di opere d'arte, soprattutto se paragonati alle tecniche più tradizionali di espressione artistica. Inoltre per molti, il Creative Coding costituisce un tassello importante nel loro percorso di istruzione in quanto le sue proprietà sono adatte all'insegnamento della programmazione e permettono di dare libero sfogo alla creatività degli studenti sia a scuola che in campo universitario.

Tuttavia non esistono strumenti che agevolino la gestione delle versioni delle opere in maniera che si adatti naturalmente alla natura esplorativa del Creative Coding, lasciando gli artisti a dover usare strumenti pensati per lo sviluppo software classico e portando molti di loro ad ideare sistemi manuali di organizzazione dei propri lavori.

I sistemi per la gestione delle versioni attuali (Version Control System), infatti, risultano difficili da imparare e non forniscono meccanismi per far fronte alle numerose e frequenti modifiche che vengono apportate alle opere durante il processo di creazine ed esplorazione. Inoltre non forniscono feedback visivo sull'aspetto dell'opera rendendo inutilmente macchinoso e controintuitivo spostarsi in un'altra versione desiderata.

Per questo motivo si rende necessario uno strumento che possa essere utilizzato sia da neofiti che da esperti e realizzato su misura per risolvere le criticità delle soluzioni esistenti.

1.1 Obiettivo

Data la natura esplorativa e open-ended del Creative Coding e data la mancanza di uno strumento che supporti tale uso è stato realizzato uno strumento per la gestione di versioni (VCS), con l'obiettivo di correggere le mancanze degli attuali strumenti di versionamento (molto comuni nello sviluppo software) che però risultano inadeguati per il Creative Coding e quindi risultano sottoutilizzati dai programmatori creativi. In particolare lo strumento deve permettere all'utente di seguire l'evoluzione del

La prima fase è consistita interamente nell'analisi degli articoli accademici disponibili riguardo Creative Coding e nell'analisi degli strumenti disponibili, sia popolari e ben conosciuti nel campo, che di strumenti più sperimentali e meno conosciuti.

È seguita poi una fase di realizzazione di sketch p5 con versionamento manuale tramite Git in modo da sperimentare in prima persona le differenze tra sviluppo software "convenzionale" e Creative Coding, oltre ai punti di forza e criticità degli attuali strumenti di versionamento quando applicati a quest'ultimo.

Individuate le principali criticità, si sono discusse quali funzionalità basate sul parsing del codice e sull'analisi del relativo Abstract Syntax Tree potessero maggiormente migliorare il processo creativo degli utenti e si è passati all'implementazione.

1.2 Struttura della tesi

La tesi è strutturata su sei capitoli, partendo dal Capitolo 1 che contiene una breve introduzione al concetto di Creative Coding e la descrizione degli obiettivi.

Il Capitolo 2 *Background e Lavori Correlati* descrive lo stato attuale della strumentazione per il versionamento di codice per il Creative Coding, le limitazioni delle soluzioni esistenti, le principali barriere riscontrate dai Programmatori Creativi e l'analisi di alcuni studi e progetti realizzati nel campo.

Nel Capitolo 3 *Progettazione* vengono esposti i requisiti identificati e la scelta della piattaforma impiegata.

Nel Capitolo 4 *Implementazione* vengono descritte le tecnologie e gli algoritmi utilizzati, la struttura e la relazione tra i vari elementi funzionali dell'applicazione e le funzionalità fornite all'utente oltre a descriverne i dettagli implementativi rilevanti, con considerazioni di UI/UX.

Il Capitolo 5 *Valutazione* descrive il test di usabilità condotto con utenti non necessariamente familiari con il Creative Coding e ne discute i risultati.

Nel Capitolo 6 *Conclusioni* vengono commentati i risultati ottenuti, valutandone la compatibilità con l'obiettivo della tesi. Vengono successivamente esplorate le direzioni che si potrebbero intraprendere a partire da questo lavoro per sviluppi futuri.

Capitolo 2

Background e Lavori Correlati

2.1 Creative Coding

Il Creative Coding è una forma di programmazione in cui l'obiettivo principale è l'espressione artistica piuttosto che l'efficienza funzionale. Consiste nello scrivere codice per creare opere artistiche. Esso nasce dall'intersezione tra arte e tecnologia e di tale pratica si possono trovare esempi fin dagli albori dell'informatica (anni '60).

Il dominio del Creative Coding è molto vasto e se ne possono trovare manifestazioni in innumerevoli forme: da videogiochi ed esperienze interattive a installazioni artistiche visive, fino ad arrivare alla produzione di audio e musica. Ogni opera può far uso di diverse fonti di informazioni, generando così opere che possono essere statiche e sempre uguali ad ogni esecuzione, fino ad arrivare a lavori procedurali, che producono un risultato potenzialmente diverso ad ogni esecuzione e che possono reagire agli input dello spettatore [1].

Visti i numerosi settori in cui gli artisti si trovano a lavorare, spesso devono interagire con programmatori per integrare i loro lavori in un contesto più ampio (ad esempio nel campo dei videogiochi). Viste le differenze tra i due campi, la comunicazione tra sviluppatore e artista può risultare difficile, sarebbe quindi utile far avvicinare i creativi al mondo della programmazione, permettendo loro di capire i processi degli sviluppatori agevolando la comunicazione e potenzialmente permettendo loro anche di partecipare attivamente al prodotto completo. Per questo motivo, sono stati istituiti anche corsi che forniscano sia conoscenze estetiche che informatiche per agevolare l'avvicinamento di questi due mondi e creati numerosi strumenti dedicati al Creative Coding.

Negli ultimi decenni, complice anche la crescente accessibilità dei calcolatori, il CC è gradualmente acquisito popolarità e ha visto un rapido sviluppo di librerie e strumenti dedicati.

Tra i più importanti e comuni possiamo trovare Processing, creata da Casey Reas e Ben Fry nel 2001 [2]: si tratta di una libreria per il linguaggio di programmazione Java e di un Integrated Development Environment realizzato per la realizzazione di *sketch* con lo scopo di insegnare i concetti fondamentali di programmazione in un contesto visivo a coloro che non si occupano di programmazione. Esso ha un'interfaccia molto semplice: si presenta come una finestra con un ampio campo di testo in cui scrivere il codice e due pulsanti che permettono di eseguirlo o di fermarne l'esecuzione. Durante l'esecuzione dello *sketch*, viene creata una nuova finestra che ne mostri il risultato.

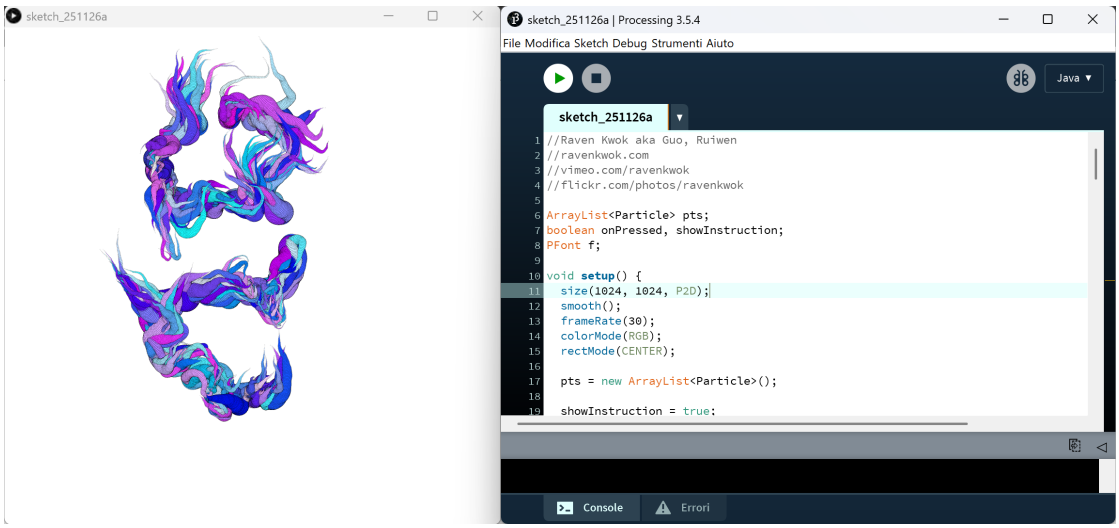


Figura 2.1: L'IDE di Processing che esegue uno *sketch* ottenuto da OpenProcessing.

Vista la popolarità di Processing, nel corso del tempo sono state realizzate numerose altre librerie con lo scopo di adattare i suoi principi ad altri linguaggi di programmazione. Di particolare importanza è p5: si tratta di una libreria scritta in JavaScript che permette la realizzazione di *sketch* che vengono eseguiti direttamente nel browser, e pertanto non è necessaria l'installazione di strumenti dedicati.

Non è raro che, in letteratura, il Creative Coding venga usato come sinonimo di Generative Art, seppure i due termini identifichino concetti differenti: la Generative Art, infatti, indica l'utilizzo di algoritmi o processi automatici dedicati alla generazione di opere a partire da un input ridotto fornito dall'utente, come avviene per la generazione di immagini a partire da testo. È importante anche notare che l'Arte Generativa non è necessariamente legata al mondo informatico. Il Creative

Coding, invece, si focalizza sul programmatore come responsabile in prima persona dell'opera della la quale sceglie ogni particolare.

Il Creative Coding si può classificare come un ramo della Programmazione Esplorativa (Exploratory Programming), con la quale condivide una mancanza di un preciso obiettivo e che invece incentiva libera sperimentazione "open-ended" e un ciclo di sviluppo rapido con feedback immediato. Per questo la Programmazione Esplorativa e il Creative Coding vengono spesso utilizzati nel campo dell'insegnamento della programmazione a neofiti [3].

La rapidità dello sviluppo, l'alta frequenza delle iterazioni e la natura "open-ended" del Creative Coding, rendono desiderabile la possibilità di tenere traccia delle modifiche apportate nel tempo in modo da poter visualizzare l'evoluzione delle proprie opere e permettendo una sperimentazione più efficace nel processo artistico. Una simile necessità è riscontrata anche dai programmatori non creativi, per i quali sono stati quindi creati strumenti appositi per la gestione delle versioni del software. Tali strumenti prendono il nome di Version Control System, una classe di software che per l'appunto consente all'utente di gestire l'evoluzione nel tempo di file. Sono principalmente utilizzati per codice sorgente (e quindi file di testo), ma in generale sono applicabili a qualunque tipo di file.

Durante lo sviluppo di software, infatti, i programmatori apportano modifiche in maniera continua e spesso di dimensione ridotta se paragonate alla dimensione complessiva del progetto, ma che possono cambiarne significativamente il funzionamento. Prima del rilascio di una nuova versione, è normale che vengano applicate un gran numero di tali modifiche, da parte di individui differenti e in tempi diversi. Tenere traccia di ogni modifica manualmente diventa estremamente complicato, se non impossibile, senza un sistema dedicato che permetta di isolare ogni modifica e metterla in relazione con altre modifiche in ordine temporale.

La soluzione più semplice, sarebbe una gestione manuale delle versioni salvando più copie del codice sorgente ed etichettando ogni versione in maniera appropriata. Tale approccio, tuttavia, richiede molto lavoro da parte dei programmatori oltre ad essere estremamente inefficiente sia in termini di spazio di archiviazione utilizzato che in termini di difficoltà nel rintracciare i cambiamenti applicati e particolarmente suscettibile ad errori umani [4].

Nel tempo sono stati quindi sviluppate diverse soluzioni software per agevolare il processo di versionamento del codice e collaborazione tra programmatori. Alcuni dei primi esempi di tale software furono SCCS (Source Code Control System) sviluppato da Marc J. Rochkind di Bell Labs nel 1972 [5] e RCS (Revision Control System) sviluppato da Walter F. Tichy della Purdue University nel 1982 [4].

Successivamente sono state realizzate altre soluzioni tra le quali CVS (Concurrent Version System) e Subversion, fino ad arrivare agli strumenti moderni come Mercurial e Git. Questi ultimi si differenziano per la loro natura decentralizzata, permettendo ai programmatori di lavorare ognuno sulla propria versione della

"repository" in autonomia e per poi richiedere l'inclusione delle proprie modifiche in un secondo momento nella "repository" principale che si differenzia dalle altre solamente per convenzione ma senza alcuna differenza dal punto di vista tecnico. Ciò si contrappone direttamente ai VCS di tipo client/server che invece pongono restrizioni al lavoro che gli sviluppatori possono svolgere offline in autonomia.

Tali strumenti, tuttavia, sono sottoutilizzati dai programmatori creativi, tanto che il fenomeno è stato analizzato nel passato in alcuni studi pubblicati su come i programmatori creativi interagiscono con i VCS, tra cui "The Art of Creating Code-Based Artworks" [6], nel quale sono stati intervistati cinque Creative Coders, con esperienza in tecnologie diverse, provenienti da quattro paesi diversi e con preparazioni che spaziano dalla produzione musicale e Graphic Design fino ad arrivare a Fisica Matematica e Scienze Ottiche Applicate.

Per adattarsi alla natura esplorativa del CC, i partecipanti allo studio generalmente tendono a creare numerosi file per sperimentare con le loro idee. In base al gradimento del risultato, questi possono fungere come campione da integrare in opere future. Per gestire la grande quantità di file risultanti dalla loro sperimentazione, i partecipanti necessitano di sviluppare strategie e meccanismi per organizzare e categorizzare il proprio lavoro.

Come evidenziato nella sezione precedente, tale strategia di gestione delle versioni è poco efficiente e porta facilmente ad errori e il fatto che gli artisti la impieghino comunque nel loro lavoro è un indizio importante sull'inadeguatezza delle soluzioni di VCS per il Creative Coding. Sebbene queste ultime si siano dimostrate estremamente utili e produttive per la programmazione "tradizionale", un partecipante nota come spesso nel CC le modifiche apportate siano troppo piccole per giustificare la creazione di un "commit" (spesso viene aggiornato il valore di una singola variabile) mentre un secondo esprime comunque il desiderio di salvare le modifiche in maniera automatica e periodica.

Un altro fattore significativo evidenziato dalle interviste riguarda la difficoltà di apprendimento dei moderni strumenti di VCS. Tali difficoltà sono esacerbate dal fatto che molti Programmatori Creativi non sono programmatori professionisti e non sono interessati a diventarlo o stanno ancora imparando a programmare.

Alcuni partecipanti notano, inoltre, come sia fondamentale per loro avere a disposizione una rappresentazione visiva degli sketch, in modo da poterli identificare più agevolmente. A tal scopo, uno di loro descrive come utilizzi la funzionalità di cattura dello schermo fornita dalla libreria da lui utilizzata per salvare anteprime degli sketch nella stessa cartella, un altro, invece, utilizza un social network per archiviare il risultato del proprio lavoro.

In sintesi, dalle interviste emerge la necessità di strumenti per il versionamento di codice pensati nello specifico per il CC, capaci di gestire modifiche molto frequenti e di dimensione ridotta, oltre alla possibilità di salvare informazioni aggiuntive come un'anteprima dello sketch.

2.2 Creative Coding e Version Control System

Nel tempo sono stati sviluppati diversi strumenti per la gestione delle versioni degli *sketch*, alcuni più sperimentali e altri che invece hanno acquisito popolarità e sono diventati di uso comune per i programmatori creativi. Tra quelli più sperimentali si annoverano SHARP e Quickpose, mentre tra gli altri è di spicco OpenProcessing.

OpenProcessing è una comunità online e piattaforma focalizzata sulla creazione e la condivisione di progetti di Creative Coding e supporta sketch scritti per le librerie p5 e Processing.

Essa è gratuita e permette di scrivere codice per uno sketch e visualizzarne il risultato direttamente nel browser, rendendo la pratica del CC estremamente accessibile.

La funzionalità di OpenProcessing più rilevante per questa tesi è la possibilità di creare un *fork* di uno *sketch*. Il processo è manuale e permette all'utente, tramite un'interfaccia intuitiva, di "congelare" il codice di uno *sketch* nel tempo, dando poi la possibilità di riutilizzarlo come punto di partenza per versioni future. È possibile salvare un numero arbitrario di versioni e, funzionalità molto importante per l'aspetto di comunità della piattaforma, si possono visualizzare le versioni salvate da altri utenti che chiunque può utilizzare come punto di partenza per il proprio lavoro. Le versioni vengono visualizzate tramite una rappresentazione ad "albero" che mostra l'evoluzione dell'opera nel tempo sia per mano dell'autore originale che per mano degli altri utenti (Figura 2.2).

Tale struttura incentiva la condivisione degli *sketch* e porta alla formazione di una comunità creativa e vivace.

Nel 2023, Sabbaraman et al. [7], hanno pubblicato uno studio che esamina la comunità di OpenProcessing, e nello specifico le abitudini degli utenti per quanto riguarda l'utilizzo della funzione di *fork*, analizzando tutte le opere pubbliche presenti sulla piattaforma (oltre un milione ai tempi dello studio). Lo studio ha evidenziato quattro strategie di utilizzo della funzionalità *fork*: come modo di salvare gli *sketch* fatti da altri senza apportare ulteriori modifiche, come modo di annotare il codice con commenti per tenere traccia del progresso personale, come modo per estendere uno *sketch* tramite aggiunte di codice sostanziali e per finire come modo per perfezionare lo *sketch* tramite piccole variazioni dei valori di variabili esistenti (ma che possono portare a risultati estremamente diversi). Il lavoro di questa tesi si concentra prevalentemente sulle ultime due casistiche.

OpenProcessing, tuttavia, non è in grado di tenere traccia di cambiamenti piccoli tra una versione e l'altra, necessitando dell'input dell'utente. Tale modo di procedere è utile, come appena discusso, per gli scopi della condivisione e dell'evoluzione degli *sketch* da parte di più individui indipendenti ma non risolve le criticità dei VCS analizzate sopra, non fornendo agli utenti informazioni dettagliate sull'evoluzione dei loro lavori.

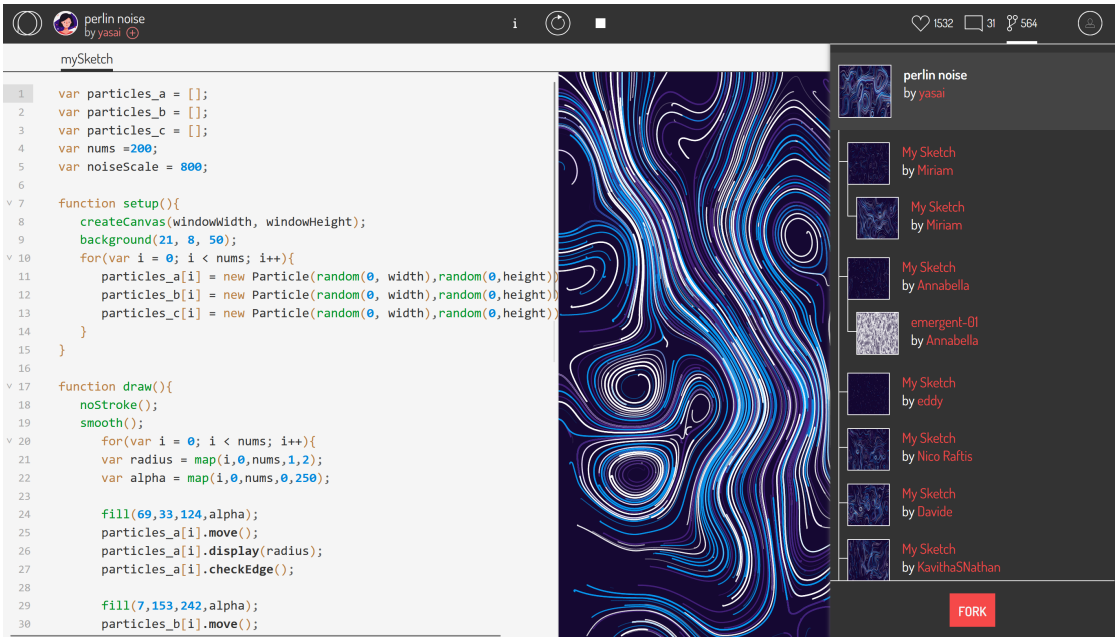


Figura 2.2: Esempio di un popolare sketch su OpenProcessing, con annessa visualizzazione dei fork.

Quickpose [8] è un VCS realizzato da Eric Rawn et al. nel 2023 pensato per Processing. Esso permette agli utenti di salvare versioni del proprio codice per poi visualizzarle in un grafo orientato dove i nodi rappresentano le versioni dello *sketch* con anteprima. L'utente può riorganizzare il grafo e aggiungere annotazioni a proprio piacimento e può iniziare dai *fork* a partire da un qualunque nodo. Il processo di salvataggio di una nuova versione è manuale e deve essere iniziato dall'utente.

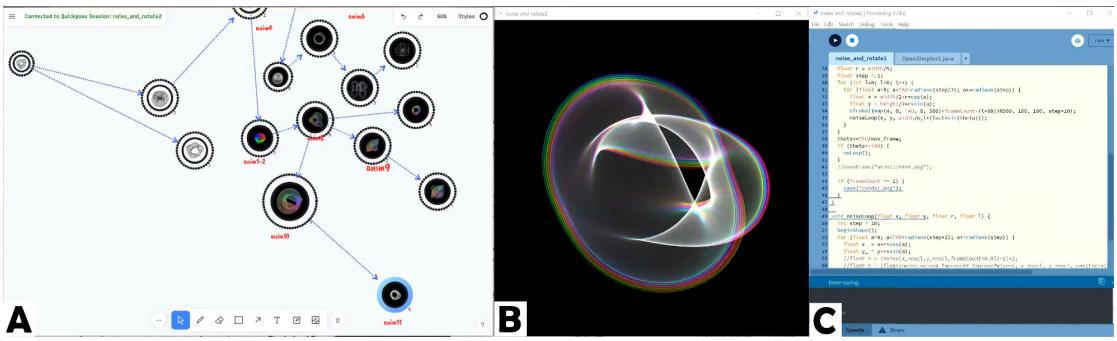


Figura 2.3: Interfaccia grafica per la visualizzazione delle versioni di Quickpose. Immagine tratta da [8].

Infine SHARP [9] è stato realizzato per il "live coding" musicale, una forma di Creative Coding nella quale si produce musica dal vivo anche di fronte ad un pubblico. Realizzato da Manesh et al. nel 2024, tale strumento mira a fornire agli artisti un VCS su misura per la loro variante di CC.

Nel "live coding" musicale, il codice è strutturato in *pattern*, ovvero sezioni di codice paragonabili ad uno strumento in una banda musicale o una traccia in una Digital Audio Workstation (DAW). Durante una performance, i *pattern* possono venire eseguiti più volte, con modifiche eseguite dal vivo tra un'esecuzione e l'altra. Per questo SHARP si focalizza nel tracciare i cambiamenti di ogni *pattern* ogni volta che vengono eseguiti in maniera completamente automatica e che non richieda comandi espliciti da parte dell'utente.

SHARP visualizza le versioni in molteplici *timeline* direttamente in linea, ognuna nella riga immediatamente antecedente al *pattern* corrispondente. Interagendo con le *timeline*, l'utente può riportare il singolo *pattern* ad una versione precedente in modo da eseguirlo nuovamente o come punto di partenza per nuovi *pattern*.

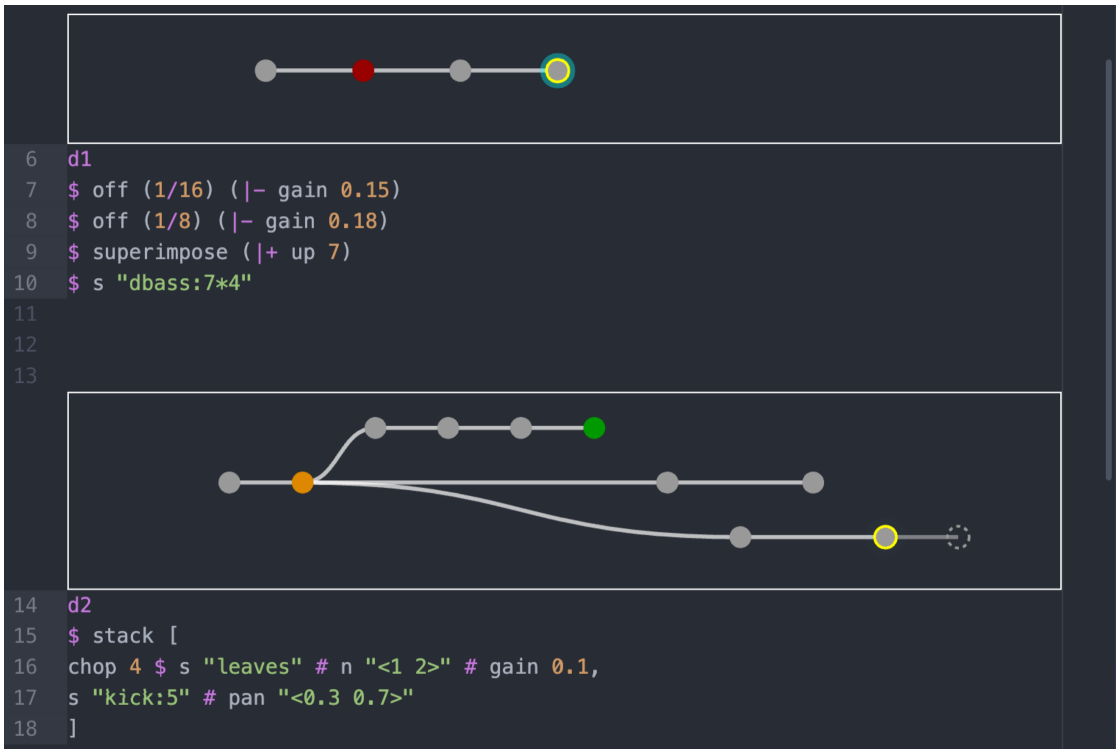


Figura 2.4: Interfaccia grafica per la visualizzazione di *timeline* di SHARP. Immagine tratta da [9].

2.3 Ideazione di un Version Control System per il Creative Coding basato su Abstract Syntax Tree

Viste le difficoltà riscontrate dai programmatori creativi descritte in questo capitolo, si possono pensare funzionalità che mirino a risolverle o mitigarle. Tra queste, figurano la proposta di un partecipante ad uno degli studi analizzati in precedenza di implementare un salvataggio periodico e automatico di nuove versioni, ma anche un'analisi precisa ed ad-hoc del codice sorgente dello *sketch* in modo da individuare con cognizione di causa le modifiche apportate e la loro natura.

Questo lavoro, seppure tocchi entrambi questi aspetti, tenta di innovare soprattutto nel secondo caso impiegando particolari strutture dati dette Abstract Syntax Tree. Si tratta di una particolare rappresentazione strutturata del codice sotto forma di albero che viene generato a partire da quella testuale in un qualunque linguaggio di programmazione. Ogni nodo in tali strutture fa riferimento ad un preciso elemento semantico (con annessa tipologia) del codice e ne descrive la relazione con gli altri elementi. In questo modo, viene quindi rappresentato il codice in maniera indipendente da formattazione del testo e spazi bianchi, elementi che modificano l'aspetto del codice senza influenzarne la funzionalità. Ciò permette di concentrarsi sulle modifiche funzionali ignorandone tutte le minuzie della rappresentazione testuale.

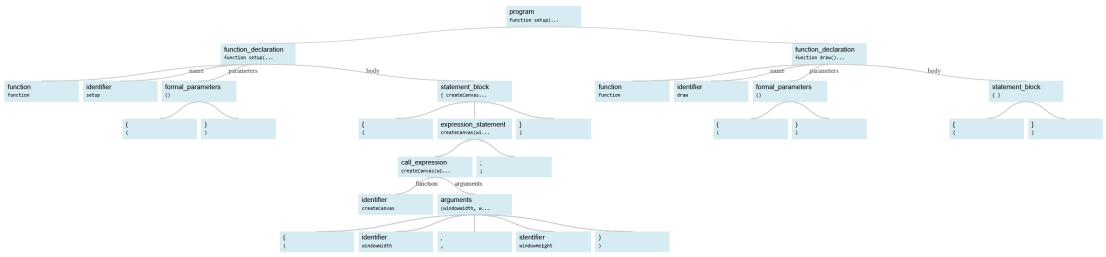


Figura 2.5: Rappresentazione grafica dell'AST di uno *sketch* vuoto.

```
1 function setup() {
2   createCanvas(windowWidth, windowHeight);
3 }
4 function draw() {
5 }
```

Listing 2.1: Codice relativo alla figura 2.5.

Gli AST sono molto utilizzati nella realizzazione di compilatori e vengono prodotti da programmi chiamati *parser*.

Tramite le loro proprietà sarebbe quindi possibile rilevare in maniera più precisa e capillare le modifiche che sono state apportate ad uno *sketch* in modo da metterle in relazione con le altre versioni così da realizzare un VCS che possa adattarsi alle esigenze dei programmatori creativi interpretandone con maggior precisione le intenzioni.

Capitolo 3

Progettazione

3.1 Requisiti

Viste le difficoltà descritte dai programmatori creativi nell'adottare VCS per i loro *sketch*, sono stati identificati i requisiti da soddisfare con lo sviluppo dell'applicazione. Il lavoro degli artisti è infatti molto iterativo, per cui vengono cambiate con alta frequenza piccole porzioni di codice. Pertanto creare dei "commit" manualmente per ogni cambiamento non è assolutamente pratico, sarebbe pertanto auspicabile un automatismo per il salvataggio di nuove versioni del codice. Per affrontare i problemi appena descritti sono stati pensati i primi due requisiti (3.1.1 e 3.1.2).

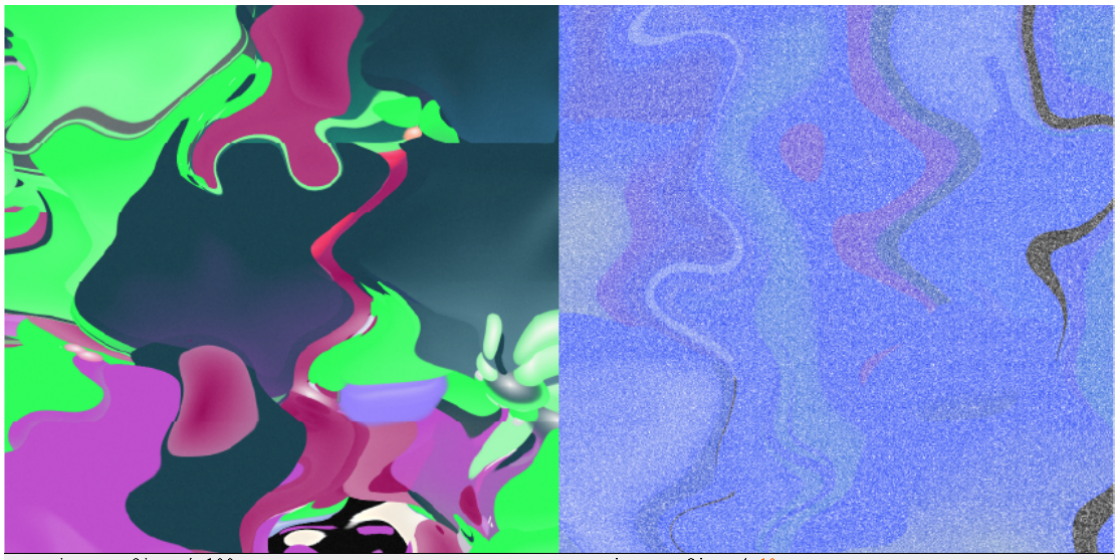
Inoltre, un ulteriore obiettivo è quello di permettere l'utilizzo dell'applicazione anche da parte di utenti inesperti, che rende quindi auspicabile dare priorità alla facilità d'utilizzo (3.1.4).

3.1.1 Versionamento capillare

Data la natura del Creative Coding, ogni piccolo cambiamento può avere un impatto significativo sul risultato finale e, dal momento che esso valorizza lo *sketch*, è importante essere in grado di seguire ogni modifica nel tempo indipendentemente dalla sua dimensione.

Per distinguere il tipo di modifica, sono stati definiti due termini simili che però implicano una sostanziale differenza *version* e *variation*: con *version*, infatti, si intende una modifica strutturale al codice sorgente dello *sketch* che ne cambia le funzionalità e la rappresentazione durante il parsing. Con *variation*, invece, ci si riferisce a modifiche dei valori di variabili già esistenti. Come detto in precedenza, tali modifiche possono portare ad un risultato molto diverso seppure apportino al codice cambiamenti non strutturali e di dimensione ridotta.

L'applicazione deve essere quindi in grado di salvare nuove *version* quando è necessario, e deve poter rilevare e tenere traccia delle *variation*.



```
margin = mySize / 100;
for (let i=0; i<int(random(50, 100)); i++) { ... }
theShader.setUniform(
  'u_time',
  millis() / 1000
);
let version = random([1,2,4,6,8])*100;
let c = random(2000, 5000);
colorMode(HSB, 360, 100, 100, 100);
```

```
margin = mySize / 10;
for (let i=0; i<int(random(500, 100)); i++) { ... }
theShader.setUniform(
  'u_time',
  millis() / 1
  );
let version = random([200,150,77,50,140]*100;
let c = random(1000, 2000);
colorMode(HSB, 21, 10, 10, 10);
```

Figura 3.1: Esempio di *variation* di uno sketch che ottiene uno stile distinto unicamente modificando le variabili già esistenti. Immagine tratta da [7].

La tesi esplora l'utilizzo di parser per il linguaggio di programmazione usato per lo *sketch* con generazione di Abstract Syntax Tree per distinguere tra *version* e *variation*.

3.1.2 Salvataggio automatico

Dalle interviste analizzate nel Capitolo 2, è emerso che alcuni partecipanti abbiano ritenuto eccessivo creare un *commit* con un Version Control System per apportare delle modifiche che erano frequentemente molto piccole (pochi caratteri) e uno di loro aveva suggerito di salvare una nuova versione automaticamente e periodicamente ogni pochi secondi. Per questo motivo, in questa tesi si è scelto di rendere il processo di salvataggio automatico nel momento in cui venisse rilevata una modifica allo *sketch*.

Operando in questo modo, tuttavia, ci si espone alla possibilità di salvare *sketch* contenenti errori nella cronologia. In questo modo, però, l'utente non potrebbe fare affidamento ad una qualunque versione salvata in quanto potrebbe contenere errori che ne impediscono l'esecuzione o la visualizzazione del risultato, rendendo

potenzialmente impossibile capire se vi siano informazioni utili per il processo creativo. Inoltre, uno sketch contenente errori rende impossibile il parsing e l'analisi dello stesso, negando la ragion d'essere del progetto.

Si è resa quindi necessaria una metodologia per tentare di rilevare errori nello sketch prima di salvarli nel database e che possa operare con sufficiente affidabilità da rilevare la maggior parte degli sketch non validi.

Vista la mole di operazioni da compiere ad ogni salvataggio, è imperativo porre particolare attenzione alle prestazioni dell'applicazione in modo che l'esperienza dell'utente non venga intralciata da blocchi e rallentamenti improvvisi, soprattutto su hardware di fascia più bassa.

3.1.3 Navigazione tra versioni

Come già spiegato, la possibilità di rivisitare versioni passate in maniera facile e intuitiva è di fondamentale importanza per i Creative Coders. Per questo motivo si è pensato ad una *timeline* che rappresentasse ogni versione dello *sketch* con annessa anteprima mediante la quale l'utente possa interagire per spostarsi tra versioni.

L'intenzione iniziale era la realizzazione di una *timeline* ad "albero" che permettesse *forks*. Successivamente, tuttavia, si è preferito utilizzare una *timeline* lineare per via della natura "single user" dello strumento.

3.1.4 Facilità d'uso

Nel Capitolo 2 sono state evidenziate le difficoltà riscontrate dai Programmatori Creativi nell'utilizzo dei Version Control System. L'utilizzo dei VCS moderni come Git, richiede infatti la conoscenza di diversi concetti come *commit*, *branch*, *merge*, *merge conflict* e *repository* oltre che dimestichezza con la linea di comando.

Molti IDE moderni forniscono delle interfacce grafiche proprio per agevolare l'utilizzo di Git ed evitare all'utente di utilizzare la linea di comando. Tuttavia, tali GUI per essere funzionali devono necessariamente esporre l'utente ai concetti del VCS sottostante, e quindi possono comunque risultare troppo complicati.

Dal momento che il Creative Coding è una pratica spesso adottata da individui che stanno ancora imparando a programmare o che non sono interessati ad approfondire le proprie conoscenze generali di programmazione [6], è importante che la soluzione adottata per questo progetto sia il più semplice e intuitiva possibile.

3.1.5 Gestione efficiente di un gran numero di versioni

Visti i requisiti descritti nelle sezioni 3.1.2 e 3.1.1, è facile immaginare che, durante un uso normale dello strumento, verranno salvate un gran numero di versioni. È

quindi desiderabile minimizzare il più possibile lo spazio su disco occupato da ogni versione e garantire una navigazione tra versioni il più rapida ed efficiente possibile.

3.2 Prototipo su carta

La prima fase nella progettazione dell'interfaccia utente è consistita nella realizzazione di *paper prototype*, ovvero di schizzi che rappresentano l'applicazione eseguiti con carta e penna in modo da farsi un'idea di come disporre gli elementi nell'applicazione ponendo importanza sulla natura dei dati mostrati e sulla loro posizione relativa, senza doversi preoccupare degli aspetti estetici.

La progettazione ha visto la creazione di diverse interfacce. Inizialmente, l'idea era di permettere all'utente di creare *branch*. Di conseguenza i design iniziali riflettono tale intenzione.

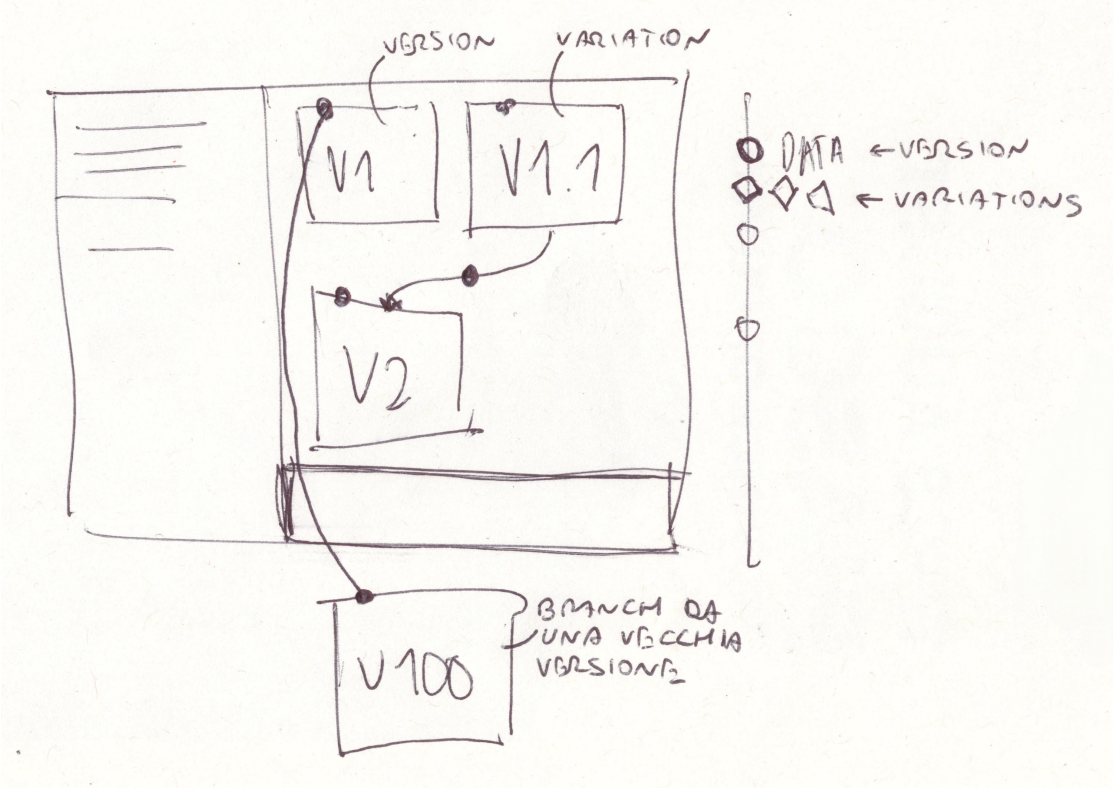


Figura 3.2: Prima idea di timeline.

La figura 3.2, mostra un design della *timeline* che occupa gran parte dello schermo. È strutturata come un insieme di nodi, ognuno dei quali rappresenta una versione. In questa fase, l'intenzione era ancora distinguere le *version* dalle

variation. Ogni riga rappresenta una *version*, mentre ogni colonna rappresenta una *variation*. Sulla destra è visibile una rappresentazione alternativa e più compatta della *timeline*, anch'essa strutturata in maniera tale da distinguere tra *version* e *variation*.

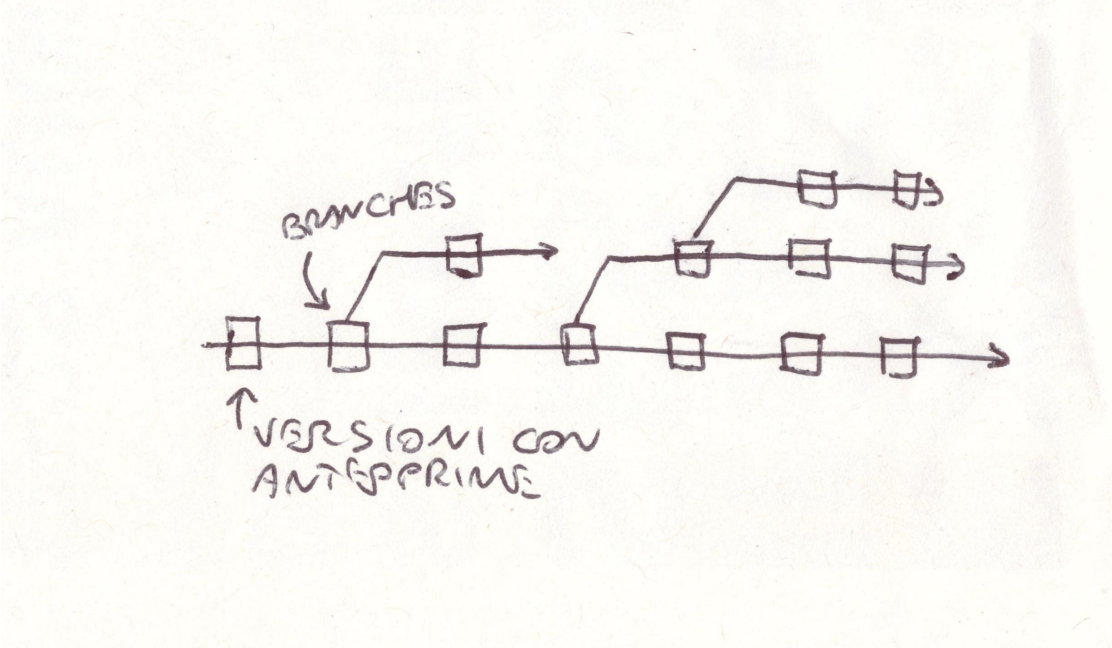


Figura 3.3: Seconda idea di timeline.

La figura 3.3, invece, mostra una visualizzazione della timeline ad albero. In questa versione è stata abbandonata la differenziazione tra *version* e *variation*. Questa versione si ispira alla rappresentazione di diverse branch di Git.

Infine, nella figura 3.4, viene mostrato uno schizzo dell'interfaccia grafica completa. In questo prototipo, fortemente influenzato dall'editor online di p5.js [10] e OpenProcessing [11], è stata abbandonato il concetto di *branch*, a favore di una *timeline* lineare e quindi più semplice da implementare ma anche per l'utente da comprendere e utilizzare. In questa versione i valori delle variabili tracciate dall'applicazione sono mostrate in un *panel* che si sovrappone all'anteprima in alto a destra nella finestra.

Quest'ultima versione ha subito ulteriori variazioni durante lo sviluppo del progetto che non sono state rappresentato in uno schizzo su carta: la *timeline* è stata spostata nella parte alta della finestra e la visualizzazione delle variabili è stata implementata come dei pannelli in-line che possono essere mostrati e nascosti a piacimento.

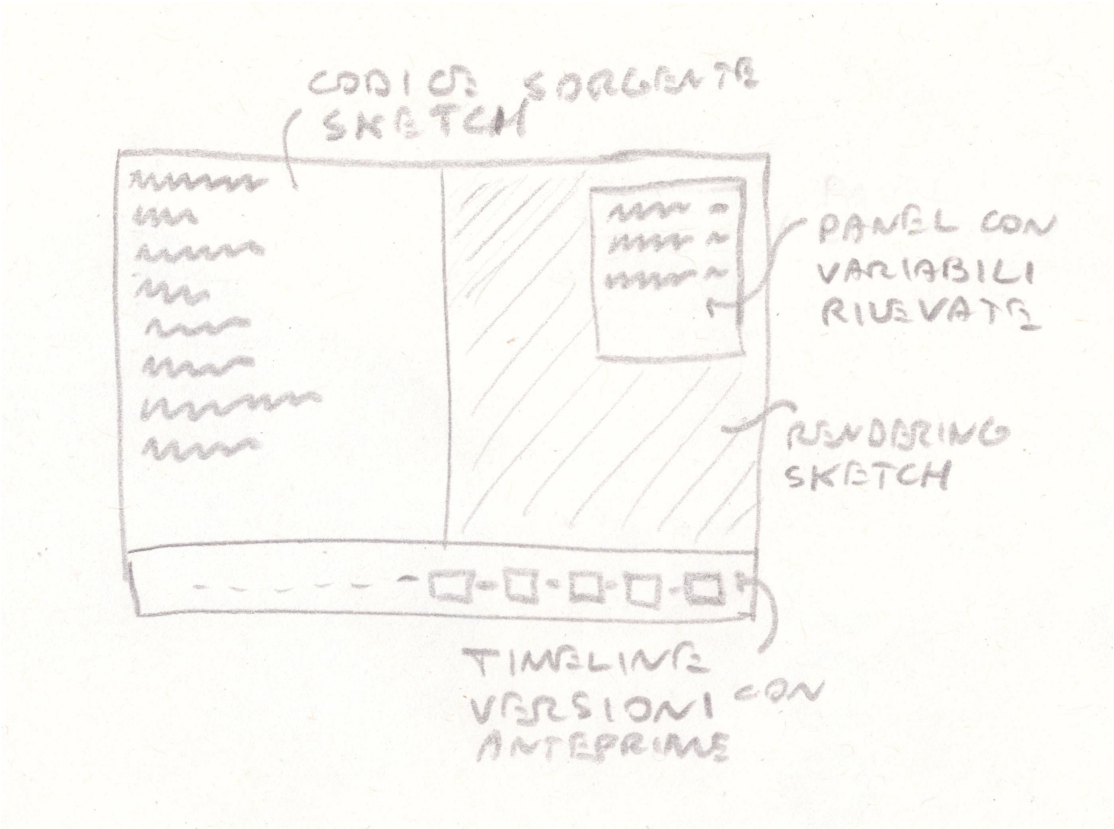


Figura 3.4: Interfaccia grafica completa.

3.3 Web Application o plug-in per un editor esistente

Una scelta importante per il progetto riguarda la piattaforma per cui sviluppare l'applicazione. Tra le possibili alternative, ne sono state isolate due in particolare: la creazione di una nuova Web Application o la realizzazione di un plug-in per un editor di codice già esistente.

Tra i numerosi editor di codice disponibili, è stato isolato Visual Studio Code (VSCode), per via della sua popolarità e relativa facilità d'uso.

Un plug-in per VSCode presenta il vantaggio di essere molto diffuso tra programmatori sia esperti che neofiti, offre un'enorme libreria di plug-in già disponibili e mette in condizione l'utente di utilizzare uno strumento possibilmente familiare e già configurato per le proprie necessità.

Al contrario, una nuova Applicazione Web non richiede installazione di software aggiuntivo (almeno nel caso in cui l'utente si avvicini per la prima volta al mondo

della programmazione), permette maggiore flessibilità di personalizzazione dell'aspetto visivo per adattarsi al meglio alle esigenze dell'utente ed è accessibile anche da piattaforme più limitate, come tablet Android e Apple.

Dal momento che gli utenti potenziali dell'applicazione potrebbero essere neofiti o non a proprio agio con gli strumenti più avanzati, li si vuole dotare di uno strumento più intuitivo in modo che esso non rappresenti per gli utilizzatori una sfida tecnica. Date queste premesse, con questo lavoro, si è deciso quindi di proseguire con la creazione di una nuova Web Application.

Capitolo 4

Implementazione

4.1 Tecnologie utilizzate

In questa sezione verranno analizzate le principali tecnologie utilizzate per lo sviluppo dell'applicazione con le annesse motivazioni che hanno portato alla loro scelta. Sono state usate p5 come libreria di base per la realizzazione degli *sketch*, Tree-sitter per il parsing e la generazione degli Abstract Syntax Tree, Yjs per la gestione nelle versioni, Monaco per l'implementazione dell'editor di codice e React per la realizzazione dell'interfaccia grafica.

4.1.1 p5.js

Vista la necessità di supportare la creazione di *sketch* all'interno del browser, è necessario scegliere una libreria per il Creative Coding che sia realizzata nativamente per il Web o che possa essere in qualche modo eseguita in tale contesto oltre ad essere sufficientemente popolare da permetterne l'utilizzo ai programmatori creativi con esperienza e facilitare la ricerca di informazioni nel caso in cui si necessitino aiuti o chiarimenti. Tra le alternative disponibili spiccano p5 e Processing. Quest'ultima, sebbene sia in grado di essere eseguita nel browser come dimostrato da OpenProcessing, necessita di accorgimenti aggiuntivi e richiede la conoscenza del linguaggio di programmazione Java (più complesso da imparare per un neofita). Per questo motivo si è scelto di basare il progetto su p5.

p5 [12] è una libreria JavaScript molto conosciuta e utilizzata nell'ambito del Creative Coding creata da Lauren Lee McCarthy nel 2013 come alternativa alla popolare libreria Processing [2] per il web. Essendo il progetto realizzato per questa tesi un'applicazione web e vista la popolarità di p5, quest'ultima è stata scelta come base degli *sketch* supportati.

p5 semplifica molto la creazione di uno *sketch* dal momento che accede direttamente al DOM del browser e crea autonomamente gli elementi necessari, come un

canvas HTML. Come verrà descritto in seguito, tuttavia, tale funzionalità, seppure di grande aiuto per gli utenti di p5, non è appropriata per questo progetto, in quanto è necessario che p5 si occupi solamente di "disegnare" sul *canvas*, lasciandone la creazione e la gestione all'applicazione.

Per questo motivo, è stato compiuto il *vendoring* (ovvero l'inclusione del codice sorgente all'interno del progetto) dell'ultima versione stabile di p5 disponibile (v2.0.5 al momento della creazione del progetto) in modo da modificarla per adattarla alle esigente di questa tesi.

4.1.2 Tree-sitter

Per generare l'AST a partire dal codice JavaScript dell'utente, è necessario un parser. Ne esistono diverse alternative, tra le quali sono state analizzate Tree-sitter e Babel. La seconda è una libreria realizzata appositamente per il web per il parsing dei linguaggi JavaScript e TypeScript; vista la rappresentazione dei dati e le funzionalità fornite per l'esplorazione dell'AST meno ergonomiche al fine di implementare l'algoritmo GumTree descritto in seguito, si è preferito utilizzare Tree-sitter per le sue prestazioni superiori e l'API matura e completa oltre alla enorme disponibilità di *parser* per gran parte dei linguaggi di programmazione (utile nel caso si decidesse di adattare questo progetto per l'uso con altre tecnologie).

Tree-sitter [13] è uno strumento per la generazione di *parser* e una libreria per il *parsing* incrementale. È stata creata da Max Brunsfeld nel 2018 e si è velocemente diffusa per la sua solidità e prestazioni. Grazie a questo strumento, sono stati realizzati *parser* per numerosi linguaggi di programmazione, tra cui JavaScript.

I *parser* Tree-sitter, normalmente compilati in codice macchina, sono anche compilabili per il web come binari WebAssembly e disponibili su *npm*.

In particolare, per questa tesi, viene utilizzato il *parser JavaScript*.

4.1.3 Yjs

Vista la necessità di tenere traccia delle modifiche apportate al codice da parte dell'utente, necessitando di una rappresentazione su disco efficiente e la capacità di gestire una gran numero di modifiche, è stata scelta in maniera insolita la libreria Yjs, la quale supporta le funzionalità necessarie anche se pensata principalmente per un altro scopo.

Essa [14] è un Conflict-free Replicated Data Type ad alte prestazioni scritto in JavaScript. I CRDT sono particolari strutture dati che possono essere replicate tra molteplici computer, dove ogni replica può essere aggiornata indipendentemente dalle altre. Per quanto ogni copia diverga dalle altre, è sempre possibile riconciliarle in maniera automatica e senza conflitti [15]. Sono molto utilizzati per

realizzare applicativi che permettono a più utenti di modificare lo stesso documento contemporaneamente, tra cui Google Docs e Notion.

Esistono diversi tipi di CRDT, ma i Commutative Replicated Data Types (CmRDT) come Yjs funzionano salvando ogni modifica apportata al documento a partire dalla sua creazione (nel caso di documenti di testo, ogni carattere inserito o rimosso). Tali informazioni possono essere utilizzate per implementare un sistema di versioning, possibilità offerta da Yjs tramite la sua funzionalità di *snapshot*.

Tra le implementazioni di CRDT disponibili, Yjs è stato scelto per via della sua affidabilità, prestazioni elevate, ridotto uso di spazio di archiviazione e disponibilità di integrazioni per Monaco e IndexedDb.

4.1.4 Monaco

Per quanto riguarda l'editor di testo, ne serviva uno che fosse ricco di funzionalità e in grado di supportare l'utilizzo sia da parte di utenti neofiti che avanzati. Per questo è stato scelta la libreria Monaco [16].

È questa la libreria sviluppata da Microsoft alla base di Visual Studio Code, estratta da quest'ultimo per permetterne l'utilizzo sul web. Essa fornisce supporto per la colorazione della sintassi per JavaScript, completamento del codice e numerose API che permettono di estenderne le funzionalità oltre a personalizzarne aspetto e comportamento.

4.1.5 React

Per la realizzazione dell'interfaccia grafica, al fine di impiegare una tecnologia matura e ben conosciuta basata su un paradigma ormai molto diffuso, è stata scelta la libreria React [17], scritta in JavaScript e sviluppata da Meta, con l'obiettivo di semplificare la creazione di interfacce grafiche basandosi su "componenti", ovvero parti di codice modulari, riutilizzabili e indipendenti.

La libreria adotta uno stile "dichiarativo", ovvero che permette al programmatore di specificare la struttura dell'interfaccia grafica senza definire i dettagli di come questa verrà effettivamente realizzata: la libreria si occuperà in maniera autonoma di aggiornare il DOM in modo da ottenere il risultato desiderato. Per migliorare le prestazioni, React impiega il Virtual DOM, che permette di aggiornare solamente le parti dell'interfaccia che sono cambiate senza dover ricostruire l'intera pagina.

4.2 AST diff

I Version Control System per lo sviluppo software più utilizzati come Git, tengono traccia dell'evoluzione dei file basandosi sulle differenze tra le righe di codice nei file di testo con algoritmi come quello di Myers [18, 19].

Libreria	**Funzionalità**
@js-sdsl/hash-map	HashMap. Usata per impl. GumTree
@js-sdsl/ordered-map	Mappa (Red Black Tree). Usata in impl. GumTree
fast-deep-equal	Deep equality. Usata per impl. GumTree
@monaco-editor/react	Adattatore per includere Monaco in un'app React
async-mutex	Implementazione di Mutex FIFO
comlink	Wrapper `async/await` per *WebWorker*
idb	Wrapper `async/await` per IndexedDb
lib0	Utility functions. Usato per Observables
monaco-editor	Vedi sezione 4.1.4
monaco-themes	Temi per Monaco editor
overlayscrollbars	Scrollbar che si sovrappongono a contenuto
overlayscrollbars-react	Adattatore React per overlayscrollbars
p5	Vedi sezione 4.1.1
react	Vedi sezione 4.1.5
yjs	Vedi sezione 4.1.3
y-monaco	Adattatore per collegare Yjs a Monaco

Tabella 4.1: Principali librerie utilizzate.

Lo svantaggio di tale approccio è che il sistema non è in grado di capire se vi siano state modifiche strutturali del codice o meno. Ciò si può risolvere eseguendo il parsing del codice sorgente ottenendo un AST, che ne descrive la struttura senza essere influenzato da spazi bianchi o formattazione del documento in generale e successivamente confrontando gli AST del codice prima e dopo la modifica. Tale operazione consiste nella generazione di un *edit script*, ovvero una lista di operazioni che descrive le differenze tra i due alberi in termini di aggiunta, rimozione e spostamento di nodi.

Tuttavia, la generazione di un *edit script* ottimale di dimensioni minime che supporti operazioni di *add*, *delete*, *update* è un problema NP-hard [20] e il miglior algoritmo noto al momento, RTED, ha complessità $O(n^3)$, dove n è il numero di nodi dell'AST [21].

Da un punto di vista pratico, però, per gli utenti di un VCS la garanzia di ottenere un *edit script* ottimale è superflua: in questo contesto un algoritmo dalle prestazioni elevate basato su euristiche è da preferirsi ad un algoritmo ottimale ma molto più lento.

Un algoritmo che rispetta le esigenze appena descritte è GumTree, realizzato da Jean-Rémy Falleri nel 2014 [22]. Seppure abbia complessità $O(n^2)$ nel caso peggiore, risulta essere sufficientemente performante per l'uso in un VCS come quello sviluppato in questa sede.

La principale implementazione di questo algoritmo è disponibile su GitHub ed è stata realizzata dallo stesso Falleri in Java [23]. Ad oggi, non sembrano esistere implementazioni dell'algoritmo che possano essere utilizzate in un contesto web, quindi nella prima fase del progetto è stato eseguito il *porting* della suddetta implementazione Java in TypeScript.

Dettagli di implementazione

Il *port* re-implementa la struttura dati che rappresenta l'AST, chiamata `LightTree` che implementa l'interfaccia `ILightTree`:

```
1  export interface ILightTree {
2      type: string;
3      text: string;
4      startPosition: Point;
5      endPosition: Point;
6      startIndex: number;
7      endIndex: number;
8      children: Array<ILightTree>;
9      childCount: number;
10     parent: ILightTree | null;
11     nextSibling: ILightTree | null;
12     descendantCount: number;
13     metrics?: Metrics;
```

```
14
15      walk(): LightTreeCursor;
16      insertChild(t: ILightTree, pos: number): void;
17      removeChild(pos: number): ILightTree | undefined;
18      isIsomorphicTo(tree: ILightTree): boolean;
19      isIsoStructuralTo(tree: ILightTree): boolean;
20      getLabel(): string;
21      hasSameTypeAndLabel(t: ILightTree): boolean;
22  }
```

`ILightTree` si ispira alla rappresentazione di Tree-sitter, aggiungendo campi e metodi utili per questo progetto (`metrics`, `insertChild`, `isIsomorphicTo`, etc.) e rimuovendone di superflui.

L'algoritmo si può suddividere in quattro fasi:

- Generazione degli AST: tramite Tree-sitter, vengono generati i due AST corrispondenti al codice sorgente prima e dopo una modifica per poi convertirli in `LightTree`;
- Greedy top-down matching: in questa fase avviene la ricerca dei sotto-alberi *isomorfici* di dimensione massima;
- Bottom-up matching: questa fase parte dai risultati della fase precedente e determina se due nodi corrispondano verificando che vi sia una buona corrispondenza tra i rispettivi discendenti e successivamente utilizzando un algoritmo ottimale per la generazione delle mappature finali tra AST sorgente e destinazione;
- Generazione dell'*edit script*: le mappature della fase precedente vengono poi fornite in input all'algoritmo di Chawathe [24], che restituirà l'*edit script* finale.

L'*edit script* è un array di `Action`, che possono rappresentare inserzioni, rimozioni, aggiornamenti o spostamenti dei nodi.

Valutazione qualitativa delle prestazioni

Per decidere come procedere con la strategia di salvataggio automatico, sono stati eseguiti alcuni semplici benchmark per verificare il tempo di esecuzione su alcuni *sketch* di esempio. Lo scopo del benchmark non è ottenere risultati precisi e riproducibili, quanto farsi un'idea dell'ordine di grandezza del tempo impiegato per l'esecuzione del port dell'algoritmo GumTree nel caso di modifiche molto piccole (pochi caratteri) allo sketch, come potrebbero esserle in pratica nell'utilizzo effettivo dell'applicativo.

Sketch	Durata media (ms)	Errore (%)	Runs	Bytes (src/dst)
Waves	18.1	±1.45	71	1929/1928
Sea Weeds	57.8	±5.43	32	5803/5802
Frozen Brush	26.0	±3.24	52	3340/3340

Tabella 4.2: Risultati del benchmark.

I risultati del benchmark rivelano che, almeno su hardware di fascia bassa e per *sketch* di medie dimensioni, l'esecuzione dell'algoritmo è sì molto rapida (decine di millisecondi), ma comunque non a sufficienza da poter essere eseguito sul thread principale insieme all'interfaccia grafica, soprattutto vista la frequenza con cui viene eseguito l'algoritmo. Si potrebbero verificare infatti, specialmente nel caso di sketch più grandi, *stutter* e "impuntamenti", che porterebbero ad un peggioramento significativo dell'esperienza utente.

Note sulla licenza del progetto

L'implementazione dell'algoritmo GumTree di J. R. Falleri, è rilasciato al pubblico sotto licenza GNU Lesser General Public License Version 3 (LGPL-3.0) [25]. Dal momento che il progetto contiene il *porting* parziale di tale codice, è stata adottata la stessa licenza.

4.3 Modifiche apportate a p5

Come accennato nella sezione 4.1.1, è stato eseguito il *vendoring* della libreria p5 e sono state apportate alcune modifiche per permetterne il funzionamento "headless" in un WebWorker. In questa sezione verranno descritte tali modifiche alla libreria e come essa viene utilizzata all'interno del progetto.

Semplificando molto, quando viene importata in una pagina web, p5 inizializza i propri moduli, che implementano le funzionalità della libreria tra cui input e output, eventi (ad esempio mouse e tastiera), funzionalità matematiche, webgl e altri e, di default, rende il tutto disponibile nello scope globale di esecuzione JavaScript. Successivamente, tenta di individuare il codice dello *sketch* dell'utente, lo analizza utilizzando un parser chiamato "acorn" in modo da rilevare dichiarazioni di funzioni o variabili che vanno in conflitto con le quelle fornite da p5 stessa, verifica la presenza delle funzioni `setup` e `draw` nello *sketch*, esegue la funzione `setup` e successivamente da inizio ad un ciclo infinito nel quale ad ogni iterazione viene chiamata la funzione `draw` e che si ripete ad ogni frame grazie alla funzione `requestAnimationFrame` fornita dai browser.

Nella funzione `setup` dello *sketch*, l'utente deve chiamare `createCanvas`, che si occupa della creazione di un `Renderer`, ovvero di una classe che fornisce le funzionalità per il disegno sul *canvas* e che può basarsi semplicemente sul *canvas* stesso o su *WebGL*. Il `Renderer` si occupa a sua volta, tra le altre cose, della creazione di un *canvas HTML* che viene automaticamente aggiunto nella pagina accedendo direttamente al DOM.

Rimpiazzo di `window` e `document`

Dal momento che, dall'interno di un WebWorker, p5 non può accedere alle proprietà fornite normalmente dal browser, queste vengono passate dall'esterno durante la creazione del WebWorker in un oggetto che rispetta l'interfaccia `P5Dom`:

```
1  export interface P5Dom {
2      screen: {
3          width: number,
4          height: number
5      };
6      innerWidth: number;
7      innerHeight: number;
8      devicePixelRatio: number;
9      orientation: number;
10 }
```

Separatamente, come parametri del costruttore del WebWorker, vengono anche passati il codice sorgente dello sketch, i *callback* necessari per la comunicare con il thread principale e il *canvas* sotto forma di un `OffscreenCanvas` [26], "collegato" ad un *canvas HTML* creato nel thread principale. Tali dati e il contenuto di P5Dom, verranno copiati all'interno della proprietà `self`.

Per poter accedere a tali informazioni, p5 è stato modificato in modo da non utilizzare le proprietà `window` e `document`, rimpiazzandole per l'appunto con `self`. Invece di creare un nuovo *canvas*, infine, p5 è stato modificato in modo da utilizzare quello fornito dal thread principale durante l'inizializzazione.

Implementazione funzionalità di heartbeat

Per verificare che gli *sketch* non si blocchino ad esempio a causa di un ciclo infinito, portando quindi al salvataggio di uno *sketch* non funzionante, è stata implementata una funzionalità di *heartbeat*, con la quale il WebWorker notifica periodicamente il thread principale permettendogli di capire che l'esecuzione procede come previsto. L'implementazione è stata aggiunta all'interno del metodo `_draw` di p5 nel file `main.js`, che viene eseguita per ogni *frame* e che chiama a sua volta la funzione `draw` dell'utente.

Ad ogni esecuzione viene misurato il tempo impiegato dal *frame* precedente in millisecondi e viene sommato ad un contatore. Quando il contatore raggiunge o supera una soglia arbitraria (in questo caso di $3s$), viene emesso un segnale e il contatore viene resettato.

Un sistema simile è utilizzato da diversi sistemi come Microsoft Windows, Android e gli stessi browser per determinare se gli applicativi in esecuzione si siano bloccati.

Questa semplice strategia, ha il limite che nel caso in cui la produzione di un frame dovesse impiegare troppo tempo, il sistema terminerebbe lo *sketch*, anche se questo non dovesse essere effettivamente bloccato.

Esposizione dei warning di p5 al thread principale

Nel file `core/friendly_errors/sketch_verifier.js`, p5 utilizza il parser "acorn" precedentemente citato per eseguire il parsing dello *sketch* e per verificare che esso non contenga definizioni di funzioni o variabili che vadano in conflitto con i simboli dichiarati da p5 stesso. Nella versione originale, p5 può segnalare all'utente la presenza dei conflitti solamente uno alla volta e tramite un messaggio in console che risulta fuori luogo se visualizzato in linea.

È stato quindi modificato il template del messaggio di errore ed è stata modificata la logica che produce gli errori in modo da poterne generare più di uno alla volta per poi mandarli al thread principale all'interno di un oggetto che contiene informazioni aggiuntive, come la riga e la colonna in cui è stato trovato il problema, tramite uno dei callback forniti durante la creazione del WebWorker.

```
1 const message = `${errorType} "${name}" on line ${line} is being
    redeclared and conflicts with a p5.js ${errorType.toLowerCase()
    }. p5.js reference: ${url}`;
```

Listing 4.1: Template originale per la generazione dei messaggi di errore di p5

```
1 const message = `${errorType} "${name}" is being redeclared and
    conflicts with a p5.js ${errorType.toLowerCase()}. p5.js
    reference: ${url}`;
```

Listing 4.2: Template modificato per la generazione dei messaggi di errore in linea

Vantaggi e svantaggi di questo approccio e soluzione alternativa

Gli editor esistenti basati su p5 come quello "originale" della libreria [10] e OpenProcessing [11], usano un approccio diverso: impiegano un `iframe` nel quale vengono caricati lo *sketch* e p5 stesso, che poi vengono eseguiti in maniera isolata dal resto dell'applicazione. Tale approccio ha diversi vantaggi, tra cui la possibilità di utilizzare la libreria p5 senza modificarla, che garantisce anche una maggiore facilità nell'adottarne gli aggiornamenti. Potenzialmente sarebbe anche possibile eseguire molteplici sketch allo stesso tempo (necessario per questo progetto), creando diversi `iframe` nascosti e mostrando solo il più recente.

Questo modo di procedere, tuttavia, non garantisce l'esecuzione in parallelo degli *sketch*, dal momento che il browser è libero di decidere se eseguire il codice dell'`iframe` in un thread separato o nello stesso thread del resto dell'applicazione.

Oltre a causare potenziali rallentamenti per via del tempo di esecuzione fisiologico degli *sketch*, nel caso di *sketch* particolarmente pesanti o di scrittura accidentale di cicli infiniti da parte dell'utente, l'intera applicazione verrebbe bloccata peggiorando l'esperienza utente, potenzialmente causando la perdita di dati nel caso in cui fossero in corso delle scritture sul database.

Note sulla licenza di p5 in relazione a quella del progetto

Come accennato in precedenza, questo progetto è pubblicato con licenza LGLP-3.0, p5 invece con licenza LGPL-2.0 [27], che è incompatibile con quella del progetto. Per rispettarne i termini, quindi, p5 non è stata inclusa nel *bundle* con il resto del codice ma viene caricata dinamicamente quando necessario, permettendo all'utente di rimpiazzarla con una copia personalizzata a proprio piacimento (qualora lo desiderasse).

4.4 Struttura dei thread

Per migliorare l'esperienza utente ed evitare "impuntamenti", il codice relativo all'interfaccia grafica e la "business logic" vengono eseguite su thread diversi che, nel mondo web, prendono il nome di *Web Workers*. Oltre ad eseguire codice su un thread separato, essi forniscono anche una "sandbox" che impedisce loro di accedere direttamente alle variabili negli altri *Workers* e al DOM nel thread principale. Ogni comunicazione tra thread deve essere effettuata tramite messaggi inviati usando il metodo `postMessage` e ricevuti registrando il callback `onmessage` [28]. Tali limitazioni, specialmente utili per garantire la sicurezza dell'applicazione (considerazione molto importante ogni qual volta si esegue l'input dell'utente), hanno però lo svantaggio di renderne l'utilizzo più macchinoso, motivo per cui è stata utilizzata la libreria *Comlink* sviluppata da Google [29], libreria che espone le funzioni definite all'interno dei *Web Worker* ad altri thread come funzioni con la stessa *signature* ma asincrone.

Oltre a quello principale, nell'applicazione vengono usate due ulteriori tipologie di *Worker*: `DrawerWorker` e `ParserWorker`.

4.4.1 DrawerWorker

Lifetime e ordine di esecuzione

Il `DrawerWorker` si occupa di eseguire p5 e lo *sketch* dell'utente. Disegna direttamente su un `OffscreenCanvas` che gli viene passato dal thread principale al quale può essere associato un *canvas HTML* in modo da mostrare all'utente il risultato del proprio lavoro in tempo reale. Il numero di istanze in esecuzione contemporaneamente non è fisso e può variare nel tempo. Ogni qual volta il thread principale decida di

salvare una nuovo *snapshot*, infatti, viene creato un nuovo `DrawerWorker` che esegua lo *sketch*. Il thread principale verifica quindi che il *Worker* emetta due *heartbeat* (la cui generazione è stata descritta in 4.3) e che non produca errori. Una volta superati tali controlli, il thread principale inizia la procedura di salvataggio di un nuovo *snapshot* e quando questo viene completato con successo, il *Worker* viene terminato, ma solo nel caso in cui durante il salvataggio ne sia stato eseguito uno nuovo che possa rimpiazzarlo, altrimenti viene lasciato in esecuzione. Come accennato in precedenza, dal momento che l'utente potrebbe apportare delle modifiche al codice dello *sketch* durante le verifiche appena descritte o durante la scrittura su disco, è possibile che vengano eseguiti più `DrawerWorker` contemporaneamente, ognuno dei quali relativo ad una versione diversa dello *sketch*. Anche se è possibile che molteplici *sketch* vengano eseguiti contemporaneamente, solo uno alla volta di essi, ovvero quello creato per ultimo, avrà un *canvas HTML* associato che ne mostri l'esecuzione all'utente, gli altri lavoreranno interamente "headless".

Vista la possibilità che vengano eseguiti molteplici `DrawerWorker` contemporaneamente, ognuno associato a *sketch* con tempi di avvio ed esecuzione potenzialmente significativamente diversi, è importante garantire che le operazioni di salvataggio che vengono lanciate successivamente al superamento dei controlli appena descritti si svolgano nell'ordine corretto. Per questo motivo, le "sezioni critiche" sono protette da `Mutex` [30] la cui implementazione garantisce in ordine *FIFO* lo smaltimento delle operazioni in attesa.

Gestione degli input

p5 permette la creazione di *sketch* interattivi, fornendo API per l'accesso a dispositivi di input tra cui mouse e tastiera basate sulle API analoghe fornite dal browser. Tali API, tuttavia, non sono accessibili dall'interno di un *WebWorker*: si rende quindi necessario catturare gli eventi rilevanti nel thread principale ed inoltrarli al *Worker*, operazione non possibile direttamente in quanto gli oggetti che rappresentano gli eventi non sono copiabili tramite una `structuredClone`. Per questo motivo, in `utils/events.ts` sono stati definiti i campi necessari per il corretto funzionamento di p5 oltre a funzioni d'appoggio per poter convertire gli eventi generati dal browser in oggetti strutturalmente simili, ma trasferibili al *WebWorker*.

Sono stati realizzati surrogati degli eventi `MouseEvent`, `KeyboardEvent`, `WheelEvent`, `UIEvent` e `FocusEvent`. Per l'evento `DragEvent`, invece, nonostante sia supportato da p5, non è stata realizzata un'alternativa in quanto può contenere dati arbitrari al suo interno di cui non è triviale garantire la trasferibilità.

4.4.2 ParserWorker

Il `ParserWorker` ha una durata di esecuzione che coincide con quella dell'applicazione. Per tutta la vita del programma ne esiste una sola istanza. Esso si comporta come servizio che fornisce all'applicazione le funzionalità di lettura e il salvataggio di dati da e nel database, oltre ad occuparsi del parsing e dell'analisi (parzialmente descritta nella sezione 4.2) degli *sketch* fornitigli dal thread principale.

`Keypoint` e tracciamento delle variabili tra *snapshot*

Durante il processo di salvataggio di uno *snapshot*, viene impiegato l'algoritmo GumTree descritto nella sezione 4.2, il cui risultato consiste nell'*edit script*, ovvero un array di `Action` che descrive le modifiche apportate allo *sketch*. Per poter determinare se tali modifiche siano state apportate al valore iniziale di una variabile, è necessario individuare i nodi di tipo `variable_declarator`. I nodi dell'AST associati ad ogni `Action`, però, possono essere di qualunque tipo, non necessariamente riferiti a dichiarazioni di variabili: per questa ragione, è necessario verificare che il cambiamento rilevato coinvolga effettivamente i sotto-alberi rilevanti. Ciò avviene esplorando l'AST "verso il basso" a partire dal nodo associato all'`Action` per verificare se vi siano discendenti del tipo desiderato; nel caso in cui non ne vengano trovati, procedendo "verso l'alto" per verificare se la modifica sia avvenuta all'interno di una dichiarazione di variabile.

Una volta disponibile la lista di nodi `variable_declarator` associati ad ogni `Action`, è possibile, per ognuna di esse, calcolare le azioni di creazione o modifica di relativi `Keypoint`:

```
1  export interface Keypoint {
2      /**
3       * The id of the series of updates referring to this node
4       */
5      seriesId: number;
6
7      /**
8       * The value of the current node or null if this is a
       temporary node to link to an older snapshot
9       */
10     value: string | null;
11
12     /**
13      * The position of the text snipped related to this Update
14      */
15     pos: {
16         startIndex: number,
17         endIndex: number,
18         identifierEndIndex: number,
19     };
```

```
20
21      /**
22       * The id of the Snapshot in which this change was made
23       */
24      snapshotId: number;
25
26      /**
27       * The id of the Update object referring to the same node in a
        previous Snapshot
28       */
29      predecessorId: number | null;
30  }
```

Ad ogni variabile viene associato un "identificatore di serie", salvato in `seriesId`, e ogni `Keypoint` è associato allo *snapshot* in cui è stata rilevata la modifica. La posizione di ogni variabile in termini di indici nella stringa che rappresenta lo *sketch*, è salvata nel campo `pos` e il valore dell'inizializzazione è salvato nel campo `value` come stringa. Per semplificare l'accesso ai `Keypoint` in modo che sia possibile accedervi con una singola query al database, il campo `value` può avere come valore `null`, per segnalare che si tratta di un `Keypoint` "temporaneo". Esso viene aggiornato ad ogni creazione di uno *snapshot* e permette di tenere traccia della posizione delle variabili nello *sketch* (visto che modificando il codice dello *sketch*, gli indici dei nodi successivi possono cambiare) e di determinare se una serie di `Keypoint` sia terminata o meno. Se tra i `Keypoint` dell'ultimo *snapshot* non ne sono presenti di associati ad una certa serie, significa che essa è terminata, ovvero che la dichiarazione di variabile corrispondente è stata rimossa. Per ogni serie, può esistere al più un `Keypoint` "temporaneo", ognuno dei quali sarà sempre associato allo *snapshot* più recente.

Transazionalità dei salvataggi

Per garantire l'integrità dei dati nel database, tutte le operazioni di elaborazione ed analisi dello *sketch* descritte fino ad ora, vengono eseguite all'interno di un'unica transazione. Nel caso in cui qualunque parte dell'analisi dovesse fallire inaspettatamente, la transazione viene annullata prevenendo la scrittura di dati parziali o incorretti su disco.

Tra i dati da salvare, vi è anche lo stato del documento di Yjs (libreria descritta nella sezione 4.1.3). Come già accennato, tra le caratteristiche che hanno portato alla sua scelta vi era la disponibilità della libreria *y-indexeddb*, realizzata dagli stessi autori, che permette la sincronizzazione automatica dello stato di un documento Yjs con un database IndexedDb. Tuttavia, *y-indexeddb* accede al database ed

esegue le scritture tramite transazioni in maniera indipendente e non personalizzabile dall'utente, caratteristica che viola i requisiti transazionali dell'algoritmo di salvataggio descritti poc'anzi.

Per questa ragione il codice di *y-indexeddb* è stato incluso nel progetto e significativamente modificato in modo da poter utilizzare la stessa connessione al database e la stessa transazione usate dal resto della funzione di salvataggio, operazione agevolata dalla brevità del codice originale.

4.5 UI/UX

In questa sezione verranno descritte le scelte di User Interface e User Experience ovvero gli elementi grafici e le funzionalità che compongono l'applicazione e come esse sono esposte all'utente.

4.5.1 Timeline

La *timeline* è posizionata nella parte superiore dell'applicazione. È composta da una lista orizzontale di elementi, ognuno dei quali rappresenta uno *snapshot*. Cliccando su un'anteprima, l'utente può visualizzare il codice di quella versione dello *sketch*. Passando il mouse sulle anteprime, vengono visualizzati i relativi numeri di versione.

Ai lati della *timeline*, sono stati aggiunti due bottoni a forma di freccia. Posizionando il mouse su di essi, la *timeline* avanza nella direzione corrispondente a velocità costante. Cliccandoci sopra, si viene portati all'inizio o alla fine, in base alla direzione indicata.

Nell'arco temporale necessario per il salvataggio di una nuova versione, viene impedito all'utente di spostarsi in altri *snapshot* per evitare la perdita di dati nel caso di fallimento e viene mostrata un'animazione di caricamento all'interno di un nuova anteprima vuota, per comunicare che sta avvenendo la creazione di un nuovo *snapshot*. Per segnalare l'impossibilità di spostarsi in altre versioni, invece, viene applicato un filtro *grayscale* a tutte le altre anteprime che vengono anche ruotate leggermente, in modo da far notare la diversa interfaccia anche nel caso di sketch in bianco e nero o con colori poco saturi.

Figura 4.4: La *timeline*, durante la creazione di un nuovo *snapshot*.

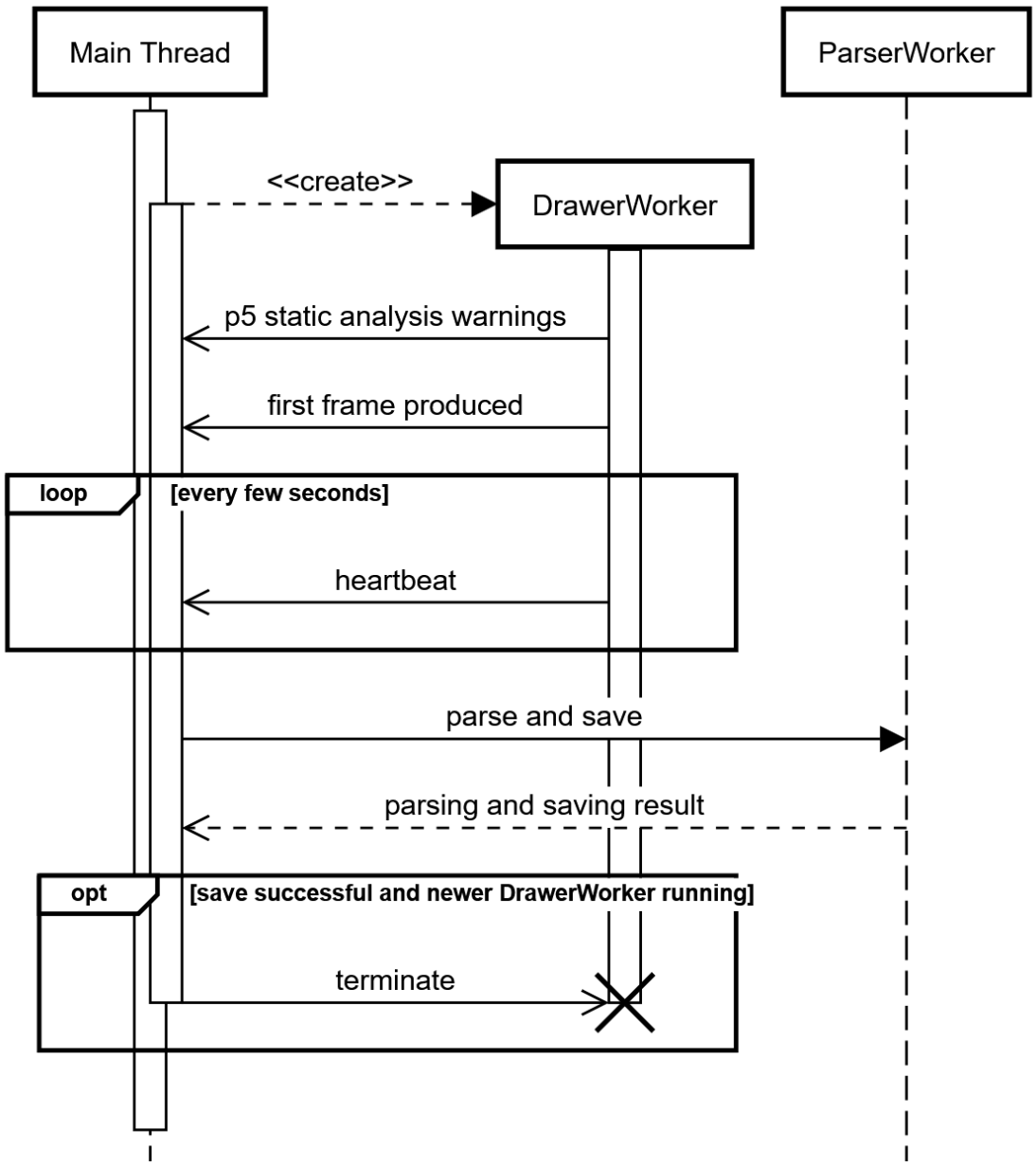


Figura 4.1: Una rappresentazione delle interazioni descritte nella sezione 4.4.

Se durante il salvataggio venire rilevato un errore, la possibilità di spostarsi in altre versioni rimane disabilitata fino a quando questi non vengano corretti in modo da evitare la perdita delle modifiche effettuate e viene mostrata una pseudo anteprima che segnali l'errore e che, una volta cliccata, mostra maggiori dettagli a riguardo.

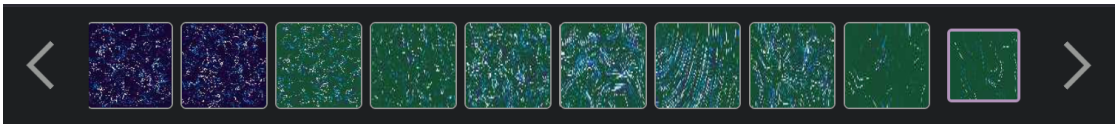


Figura 4.2: La *timeline*, che presenta molteplici *snapshot* e con l'ultima versione selezionata.

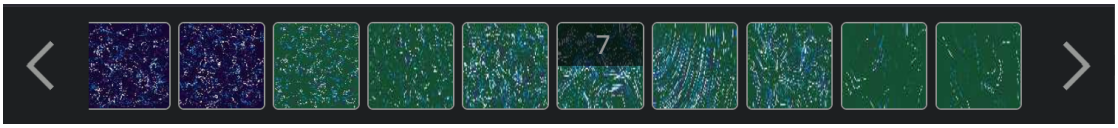


Figura 4.3: La *timeline*, che presenta molteplici *snapshot*. Su uno di essi è stato passato il cursore del mouse, rivelando il numero di versione.

4.5.2 Cronologia variabili

Quando si sta visualizzando lo *snapshot* più recente, nel codice sorgente appaiono delle icone che rappresentano un cronometro a destra di ogni dichiarazione di variabile tracciata. Cliccandole, viene mostrato un inserto in linea che mostra tutti i valori che quella variabile ha assunto nel tempo a partire dalla sua dichiarazione, compresi i numeri di versione in cui essa ha assunto il valore corrispondente (fig. 4.6).

```
1   var particles_a ⏱ = [];
2   var particles_b ⏱ = [];
3   var particles_c ⏱ = [];
4   var nums ⏱ = 200;
5   var noiseScale ⏱ = 800;
6
```

Figura 4.6: Dichiarazione di alcune variabili tracciate in uno *sketch*.

Interagendo con i valori mostrati, viene visualizzata un'anteprima dell'aspetto dello *sketch* con la variabile che assume il valore scelto. Se si è soddisfatti del risultato si può usare il bottone "Apply" per applicare la modifica e creare un nuovo *snapshot*, altrimenti, con il tasto "Cancel" viene nascosto l'inserto e ricaricato lo *sketch* salvato nel database (fig. 4.7).

Figura 4.5: La *timeline*, in caso di errore durante la creazione di un nuovo *snapshot*.

Figura 4.7: L'inserto contenente la cronologia della variabile `noiseScale`.

Se il valore della variabile dovesse risultare troppo lungo per essere mostrato nell'inserto, verrà troncato e poi espanso nel momento in cui l'utente vi passi sopra il cursore del mouse. Se il valore risultasse comunque troppo lungo, viene applicato uno scorrimento automatico in modo da mostrarne comunque completamente il valore.

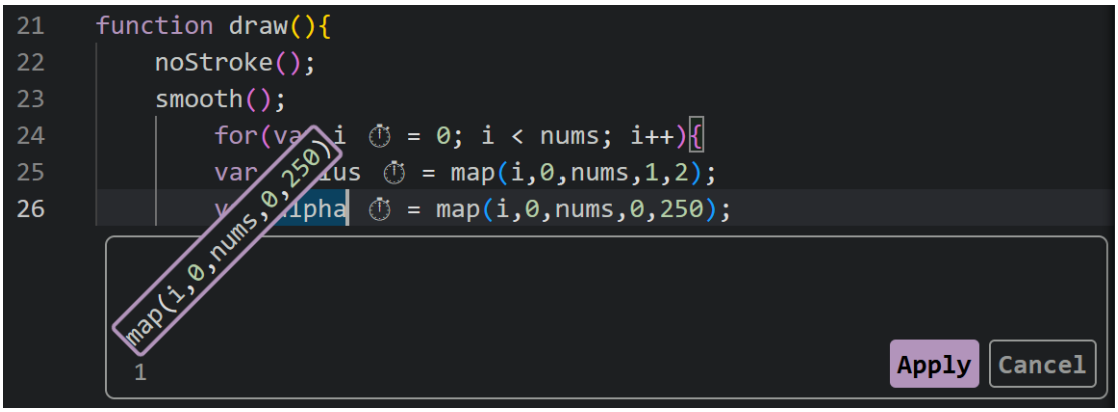


Figura 4.8: L'inserto contenente la cronologia della variabile `alpha`, espanso in modo da mostrarlo interamente.

```
1  // Welcome! You can use this tool to make sketches
2  // based on p5 library (v2.0.5) and explore it across
3  // different versions. You can find p5's documentation
4  // with many examples at https://p5js.org/
5
6  // Here's an example to get you started:
7
8  function setup() {
9    createCanvas(windowWidth, windowHeight);
10   background(100);
11 }
12
13 function draw() {
14   circle(mouseX, mouseY, 20);
15 }
```

Listing 4.3: *sketch* predefinito mostrato all'utente. Tramite Ctrl + click, è possibile visitare direttamente il link alla documentazione di p5.

4.5.3 Aiuti per l'utente

Sketch predefinito

In modo da guidare l'utente nella creazione di uno *sketch*, al primo avvio ne viene mostrato uno predefinito che contiene un commento con un link che porta direttamente alla documentazione di p5. Nel caso in cui l'utente dovesse rimuoverlo, viene mostrato un messaggio per evitare di lasciare un utente potenzialmente inesperto senza alcuna indicazione. Inoltre, tramite la funzionalità di gestione degli errori di p5 modificata e descritta nella sezione 4.3, vengono mostrati messaggi di errori che guidano l'utente alla definizione delle funzioni `setup` e `draw`.

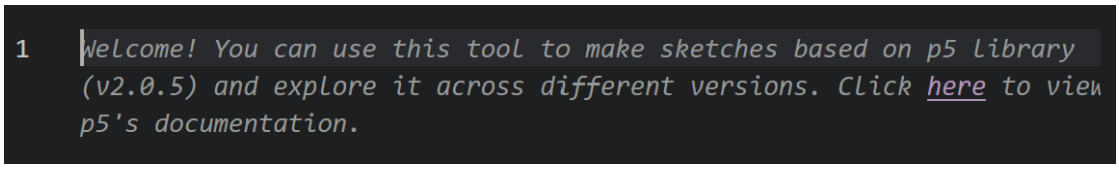


Figura 4.9: Messaggio *placeholder* mostrato quando l'editor non contiene testo.

Integrazione con Monaco

Per agevolare la creazione di *sketch*, vengono importate in Monaco le definizioni dei tipi di p5, permettendo all'editor di fornire completamento automatico del codice

in tempo reale e di mostrare la documentazione delle funzionalità fornite da p5 posizionando il cursore del mouse su funzioni e variabili.

Inoltre, al fine di fornire all'utente feedback preciso e accurato, gli errori e i warning emessi durante l'esecuzione degli *sketch* vengono mostrati tramite la funzionalità di evidenziazione di Monaco, permettendo di capire a colpo d'occhio a quali parti del codice siano relativi gli avvisi e dando la possibilità di visualizzare maggiori informazioni nuovamente posizionando il cursore sopra il testo evidenziato.

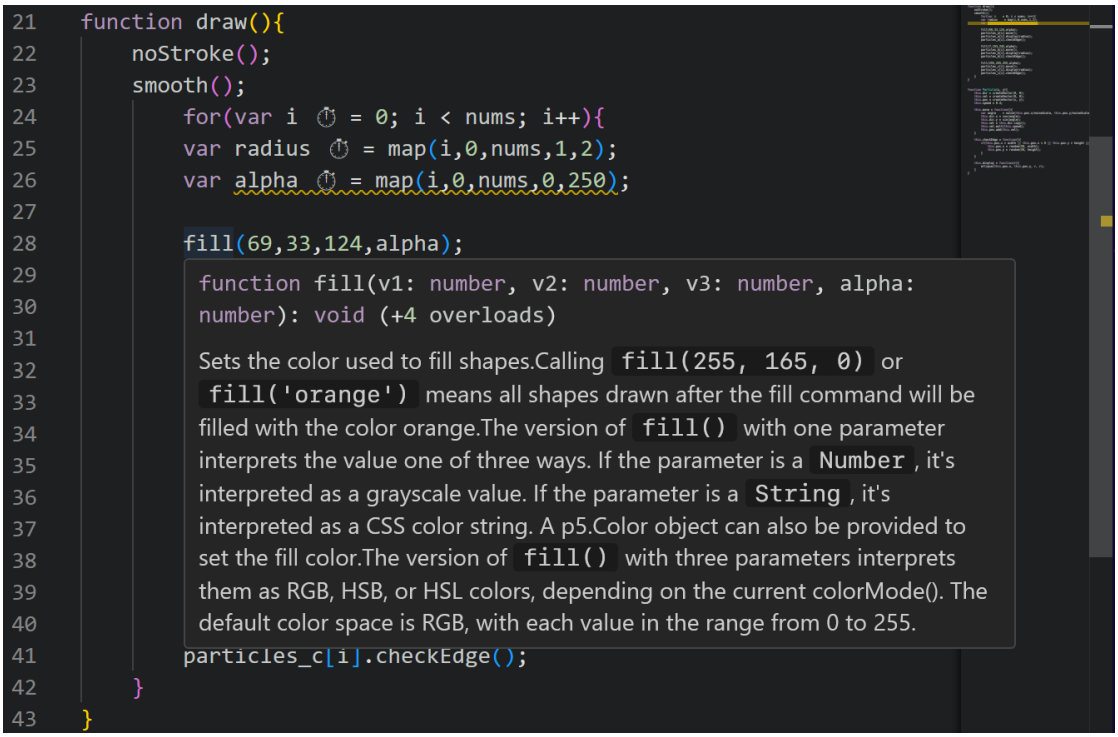


Figura 4.10: Esempio in cui viene mostrata la documentazione della funzione `fill` insieme ad un warning riguardante la variabile `alpha`.

4.5.4 Alcune considerazioni estetiche

Palette colori

I colori utilizzati per gli elementi dell'interfaccia grafica dell'applicazione, vengono estratti direttamente dal tema usato da Monaco e assegnati a variabili CSS, usate a loro volta dalle definizioni degli stili di tutti gli elementi della UI: nel momento in cui si decidesse di usare un tema di Monaco diverso, il resto dell'interfaccia grafica si adatterebbe automaticamente al cambiamento risultando in una palette cromatica sempre coerente.

Ridimensionamento automatico del canvas dello *sketch*

Durante la creazione del canvas, è necessario sceglierne la dimensione in modo che lo *sketch* sia sempre interamente visibile e con le corrette proporzioni. In base alle dimensioni dello *sketch* scelte dall'utente con la chiamata a `createCanvas` (ad esempio utilizzando i valori predefiniti `windowWidth` e `windowHeight` per ereditare le dimensioni del *viewport* o usando dimensioni maggiori dello spazio disponibile), ciò potrebbe portare alla creazione di un canvas che, nel caso di ridimensionamento della finestra del browser da parte dell'utente, non si adatta correttamente alle nuove dimensioni. Per mitigare tale effetto, l'applicazione rileva tali ridimensionamenti e ricrea il `DrawerWorker` con l'annesso canvas usando le dimensioni corrette, mostrando all'utente lo *sketch* nitidamente e nella sua interezza.

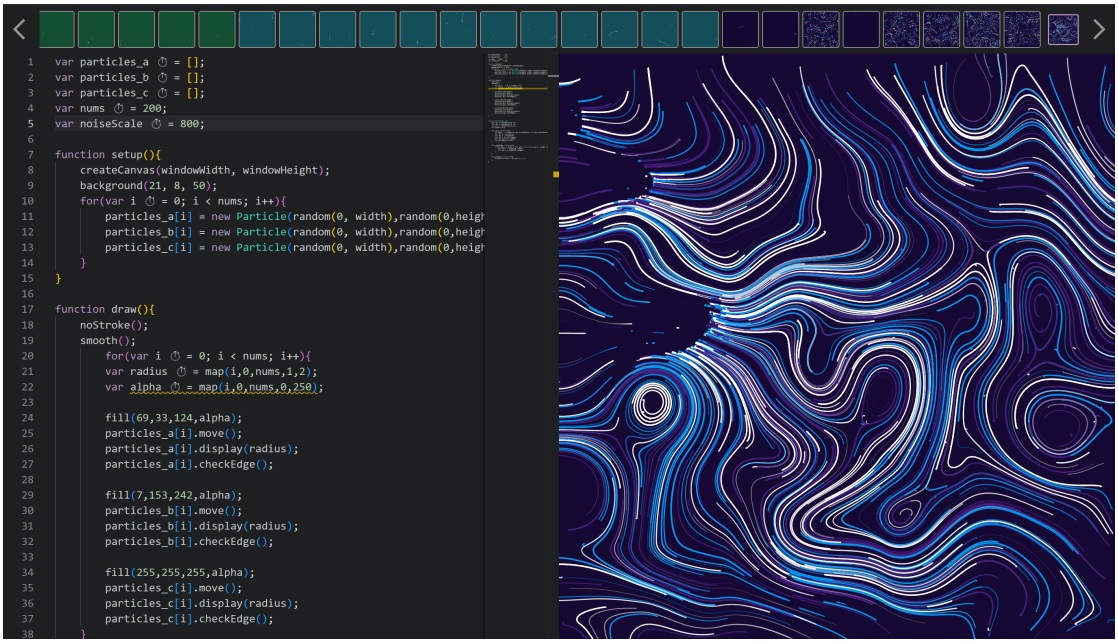


Figura 4.11: L'interfaccia completa dell'applicazione mentre esegue il popolare *sketch* "*perlin noise*" [31] ottenuto da OpenProcessing.

Capitolo 5

Valutazione

5.1 Introduzione

Successivamente all'implementazione, è stato svolto un piccolo test preliminare dell'usabilità in ambiente controllato per valutare l'intuitività e l'efficacia dell'applicazione. L'obiettivo era verificare se gli utenti ritenessero le funzionalità implementate utili allo sviluppo e facili da usare con l'obiettivo di correggerne le eventuali mancanze, criticità e punti deboli.

Un aspetto importante del test sono le metriche usate per valutare l'operato degli utenti. Esse possono essere soggettive o quantitative. Le prime riguardano il background di ciascun soggetto (come età, conoscenze, competenze) e servono per contestualizzare il risultato del test in rapporto alle esperienze pregresse di ogni partecipante. Esse comprendono anche le risposte alle domande aperte poste alla fine del test. Le metriche quantitative, invece, riguardano come gli utenti interagiscono con l'applicazione durante la prova, il tempo impiegato per il completamento di ciascuna Task e, ad esempio, quante persone hanno completato ogni Task. Altre metriche quantitative includono dati aggregati su tutti i partecipanti, come gli Errori Critici, gli Errori non critici, il Tasso di completamento e il Tasso di completamento senza errori. I primi riguardano errori che hanno impedito all'utente di completare la Task, i secondi riguardano gli errori che l'utente è riuscito ad individuare e correggere autonomamente e che quindi non hanno impedito il completamento della Task.

Il test si divide in tre fasi: preparazione, esecuzione e analisi.

5.2 Preparazione

Durante la preparazione sono state pensate otto *task* da far svolgere agli utenti. In questi casi, il numero ideale di task va dalle cinque alle dieci e devono essere

chiare e precise. Per ogni task è stato definito una metrica per determinarne il completamento e il tempo massimo in cui essa deve essere svolta per risultare valida. Le *task* utilizzate sono riportate tradotte in italiano nella Tabella 5.1 e in lingua inglese nell'Appendice A.

Per questa valutazione è stato scelto di osservare i partecipanti e prendere nota delle loro azioni durante lo svolgimento delle task per poi fare loro alcune domande alla fine del test.

Per garantire che ogni partecipante riceva le stesse informazioni, è stato scritto un "copione" da seguire all'inizio della sessione, anch'esso riportato nell'Appendice A. Il copione include una breve introduzione, tutte le task, le metriche di completamento, le domande aperte da porre alla fine del test e una breve sezione per salutare e ringraziare i partecipanti.

Per l'ultima fase del test, sono state preparate alcune domande aperte in modo da ricevere riscontro direttamente dall'utente stesso:

- Quale task ti ha causato maggior difficoltà?
- Useresti queste funzionalità per i tuoi *sketch*?
- Ritieni che queste funzionalità siano utili?
- Ritieni che salvare un'immagine del tuo *sketch* sia difficile?
- Ti è chiaro cosa sia uno *snapshot*?
- Hai altri suggerimenti o commenti da farci?

5.3 Esecuzione

Al test hanno partecipato quattro persone con l'età compresa tra i 25 e i 30 anni, tutti con esperienze pregresse di programmazione in JavaScript e nell'uso di Version Control System, ma senza esperienza nel Creative Coding. Due di loro erano neo-dottorandi, i rimanenti invece lavoratori nel campo dell'ingegneria informatica anche se con diversa maturazione lavorativa. Nessuno di loro ha esperienza nel campo del Creative Coding e sono stati scelti proprio per questo motivo: si voleva infatti verificare l'immediatezza e la facilità d'uso per un utente che non conosce il campo, seppure abbiano competenze di programmazione. Tale campione si posiziona a metà strada tra coloro che non hanno alcun tipo di esperienza nella programmazione e i programmatori creativi professionisti, fornendo quindi un buon punto di partenza per ulteriori indagini future.

Il test è stato svolto di persona utilizzando lo stesso computer per tutti i partecipanti. Durante la prova i partecipanti hanno avuto modo di fare domande e

N	Task	Criterio successo	Tempo massimo
1	Vuoi continuare a lavorare su uno *sketch*, ma non ricordi cosa faccia la funzione `background()`. Trova un modo di leggerne la documentazione.	Il partecipante è in grado di leggere la documentazione della funzione.	2 minuti
2	Vuoi iniziare a creare: aggiungi una linea che vada da $(10, 10)$ a $(100, 100)$.	Il partecipante è in grado di capire quali funzioni siano necessarie per disegnare una linea e le usa correttamente.	5 minuti
3	La linea non ha un bell'aspetto: assegnale un colore salvandolo in una variabile.	Il partecipante è in grado di creare una nuova variabile contenente il colore della linea.	2 minuti
4	Vorresti sperimentare con altri colori per la linea: provane un altro.	Il partecipante è in grado di cambiare il valore della variabile	2 minuti
5	Ti ricordi che qualche versione fa, lo *sketch* aveva uno sfondo di un bel colore. Passa ad una versione precedente per vedere che aspetto avesse.	Il partecipante nota la *timeline* in cima alla finestra e la usa per passare ad una versione precedente.	2 minuti
6	Preferisci il colore di sfondo dell'ultima versione. Ritorna all'ultimo *snapshot*.	Il partecipante è in grado di ritornare all'ultima versione.	2 minuti
7	Preferivi il colore che avevi assegnato alla linea nella Task 2. Ripristina quel valore senza usare la tastiera.	Il partecipante nota le icone a forma di cronometro a fianco dei nomi delle variabili, le clicca e usa l'interfaccia in linea per ripristinare il precedente colore.	2 minuti
8	Ti piace l'aspetto di questo *sketch*. Salvalo come immagine sul disco.	Il partecipante riesce a salvare lo sketch come immagine su disco.	2 minuti.

Tabella 5.1: Elenco delle task.

chiedere chiarimenti, anche se non è sempre stato possibile fornire maggiori dettagli per evitare di influenzarne il risultato. Alla conclusione del test è stato chiesto loro di rispondere alle domande aperte descritte sopra e, se lo desiderassero, di aggiungere ulteriori commenti.

5.4 Risultati

In questa sezione verranno analizzati e commentati i risultati del test con utenti ponendo attenzione su quali Task abbiano causato maggiore difficoltà durante la prova e riportando alcune delle risposte alle domande aperte insieme al pensiero generale degli stessi.

5.4.1 Task

Task	**Errori Critici**	**Errori non critici**	**Tasso di completam.**	**Tasso di compl. senza errori**
1	0	0	1	1
2	1	2	0.75	0.33
3	0	0	1	1
4	0	0	1	1
5	0	1	1	0.75
6	0	0	1	1
7	0	2	1	0.50
8	0	0	1	1

Tabella 5.2: Misure ottenute dal test.

Durante la prova, gli utenti sono riusciti a completare tutte le task, anche se in tempi diversi. Le principali difficoltà osservate hanno riguardato le Task 2 e 7.

Nel caso della Task 2, le problematicità riscontrate sono state dovute all'inesperienza degli utenti sia nell'utilizzo della libreria p5 e sia nella mancanza di conoscenza di come gli *sketch* realizzati per essa siano strutturati.

Nel caso della Task 7, i partecipanti non sono riusciti a notare immediatamente le icone a forma di cronometro a fianco dei nomi delle dichiarazioni di variabili, e hanno invece cercato di interagire principalmente con la *timeline*. Le altre task sono state eseguite senza troppe difficoltà ed hanno portato al risultato sperato.

5.4.2 Domande aperte

Per quanto riguarda le domande poste alla conclusione del test, gli utenti hanno fornito feedback in generale positivo e hanno espresso interesse nell'utilizzo di questo strumento qualora avessero necessità di lavorare su degli *sketch*. È anche emerso che delle funzionalità analoghe sarebbero state apprezzate anche al di fuori del Creative Coding.

In particolare un utente mi ha fatto notare che "sarebbe bello avere un modulo per vedere elencate le funzioni disponibili, perché da persona che non sa quali funzioni sono disponibili è stato difficile capire cosa potessi fare". Un altro dice "in realtà una cosa del genere [la cronologia delle versioni] la userei anche in generale, non solo per questo [il CC]", opinione condivisa anche da altri. Un ulteriore commento è stato "non le vedevo [icone per accedere alla cronologia delle versioni]... non gli cambierei il colore perché sapendo che ci sono si notano, ma farei tipo un pop-up che spiega cosa di può fare la prima volta che si apre l'applicazione".

La principale critica si ricollega alle difficoltà osservate nella Task 2: in questo caso è stato suggerito di apportare un miglioramento alla UI al fine di rendere più evidenti i pulsanti per la visualizzazione della cronologia dei valori di una variabile. È stato anche suggerito di aggiungere un pannello per la visualizzazione delle funzioni di p5 disponibili in modo da agevolare gli utenti meno esperti. Infine è stata anche notato il numero eccessivo di *snapshot* creati durante la modifica dello *sketch*, problematica migliorabile tramite l'implementazione di logica più raffinata per la determinazione di cosa costituisca una nuova versione.

Capitolo 6

Conclusioni

Quello che mi sono prefissato con la stesura di questa tesi relativa al Creative Coding è stato realizzare un'applicazione web che faciliti la creazione e l'evoluzione di *sketch* sia da parte dei programmatori creativi che da parte di coloro si avvicinano per la prima volta al mondo della programmazione.

Viste le limitazioni che gli strumenti relativi al versionamento del codice presentano quando utilizzati nello specifico per il Creative Coding, sono state prese in considerazione le principali criticità lamentate dagli utenti (evidenziate anche dai molteplici studi sull'argomento analizzati durante questo elaborato nel Capitolo 2) e che sono state utilizzate come spunto per la creazione di uno strumento che, basandosi sull'analisi della struttura del codice sorgente relativo agli *sketch*, possa fornire delle funzionalità e un'esperienza il più utile e organica possibile. Tale strumento cerca, infatti, di permettere all'utente di interagire con gli elementi costitutivi del codice in maniera intuitiva e di non richiedere conoscenze nell'ambito dei Version Control System, prediligendo una modalità di operazione il più possibile automatizzata e autonoma.

A tal fine sono stati definiti e descritti i requisiti dell'applicazione e le funzionalità desiderate in modo da affrontare i problemi individuati nel corso della tesi, sono state mostrate le fasi di ideazione dell'interfaccia utente tramite prototipi su carta con annessa evoluzione della UX nel corso della progettazione; è stata poi motivata la scelta di realizzare una nuova applicazione invece di creare un'estensione per uno strumento già esistente al fine di realizzare un applicativo facilmente accessibile anche da utenti meno esperti.

Successivamente sono state descritte le tecnologie utilizzate e le scelte di implementazione delle quali la tecnologia alla base della generazione degli AST, l'uso e il *vendoring* di p5, l'algoritmo alla base del confronto tra versioni dello *sketch*, la struttura e l'interazione tra i vari elementi del programma e la strategia adottata per il salvataggio delle informazioni rilevanti su disco, sono le più significative.

L'applicazione realizzata ha la potenzialità di integrarsi naturalmente nell'ecosistema degli strumenti basati su p5 come OpenProcessing e dai quali ha anche tratto ispirazione.

Il lavoro svolto non ha l'intenzione di essere esaustivo o di risolvere tutte le criticità riscontrate ma, come ogni ricerca, vuole proseguire il lavoro svolto da altri individui e tesisti prima di me; nel contempo, vuole fornire un punto di partenza e ulteriori informazioni a coloro che decidessero di proseguire la ricerca in questo ambito.

A coronamento della realizzazione del programma, è stato eseguito un test di usabilità con utenti non necessariamente avvezzi al Creative Coding in modo da valutarne quindi la facilità di utilizzo anche per individui poco esperti. In seguito a tale processo di valutazione, il feedback ottenuto è stato riportato in questo documento nel Capitolo 5 in modo da essere utile per sviluppi futuri.

6.1 Sviluppi futuri

Durante la creazione dell'applicazione, sono state proposte delle funzionalità che non sono state implementate per via di vincoli di tempo. In aggiunta, durante i test di usabilità gli utenti hanno fornito suggerimenti riguardo a funzionalità che gradirebbero usare nell'applicazione. In questa sezione verranno descritte tali "desiderata", implementabili usando questo lavoro come punto di partenza.

Una delle aggiunte riguarda la UX dei nuovi utenti, che quindi non sono a conoscenza delle funzionalità fornite dall'applicazione. Per questi ultimi sarebbe utile aggiungere un pop-up visibile solamente alla prima apertura dell'applicazione che spieghi cosa sia possibile fare e che, in particolare, porti l'attenzione sulle icone "cronometro" usate per mostrare la cronologia dei valori di una variabile. È infatti emerso che esse non risultano immediatamente evidenti agli utenti neofiti, ma che sono chiare e ben visibili per coloro che invece ne conoscono l'esistenza.

Un'altra funzionalità suggerita riguarda la possibilità di avere a disposizione una sezione dell'interfaccia che elenchi le funzioni fornite da p5 in modo da rendere l'applicazione più accessibile anche per coloro che si avvicinano per la prima volta alla libreria e per i quali la documentazione consultabile in linea non è sufficiente.

Sempre durante i test, è emerso anche che, vista la natura fortemente iterativa del Creative Coding, il salvataggio automatico porta alla creazione di un grande numero di *snapshot* molto simili tra di loro. Ne consegue che potrebbe essere desiderabile ampliare la logica che determina la creazione di una nuova versione in modo che venga unita alla versione precedente qualora il risultato dello *sketch* risultasse uguale o molto simile, ad esempio tramite l'impiego di soluzioni di *Computer Vision*. Inoltre, risulterebbe utile permettere agli utenti di assegnare agli

snapshot una descrizione testuale arbitraria in modo da avere un'organizzazione più navigabile delle versioni o di eliminare quelli che non ritengono più utili.

Durante lo sviluppo dell'applicazione è stata data priorità alle funzionalità descritte in questo documento, mentre non sono ne sono state implementate altre utili come la creazione di più di uno *sketch* alla volta. La struttura interna, tuttavia, rende possibile l'implementazione di una galleria che permetta di gestire molteplici *sketch*, in maniera relativamente semplice e immediata.

Un'ulteriore possibile sviluppo riguarda un maggiore utilizzo dell'AST e dell'*edit script* per tenere traccia di ulteriori aspetti dell'applicazione. Se in questo momento infatti vengono solamente tracciate le dichiarazioni di variabili, le informazioni disponibili permettono di analizzare qualunque aspetto del codice degli *sketch*. Alcuni esempi delle funzionalità che le informazioni disponibili rendono possibili potrebbero essere:

- la deduzione dei tipi di valori immagazzinati in una variabile con la conseguente possibilità di utilizzare elementi grafici specifici per quel valore (come un *color picker* per variabili che rappresentano un colore);
- il tracciamento anche degli argomenti delle funzioni;
- la possibilità di visualizzare la corrispondenza tra una determinata sezione di codice e un particolare elemento riprodotto nel *canvas* e viceversa.

Inoltre, visto l'utilizzo di Yjs, sarebbe possibile adattare l'applicazione per permetterne l'utilizzo a più individui contemporaneamente. Tale funzionalità potrebbe dare agli utenti la possibilità di realizzare nuove forme di opere collaborative o agevolare l'interazione tra insegnanti e studenti durante l'apprendimento dei concetti di programmazione anche da remoto.

In ultimo, viste le limitazioni del test con utenti e basandosi sui suoi risultati incoraggianti, sarebbe auspicabile eseguirne un altro con più partecipanti che abbiano esperienze pregresse nel Creative Coding, in modo da verificare ulteriormente l'adeguatezza dell'applicazione e il raggiungimento degli obiettivi preposti per tutte le categorie di utenti individuate nel corso della tesi.

6.2 Utilizzo dell'applicazione con altre librerie o in altri contesti

Durante il testing con utenti uno di loro ha espresso interesse riguardo alla possibilità di utilizzare le funzionalità presentate anche in contesti esterni al Creative Coding o con altre librerie. Sebbene l'algoritmo GumTree e la logica relativa al tracciamento delle variabili siano generiche e applicabili a qualunque linguaggio di

programmazione (a patto che esista una corrispondente *grammar* Tree-sitter), questa applicazione è costruita attorno e per p5. Aggiungere supporto per altre librerie come Processing, seppure possibile a livello teorico, necessiterebbe la creazione di *layer* di astrazione e cambiamenti strutturali non previste durante lo sviluppo e quindi non semplici da implementare in maniera organica.

Appendice A

Test di usabilità

In questa sezione viene riportato il "copione" da seguire per guidare gli utenti durante il test e le task che questi ultimi dovranno svolgere.

A.1 Introduzione

Hello, _____. Thank you for participating in this test. My name is _____ and I'll be guiding you through this session.

Before we begin, here are a few informations to ensure we are on the same page. We are testing a Web Application built to help Creative Coders create and evolve their *sketches*, which should also be useful for people learning programming. With this test, we want to better understand how this application behaves in the hands of actual users and to see if they find it intuitive and if they like the features they see.

This session is going to take around 20 minutes. While you work, I'm going take notes about how you interact with the application. Keep in mind, however, that we are not evaluating you, we are evaluating the application itself, so do not worry about making mistakes. Feel free to ask question at any time during the test, but keep in mind that I won't be able to answer some of them right away for the sake of the test.

At the end of the session, I'll ask some follow-up questions to better understand what could be inproved. Do you have any questions so far?

You can navigate around the app if you want, once you are ready we'll start with the tasks.

[Let the user navigate around the application for a few seconds]

A.2 Task

Task 1

You want to continue working on a *sketch*, but you do not remember what the `background()` function does. Find a way to read its documentation.

Success Criteria: The participant is able to view the function's documentation.

Metrics:

- Successful task completion (T/F)
- Time on Task: 2 minutes

Task 2

You want to start creating. Add a line going from $(10, 10)$ to $(100, 100)$.

Success Criteria: The participant is able to understand which functions are needed to draw a line and uses them correctly.

Metrics:

- Successful task completion (T/F)
- Time on Task: 5 minutes

Methodology: Cooperative

Task 3

The line looks bland, give it a color, but store it in a variable.

Success Criteria: The participant is able to create a new variable containing the line's color.

Metrics:

- Successful task completion (T/F)
- Time on Task: 2 minutes

Task 4

You'd like to experiment with some other color for that line, change it.

Success Criteria: The participant is able to change the variable's value.

Metrics:

- Successful task completion (T/F)
- Time on Task: 2 minutes

Task 5

You remember that a few versions ago, the sketch had a nice background color. Try peeking old *sketches* to see how they look like.

Success Criteria: The participant notices the *timeline* on the top of the window and is able to use it to jump to a previous *snapshot*.

Metrics:

- Successful task completion (T/F)
- Time on Task: 2 minutes

Task 6

You decided you like the background color of the latest version, go back to the latest *snapshot*.

Success Criteria: The participant is able to return to the latest version.

Metrics:

- Successful task completion (T/F)
- Time on Task: 2 minutes

Task 7

You liked the first color you chose for the line in task 2. Change it back to that without typing it on the keyboard.

Success Criteria: The participant notices the *clock* icon next to the variable's name, clicks it and uses the inline interface to restore the previous value.

Metrics:

- Successful task completion (T/F)
- Time on Task: 2 minutes

Task 8

You like the look of this sketch. Save it as an image on the hard drive.

Success Criteria: The participant is able to save the *sketch* on the hard drive as an image.

Metrics:

- Successful task completion (T/F)
- Time on Task: 2 minutes

A.3 Domande di approfondimento

Thank you for participating. I'd now like to ask you a few questions regarding your experience using this application to gather some feedback about the test:

- Which task did you find more difficult?
- Would you use these functionalities to create your own sketches?
- Do you think these functionalities are useful?
- Did you find saving an image of your sketch difficult?
- Is it clear what a snapshot is?
- Do you have any suggestions for us?

Thanks again for participating. Now that we are done, do you have any other question for me?

Bibliografia

[1] Tyler Angert, Miroslav Suzara, Jenny Han, Christopher Pondoc e Hariharan Subramonyam. «Spellburst: A Node-based Interface for Exploratory Creative Coding with Natural Language Prompts». In: *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. UIST '23. San Francisco, CA, USA: Association for Computing Machinery, 2023. ISBN: 9798400701320. DOI: 10.1145/3586183.3606719. URL: https://doi.org/10.1145/3586183.3606719 (cit. a p. 4).

[2] *Processing*. https://processing.org/. Visitato il giorno 09/11/2025 (cit. alle pp. 5, 20).

[3] Mary Beth Kery e Brad A. Myers. «Exploring exploratory programming». In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2017, pp. 25–29. DOI: 10.1109/VLHCC.2017.8103446 (cit. a p. 6).

[4] N. Deepa, B. Prabadevi, L.B. Krithika e B. Deepa. «An analysis on Version Control Systems». In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. 2020, pp. 1–9. DOI: 10.1109/ic-ETITE47903.2020.39 (cit. a p. 6).

[5] Marc J. Rochkind. «The source code control system». In: *IEEE Transactions on Software Engineering* SE-1.4 (1975), pp. 364–370. DOI: 10.1109/TSE.1975.6312866 (cit. a p. 6).

[6] Mauricio Verano Merino e Juan Pablo Sáenz. «The Art of Creating Code-Based Artworks». In: *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI EA '23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394222. DOI: 10.1145/3544549.3585743. URL: https://doi.org/10.1145/3544549.3585743 (cit. alle pp. 7, 15).

[7] Blair Subbaraman, Shenna Shim e Nadya Peek. «Forking a Sketch: How the OpenProcessing Community Uses Remixing to Collect, Annotate, Tune, and Extend Creative Code». In: *Proceedings of the 2023 ACM Designing*

Interactive Systems Conference. DIS '23. Pittsburgh, PA, USA: Association for Computing Machinery, 2023, pp. 326–342. ISBN: 9781450398930. DOI: 10.1145/3563657.3595969. URL: https://doi.org/10.1145/3563657.3595969 (cit. alle pp. 8, 14).

[8] Eric Rawn, Jingyi Li, Eric Paulos e Sarah E. Chasins. «Understanding Version Control as Material Interaction with Quickpose». In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394215. DOI: 10.1145/3544548.3581394. URL: https://doi.org/10.1145/3544548.3581394 (cit. a p. 9).

[9] Daniel Manesh, Douglas Bowman Jr. e Sang Won Lee. «SHARP: Exploring Version Control Systems in Live Coding Music». In: *Proceedings of the 16th Conference on Creativity & Cognition*. C&C '24. Chicago, IL, USA: Association for Computing Machinery, 2024, pp. 426–437. ISBN: 9798400704857. DOI: 10.1145/3635636.3656195. URL: https://doi.org/10.1145/3635636.3656195 (cit. a p. 10).

[10] *p5.js Web Editor*. https://editor.p5js.org. (Visitato il giorno 08/11/2025) (cit. alle pp. 17, 29).

[11] *OpenProcessing Web Editor*. https://openprocessing.org/sketch/create. (Visitato il giorno 08/11/2025) (cit. alle pp. 17, 29).

[12] *p5.js*. https://p5js.org. Visitato il giorno 08/11/2025 (cit. a p. 20).

[13] *Tree-sitter*. https://tree-sitter.github.io/tree-sitter/. Visitato il giorno 09/11/2025 (cit. a p. 21).

[14] *Yjs*. https://docs.yjs.dev/. Visitato il giorno 09/11/2025 (cit. a p. 21).

[15] Marc Shapiro, Nuno Preguiça, Carlos Baquero e Marek Zawirski. «Conflict-Free Replicated Data Types». In: *Stabilization, Safety, and Security of Distributed Systems*. A cura di Xavier Défago, Franck Petit e Vincent Villain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 386–400. ISBN: 978-3-642-24550-3 (cit. a p. 21).

[16] *Monaco*. https://microsoft.github.io/monaco-editor/. Visitato il giorno 09/11/2025 (cit. a p. 22).

[17] *React*. https://react.dev/. Visitato il giorno 09/11/2025 (cit. a p. 22).

[18] *git-diff Documentation*. https://git-scm.com/docs/git-diff. Visitato il giorno 10/11/2025 (cit. a p. 22).

[19] Eugene W. Myers. «AnO(ND) difference algorithm and its variations». In: *Algorithmica* 1.1 (nov. 1986), pp. 251–266. ISSN: 1432-0541. DOI: 10.1007/BF01840446. URL: https://doi.org/10.1007/BF01840446 (cit. a p. 22).

[20] Philip Bille. «A survey on tree edit distance and related problems». In: *Theoretical Computer Science* 337.1 (2005), pp. 217–239. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2004.12.030. URL: https://www.sciencedirect.com/science/article/pii/S0304397505000174 (cit. a p. 24).

[21] Mateusz Pawlik e Nikolaus Augsten. «RTED: a robust algorithm for the tree edit distance». In: *Proc. VLDB Endow.* 5.4 (dic. 2011), pp. 334–345. ISSN: 2150-8097. DOI: 10.14778/2095686.2095692. URL: https://doi.org/10.14778/2095686.2095692 (cit. a p. 24).

[22] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez e Martin Monperrus. «Fine-grained and accurate source code differencing». In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ASE '14. Vasteras, Sweden: Association for Computing Machinery, 2014, pp. 313–324. ISBN: 9781450330138. DOI: 10.1145/2642937.2642982. URL: https://doi.org/10.1145/2642937.2642982 (cit. a p. 24).

[23] *gumtree: an awesome code differencing tool*. https://github.com/GumTreeDiff/gumtree. Visitato il giorno 10/11/2025 (cit. a p. 24).

[24] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina e Jennifer Widom. «Change detection in hierarchically structured information». In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. SIGMOD '96. Montreal, Quebec, Canada: Association for Computing Machinery, 1996, pp. 493–504. ISBN: 0897917944. DOI: 10.1145/233269.233366. URL: https://doi.org/10.1145/233269.233366 (cit. a p. 25).

[25] *GNU Lesser General Public License v3.0*. https://www.gnu.org/licenses/lgpl-3.0.en.html. Visitato il giorno 12/11/2025 (cit. a p. 27).

[26] *OffscreenCanvas*. https://developer.mozilla.org/en-US/docs/Web/API/OffscreenCanvas. Visitato il giorno 11/11/2025 (cit. a p. 28).

[27] *GNU Lesser General Public License v2.0*. https://www.gnu.org/licenses/lgpl-2.0.en.html. Visitato il giorno 12/11/2025 (cit. a p. 30).

[28] *Using Web Workers*. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers. Visitato il giorno 12/11/2025 (cit. a p. 30).

[29] *Comlink*. https://github.com/GoogleChromeLabs/comlink. Visitato il giorno 12/11/2025 (cit. a p. 30).

[30] *async-mutex*. https://github.com/DirtyHairy/async-mutex. Visitato il giorno 15/11/2025 (cit. a p. 31).

[31] *perlin noise*. `https://openprocessing.org/sketch/494102`. Visitato il giorno 16/11/2025 (cit. a p. 40).