Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica

A.a. 2022/2023

Sessione di laurea Aprile 2023

P5$^+$: Uno strumento per migliorare l'esperienza degli artisti nel mondo del creative coding

Relatori:

Luigi De Russis

Juan Pablo Sáenz Moreno

Candidato:

Giacomo Vitali

Ringraziamenti

Desidero, innanzitutto, ringraziare tutti coloro che mi hanno aiutato nella realizzazione del progetto e nella stesura di questa tesi di laurea. Ringrazio il mio relatore, il Professore Luigi De Russis, per la sua competenza e per la sua disponibilità nel curare e correggere la tesi, capitolo dopo capitolo. Ringrazio anche Juan Pablo Sáenz Moreno per i numerosi consigli durante lo sviluppo dell'applicazione e per i test di usabilità, che non avrei potuto portare a termine nello stesso modo senza il suo aiuto.

Vorrei ringraziare la mia famiglia, che ha sempre creduto in me e mi ha sempre sostenuto durante questo percorso di studi. Ringrazio Caterina, per essere stata al mio fianco.

Ringrazio, inoltre, tutti i miei amici e l'Azione Cattolica, che mi hanno aiutato in questi anni e, in particolare, il mio collega Marco La Gala, con il quale ho condiviso la maggior parte del mio percorso di studi.

Indice

Elenco delle figure

Capitolo 1

Introduzione

La storia dell'arte mostra come gli artisti cerchino sempre nuovi metodi per esprimere le loro idee e le loro forme d'arte. L'evoluzione della tecnologia, negli ultimi decenni, ha cambiato sempre di più il modo di pensare, creare e condividere le proprie opere. Al giorno d'oggi, gli artisti non utilizzano la tecnologia solamente per aiutare il proprio processo creativo, ma molti di loro trasformano completamente la propria arte utilizzando tutta la potenza e i mezzi che la tecnologia gli fornisce [1].

Tutte le opportunità presentate dall'evoluzione della tecnologia hanno cambiato completamente il modo in cui un artista si approccia al proprio lavoro. Con l'avvento del digitale, le opere hanno smesso di essere statiche, diventando, per esempio, sculture realizzate al computer, gallerie virtuali e molto altro. Queste nuove forme di arte digitale vengono sempre più riconosciute anche dai musei, permettendo agli artisti di creare opere tramite nuovi paradigmi.

Fino al giorno d'oggi, la creazione di oggetti e immagini era il principale obiettivo dell'arte visiva, e tutto il processo in cui l'opera veniva pensata era secondario. Oggi, invece, il processo creativo attraversato dall'artista per creare il suo lavoro è passato in primo piano. L'opera finale, infatti, non esiste da sola, ma quello che diventa importante è tutto ciò che la circonda [2].

Il sovrapporsi del mondo artistico con quello informatico, spinge gli artisti ad affrontare un livello di complessità maggiore nella creazione delle proprie opere, visto l'aumentare degli elementi messi a disposizione. Però, questo apre anche un nuovo mondo di possibilità che possono essere utilizzate nella creazione delle proprie opere [3].

Negli ultimi decenni, sempre più artisti hanno cominciato ad avvicinarsi al mondo della programmazione, cercando di utilizzarlo come mezzo per esprimere la propria arte e le proprie idee.

Infatti, la programmazione offre agli artisti una nuova forma di espressione, tramite la quale possono essere create opere d'arte interattive, capaci di comunicare con lo spettatore in modo dinamico.

Tra le discipline che vanno ad unire l'arte e la programmazione è presente il creative coding dove lo scopo non è creare qualcosa di funzionale, bensì qualcosa di espressivo che unisce la programmazione di algoritmi con la creazione di opere digitali [4]. Tradizionalmente, il fine dell'informatica e della programmazione è quello di creare qualcosa di funzionale. Con l'avvento del creative coding, invece, si passa dalla creazione di qualcosa puramente funzionale, alla creazione di qualcosa il cui fine diventa quello espressivo, come per esempio un'arte visiva, un suono, un video e così via.

Questa, però, può essere una sfida per molti artisti, in quanto la programmazione necessita di una comprensione della logica computazionale e della sintassi del codice, che richiedono del tempo per essere capiti a fondo e assimilati. Inoltre, è anche presente un cambiamento di prospettiva rispetto all'arte classica, basata sull'utilizzo dei materiali fisici come un pennello o un foglio di carta, che vengono sostituiti da istruzioni di un programma, tramite le quali è possibile creare le proprie opere.

Tuttavia, nonostante le difficoltà, col tempo sempre più artisti utilizzano il creative coding, anche grazie all'aumento dei corsi di programmazione all'interno delle scuole e delle università, dove si è notato come questi aiutino l'artista a sfruttare pienamente il potenziale di nuove forme artistiche per dare libero sfogo alla propria creatività e alle proprie idee [5].

Un ulteriore fattore che ha portato al maggiore utilizzo del creative coding è l'aumento di programmi e strumenti open-source tramite i quali è possibile creare opere sia scrivendo codice, sia tramite interfacce grafiche. Inoltre, anche il numero di librerie che estendono le funzionalità degli ambienti di programmazione è molto aumentato [6].

Molti artisti, però, senza nessun background informatico fanno fatica ad approcciarsi a questo mondo, visto che molti di questi programmi non sono facili da utilizzare e da comprendere da un'artista inesperto, in quanto richiedono alcune conoscenze sia del linguaggio di programmazione che utilizzano, sia del programma stesso, che spesso nasconde al suo interno molte potenzialità con le quali l'utente non riesce ad entrare facilmente in confidenza. Proprio per questo i nuovi strumenti tendono ad essere sempre più intuitivi e semplici anche da chi non ha esperienza nella programmazione, aiutando l'utente finale ad avere un approccio sempre più creativo alla programmazione.

Tramite questo nuovo approccio per gli artisti è stato possibile spingersi oltre la creazione di opere statiche, andando a creare opere dinamiche e interattive che possono anche reagire agli input dell'utente.

1.1 Obiettivo

L'obiettivo della tesi è quello di rendere più semplice e intuitivo andare a realizzare animazioni e visualizzazioni grafiche attraverso il creative coding, focalizzandosi sulle arti visive, statiche o animate, che si possono creare tramite questa disciplina.

All'interno della tesi viene preso in analisi l'esistente editor di p5, uno degli strumenti esistenti per creare opere con il creative coding, andando a verificare, tramite uno studio della bibliografia e di alcune interviste, cosa in esso può essere migliorato e semplificato risolvendo alcuni dei problemi evidenziati da chi lo utilizza.

Per i problemi maggiormente riscontrati, sono state proposte delle soluzioni che potessero andare a risolverli e per ognuna di esse, sono stati creati dei mockup cercando di definire le principali nuove funzionalità che successivamente sono state implementate all'interno dell'attuale editor di p5, rinominato p5$^+$, con l'obiettivo di aiutare l'utente finale nella creazione dei suoi progetti, semplificando e rendendo maggiormente intuitivo lo strumento utilizzato.

Infine, per verificare che le funzionalità fossero ritenute intuitive e semplici da chi utilizza questo strumento per creare le proprie opere, è stato svolto un test di usabilità con alcuni artisti che utilizzano il creative coding e conoscono l'editor di p5.

1.2 Struttura della tesi

La tesi presenta nel Capitolo 1 una breve introduzione iniziale sull'evoluzione della programmazione e su come questa si intrecci al mondo dell'arte, introducendo il concetto del creative coding. Successivamente, è presentato l'obiettivo della tesi e una descrizione della struttura della stessa.

Nel Capitolo 2 *Background e Stato dell'Arte* si affronta uno studio della letteratura esistente, andando a definire in cosa consiste il Creative Coding e come questo negli ultimi anni è stato sempre più utilizzato, per poi andare ad elencare e approfodnire quali sono i principali programmi e strumenti utilizzati dagli artisti.

Il Capitolo 3 *Progettazione* descrive in maniera generale la struttura di Processing e p5, analizzando pro e contro per poi evidenziare quale dei due è stato scelto per l'implementazione delle funzionalità. Successivamente, dopo aver presentato le nuove funzionalitò, vengono presentati i mockup per ogni funzione proposta.

Il Capitolo 4 *Implementazione* mostra come ogni funzionalità è stata implementata all'interno dell'editor di p5, dopo aver approfondito le tecnologie utilizzate.

Nel Capitolo 5 *Valutazioni* viene mostrato il test di usabilità condotto su alcuni utenti che utilizzano il creative coding e conoscono l'editor di p5.js, per poi discutere i risultati ottenuti.

Il Capitolo 6 *Conclusioni* esplora i risultati ottenuti, per poi fornire qualche spunto per degli sviluppi futuri.

Capitolo 2

Background e Stato dell'Arte

2.1 Creative Coding

Il creative coding è il risultato dell'incontro tra l'arte e la tecnologia, e la sua storia è strettamente legata allo sviluppo dell'arte digitale e all'informatica. È un processo basato sull'esplorazione, la riflessione e la scoperta, dove il codice viene utilizzato come mezzo per creare una vasta gamma di arte [7].

Il creative coding nasce negli anni '60, ma comincia ad essere utilizzato solamente a partire dagli anni '80 e '90. Negli anni 2000 aumenta la sua popolarità, diffondendosi sempre di più. Negli ultimi 10/15 anni si è infatti visto un gran numero di artisti, designer e sviluppatori aderire a questa corrente artistica.

Questa disciplina che combina la teoria e la metodologia dell'informatica e dell'ingegneria con principi estetici e approcci pedagogici delle arti grafiche [5], può essere vista sia come un uso artistico del codice, sia come un modo per comprendere il mondo digitale che ci circonda [8].

Tramite il creative coding, invece di creare una singola opera, l'artista crea un processo che segue una serie di regole per creare arte, attraverso un computer o altri dispositivi, come mostrato in Figura 2.1. Il creative coding può essere inteso come creare tool che non esistono, con la possibilità di prendere ciò che altri hanno prodotto e modificarlo affinché venga creato qualcosa di unico. L'opera può variare dall'essere un'immagine visibile su uno schermo con suoni e movimenti ad essere stampata in qualche modo da un dispositivo [9].

Questo viene utilizzato da designer, artisti, pittori, poeti, animatori, sviluppatori e molti altri e per questo motivo si possono costruire arti visive, suoni, videogiochi, prototipi di prodotti e così via. Ciò che accomuna tutti questi diversi ambiti è il fatto che bisogna conoscere e saper capire come dev'essere strutturato ciò che si

Figura 2.1: Ryan Alexander. Mycelium, 2007. Opera creata utilizzando Processing.

vuole creare e per questo nasce il bisogno di avere delle basi di informatica per poter cominciare a scrivere del codice e creare la propria opera.

Da diversi anni, ci si interroga su come poter avvicinare gli artisti al mondo dell'informatica, per esempio dedicando appositi corsi al creative coding per mettere insieme conoscenze di informatica e di principi estetici, in modo che gli studenti siano in grado di apprendere concetti non solo nel mondo delle arti, ma anche nel mondo informatico, come algoritmi e programmazione [10, 11, 12].

Ci sono diversi motivi per cui è ritenuto importante avvicinare gli artisti al mondo della programmazione. Prima di tutto, bisogna considerare che gli artisti nello sviluppo delle loro opere, devono collaborare con persone che vengono da mondi differenti dal loro. Per esempio, se un designer disegna qualcosa di digitale, la figura chiave a cui fa riferimento potrebbe essere uno sviluppatore. Il rapporto tra un designer e uno sviluppatore potrebbe essere complicato per la differenza tra gli ambienti da cui provengono. Conoscere qualcosa sulla programmazione, permetterebbe di diminuire il divario tra i due mondi.

Un secondo motivo è che l'artista deve poter partecipare non solo alla realizzazione dell'opera, ma anche nel processo in cui questa viene pensata. Avere qualche conoscenza di programmazione, gli permetterebbe di poter partecipare più attivamente.

Inoltre, la programmazione non deve essere considerata come qualcosa di noioso, ma può essere qualcosa che permette di scoprire nuovi modi per esprimersi e per avere nuove idee. Molti designer utilizzano già la programmazione nei loro lavori. Grazie ad essa, creare forme o interazioni tra gli elementi, che può rivelarsi complicato da fare a mano, diventa molto più semplice e veloce.

Infine, la programmazione non è qualcosa legato strettamente al mondo dell'ingegneria dell'informazione. Infatti, visto che i software influenzano tutta la cultura e le opere contemporanee, incluso il design, gli artisti devono poter comprendere a pieno tutto ciò che li circonda e le opere che andranno a creare [3].

All'interno di questa tesi si è deciso di focalizzarsi sulle arti visive, che siano esse statiche o animate. Per questo motivo, si sono presi in considerazione i tool che aiutano l'artista in questo ambito. Più specificamente, si è scelto di concentrarsi su Processing e p5.js, entrambi appartenenti alla Processing Foundation, in quanto vengono molto utilizzati per questo genere di opere. Questi verranno approfonditi nei paragrafi successivi.

2.2 Differenze tra creative coding e generative art

Spesso, in letteratura, il termine creative coding viene sostituito dal termine generative art, nonostante i significati siano leggermente diversi.

Con generative art ci si riferisce a tutte le pratiche dove l'artista crea un processo, inteso come un programma per computer, una macchina o altre invenzioni, che poi compie autonomamente del lavoro tramite alcuni gradi di libertà contribuendo o creando da zero un'opera d'arte. Quest'ambito si concentra sull'utilizzo di processi automatici e algoritmi per generare opere.

La generative art, infatti, non è strettamente legata al mondo della programmazione, come il creative coding.

Il generative design può essere descritto come la trasformazione di un input di dati esistente in un output con una forma diversa. Un esempio può essere quello della conversione di un testo in un'animazione.

2.3 Abitudini e difficoltà degli artisti

Avvicinarsi al mondo della programmazione, come evidenziato precedentemente, oltre ad aiutare gli artisti ad espandere le proprie idee e approcciarsi a nuovi modi

per realizzare le proprie opere può essere complicato, in quanto la programmazione segue determinati paradigmi che devono essere assimilati prima di poter creare i propri lavori.

Per gli artisti, potrebbe anche essere complicato entrare nella logica strutturata proposta da alcuni strumenti per creare opere tramite il creative coding, in quanto queste modalità di progettazione sono poco compatibili con la manipolazione manuale che per molto tempo ha contraddistinto il mondo dell'arte, come evidenziato nell'articolo di Li et al. [13], dove vengono intervistati alcuni artisti per comprendere meglio come loro utilizzano e sviluppano software per creare opere tramite il creative coding. Nelle interviste, viene approfondito come l'artista è motivato a costruire software come un meccanismo di impegno intellettuale e sociale, come i software esistenti lavorino insieme o contro il flusso di pensiero di un artista e come questi strumenti vengano utilizzati non solo per creare le proprie opere, ma anche come strumento per organizzarsi, motivarsi e riflettere.

Dalla ricerca viene evidenziato come i visual designer cerchino tool che siano direttamente integrati con strumenti di visual design e utilizzino tool di alto livello creati per alcuni specifici obiettivi diversi rispetto a ciò per cui vengono utilizzati, invece di imparare ad utilizzare un nuovo e differente framework.

Dalle interviste effettuate all'interno del lavoro, emerge il fatto che gli artisti preferiscono utilizzare spesso lo stesso strumento per generare le loro opere d'arte. In più, viene evidenziato come gli artisti tendano a sviluppare dei software dove lavorare per poter aggiungere nuove funzionalità che non vengono riscontrate all'interno degli strumenti esistenti, così da non avere dei limiti imposti dagli strumenti stessi. Un intervistato, per esempio, rimarca come, lavorando con Processing, debba effettuare gli screenshot per ogni frame dello sketch, in modo da poterli riguardare chiaramente, visto che il tool non offre nessuna funzionalità che permette di fare ciò. Un altro artista evidenzia invece come utilizzi alcuni software digitali per collezionare e organizzare i propri sketch, in modo da averli sempre a portata di mano per poterli modificare o riutilizzare all'interno di uno sketch differente.

2.4 Strumenti e alternative per supportare gli artisti

Oltre a intervistare gli artisti sui principali problemi o mancanze da loro rinvenuti durante l'utilizzo di strumenti per creare opere con il creative coding, vengono anche proposti diversi strumenti e diverse funzionalità per cercare di rendere la creazione di opere d'arte semplice e intuitiva, sia tramite la modifica di alcuni tool esistenti, sia creando da zero nuovi strumenti che gli artisti possano utilizzare.

Gli obiettivi di un ambiente di programmazione dovrebbero essere quelli di incoraggiare nuovi modi di pensare e permettere a chi scrive codice di vedere e capire l'esecuzione di ciò che sta scrivendo. Secondo Victor [14], Processing e JavaScript vengono considerati linguaggi che non rispettano questi obiettivi e l'unico modo per far sì che questi vengano rispettati è quello di modificare il programma su cui l'utente andrà a lavorare. Il programma viene diviso in due parti: una riguarda l'ambiente di programmazione, che è la parte installata all'interno del computer e l'altra riguarda il linguaggio, che è la parte che deve imparare l'utente finale che andrà a scrivere il codice. Ognuna di queste due parti dovrebbe adottare qualche accorgimento per rispettare gli obiettivi citati inizialmente.

Il primo punto si concentra sul comprendere il lessico utilizzato dal linguaggio. L'ambiente di programmazione deve essere in grado di permettere all'utente di leggere facilmente ciò che viene scritto, permettendogli di capire a fondo tutto ciò che il programma compie al suo interno, in modo che si possa concentrare sul creare la propria opera e non sul capire cosa comporta l'esecuzione di ogni istruzione. Un modo per risolvere questo problema potrebbe essere quello di aggiungere le definizioni delle singole istruzioni all'interno del codice, da mostrare quando l'utente passa con il mouse sopra una riga, come mostrato in Figura 2.2.

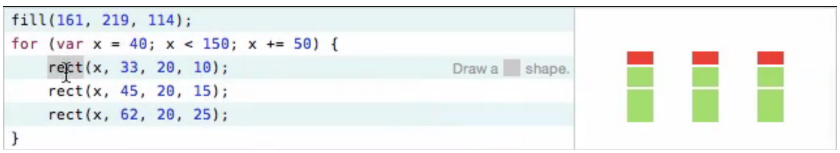


Figura 2.2: Definizioni mostrate per aiutare l'utente a comprendere il codice.

Il passaggio successivo è quello di permettere all'utente di seguire l'esecuzione del codice, specialmente se questo al suo interno presenta dei cicli, per esempio visualizzando i singoli frame dell'applicazione, in modo da permettere all'artista di scorrere tra di essi e comprendere bene come il codice si evolve man mano che l'animazione avanza.

Integrando questi accorgimenti, l'utente riesce meglio a seguire il flusso del programma, comprendendo meglio cosa ogni istruzione produce e come l'animazione si evolve nel tempo.

Successivamente, viene anche spiegato come la visualizzazione dell'anteprima di ciò che l'artista sta creando dovrebbe essere riprodotta il prima possibile dopo ogni istruzione, così che l'utente possa comprendere meglio ciò che sta creando.

Alcuni utenti hanno anche creato un nuovo editor di p5 da integrare all'interno di un ambiente di programmazione che, tra le altre funzionalità, permette anche di poter visualizzare le definizioni di ogni singola istruzione che viene utilizzata [15], cosa che non è possibile fare nell'attuale editor online.

All'interno dello studio di Hoffswell et al. [16] viene proposta un'alternativa agli strumenti esistenti, ovvero la possibilità di visualizzare le variabili del programma direttamente accanto al codice sorgente, così da poter seguire in modo chiaro il loro cambiamento. In questo modo, diversi snapshot del programma possono evidenziare precisamente il valore di una qualsiasi variabile in un determinato momento, o la distribuzione di una variabile nel tempo, come mostrato in Figura 2.3. Questo è stato molto apprezzato dai partecipanti dello studio che successivamente hanno valutato l'usabilità della nuova funzione.

Figura 2.3: Istogramma che mostra come le variabili cambiano nel tempo.

Un'ulteriore modifica viene proposta da Sasson, il quale propone un'estensione al tool di Processing chiamata TweakMode [17], mostrata in Figura 2.4.

Quando uno sketch viene eseguito tramite questa modalità, tutti i valori all'interno dello sketch che non appartengono a variabili globali diventano interattivi e possono essere modificati semplicemente cliccando su di loro e spostando il cursore a destra o a sinistra. Quando un valore cambia, l'anteprima viene aggiornata immediatamente. Questa estensione può essere utile, per esempio, per andare a modificare alcuni particolari di un'immagine, così da vedere immediatamente il cambiamento applicato all'anteprima, oppure per comprendere il codice scritto da un altro utente. Tramite questa modalità viene anche messa a disposizione una finestra con un color picker dove è possibile scegliere un colore per andare a modificarne uno già presente.

Lo studio portato avanti da Chugh et al. all'interno di due diversi articoli [18, 19] propone un nuovo strumento chiamato Sketch-n-Sketch, mostrato in Figura 2.5, il quale permette una manipolazione diretta dell'output visivo. Ogni volta che la parte di codice viene modificata dall'utente, la parte dell'output visivo viene aggiornata e viceversa, tramite un processo chiamato *live synchronization*. Gli

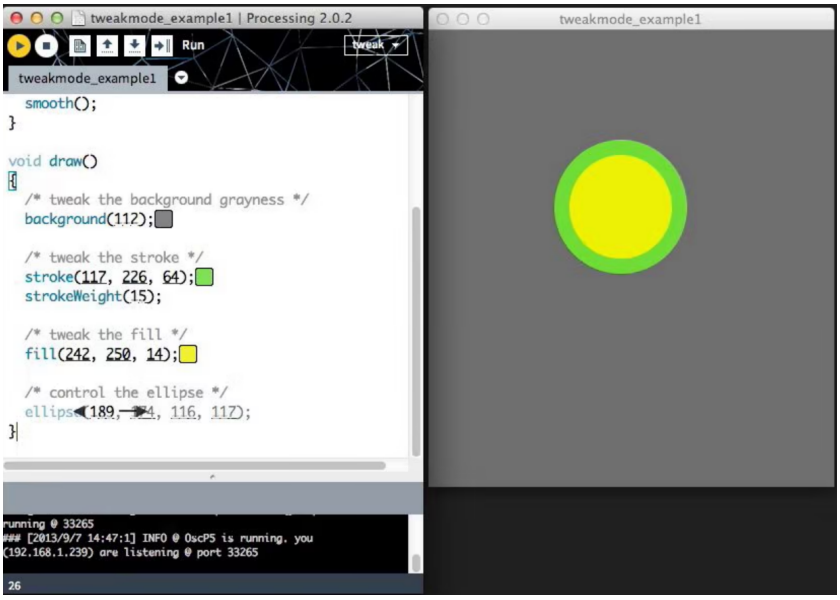


Figura 2.4: Tweak Mode in Processing.

autori propongono diverse tecniche per andare a realizzare lo strumento Sketch-n-Sketch, lavorando però maggiormente sui vettori grafici, senza considerare la possibilità di poter creare o gestire animazioni.

Viene però evidenziato dagli autori stessi come l'obiettivo dello studio non sia quello di creare un nuovo tool di disegno, ma utilizzare questo ambiente come uno studio per aumentare l'espressività di un programma su cui è possibile modificare direttamente l'output.

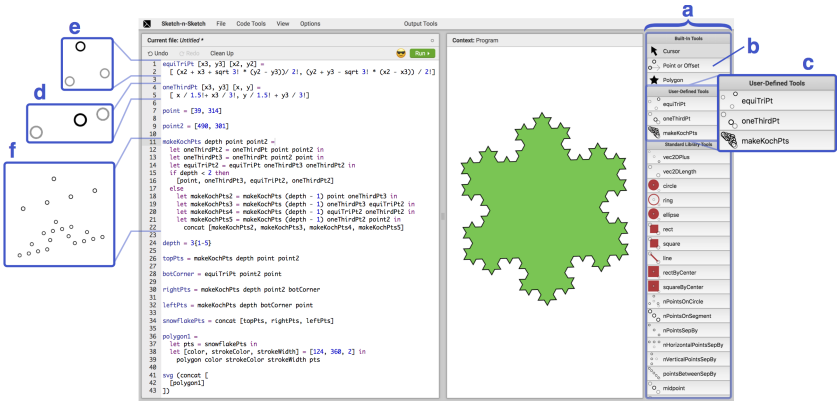


Figura 2.5: Interfaccia principale di Sketch-n-Sketch.

Il lavoro di Li et al. [20] evidenzia come la programmazione offra nuove opportunità per la creazione di arti visive, ma capire e manipolare la rappresentazione che rende potenti gli strumenti utilizzati può porre delle barriere per artisti abituati a interazioni visive. Viene quindi sviluppato e creato un tool chiamato *Demystified Dynamic Brushes (DDB)* che va a legare codice, dati numerici e arte grafica tramite un'interfaccia di programmazione. Questo strumento va a registrare tutti gli input disegnati manualmente da un artista, in modo che possano essere approfonditi sia quelli attuali che quelli passati e possa essere controllata l'esecuzione del programma.

All'interno dell'articolo viene evidenziato come gli artisti, per comprendere meglio ciò che stanno creando, spesso compiono dei piccoli cambiamenti, ridisegnando solo piccoli pezzi del lavoro finale.

Nel lavoro sviluppato da Jacobs et al. [21] viene implementato Para, un tool per l'illustrazione digitale che supporta la creazione di arte tramite la manipolazione dell'output con l'utilizzo di vettori. Viene evidenziato come i linguaggi di programmazione presentano al loro interno barriere, richiedendo agli artisti di imparare molti concetti prima di riuscire a portare a termine anche semplici obiettivi. Viene anche fatto notare come, per esempio, per poter utilizzare Processing, gli artisti debbano imparare un sacco di convenzioni del linguaggio prima di produrre opere d'arte basiche.

Riassumendo, la principale funzionalità trattate all'interno della maggior parte degli studi riguarda la modifica dell'output tramite la sua diretta manipolazione. Questo però non verrà approfondito all'interno del lavoro di tesi.

Molti altri spunti, invece, come il poter vedere tutti i frame di un determinato sketch, o la possibilità di modificare parzialmente uno sketch tornando alla sua ultima modifica, verranno riutilizzate successivamente in fase di progettazione.

2.5 Programmi e strumenti utilizzati nel creative coding

Esistono diversi tool, librerie ed estensioni che supportano gli artisti nello sviluppo delle loro opere, come immagini, video, suoni o videogiochi.

Questi possono essere distinti in diverse categorie: i framework, i visual programming languages, i sound programming languages, i web programming, i projection mapping e altre ancora [22]. Di seguito, per ogni categoria, verranno elencati alcuni dei tool che ne fanno parte e verranno approfonditi.

2.5.1 Framework

In questa prima categoria sono presenti i framework, le librerie e gli ecosistemi. Questi tre termini hanno significati leggermente diversi tra di loro.

Con framework si intende un insieme di librerie e strumenti preconfigurati che forniscono una struttura per lo sviluppo di un'applicazione o di un sistema. Il programmatore utilizza il framework per costruire l'applicazione, ma deve seguire le linee guida fornite dal framework.

Con libreria si intende un insieme di funzioni e strutture di dati riutilizzabili che possono essere utilizzati per sviluppare un'applicazione. Inoltre, possono essere utilizzate per semplificare lo sviluppo del software.

Con ecosistema, invece, si intende un insieme di componenti interconnessi che lavorano insieme per supportare lo sviluppo e la distribuzione di software.

In questa categoria possiamo trovare i seguenti tool:

- Processing
- Cinder
- openFrameworks
- Unity
- Godot

Processing è stato sviluppato da Ben Fry e Casey Reas a partire dal 2001. Questo è un ambiente di programmazione open source basato su Java che mira a facilitare la creazione di grafica, animazioni e interazioni. Esso offre una vasta libreria di funzioni per la creazione di forme, colori e animazioni, rendendo per gli artisti più semplice la creazione di contenuti digitali. Processing verrà approfondito successivamente nella Sezione 3.1.1.

Cinder è una libreria open-source per la programmazione di applicazioni interattive. È scritta in C++ ed è pensata per sviluppatori e artisti che vogliono produrre contenuti per Windows, macOS, Linux, iOS e Android. Offre una serie di strumenti e funzionalità avanzate per la creazione della grafica e un'architettura flessibile che permette di sviluppare progetti che possono essere in seguito estesi con funzionalità più avanzate [23].

openFrameworks è una libreria open-source per la creazione di applicazioni che possono anche essere interattive. È scritto in C++, ma presenta alcune estensioni del linguaggio utilizzato e supporta varie piattaforme, tra cui Windows, macOS, iOS e Android. Offre al suo interno un'architettura modulare che permette di estendere facilmente le funzionalità della libreria o di integrarla con altre tecnologie [24]. La struttura della libreria e del framework, però, può renderlo qualche volta più complicato per chi non lo conosce bene [25].

Cinder e openFrameworks sono molto simili, ma si differenziano in diversi aspetti, tra cui l'architettura e le funzionalità. Il primo ha un'architettura più definita, mentre il secondo ne presenta una più flessibile e modulare. Per quando riguarda le funzionalità, il secondo permette di estendere facilmente la libreria con altre tecnologie.

Unity è una piattaforma di sviluppo di giochi e applicazioni interattive che fornisce una serie di strumenti per la creazione di contenuti 2D e 3D. Con Unity possono essere creati giochi, app interattive, visualizzazioni 3D e molto altro. Offre un'interfaccia visuale intuitiva per la creazione dei contenuti e molte funzionalità avanzate per la programmazione.

Godot è una piattaforma open-source per la creazione di giochi e applicazioni interattive. Con questo possono essere creati giochi e app interattive utilizzando una combinazione di codice e grafica. Godot è scritto in C++ e supporta una vasta gamma di linguaggi di programmazione, tra cui GDScript, un linguaggio script proprietario pensato appositamente per Godot [26].

Unity e Godot sono molto simili, ma si differenziano principalmente per l'architettura, e per l'interfaccia utente. Il primo ha un'architettura più rigida e definita, e un'interfaccia utente più complessa ma più potente. Godot, invece, ha un'architettura più flessibile e modulare e un'interfaccia utente più semplice e più intuitiva.

2.5.2 Visual Programming Languages

Con visual programming language, traducibile in *linguaggio di programmazione visuale*, si intende un linguaggio che utilizza rappresentazioni grafiche come blocchi o diagrammi invece di righe di codice testuale per creare programmi. Questi linguaggi vengono pensati per essere intuitivi e facili da comprendere anche per chi non ha esperienza di programmazione. Allo stesso tempo, però, potrebbero essere meno potenti e flessibili rispetto ai linguaggi tradizionali.

In questa categoria possiamo trovare i seguenti tool:

- Vvvv
- NodeBox
- Isadora
- JOY.JS

Vvvv è un software open-source per la creazione di applicazioni interattive, visive e di realtà virtuale. Questo utilizza un approccio visuale, consentendo agli artisti di creare progetti utilizzando diagrammi a blocchi che rappresentano i vari flussi di lavoro. Questi blocchi possono essere collegati fra loro senza dover scrivere codice.

Questo perché, nonostante sia un ambiente di programmazione, usa al suo interno un editor grafico, il quale permette di creare ambienti e interazioni visualmente. Questo tool è molto popolare per artisti e designer che cercano una soluzione da utilizzare per la creazione di progetti interattivi e visivi, ed è molto usato per installazioni artistiche, giochi interattivi e progetti di realtà virtuale. L'interfaccia, però, non è molto intuitiva e progetti più complessi possono velocemente creare un grande intreccio di blocchi e flussi di lavoro [25].

NodeBox è un software open-source per la creazione di grafica e animazione, tramite la creazione e il collegamento di blocchi che rappresentano i vari flussi di lavoro. Ogni blocco, o nodo, rappresenta una funzione, e i suoi dettagli possono essere modificati selezionandoli dal menu di ogni nodo. I dati esterni possono essere aggiunti al nodo e modificati direttamente senza scrivere righe di codice. Nonostante i singoli blocchi possano essere personalizzati tramite Python, l'ambiente di programmazione rimane meno espressivo e con meno funzionalità rispetto ad altri programmi come vvvv [25]. Il software è molto flessibile e presenta al suo interno diverse funzionalità per la creazione di forme, testo, immagini e animazioni.

Isadora è un software per la creazione di spettacoli e installazioni interattive basato sulla realtà virtuale. Il software è molto flessibile e può essere utilizzato per spettacoli al vivo, installazioni interattive o progetti di realtà virtuale.

JOY.JS è un framework JavaScript che permette di creare giochi interattivi e animazioni per il web. Offre, al suo interno, anche una serie di funzionalità avanzate per la gestione degli effetti di un'animazione, che lo rendono molto flessibile e capace di creare prodotti di alta qualità.

2.5.3 Web Programming

Il web programming, traducibile in *programmazione web*, comprende tutti i programmi accessibili tramite un browser web da qualsiasi dispositivo connesso a Internet.

In questa categoria possiamo trovare i seguenti tool:

- Three.js
- Paper.js
- P5.js
- Babylon.js
- Theatre.js

Three.js è una libreria open-source JavaScript per la creazione di effetti 3D e animazioni in tempo reale. È facile da utilizzare anche da chi non ha molta

esperienza con la programmazione 3D. Utilizza al suo interno WebGL, che è un'interfaccia di programmazione delle grafiche per il browser, in modo da reindirizzare i contenuti 3D. La libreria include molte funzionalità per la creazione di oggetti 3D e la gestione delle animazioni [27].

Paper.js è una libreria open-source per la creazione di animazioni e grafici vettoriali. Si basa sul Canvas di HTML5, quindi tutti i contenuti generati all'interno di questa libreria possono essere visualizzati su qualsiasi browser senza installare plug-in particolari. Fornisce un'interfaccia intuitiva e facile da utilizzare, tramite la quale è possibile creare facilmente oggetti 2D e 3D [28].

P5.js è una libreria JavaScript open-source che fornisce un ambiente di programmazione per la creazione di contenuti interattivi e animazioni. La libreria si basa su Processing. Tramite p5, gli artisti possono facilmente creare forme, animazioni e immagini e gestire le animazioni. La libreria include al suo interno anche molti strumenti avanzati e interattivi. P5 verrà approfondito successivamente nella Sezione 3.1.2.

Babylon.js è una libreria open-source JavaScript per la creazione di contenuti interattivi 3D, come giochi o applicazioni. La libreria è stata ottimizzata per la creazione dei contenuti 3D anche su dispositivi mobili.

Theatre.js è una libreria open-source JavaScript che permette di creare contenuti animati e interattivi in 2D e 3D. È possibile creare il proprio progetto sia lavorando direttamente sul codice, sia andando a modificare direttamente l'output tramite l'interfaccia grafica, con la possibilità di creare animazioni più o meno complesse.

Capitolo 3

Progettazione

In questo capitolo vengono presentate le opzioni scelte per la fase di progettazione e realizzazione del prototipo. Si è deciso inizialmente di approfondire p5 e Processing, concentrandosi sulle principali funzionalità offerte e sulle principali mancanze riscontrate su ognuno di essi. Successivamente, riprendendo l'analisi della letterature del creative coding effettuata nel capitolo precedente, sono stati elencati i principali problemi riscontrati dagli utenti e sono state elencate alcune funzionalità che potessero andare a risolvere le principali criticità rinvenute.

Si è poi deciso di implementare le nuove funzionalità all'interno dell'editor di p5. Successivamente, sono stati proposti una serie di mockup in modo da decidere come e dove inserire le nuove funzionalità all'interno dell'editor, cercando di mantenere la sua interfaccia semplice e intuitiva.

3.1 Analisi dei programmi esistenti

All'interno di questa sezione verranno approfonditi singolarmente Processing e p5, entrambi facenti parte della Processing Foundation. Si è deciso di concentrarsi su questi strumenti in quanto sono tra i più utilizzati dagli artisti per creare arti visive statiche o dinamiche tramite il creative coding.

Infine dopo aver elencato pro e contro di entrambi, si è deciso in quale dei due tool implementare le nuove funzionalità.

La Processing Foundation è un'organizzazione no-profit basata sulla promozione e lo sviluppo del linguaggio Processing. È stata fondata nel 2012 da Ben Fry e Casey Reas, per garantire che il proprio linguaggio di programmazione fosse accessibile a quanti più programmatori e artisti possibili. Fornendo una vasta gamma di risorse e strumenti agli sviluppatori, e organizzando eventi e workshop, l'organizzazione cerca di promuovere e sviluppare sempre di più il proprio linguaggio, tramite una cultura aperta e inclusiva. Tutto questo viene fatto sviluppando e distribuendo

diversi software, che includono al loro interno Processing, p5.js, Processing.py, Processing Pi e Processing per Android.

3.1.1 Processing

Processing è un ambiente di programmazione visuale open-source creato nel 2001 da Ben Fry e Casey Reas, con lo scopo di avvicinare l'arte alla tecnologia e all'informatica. Tramite un linguaggio semplice e accessibile, il suo scopo è quello di facilitare la creazione di forme, immagini e animazioni anche agli utenti meno esperti nel campo della programmazione. Anche per questi motivi, è uno dei programmi più utilizzati nell'ambito del creative coding.

È stato chiamato così perché permette di modificare, muovere e combinare simboli a basso livello per costruire rappresentazioni ad alto livello. Oltre a questo, permette anche di concentrarsi sul processo della creazione piuttosto che sul risultato [29].

Nasce inizialmente con l'idea di essere utilizzato come software per creare sketch e per insegnare le basi della programmazione. Si parla di sketch e non di progetti, proprio per rendere questo strumento il più vicino possibile al mondo dell'arte, cercando di rendere intuitivo ed accessibile anche il linguaggio utilizzato nella definizione dei termini.

Processing è estremamente flessibile e include molte librerie che vanno a migliorarlo, anche grazie alla ricca documentazione e ai molti contributi degli utenti che lo utilizzano, che possono aggiungere plug-in e add-ons per arricchire il programma ed estendere le sue capacità. Fornisce inoltre anche un supporto per i lavori in 3D.

Grazie a Processing, molti artisti, designer e sviluppatori possono accedere a uno strumento molto potente e personalizzabile, tramite il quale è possibile esprimere le proprie idee e il proprio talento artistico in maniera molto intuitiva e semplice.

Inoltre, ha reso la programmazione più accessibile e aperta ad un pubblico più ampio, creando alle sue spalle una grande comunità di persone che condividono le proprie conoscenze per aiutarsi nella creazione dei propri progetti. La comunità di Processing, infatti, è molto attiva.

Processing è anche molto utilizzato da utenti più esperti, in quanto, nonostante la sua semplicità, è uno strumento che riesce a gestire anche progetti molti complessi, utilizzando librerie e strumenti avanzati.

Processing permette quindi di scrivere, modificare, compilare ed eseguire codice in Java. Java è anche considerato un buon punto di partenza perché è molto più tollerante di altri linguaggi, come il C++, e permette agli utenti la distribuzione dei propri sketch tra diverse piattaforme. Il Java è anche stato considerato un linguaggio che bilancia correttamente velocità e semplicità d'uso. Se, per esempio, consideriamo linguaggi come Ruby o Python, questi sono molto semplici da imparare,

ma la loro velocità è minore. Al contrario, linguaggi come il C e il C++ sono molto più veloci, ma più difficili da comprendere.

Tramite un preprocessore, il codice scritto in Processing viene convertito in codice Java durante la compilazione. All'interno di Processing sono state aggiunte alcune API grafiche per semplificare il linguaggio all'utente finale.

L'interfaccia di Processing è molto minimale. Fornisce un editor di testo dove l'utente può scrivere il suo codice, un compiler e una finestra dove è possibile visualizzare l'output una volta eseguito il proprio codice come mostrato in Figura 3.1a [29].

Nell'editor è disponibile la colorazione della sintassi e la compilazione automatica del codice. È inoltre possibile aggiungere dei breakpoint per poter effettuare un debug del codice, in modo da vedere chiaramente step by step come il proprio codice viene eseguito e cosa ogni linea di codice e ogni variabile rappresentano.

All'interno del programma si possono trovare anche diverse modalità con cui compilare il codice. Quella che viene selezionata di default è il Java, ma l'utente può anche compilare il suo codice tramite JavaScript, Python, R e altre modalità e linguaggi che possono essere facilmente aggiunti al proprio IDE.

Processing permette anche di esportare i propri lavori in diversi formati, così che possano essere visionati correttamente su MacOS, Linux o Windows.

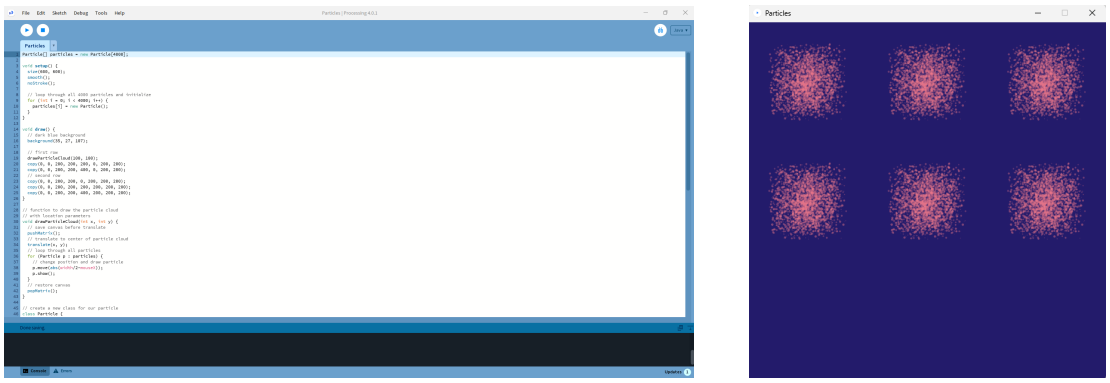
Processing non offre la capacità di modificare il proprio sketch direttamente dall'interfaccia grafica, ma solamente tramite codice. Per questo motivo, gli artisti hanno bisogno di imparare a utilizzare correttamente il codice prima di cominciare a creare le loro opere.

Allo stesso modo, non è possibile visualizzare graficamente ciò che viene creato fino a quando il codice non viene compilato. Questo significa che per visualizzare ciò che si sta costruendo, bisogna ogni volta compilare ed eseguire il codice, come mostrato in Figura 3.1.

Quando si lavora su un'animazione non esiste, all'interno di Processing, la possibilità di fermarsi su un singolo frame, o la possibilità di scorrere l'animazione un frame alla volta, in modo da vedere come essa si comporta. Allo stesso modo, non è possibile spostarsi avanti o indietro tra i vari frame dell'animazione nel caso in cui questa sia animata.

3.1.2 P5.js

P5.js è una libreria JavaScript gratuita e open-source per il creative coding rilasciata nel 2014 da Ben Fry e Casey Reas, che si concentra sul rendere il codice accessibile e inclusivo per gli artisti, i designer, i principianti e chiunque altro. Offre un'interfaccia semplice e intuitiva per creare le proprie grafiche e animazioni.



(a) La schermata principale di Processing, senza anteprima.

(b) Finestra separata con l'anteprima.

Figura 3.1: Funzionamento dell'anteprima dello sketch in Processing.

Essendo basato su Processing, p5 consente ai designer, agli artisti e agli sviluppatori di creare facilmente progetti interattivi con forme e immagini e tramite le quali l'utente può interagire.

Tramite una serie di funzioni predefinite, gli utenti possono creare facilmente diversi tipi di forme più o meno complesse con la possibilità di manipolarle, rendendo p5 uno strumento molto versatile anche per la creazione di progetti interattivi.

Come Processing, anche p5 ha alle sue spalle una grande comunità di sviluppatori, i quali tengono sempre aggiornata la piattaforma, rendendola sempre più intuitiva e inclusiva verso chi si approccia per la prima volta al creative coding, senza dimenticarsi degli utenti più esperti, i quali, grazie alla potenza di questo strumento, possono sempre trovare nuove sfide e produrre opere più complesse.

Inoltre, nonostante in questo caso siano presenti in numero minore, la documentazione e gli esempi messi a disposizione rendono il tutto più semplice da imparare per gli artisti che si approcciano per le prime volte a questo editor. Questo può aiutare anche chi ha più esperienza nella creazione di grafiche più complesse.

Grazie a p5, non si è limitati alla creazione di semplici disegni o animazioni, ma è anche possibile pensare a tutta la pagina del browser come un vero e proprio sketch, con la possibilità, tramite HTML5, di aggiungere anche testi, campi di input, suoni, video o interagire con la webcam dell'utente [30].

P5, oltre a mettere a disposizione il suo editor online rilasciato nel 2018, in modo che sia facilmente accessibile a chiunque abbia un dispositivo con una connessione a Internet offrendo la possibilità di non scaricare nulla sul proprio computer e rendendo la piattaforma accessibile da un maggior numero di dispositivi rispetto a Processing, mette anche a disposizione la sua libreria che può essere facilmente scaricata per creare progetti direttamente sul proprio dispositivo, rendendolo ancora

più versatile e flessibile. In questo modo, p5 è utilizzabile anche all'interno di un diverso editor, e il codice poi verrà compilato attraverso la sua libreria.

Uno sketch di p5 è composto da alcuni differenti linguaggi messi insieme. L'HTML collega tutti gli elementi che si trovano sulla pagina, il JavaScript permette di creare alcune grafiche interattive che verranno poi mostrate sulla pagina HTML e, infine, il CSS può essere usato per personalizzare alcuni componenti che si trovano sulla pagina come input di testo o caricamenti di file, rendendo l'opera finale più interattiva rispetto a Processing, che permette di utilizzare solamente mouse e tastiera.

L'interfaccia dell'editor online di p5 è costituita da un editor di testo, che già propone uno sketch basico da cui partire, una console dove è possibile visualizzare eventuali errori o alcune stampe che si vogliono effettuare all'interno del programma e una zona di anteprima, dove è possibile visualizzare il proprio sketch una volta compilato ed eseguito come mostrato in Figura 3.2.

All'interno dell'editor di testo è anche disponibile la colorazione della sintassi per rendere ancora più semplice e intuitivo comprendere il codice che viene scritto.

In p5.js l'anteprima grafica del proprio progetto viene sempre visualizzata sullo schermo e può anche essere aggiornata automaticamente tramite l'auto-refresh ogni volta che viene fatta una modifica sul codice, così da vedere in tempo reale come lo sketch, statico o animato, si evolve, come mostrato in Figura 3.2.

Inoltre, l'editor di p5 permette la creazione di un proprio account, dove possono essere salvati i vari sketch creati, che possono anche essere raccolti in una o più collezioni. Successivamente, i singoli sketch o le collezioni possono essere condivisi all'esterno tramite un link generato dall'editor, in modo da renderle accessibili anche ad altri utenti, che possono visualizzare il codice e, volendo, crearne una copia così da modificarla.

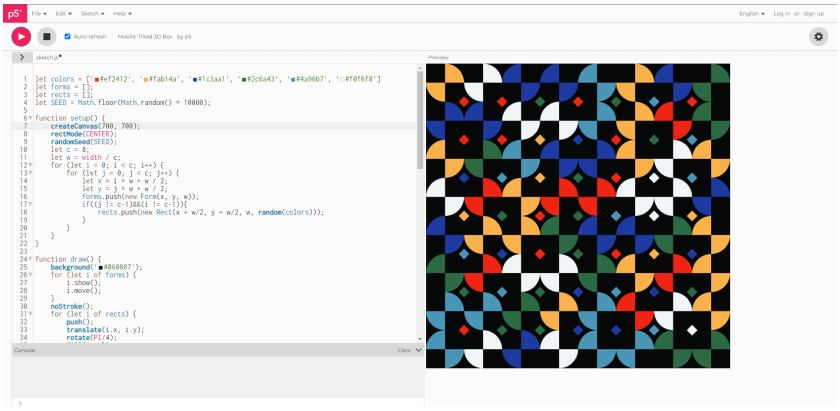


Figura 3.2: La schermata principale di p5.js, con l'anteprima a destra del codice.

Differentemente da Processing, p5 non mette a disposizione il debug del codice,

limitandosi a segnalare gli errori presente nello sketch all'interno della console. In più, in caso di sketch con grafiche molto pesanti, l'efficenza dell'editor potrebbe risentirne, in quanto la sua potenza è vincolata al browser utilizzato per aprire l'editor. Infine, è più complicato, rispetto a Processing, leggere file da computer o interfacciarsi con la telecamera.

3.1.3 Differenze tra Processing e p5.js

In questa sezione verranno riassunte le principali differenze tra Processing e p5.

Prima di tutto, il linguaggio di programmazione all'interno dei due programmi è differente. Processing utilizza il Java, mentre p5 JavaScript. Questo significa che il secondo può essere utilizzato in un ambiente web, mentre il primo richiede di essere scaricato.

Proprio grazie al linguaggio che utilizza, Processing può essere integrato facilmente con altri linguaggi come Python e Ruby, mentre per p5 questo risulta più complicato.

Per quanto riguarda le performance, quelle di Processing sono molto elevate e questo ha una grande potenza alle sue spalle. P5, invece, essendo limitato alle prestazioni del browser potrebbe non essere in grado di gestire allo stesso modo progetti molto complessi o molto grandi. Questo viene compensato dal fatto che, per un nuovo utente, potrebbe essere più facile imparare il JavaScript e utilizzare p5, in quanto questo linguaggio è meno rigido riguardo gli errori che l'utente potrebbe commettere.

Infine, la comunità di Processing, esistendo da più anni, è più affermata e presente. Negli ultimi anni però anche quella di p5 si sta ingrandendo, anche grazie alla sua facilità d'uso e alla sua accessibilità, soprattutto per gli utenti che non hanno esperienza nella programmazione.

3.1.4 Quale scegliere

Fra i due programmi, si è deciso di implementare le nuove funzionalità all'interno dell'editor di p5, successivamente rinominato p5$^+$, in quanto è il più recente e aggiornato fra i due ed è possibile utilizzarlo all'interno di un maggiore numero di dispositivi, senza dover scaricare nessuna applicazione, differentemente da Processing. Non serve infatti nessun componente aggiuntivo per poter creare una propria opera d'arte. In più, essendo p5 creato tramite tecnologie web, anche l'opera finale creata dall'artista può essere visualizzata su un qualsiasi dispositivo.

Infine, sempre più artisti utilizzano il JavaScript per creare e condividere le proprie opere; è possibile notarlo attraverso community molto attive come OpenProcessing, dove gli artisti possono condividere le proprie opere scritte in JavaScript.

3.2 Alternative e scelte progettuali

Per andare ad esplorare i principali problemi riscontrati dagli artisti all'interno di Processing e p5, si è scelto di non condurre una fase di interviste, ma di compiere una ricerca all'interno della bibliografia e dei forum visitati quotidianamente da artisti che utilizzano il creative coding per creare le loro opere.

Analizzando la bibliografia legata al creative coding, approfondita nei Capitoli 2.3 e 2.4, insieme all'analisi di Processing e p5, effettuata nella sezione precedente, sono state riscontrate alcune problematiche poste dagli artisti all'interno di interviste o articoli.

Una delle alternative che potrebbero essere implementate all'interno dell'editor di p5 è quella di aggiungere la possibilità di vedere come vengono modificate le variabili all'interno del proprio progetto mentre questo viene eseguito, associando ogni frame dell'animazione ai valori che ha ogni variabile in quel preciso momento, come evidenziato all'interno dello studio di Hoffswell et al. [16] e all'interno del lavoro di Victor [14]. Questo meccanismo viene presentato per aiutare gli utenti meno esperti a comprendere meglio il codice che viene eseguito, con la possibilità di vedere come questo si evolve, e con la possibilità di comparare un valore con il precedente o successivo. Il suo svantaggio, però, potrebbe essere quello di appesantire molto l'interfaccia visualizzata dall'utente, riempiendo l'editor e rendendolo più complicato da comprendere per i meno esperti. Questo potrebbe rendere il tutto meno intuitivo.

Per risolvere i problemi evidenziati nelle interviste all'interno dello studio di Li et al. [13] dove un artista, parlando delle sue opere, ha dichiarato come salvasse in continuazione screenshot di ogni frame, in modo tale da poter vedere come l'animazione si sarebbe comportata in ogni passaggio, si potrebbe introdurre una sezione all'interno dell'applicazione, dove l'artista può vedere singolarmente tutti i frame che compongono l'opera, con la possibilità di scaricarli, così da evitare di creare manualmente screenshot per ogni frame dell'opera stessa.

Un altro problema rinvenuto dagli artisti all'interno di Processing, è quello di dover sempre compilare ed eseguire il codice per vedere l'anteprima della propria opera e come essa si comporti, in quanto non esiste la possibilità di poter vedere in automatico nell'anteprima le modifiche fatte sul codice. Questa esigenza viene già risolta all'interno dell'editor di p5, in quanto è già presente la possibilità di vedere sulla stessa finestra del codice l'anteprima del proprio sketch, e, inoltre, tramite l'opzione *auto-refresh*, è possibile andare ad aggiornare automaticamente l'anteprima dell'opera quando il codice viene modificato, senza doverlo fare ogni volta manualmente. All'interno dell'editor è anche presente un color picker, attivabile tramite una sequenza di tasti.

Un'alternativa ritenuta utile, che già è presente come estensione all'interno di Processing, è TweakMode [17]. Al momento, l'unico svantaggio all'interno di

Processing è che l'anteprima viene comunque aperta su una finestra diversa rispetto a quella dove si trova il codice, con il rischio che questa venga messa in secondo piano o che venga persa dall'utente, se ci sono molti programmi aperti. Il grande vantaggio di questa modalità è il poter visualizzare l'anteprima nella stessa finestra del codice e la possibilità di avere l'output visivo immediatamente aggiornato dopo un cambiamento.

Un'ulteriore mancanza che hanno Processing e p5, riscontrata da un piccolo numero di artisti, è quella di non poter modificare direttamente l'output grafico, dovendo quindi imparare a utilizzare correttamente il codice prima di creare la propria opera. Questo è stato riscontrato principalmente da artisti inesperti, in quanto modificare la parte grafica può essere utile per creare opere statiche e basilari, mentre per renderle animate e più complesse bisogna lavorare sulla parte di codice. Per questo motivo, si è deciso di non approfondire quest'alternativa progettuale.

Come evidenziato anche all'interno del lavoro di Li et al. [13], per evitare di ricominciare ogni nuovo progetto da zero, potrebbe essere utile per gli utenti anche l'aggiunta di alcuni template selezionabili. L'artista potrebbe decidere di utilizzare alcuni suoi progetti o progetti di altri utenti come base per la sua prossima opera. In questo modo, una volta aggiunto un processo ai template, l'utente potrebbe facilmente richiamarlo per andarlo ad aggiungere all'interno di quello attuale, senza ricopiare manualmente tutto il codice. Con un basso livello di astrazione, l'utente potrebbe andare ad analizzare il codice inserito, in modo da comprenderlo e poterlo modificare, così da aumentare la propria conoscenza. Al contrario, con un alto livello di astrazione, l'artista non avrebbe modo di analizzare e studiare il codice che genera l'animazione finale.

Alcuni utenti hanno anche richiesto la possibilità di aggiungere dei tag a ogni sketch, in modo da poterli caratterizzare e ritrovarli facilmente tra gli altri attraverso la ricerca. In questo modo il nome dello sketch può essere breve e una descrizione di ciò che ha all'interno può essere inclusa attraverso i tag [31].

Tramite lo studio condotto da Li et al. [20], si è visto come alcuni artisti compiono piccoli cambiamenti nella loro opera e poi ridisegnano il tutto un pezzo alla volta per capire meglio come il programma si comporta. All'interno di Processing e p5 non è presente una lista delle modifiche effettuate all'interno di un progetto. Non c'è, infatti, un modo per poter tenere traccia di tutti i cambiamenti di uno sketch, con tutte le sue versioni, in modo da poter avere uno storico di quando il progetto è stato modificato e di com'era precedentemente. Potrebbe quindi essere utile aggiungere una funzionalità che permetta di poter vedere tutte le modifiche effettuate ad uno sketch e ripristinare una vecchia modifica. In più, si potrebbe anche aggiungere la possibilità di creare una copia di una vecchia modifica, così da non perdere il lavoro attuale, ma poter allo stesso tempo lavorare anche su una vecchia versione dell'opera. OpenProcessing, piattaforma che permette agli artisti

che utilizzano creative coding di condividere le proprie opere e di poter visualizzare sia codice che output delle opere di altri utenti, al suo interno implementa un meccanismo che permette di vedere tutte le versioni di una determinata opera, tenendo traccia di tutte le sue versioni create da diversi utenti o da un singolo utente. All'interno della piattaforma, questa funzionalità è molto apprezzata.

Si è deciso, infine, fra tutte le problematiche evidenziate, di implementare all'interno dell'editor di p5$^+$ la maggior parte di esse, dando la precedenza a quelle più riscontrate dagli artisti che utilizzano quotidianamente queste piattaforme, in modo che creare animazioni e visualizzazioni grafiche possa diventare il più semplice e intuitivo possibile, sia per i nuovi utenti che per quelli più esperti.

Per questo motivo, si è deciso di implementare una funzionalità per poter importare facilmente uno sketch all'interno di un altro, senza dover manualmente ricopiare ogni riga. Questa funzione va anche a risolvere in parte l'esigenza dei template selezionabili. Successivamente, gli sketch riutilizzati all'interno di nuovi progetti verranno definiti *Building Blocks*.

La seconda funzionalità che si è ritenuta utile da aggiungere riguarda la visualizzazione di tutti i frame di uno sketch in un apposito pannello, così che l'utente possa facilmente tenere traccia di tutte le modifiche effettuate.

Successivamente, si è deciso di implementare la possibilità di poter vedere le definizioni delle singole funzioni utilizzate all'interno dello sketch, così da poter rendere il linguaggio trasparente.

Un'altra funzionalità che si è deciso di aggiungere riguarda i tag, che sono stati richiesti da diversi utenti e sono stati ritenuti utili per poter caratterizzare uno sketch con precisione, per poi ritrovarlo facilmente in mezzo agli altri.

Infine, si è scelto di implementare la funzionalità delle ultime modifiche, così che un utente possa avere uno storico dei cambiamenti fatti ad un singolo sketch e possa facilmente ripristinare una vecchia versione di questo.

3.3 Mockup

Per risolvere i problemi riscontrati durante lo studio della bibliografia e delle interviste, sono stati sviluppati alcuni mockup aggiungendo delle nuove funzionalità all'editor di p5, cercando di mantenere la sua interfaccia semplice e intuitiva. Per lo sviluppo dei mockup è stato scelto Figma[32] per la facilità di utilizzo e per gli strumenti offerti.

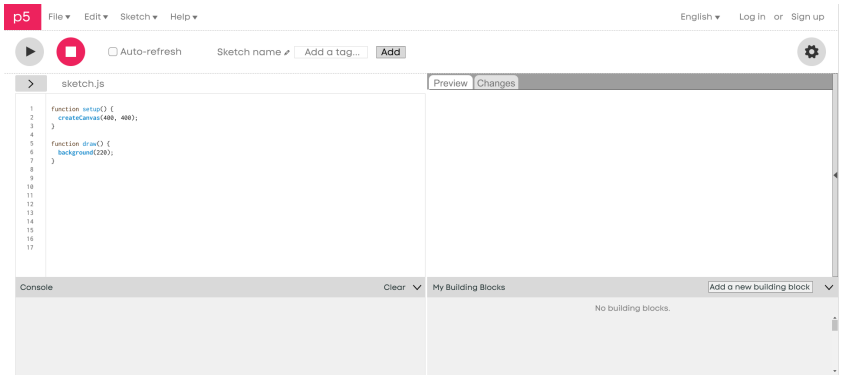
3.3.1 Building Blocks

La prima funzionalità che si è deciso di aggiungere è quella dei Building Blocks. Nata dall'esigenza di poter riutilizzare le proprie opere o quelle di altri come punto di partenza per un nuovo progetto, i Building Blocks permettono di importare

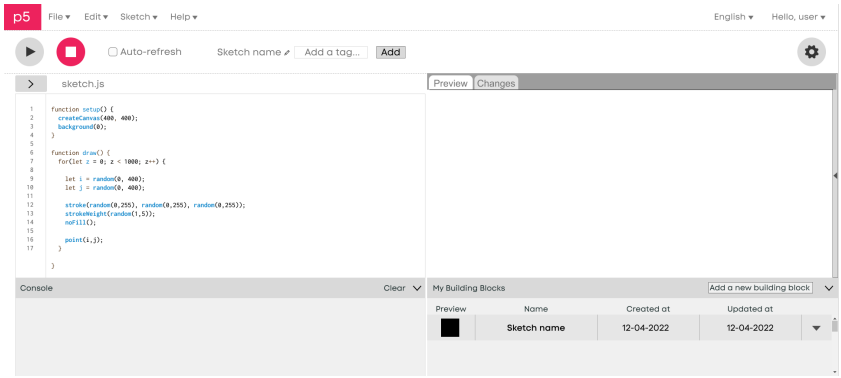
un progetto all'interno di un altro, senza dover manualmente ricopiare tutte le funzioni.

Sono stati definiti come Building Blocks per far sì che l'artista si focalizzi non tanto sulla parte di codice, ma sull'animazione finale che viene creata. Il Building Block può essere visto come un blocco di uno sketch, statico o animato, che da solo o insieme ad altri va a comporre l'opera finale.

Il mockup dei Building Blocks è mostrato in Figura 3.3. Si è deciso di aggiungere la nuova sezione in basso a destra sotto l'anteprima, riutilizzando lo stile della console che si trova sulla sinistra, in modo da non stravolgere l'interfaccia grafica. Prima che l'utente effettui il log in non viene mostrato nessun Building Block e questi non possono essere aggiunti finché l'utente non sarà autenticato. Una volta autenticato, potrà vedere la lista degli sketch salvati come Building Block e potrà gestirli aggiungendone altri o rimuovendo i presenti.



(a) L'utente non ha effettuato il log in.



(b) L'utente autenticato può gestire i suoi Building Blocks.

Figura 3.3: Mockup dell'editor di p5 con l'aggiunta dei Building Blocks.

Si è deciso di inserire i Building Block come una collezione che non può essere eliminata, riutilizzando la funzionalità delle collezioni già presenti all'interno

dell'editor di p5.

Accanto al nome dello sketch presente nei Building Block si è deciso di mostrare la sua anteprima, così che l'utente possa subito capire cosa contiene, insieme alla data di creazione e alla data di ultima modifica. In più, l'utente può gestire il singolo blocco attraverso alcune opzioni contenute in un menu a scomparsa presente sulla destra.

Aggiungendolo allo sketch attuale, verranno create delle nuove funzioni richiamate all'interno delle funzioni `setup` e `draw`, in modo da non sovrascrivere il codice che l'utente può aver scritto fino a quel momento.

3.3.2 Visualizzazione dei Frame

Per permettere agli artisti di poter vedere, un frame alla volta, come si evolve l'opera da loro creata, si è deciso di implementare una nuova funzionalità per poter visualizzare tutti i vari frame dell'animazione.

La decisione è stata quella di inserire i vari frame in un pannello a scomparsa sulla destra, così da non occupare costantemente una parte dello schermo, come mostrato in figura 3.4. All'interno, per ogni frame, è possibile effettuare il download, in modo da poter riutilizzare l'immagine anche all'esterno dell'editor.

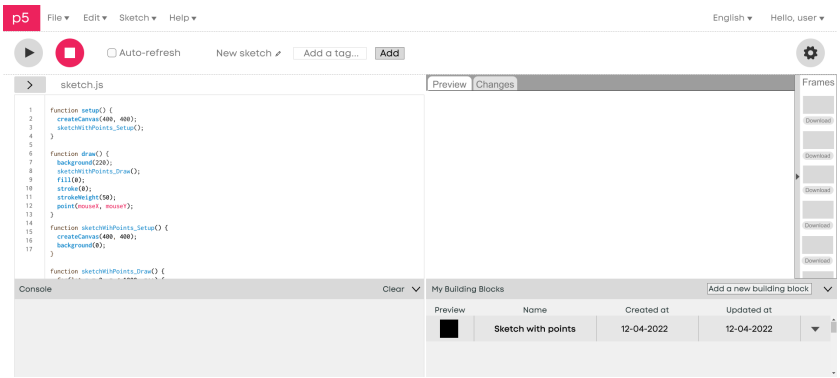


Figura 3.4: Mockup dell'editor di p5 con l'aggiunta dei Frames.

Inizialmente, la sezione sarà vuota e quando lo sketch verrà eseguito, all'interno di essa verranno mostrati tutti i frame che compongono l'animazione.

3.3.3 Linguaggio trasparente

Si è anche deciso di aggiungere la possibilità di vedere le definizioni di ogni istruzione, in modo da rendere il linguaggio trasparente e più semplice da capire per gli artisti che lo utilizzano. In questo modo, abilitando la funzionalità tramite le impostazioni,

l'utente ha la possibilità, cliccando su una determinata istruzione, di vedere la sua definizione in modo da poter comprenderla meglio, come mostrato in Figura 3.5.

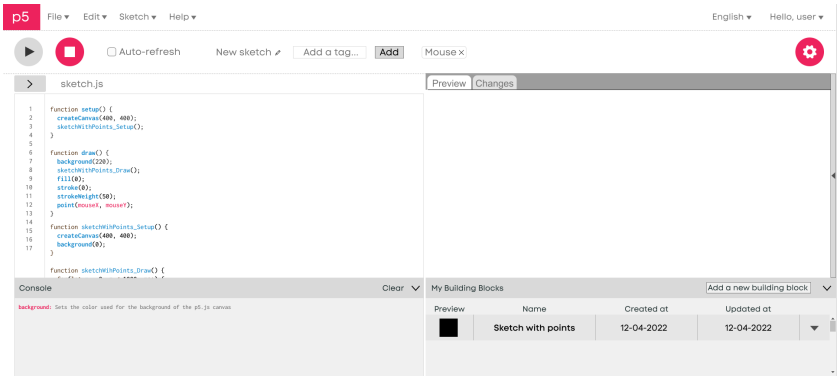


Figura 3.5: Mockup dell'editor di p5 con l'aggiunta delle definizioni.

Inizialmente, si è deciso di visualizzare la definizione a destra dell'istruzione all'interno dell'editor, per poi decidere di spostarla all'interno della console, in quanto questa può essere molto lunga e occupare molto spazio.

3.3.4 Tag

Un'altra aggiunta fatta all'editor di p5 è quella dei tag, in modo che l'utente possa andare a caratterizzare meglio la propria opera, non limitandosi al nome.

In questo modo, anche attraverso la ricerca, diventa più facile andare a trovare il proprio sketch, utilizzando anche le parole chiave inserite nei tag.

Si è deciso di gestire i tag all'interno dello sketch, sulla parte superiore dello schermo, come mostrato in Figura 3.6. Ogni tag aggiunto dall'utente può essere facilmente rimosso.

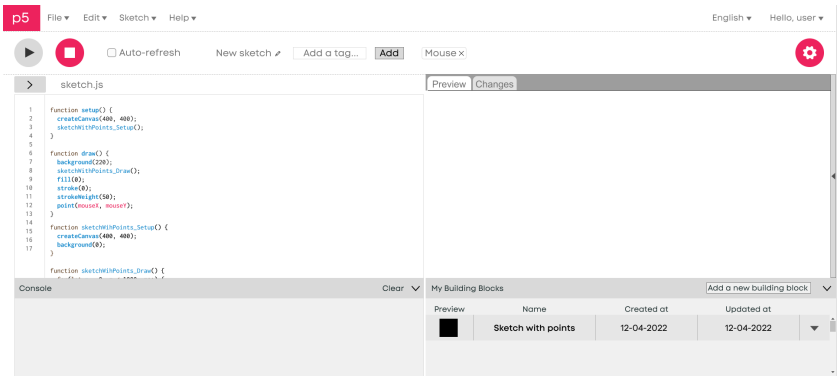


Figura 3.6: Mockup dell'editor di p5 con l'aggiunta dei tag.

3.3.5 Ultime modifiche

L'ultima funzionalità aggiunta all'editor di p5 è quella di gestione delle ultime modifiche. Questa sezione è stata aggiunta accanto all'anteprima, in modo da essere ben visibile sullo schermo quando viene aperta, come mostrato in Figura 3.7. Si è deciso, quando questa sezione viene visualizzata, di non mostrare l'anteprima dello sketch, in quanto si è considerata poco utile durante la visualizzazione e la gestione delle ultime modifiche dello sketch.

Si è anche deciso, per ogni modifica, di visualizzarne l'anteprima, così che l'artista possa immediatamente capire cosa era presente in quel preciso punto dello sketch, guardando cosa era stato prodotto.

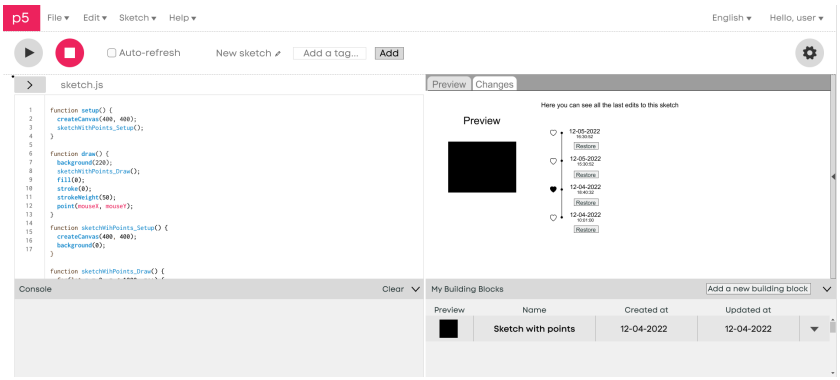


Figura 3.7: Mockup dell'editor di p5 con l'aggiunta delle ultime modifiche.

Ogni volta che lo sketch viene modificato, verrà aggiunta sulla lista una nuova modifica, in modo che l'utente possa tener traccia di tutti i cambiamenti effettuati all'interno della propria opera, ordinati per data e ora in cui la modifica è stata effettuata partendo dalla più vecchia che si trova in basso nello schermo.

È stata anche inserita la possibilità di aggiungere le modifiche ai preferiti. In questo modo, è possibile salvare una copia della modifica selezionata per poterci lavorare in modo parallelo al lavoro effettuato fino a quel momento sullo sketch principale, senza perdere nessun progresso.

Infine, si è deciso di aggiungere la possibilità di poter ripristinare una vecchia modifica, perdendo quella attuale, che potrà essere ritrovata all'interno della lista delle ultime modifiche, così da poter sovrascrivere la propria opera e ricominciare a lavorare da una versione precedente. Questo può essere utile se l'artista decide di abbandonare il lavoro fatto fino a quel momento per ricominciare da una vecchia modifica e muoversi in una direzione diversa da quella presa precedentemente.

Capitolo 4

Implementazione

In questo capitolo verrà analizzata l'implementazione delle nuove funzionalità presentate nel Capitolo 3.

Inizialmente vengono analizzate le principali tecnologie utilizzate, per poi approfondire la struttura dell'applicazione attuale, concentrandosi sulla struttura del frontend e del backend. Successivamente, verranno analizzate le nuove funzionalità implementate e, per ognuna di esse, verranno mostrate e commentate le parti più significative del codice sviluppato.

4.1 Tecnologie utililizzate

All'interno di questa sezione verranno approfondite tutte le tecnologie utilizzate per lo sviluppo delle nuove funzionalità all'interno dell'editor di p5.

È stato utilizzato React per il frontend, express per la gestione del routing e per il backend, MongoDB come database e Docker per far partire e gestire i diversi container.

4.1.1 React

React è una libreria JavaScript open-source per lo sviluppo di applicazioni web. È stata creata nel 2011 e viene utilizzata per la creazione di interfacce utente interattive e dinamiche [33]. Si basa su un paradigma di programmazione chiamato *programmazione dichiarativa*, che consente agli sviluppatori di descrivere come dovrebbe essere la UI in base allo stato dell'applicazione, piuttosto che descrivere come modificarla. Questo rende la creazione e la gestione più facile e prevedibile. In più, per quanto riguarda il flusso di dati, questi sono gestiti in modo unidirezionale, così che lo stato venga gestito sempre dal padre e passato ai figli tramite proprietà, chiamate *props*, che non possono essere modificate dai figli stessi. Per andare a

creare l'interfaccia utente, viene utilizzato un sistema di componenti riutilizzabili, così da poter creare un sistema modulare e scalabile. Nella scrittura del codice, React utilizza il JSX che permette di avere una creazione dei componenti facilitata rispetto ad altri linguaggi come l'HTML, che possono essere definiti tramite classi o funzioni. Questo codice, al momento della compilazione, verrà trasformato in codice JavaScript interpretabile da React.

Per la gestione dell'interfaccia utente, viene utilizzato il Virtual DOM, che consente a React di aggiornare solamente le parti della UI che sono state modificate, senza dover ricostruire ogni volta tutta la pagina.

Dalla versione 16.8.0, React introduce gli *hooks* per risolvere alcuni problemi. Prima di tutto, React non offre un modo facile per avere comportamenti riutilizzabili all'interno di un componente. Per questo genere di comportamenti bisogna ristrutturare il codice, rendendolo più complicato da mantenere e gestire. Con gli hooks viene messo a disposizione un modo semplice per andare a testare e riutilizzare la logica di un singolo componente.

Un altro problema è la gestione del ciclo di vita di un componente, in quanto all'interno di alcuni metodi possono andare a mischiarsi i dati dell'applicazione con la sua logica. In alcuni casi non è possibile andare a separare queste parti in componenti diversi e, quando è possibile, i componenti possono diventare più difficili da essere riutilizzati e da mantenere. Per risolvere questo problema, gli hooks permettono di dividere un singolo componente in componenti più piccoli a seconda di cosa vanno a gestire al loro interno.

L'ultimo problema è legato alle classi, che possono essere complicate da imparare. In più, i componenti di React sono sempre stati più vicini alle funzioni che alle classi per la loro struttura interna. Per questo motivo, con gli hooks è possibile utilizzare meglio le funzionalità di React senza utilizzare le classi.

Due degli hooks più utilizzati sono `useState` e `useEffect`. Il primo viene utilizzato per salvare lo stato del componente all'interno di una variabile. Essendo un hook, non può essere dichiarato all'interno di cicli o condizioni. Tramite questo hook, lo stato può essere richiamato all'interno del componente e può essere facilmente aggiornato. Un esempio del suo uso è riportato nel Listato 4.1, dove, tramite lo stato, viene salvato e aggiornato il valore di un contatore.

```
1  import { useState } from 'react';
2
3  export default function Counter() {
4    const [count, setCount] = useState(0);
5
6    function handleClick() {
7      setCount(count + 1);
8    }
9
10   return (
```

```
11      <button onClick={handleClick}>
12        You pressed me {count} times
13      </button>
14    );
15  }
```

Listing 4.1: Esempio dell'hook `useState`

L'hook `useEffect`, invece, permette di sincronizzare il componente con degli effetti esterni che non dipendono dall'evoluzione del componente stesso, come la connessione ad un server esterno. In questo caso l'hook conterrà una funzione di setup per connettersi al sistema esterno, una funzione di cleanup e una lista di dipendenze. Quando il componente viene aggiunto alla pagina, partirà la funzione di setup. Successivamente, ogni volta che il componente viene ricaricato, se le sue dipendenze sono state modificate, verrà prima eseguito il codice di cleanup con le vecchie proprietà e successivamente il codice di setup con le nuove proprietà. Quando il componente viene rimosso dalla pagina, verrà eseguito nuovamente il codice di cleanup. Un esempio può essere visto nel listato 4.2.

```
1  import { useEffect } from 'react';
2
3  function ChatRoom({ roomId }) {
4    const [serverUrl, setServerUrl] = useState('https://localhost
       :1234');
5
6    useEffect(() => {
7      const connection = createConnection(serverUrl, roomId);
8      connection.connect();
9      return () => {
10       connection.disconnect();
11     };
12   }, [serverUrl, roomId]);
13   // ...
14 }
```

Listing 4.2: Esempio dell'hook `useEffect`

4.1.2 Express

Express è un web framework per Node.js, che permette di creare applicazioni web e API RESTful [34]. Fornisce alcune funzionalità per le applicazioni web come routing, middleware, gestione degli errori e gestione delle richieste http. Express offre una gestione dettagliata degli errori, così da poter fornire risposte puntuali e precise agli utenti. Un esempio del suo utilizzo può essere visto nel Listato 4.3, dove il server viene avviato e resta in ascolto sulla porta 3000.

```
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.listen(port, () => {
10   console.log(`Example app listening on port ${port}`)
11 })
```

Listing 4.3: Configurazione di un server `Express`

Le funzioni middleware consentono di gestire in modo flessibile le richieste http, e possono essere utilizzate per autenticazione, logging, gestione delle sessioni e molto altro, potendo accedere all'oggetto richiesta `req` e all'oggetto risposta `res`. Tramite il routing possono essere facilmente gestiti diversi endpoint dell'applicazione, indicando il metodo di richiesta specifico per ogni pagina. Al loro interno è anche possibile inserire funzioni di middleware.

4.1.3 MongoDB

MongoDB è un database NoSQL open-source, che si basa su un modello orientato sui documenti. È stato creato per gestire grandi quantità di dati in modo scalabile e flessibile [35].

MongoDB memorizza i dati in documenti flessibili JSON-like, dove i campi possono variare da un documento all'altro, così da avere una struttura dati variabile. I documenti vengono salvati all'interno di collezioni; queste contengono al loro interno tutti i documenti che sono correlati fra loro. Le collezioni sono poi raggruppate all'interno di un database, che può contenere una o più collezioni di documenti. La struttura del documento consente approcci differenti quando vengono salvati dei dati.

Al suo interno supporta la replica dei dati, così da avere più copie garantendone la loro ridondanza e disponibilità. Supporta anche l'indicizzazione dei dati, così che si possa accedere ad essi in modo rapido ed efficiente. Tutte le operazioni di scrittura all'interno di MongoDB sono atomiche al livello del singolo documento. Quando una singola operazione di scrittura va a modificare più di un documento, ogni singola modifica è atomica, ma l'operazione non lo è. Dalle ultime versioni, MongoDB supporta le transazioni multi-documento, garantendole anche sulle repliche.

All'interno dell'editor viene usato per la gestione degli utenti: in particolar modo, si occupa della gestione degli sketch e delle collezioni.

4.1.4 Docker

Creare servizi distribuiti può richiedere diverse applicazioni software, come per esempio, nel caso di p5$^+$, un DBMS e un server per l'applicazione. Ogni singolo componente richiede una specifica configurazione e ognuno di essi potrebbe aver bisogno di una diversa macchina fisica richiedendo un maggior numero di memoria o macchine dedicate se ci sono molte richieste. Per fronteggiare questo tipo di sfida, sono state proposte alcune soluzioni, tra cui l'uso di container, per avere le diverse applicazioni su una singola macchina. I container si appoggiano sullo stesso kernel della macchina che li ospita, avendo però nomi differenti. Ogni container viene completamente isolato dagli altri container che si trovano sullo stesso host rappresentando per il sistema operativo un gruppo isolato di processi. Il filesystem di ogni container è isolato dagli altri. Con i container, è possibile avere una grande scalabilità e portabilità.

Docker è una tecnologia open-source che supporta la creazione e distribuzione dei container [36]. È composto da quattro principali componenti: Docker Engine, Docker Client, Docker Images e Docker Containers.

Il Docker Engine è il processo che controlla l'esecuzione dei container e rappresenta il core della piattaforma. Il Docker Client permette l'accesso alle funzionalità offerte dalle API. Il Docker Images rappresenta le immagini, che sono dei template in sola lettura con delle istruzioni per creare un container in Docker. Il DockerFile è il file che contiene le istruzioni per creare una nuova immagine partendo da una esistente. Un container è quindi un'istanza di un'immagine che può essere fatta partire.

Un possibile esempio di DockerFile viene mostrato nel Listato 4.4, dove è presente la configurazione dell'editor di p5$^+$, dove vengono esposte la porta 8000 e la porta 8002. La prima andrà a gestire tutta l'applicazione tramite la quale interagisce l'utente, mentre la seconda gestirà tutte le anteprime degli sketch generati tramite l'editor.

```
1  FROM node:16.14.2 as base
2
3  ENV APP_HOME=/usr/src/app \
4    TERM=xterm
5  RUN mkdir -p $APP_HOME
6  WORKDIR $APP_HOME
7  EXPOSE 8000
8  EXPOSE 8002
9
10 FROM base as development
11 ENV NODE_ENV development
12 COPY package.json package-lock.json ./
13 RUN npm install
14 RUN npm rebuild node-sass
15 COPY .babelrc index.js nodemon.json ./
```

```
16 COPY ./webpack ./webpack
17 COPY client ./client
18 COPY server ./server
19 COPY translations/locales ./translations/locales
20 COPY public ./public
21 CMD ["npm", "start"]
22
23 FROM development as build
24 ENV NODE_ENV production
25 RUN npm run build
26
27 FROM base as production
28 ENV NODE_ENV=production
29 COPY package.json package-lock.json index.js ./
30 RUN npm install --production
31 RUN npm rebuild node-sass
32 COPY --from=build $APP_HOME/dist ./dist
33 CMD ["npm", "run", "start:prod"]
```

Listing 4.4: Esempio di `DockerFile`

4.2 Struttura dell'applicazione

Lavorando su un progetto già esistente, nel quale collaborano diversi sviluppatori, la struttura dell'applicazione è già stata precedentemente definita, in modo da renderla il più modulare e scalabile possibile.

Le principali cartelle presenti nel progetto sono quella del client e quella del server, in cui all'interno sono rispettivamente inseriti il frontend e il backend dell'editor di p5.

4.2.1 Frontend

Il frontend è organizzato utilizzando Redux. Redux è una libreria di gestione dello stato per applicazioni JavaScript. Si basa sul concetto di avere uno stato globale, il quale può essere accessibile e modificabile da qualsiasi parte dell'applicazione [37].

La gestione dello stato in Redux avviene tramite tre concetti principali: lo stato, le azioni e i riduttori.

Lo stato è un oggetto o un array JavaScript immutabile che rappresenta l'intero stato dell'applicazione. Tutti i dati che devono essere condivisi tra i diversi componenti vengono salvati nello stato. Gli stati non possono essere modificati direttamente, ma solo attraverso le azioni.

Le azioni sono semplici oggetti JavaScript che descrivono ciò che è successo nell'applicazione. Ogni azione deve avere una proprietà `type` che indica il tipo di

azione che si sta eseguendo. Possono anche contenere altre proprietà che descrivono i dati associati all'azione. Normalmente tutti i dati in più vengono inseriti in un campo chiamato `action.payload`.

I riduttori sono le funzioni che descrivono come lo stato dell'applicazione cambia in risposta alle azioni. Ciascun riduttore è associato ad un particolare tipo di azione e riceve l'azione e lo stato corrente come parametri. La funzione riduttrice deve quindi restituire il nuovo stato in base all'azione eseguita.

In pratica, quando viene eseguita un'azione nell'applicazione, questa viene inviata a tutti i riduttori che ne hanno il tipo associato. Ogni riduttore analizza l'azione e aggiorna lo stato in base alla logica applicata. Il nuovo stato viene quindi restituito e sostituisce lo stato precedente. All'interno di un riduttore non può essere presente logica asincrona.

Insieme a Redux, all'interno del progetto, viene inserita la libreria `reselect`, che fornisce una sintassi dichiarativa per l'accesso e la manipolazione dello stato in Redux [38]. Consente di memorizzare in cache i risultati dei selectors per migliorare le prestazioni dell'applicazione. Ogni volta che lo stato viene modificato, tutti i selectors vengono ricalcolate. Ciò può comportare un aumento del carico di lavoro per l'applicazione, poiché i selectors potrebbero essere complesse o richiedere molte elaborazioni. La libreria consente di evitare questo overhead tramite la memorizzazione nella cache dei risultati dei selectors, in modo che vengano ricalcolati solo quando lo stato dell'applicazione cambia.

I file dedicati al frontend sono all'interno della cartella client, dove è presente un'ulteriore suddivisione in sottocartelle, tra cui *common* che contiene componenti riutilizzabili come i pulsanti, *components* che contiene header e footer, *styles* che contiene gli stili e *modules* che contiene tutto il resto dell'applicazione. Modules, al suo interno, presenta un'ulteriore suddivisione: contiene la cartella *preview* dove al suo interno è presente tutta la gestione della preview dello sketch che in locale viene gestita sulla porta 8002, la cartella *user* che contiene tutte le pagine riguardanti l'utente, come la pagina di login, di registrazione e così via. La cartella che contiene il resto dei file utilizzati nell'interfaccia è la cartella *IDE*, che è organizzata come segue:

- `actions`: contiene le azioni per la gestione dello stato in Redux.
- `components`: contiene tutti i componenti utilizzati per creare l'interfaccia dell'editor di p5.
- `pages`: contiene le pagine principali dell'editor, come `IDEView.jsx` che si occupa di gestire al suo interno tutti i componenti utilizzati per creare la pagina principale dell'editor di p5.
- `reducers`: contiene i riduttori per la gestione dello stato in Redux.

- `selectors`: contiene i selectors per la gestione dello stato in Redux, tramite la libreria reselect.

4.2.2 Backend

Il backend è organizzato dividendo controllers, models e routes. In Figura 4.1 è mostrato come viene implementato il flusso di dati quando si gestisce una richiesta o una risposta http.

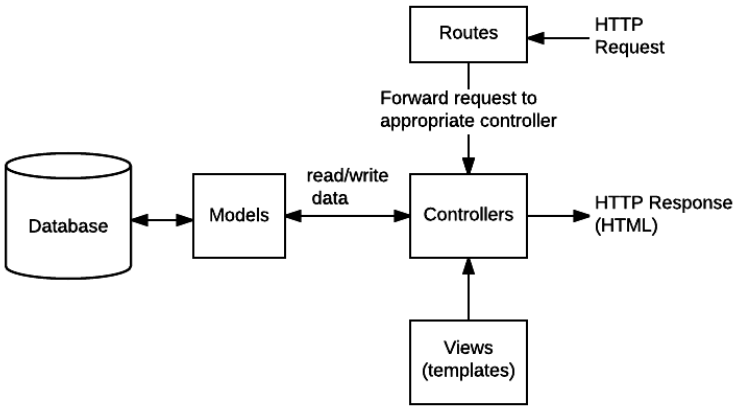


Figura 4.1: Gestione del flusso di dati in Express.

I controllers sono moduli JavaScript che contengono funzioni per gestire le richieste http. Ad esempio, un controller può gestire le richieste di una pagina di login o una richiesta di aggiornamento di un'entità nel database. I controllers possono accedere a moduli esterni come i modelli o il middleware di Express per eseguire operazioni specifiche.

I modelli, invece, rappresentano l'accesso ai dati dell'applicazione e la loro manipolazione. I modelli possono essere utilizzati per accedere a un database o a qualsiasi altra fonte di dati esterna all'applicazione. Inoltre, possono contenere la logica per la validazione dei dati inseriti dall'utente. Nel caso specifico dell'applicazione i modelli sono utilizzati per andare a definire la struttura delle collezioni presente all'interno del database MongoDB così che questo possa essere opportunatamente popolato quando viene fatta partire l'applicazione.

Infine, le routes in Express definiscono le URL e le azioni da eseguire quando una determinata richiesta viene ricevuta dal server. Una route è associata a un'istanza di un controller che si occupa della gestione della richiesta e restituisce una risposta appropriata. Le routes possono anche utilizzare middleware per eseguire operazioni

comuni come la verifica dell'autenticazione dell'utente o la manipolazione dei dati della richiesta.

4.3 Funzionamento dell'applicazione

La principale attività dell'applicazione riguarda la scrittura e la compilazione del codice, così che l'utente possa vedere graficamente, tramite l'anteprima, la sua opera. Per fare in modo che questo venga fatto, vengono utilizzati diversi componenti, che collaborano insieme per la riuscita di questo processo.

Prima di tutto, l'utente andrà a scrivere del codice all'interno dell'editor, che utilizza la libreria `CodeMirror`, tramite la quale è possibile personalizzare l'editor rendendolo semplice e intuibile da comprendere per l'utente finale. Alcune aggiunte che possono essere fatte tramite la libreria sono, per esempio, la numerazione delle righe dell'editor o la possibilità di colorare le varie parole chiave del linguaggio utilizzato.

Tramite la libreria, dopo aver definito l'editor, si può definire una particolare azione quando, per esempio, viene premuto un determinato tasto, oppure quando viene modificato il testo all'interno dell'editor. Nell'applicazione è stata utilizzata questa seconda funzionalità per fare in modo che quando il codice viene modificato, il suo stato venga correttamente aggiornato tramite un'azione, in modo che sia sempre disponibile e coerente per essere passato a un altro componente, nel caso in cui l'utente decidesse di visualizzare l'anteprima dello sketch.

```
1  this._cm.on('change',
2    debounce(() => {
3      if (this.state.isRealChange) {
4        this.props.setUnsavedChanges(true);
5        this.props.hideRuntimeErrorWarning();
6        this.props.updateFileContent(this.props.file.id, this._cm.
     getValue());
7        if (this.props.autorefresh && this.props.isPlaying) {
8          this.props.clearConsole();
9          this.props.startSketch();
10       }
11     }
12     this.setState({
13       isRealChange: true
14     });
15   }, 1000)
16 );
```

Listing 4.5: Funzionamento della modifica nell'editor.

Come è possibile vedere nel Listato 4.5, ogni volta che lo sketch viene modificato dall'utente, viene aggiornato il suo contenuto tramite l'azione `updateFileContent`

passata come props e viene aggiunta un'icona accanto al nome dello sketch per indicare che all'interno ci sono modifiche non salvate, così che, nel caso in cui l'utente decidesse di cambiare pagina prima di salvare lo sketch, venga mostrato un avviso.

A questo punto, se è stato selezionato l'auto-refresh, viene chiamata l'azione `startSketch`: la stessa azione verrà chiamata quando l'utente cliccherà il tasto Play. Una volta chiamata l'azione, questa al suo interno inizializza la console, rimuovendo eventuali errori precedenti e, tramite un messaggio di tipo `MessageTypes.SKETCH`, viene inviato un messaggio con l'intero sketch. Questo messaggio verrà catturato dalla parte dedicata alla creazione e visualizzazione la preview. All'interno di questa parte, tramite la funzione `renderSketch`, viene prodotta l'anteprima dell'animazione, che verrà poi inserita all'interno di un Frame, tramite uno `useEffect` che lo mantiene sempre aggiornato. Questo è presente nel Listato 4.6.

```
function renderSketch() {
    const doc = iframe.current;
    if (isPlaying) {
      const htmlDoc = injectLocalFiles(files, htmlFile, {
        basePath,
        gridOutput,
        textOutput
      });
      const generatedHtmlFile = {
        name: 'index.html',
        content: htmlDoc
      };
      const htmlUrl = createBlobUrl(generatedHtmlFile);
      setTimeout(() => {
        doc.src = htmlUrl;
      }, 0);
    } else {
      doc.src = '';
    }
}

useEffect(renderSketch, [files, isPlaying]);
    return (
        <Frame
          aria-label="Sketch Preview"
          role="main"
          frameBorder="0"
          ref={iframe}  />
    );
```

Listing 4.6: Funzione `renderSketch`.

La funzione `renderSketch`, tramite `injectLocalFiles` produce l'anteprima che l'utente andrà a visualizzare. Infine, la funzione `createBlobUrl` andrà a creare l'url

da inserire all'interno del Frame. Prima di produrre il blob, viene controllato se all'interno dello sketch sono presenti errori. Nel caso in cui siano presenti, l'anteprima non verrà visualizzata, ma verranno visualizzati gli errori all'interno della console, formattati in modo da essere intuitivi per l'utente finale.

Infine, una volta prodotto lo sketch, questo viene mostrato all'utente tramite il componente `previewFrame` che al suo interno presenta un Frame dove l'utente può visualizzare l'anteprima dell'opera che ha creato.

4.4 Principali funzionalità implementate

Di seguito verranno approfondite le nuove funzionalità implementate all'interno dell'editor di p5, rinominato p5$^+$. In Figura 4.2 è possibile vedere la schermata principale dell'applicazione con tutte le nuove principali aggiunte.

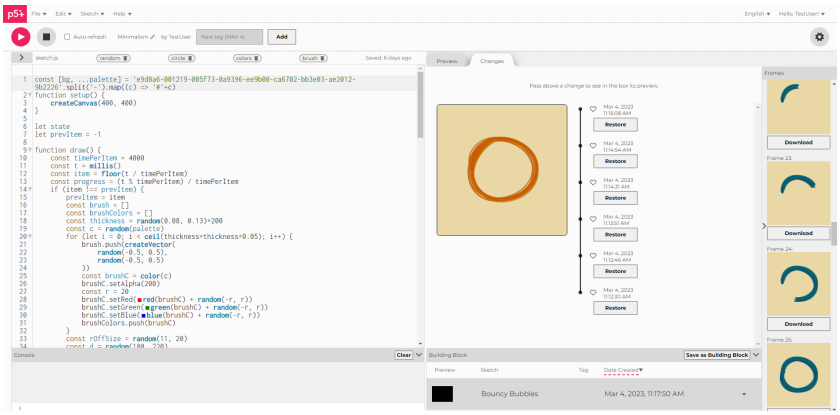


Figura 4.2: L'interfaccia di p5+.

4.4.1 Building Blocks

La prima funzionalità implementata è quella dei Building Blocks. Si è da subito pensato di creare automaticamente una nuova collezione quando l'utente si registra chiamata Building Blocks, che non possa essere eliminata. In questo modo viene riutilizzata una funzionalità già implementata all'interno dell'editor di p5. In particolare, quando uno sketch viene aggiunto o rimosso dai building block, vengono riutilizzate le stesse funzioni per l'aggiunta e la rimozione di uno sketch da una collezione.

Successivamente, per mostrare graficamente i building blocks all'interno dell'interfaccia principale, si è deciso di riutilizzare la grafica del box della console e posizionare il nuovo contenitore sulla destra dello schermo, in modo simmetrico

rispetto alla console. Al suo interno, è stato riutilizzato il componente che mostra la lista di tutti gli sketch, per cercare di mantenere l'interfaccia il più semplice possibile senza stravolgerla.

Quando l'utente si registra, viene chiamata la funzione `createCollectionInternally` per andare a creare automaticamente la collezione Building Block per il singolo utente. Questa è l'unica funzione aggiunta lato server per gestire i Building Blocks.

Per ogni building block viene anche mostrata la relativa preview accanto al nome dello sketch. Per fare in modo che queste vengano visualizzate, viene richiamata la funzione mostrata nel listato 4.7, che ritorna una Promise.

```
1  export function renderSketchBBOutside(element, basePath) {
2    const htmlFile = getHtmlFile(element.files);
3
4    return new Promise((resolve, reject) => {
5      const htmlDoc = injectLocalFiles(
6        element.files,
7        htmlFile,
8        {
9          basePath,
10         gridOutput,
11         textOutput
12       },
13       false
14     );
15     const generatedHtmlFile = {
16       name: 'index.html',
17       content: htmlDoc
18     };
19     const htmlUrl = createBlobUrl(generatedHtmlFile);
20
21     setTimeout(() => {
22       resolve(htmlUrl);
23     }, 0);
24   });
25 }
```

Listing 4.7: Funzione `renderSketchBBOutside`.

Alla funzione vengono passati i file dello sketch e il path della pagina in cui ci si trova attualmente. Tramite la funzione `injectLocalFiles` viene creato un file HTML, al cui interno viene inserito il codice JavaScript dello sketch scritto dall'utente e gli script relativi alla libreria di p5 per andare a compilare lo sketch. Successivamente, tramite la funzione `createBlobUrl`, viene creato un blob che mostra l'anteprima dello sketch selezionato, che viene rimandato indietro alla funzione chiamante.

Per fare in modo che un building block venga importato all'interno dello sketch attuale, viene eseguita la funzione `addBBToProject`, di cui la prima parte è riportata nel Listato 4.8

```
addBBToProject = () => {
  const sketchFile = this.props.sketch.files.find(
    (el) => el.name === 'sketch.js'
  );

  if (
    sketchFile.content !== defaultSketch &&
    this.props.file.name === 'sketch.js'
  ) {

    let openedProject = this.props.file.content;

    let openBrackets = 0;
    let closeBrackets = 0;
    let output = '';
    for (
      let i = openedProject.indexOf('function setup()');
      i < openedProject.length;
      i += 1
    ) {
      if (openedProject[i] === '{') {
        openBrackets += 1;
      } else if (openedProject[i] === '}') {
        closeBrackets += 1;
        if (openBrackets === closeBrackets) {

          output = [
            openedProject.slice(0, i - 1),
            `\n  ${this.props.sketch.name.replace(/\s/g, '')}Setup
   ();\n`,
            openedProject.slice(i - 1)
          ].join('');
          break;
        }
      }
    }
    if (openBrackets === closeBrackets) {
      openedProject = output;
    }

    // ...

  }
};
```

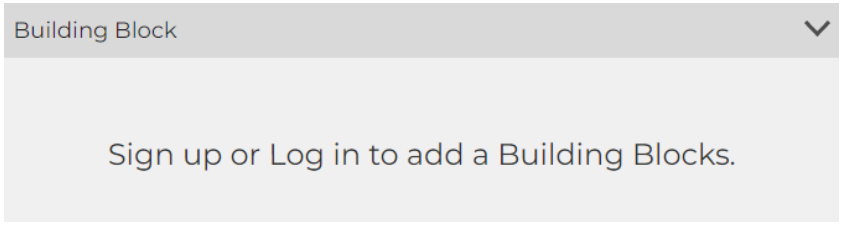
Listing 4.8: Funzione `addBBToProject`.

All'interno della funzione, dopo aver controllato se il file che l'utente ha aperto è un file JavaScript, viene fatto un conteggio delle parentesi graffe aperte e chiuse,

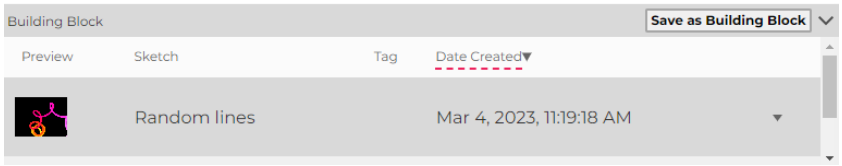
in modo da associare la prima parentesi a quella che chiude la relativa funzione. Partendo dalla prima riga della funzione `setup`, si procede in avanti finché il numero di parentesi aperte è uguale a quello di parentesi chiuse. A questo punto viene modificato il nome della funzione. Successivamente, viene fatta la stessa cosa per la funzione `draw`.

Infine, il tutto viene aggiunto al progetto in cui si sta lavorando. Nel caso in cui nel progetto attuale sia già presente il Building Block selezionato, questo non verrà copiato e questo verrà fatto presente all'utente tramite un avviso. L'utente verrà avvisato anche nel caso in cui il codice scritto sia errato e il numero dii parentesi graffe e chiuse non sia lo stesso.

Nell'implementazione finale effettuata, quando l'utente non è autenticato, non potrà aggiungere o gestire i Building Block; una volta autenticato potrà aggiungerne di nuovi, vedere i building blocks attuali e gestirli, come mostrato in Figura 4.3. Si è deciso, rispetto al mockup, di sostituire la data di ultima modifica dello sketch con i tag, ritenuti più utili.



(a) L'utente non è autenticato.



(b) L'utente ha effettuato il login.

Figura 4.3: Implementazione dei Building Blocks.

4.4.2 Linguaggio trasparente

Per l'implementazione del linguaggio trasparente, ovvero l'aggiunta delle definizioni delle principali istruzioni della libreria di p5, si è deciso di permettere all'utente di abilitare o meno questa funzionalità tramite le impostazioni. Quando l'utente entra per la prima volta l'impostazione sarà disattivata, in modo che la console non venga popolata se l'utente erroneamente clicca su un'istruzione.

Una volta attivata, cliccando un'istruzione, verrà mostrata la sua definizione all'interno della console. Le definizioni sono state prese direttamente dal sito

ufficiale di p5. Per fare questo, si è deciso di utilizzare lo stesso file JSON utilizzato dal sito ufficiale, dal quale viene estrapolata la definizione della funzione cliccata dall'utente.

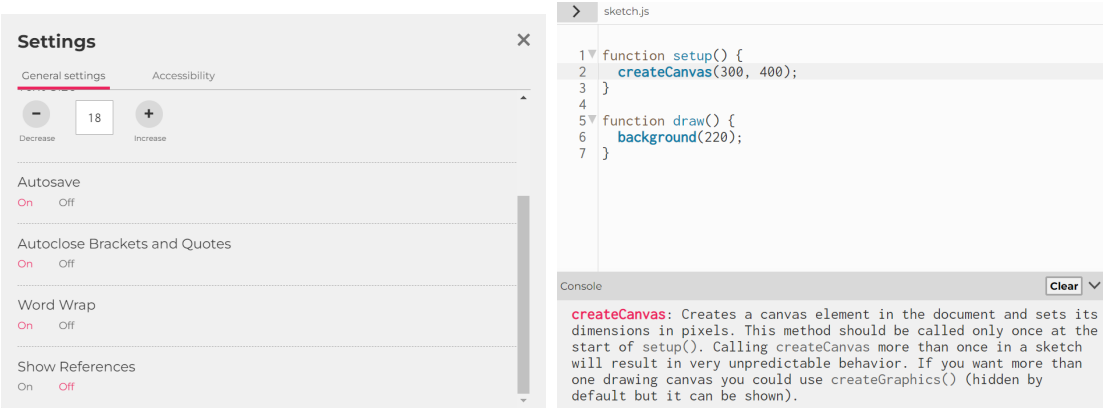
Quando l'utente clicca un'istruzione, viene chiamata l'azione `setShowReference` a cui viene passato tutto il file e il punto nel quale l'utente ha cliccato. Successivamente, in modo asincrono, viene chiamata la funzione `findReference` e la sua risposta viene aggiunta allo stato, che verrà letto e mostrato all'utente tramite la console.

La funzione `findReference`, partendo dall'offset, prenderà i caratteri immediatamente precedenti e quelli immediatamente successivi, rispetto al carattere cliccato dall'utente, fermandosi nel momento in cui incontra un carattere che non è né una lettera né un numero. A questo punto viene chiamata la funzione `search` presente nel Listato 4.9.

```
1  function search(obj, word) {
2    let find = false;
3    let myObj;
4    for (let i = 0; i < Object.keys(obj).length; i += 1) {
5      if (Object.keys(obj)[i] === word) {
6        find = true;
7        myObj = obj[Object.keys(obj)[i]];
8      }
9
10     const currentObj = obj[Object.keys(obj)[i]];
11     if (typeof currentObj === 'object') {
12       for (let x = 0; x < Object.keys(currentObj).length; x += 1)
     {
13         if (Object.keys(currentObj)[x] === word) {
14           find = true;
15           myObj = currentObj[Object.keys(currentObj)[x]];
16         }
17         if (find)
18           break;
19       }
20     }
21     if (find)
22       break;
23   }
24
25   if (find) {
26     return `<span style="color: #ed225d; font-weight: bold;">${
     word}</span>: ${myObj.description[0]}`;
27   }
28   return ''; // 'Reference not found';
29 }
```

Listing 4.9: Funzione `search`.

Il JSON in cui si effettua la ricerca è composto da oggetti e oggetti di oggetti, che presentano il campo `description` al cui interno è inserita la definizione. La funzione `search` scorre tutto il file finché non trova la definizione. Nel primo ciclo vengono controllati solamente gli oggetti esterni, per rendere la ricerca più veloce. Successivamente, se la definizione non è stata ancora trovata, verrà cercata all'interno degli oggetti più interni. Infine, se la definizione non è presente, dopo aver scorso tutto il file verrà ritornata una riga vuota, altrimenti verrà ritornata la descrizione che verrà inserita all'interno della console, come mostrato in Figura 4.4.



(a) L'utente abilita la funzionalità nelle impostazioni.

(b) L'utente visualizza la definizione.

Figura 4.4: Implementazione del linguaggio trasparente.

4.4.3 Tag

Per l'aggiunta dei tag, prima di tutto è stato aggiornato il modello degli sketch aggiungendo i tag nel backend, così che possano essere gestiti dal database.

Successivamente, per seguire la logica di Redux, nel frontend è stata creata un'azione e un reducers per la loro aggiunta e rimozione. Per fare in modo che venissero caricati insieme allo sketch, è stato aggiunto anche un selector.

I tag possono essere aggiunti tramite un apposito campo che si trova nella schermata principale, per essere visualizzati sopra lo sketch. In più, possono anche essere visualizzati nella finestra in cui sono presenti tutti gli sketch, con la possibilità di effettuare la ricerca sia per nome dello sketch, sia per tag, in modo da trovare il progetto più facilmente.

Per la loro gestione è stato deciso che l'utente ne può inserire al massimo quattro per ogni sketch e questi vengono gestiti tramite un array. Quando l'utente inserisce

il numero massimo di tag, il campo di input viene disabilitato, in modo che non ne possa aggiungere altri.

Nella Figura 4.5 è possibile vedere l'implementazione della funzionalità dei tag. Questi possono anche essere aggiunti quando l'utente non è autenticato, ma non ci sarà la possibilità di salvare lo sketch; in questo modo la modifica sarà persa.

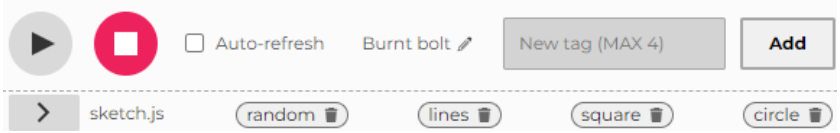


Figura 4.5: Implementazione dei tag.

4.4.4 Ultime modifiche

La nuova funzionalità delle ultime modifiche è stata aggiunta vicino alla preview, con la possibilità di visualizzare o l'anteprima dello sketch o le sue ultime modifiche.

Ogni volta che viene fatto partire lo sketch, verrà aggiunta una modifica alla lista delle ultime modifiche, che, come nome, riporta la data e l'ora in cui la modifica è stata effettuata. Per far sì che queste venissero salvate e fossero visibili anche quando la pagina viene aggiornata, si è deciso di salvare ogni modifica come un file JavaScript distinto all'interno di una cartella changes creata per l'occasione, impostandole come non visibili.

Per questo motivo, inizialmente è stata implementata la possibilità di rendere dei file invisibili, aggiungendo a ogni file la voce `isVisible` impostata a `true` per i file da rendere visibili e `false` per i file da non mostrare. In questo modo, quando una modifica viene creata, viene automaticamente resa invisibile. Questo viene fatto tramite l'action `saveNewChange`, presente nel Listato 4.10, che è chiamata ogni volta che viene riprodotto lo sketch o cliccando il pulsante play o, se l'auto-refresh è attivato, dopo ogni modifica.

```
1  export function saveNewChange() {
2    return (dispatch, getState) => {
3      const state = getState();
4
5      const openedFile = state.files.filter((file) => file.
       isSelectedFile)[0];
6
7      if (openedFile.name.match(/.*\.js$/i)) {
8        const { content } = openedFile;
9        const name = `${new Date().toLocaleString('en-us', {
10         dateStyle: 'medium',
11         timeStyle: 'medium',
12         hour12: true
```

```
13        })}.js`;
14        const formProps = {
15          name,
16          isVisible: false,
17          content
18        };
19
20        const parentId = state.files.filter((file) => file.name ===
      'changes')[0].id;
21        const { children } = state.files.filter((file) => file.name
      === 'changes')[0];
22        const myParentId = { parentId };
23
24        if (children.length === 0) {
25          dispatch(handleCreateFile(formProps, false, myParentId));
26        } else {
27          const lastId = children[children.length - 1];
28          const lastContent = state.files.filter((file) => file.id
      === lastId)[0].content;
29
30          if (lastContent !== content) {
31            dispatch(handleCreateFile(formProps, false, myParentId))
      ;
32          }
33        }
34      }
35    };
36  }
```

Listing 4.10: Funzione `saveNewChange`.

Per far sì che non vengano create modifiche con lo stesso contenuto, ogni volta che viene invocata la funzione `saveNewChange`, questa va a controllare se il contenuto della modifica è diverso da quella precedente. In questo caso verrà creato un nuovo file con `isVisible` impostato a `false`. Se l'utente vuole visualizzare una modifica tra i file del progetto, può aggiungerla ai preferiti; in questo modo, la modifica verrà resa visibile. L'utente può anche decidere di ripristinare una vecchia modifica, sostituendola allo sketch attuale: il lavoro attuale verrà sostituito da quello di una vecchia modifica, ma sarà comunque presente nella lista delle ultime modifiche.

Passando con il mouse sopra una modifica, verrà visualizzata la sua anteprima. Questa è stata creata riutilizzando la funzione utilizzata per creare l'anteprima nei Building Blocks, visibile nel Listato 4.7. Tramite le impostazioni è anche possibile modificare la grandezza dell'anteprima della sezione delle ultime modifiche, in modo da renderla più grande o più piccola, a seconda di ciò che l'artista preferisce.

In Figura 4.6 è mostrato il funzionamento della sezione delle ultime modifiche.

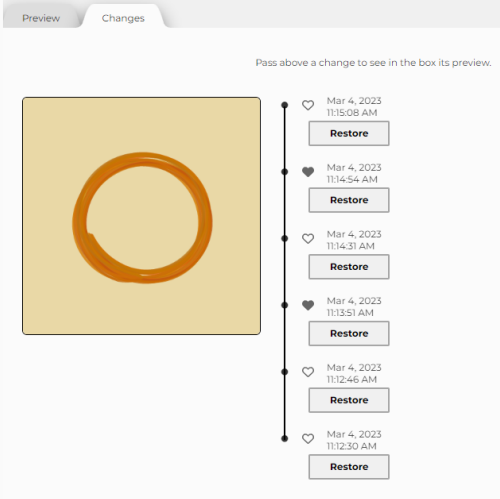


Figura 4.6: Implementazione delle ultime modifiche.

4.4.5 Visualizzazione dei Frame

Ogni volta che uno sketch viene riprodotto, è stata aggiunta la possibilità di vedere tutti i suoi frame e scaricarli, così che l'utente possa visualizzare in modo puntuale come si comporta l'animazione.

Per fare ciò, è stata creata una funzione che, in modo asincrono, compila ed esegue lo sketch per poi prendere i primi 50 frame per riproporli all'utente. Quando lo sketch viene eseguito, tramite l'azione `printFrames` viene popolato in modo asincrono lo stato del reducer con i frames dello sketch. Per popolarlo, viene invocata la funzione `printFrames` che si trova nel backend dell'applicazione.

All'interno della funzione, dopo aver prodotto i frame, viene controllato se i frames sono uguali fra loro. Se questo succede, significa che lo sketch è statico e tutti i frame saranno uguali. In questo caso, verrà visualizzato solamente un frame.

Per eseguire lo sketch e prendere i suoi frame in background è stata utilizzata la libreria Puppeteer.

Puppeteer è una libreria open-source sviluppata da Google che permette di controllare e automatizzare interazioni con un browser tramite un'interfaccia di programmazione in Node.js [39]. Consente quindi di simulare il comportamento dell'utente su un sito web, eseguire test automatizzati e raccogliere dati da pagine web per elaborarli o analizzarli. Con Puppeteer, è possibile creare script in Node.js che aprono un browser basato su Chromium, caricano una pagina web, compilano un form, scorrono un elenco di elementi, effettuano clic e interagiscono con il contenuto della pagina. È anche possibile effettuare screenshot della pagina, generare file PDF e simulare l'input dell'utente con il mouse e la tastiera.

Per consentire il download dei singoli frame è stata utilizzata la funzione

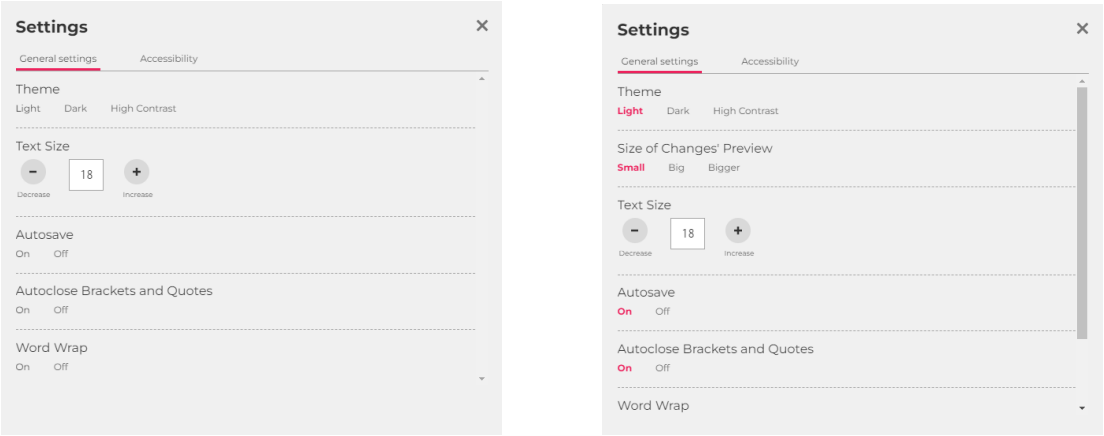
`downloadFrame`, alla quale viene passato il frame sotto forma di `data:image` che viene trasformato e fatto scaricare all'utente come file `png`.

È possibile visualizzare la funzionalità dei Frames in Figura 4.2, sulla destra dell'interfaccia.

4.5 Altri cambiamenti

Oltre alle principali esigenze risolte con le implementazioni della sezione precedente, si è deciso di applicare altre piccole modifiche all'applicazione.

La prima è stata quella di modificare il colore delle opzioni nelle impostazioni, in quanto non c'era modo di distinguere quale opzione fosse selezionata. Si è deciso di colorarle con il colore principale dell'applicazione e renderle più spesse in modo da farle vedere facilmente dall'utente che utilizza l'editor. In Figura 4.7a è mostrato come era inizialmente il pannello delle impostazioni, mentre in Figura 4.7b si può vedere la modifica effettuata.



(a) Prima della modifica. **(b)** Dopo la modifica.

Figura 4.7: Sezione delle impostazioni all'interno dell'editor.

Il cambiamento successivo riguarda i pulsanti dei pannelli che si trovano in basso sull'interfaccia, ovvero quello della Console e quello dei Building Blocks, in quanto precedentemente non si capiva bene che fossero pulsanti. Si è deciso di trasformarli in bottoni veri e propri, così che l'utente capisca facilmente che possano essere cliccati.

In Figura 4.8a è mostrata la precedente implementazione del bottone *Clear* nella Console, mentre in Figura 4.8b è mostrata l'attuale implementazione.

L'ultima modifica riguarda gli alert mostrati all'utente nei casi in cui uno sketch non venga salvato prima di cambiare pagina, oppure l'eliminazione di uno sketch o

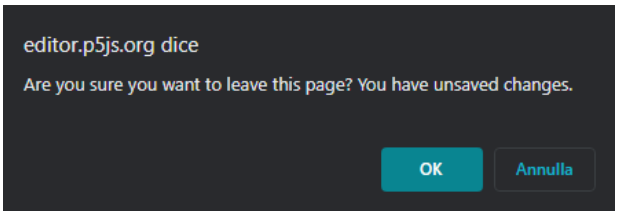
(a) Prima della modifica.

(b) Dopo la modifica.

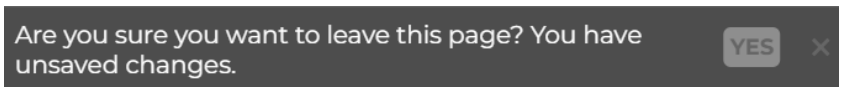
Figura 4.8: Bottone *Clear* presente nella Console.

di una collezione. In questi casi, precedentemente, veniva mostrato all'utente un alert che variava a seconda del browser in cui l'applicazione veniva visualizzata, in quanto si trattava della funzione `window.confirm`, che genera un avviso gestito dal browser e non dall'applicazione stessa. Successivamente, si è deciso di modificarlo in modo tale che resti omogeneo all'interno dell'applicazione, anche nel caso in cui si utilizzi un browser differente.

In Figura 4.9a è mostrato l'alert nel caso in cui una modifica nello sketch non venga salvata prima di spostarsi su una pagina differente, mentre in Figura 4.9b è presente lo stesso alert dopo la modifica.



(a) Prima della modifica.



(b) Dopo la modifica.

Figura 4.9: Alert di modifiche non salvate.

Capitolo 5

Valutazione

5.1 Introduzione

L'obiettivo principale nella fase di valutazione è quello di verificare l'usabilità e la funzionalità di un sistema interattivo tramite diversi criteri. Durante la valutazione, è possibile osservare come le persone utilizzano l'interfaccia utente e comprendere quali sono i punti di debolezza dell'applicazione. La valutazione è un concetto molto generale, ma ciò che si vuole verificare è identificare e correggere possibili problemi dell'applicazione oppure compiere decisioni differenti da quelle prese durante la progettazione e l'implementazione il prima possibile, in modo da evitare problemi futuri.

Il test di usabilità è composto da tre fasi principali: la pianificazione, l'esecuzione e l'analisi.

Nella fase di pianificazione, la prima cosa da fare è scegliere chi sono gli utenti a cui è rivolto il sistema su cui verificare l'usabilità. Successivamente bisogna capire di quanti utenti si ha bisogno. Secondo Nielsen, il numero di partecipanti adatto per trovare la maggior parte dei problemi di usabilità è 5 [40]. Con un numero maggiore si riescono a trovare altri problemi, ma la maggior parte di essi saranno gli stessi trovati dai precedenti.

Successivamente, bisogna decidere chi condurrà il test di usabilità e con quale ruolo. Prima di tutto, bisogna trovare un facilitatore, ovvero colui che si occuperà di dare il benvenuto ai partecipanti e spiegare loro i task aiutandoli nel caso ci siano problemi con il sistema. Tipicamente, sono presenti anche un'altra o altre due persone, una delle quali sarà l'osservatore e l'altra che prenderà appunti.

Normalmente, i creatori, gli sviluppatori e i designer del sistema possono partecipare prendendo appunti o facendo gli osservatori, ma non dovrebbero essere i facilitatori, in quanto conoscendo l'applicazione potrebbero essere imparziali

durante la spiegazione di essa e del test da condurre, per esempio dando alcuni consigli al partecipante e invalidando in questo modo il test.

Tipicamente, questo ruolo è ricoperto da una persona esperta di test di usabilità presa dall'esterno, che ha una panoramica dell'applicazione tale da poter condurre il test, ma che non fa parte del team che ha creato il sistema.

Dopo aver deciso chi saranno i partecipanti e definito i ruoli, bisogna scegliere quali saranno i task da chiedere agli utenti. Qualche volta è utile che queste siano introdotte da uno scenario, come per esempio una breve spiegazione del contesto in cui il task deve essere svolto. I task devono avere un obiettivo ben preciso che l'utente deve andare a compiere. Il numero adatto di task che si dovrebbe avere è compreso tra 5 e 10.

Il passo successivo è quello di scegliere quali sono le metriche di valutazione per ogni task. La prima cosa da decidere è se si vogliono chiedere al partecipante delle informazioni aggiuntive prima o dopo del singolo task, del test o di un gruppo di task per andare a caratterizzare meglio il test che si sta svolgendo. Successivamente, bisogna decidere l'equipaggiamento necessario, che deve essere concorde con i criteri e la metodologia scelta. Bisogna preparare un consenso informato da far compilare a ogni singolo partecipante, che dev'essere d'accordo riguardo alla registrazione della seduta, specificando come saranno usati i suoi dati. L'ultima cosa da preparare è un copione che il facilitatore dovrà seguire, così che tutti gli utenti ricevano le stesse informazioni. Il copione, al suo interno, comprende una breve parte di benvenuto, tutti i task, le loro metriche, l'equipaggiamento necessario, delle possibili domande post-test e una parte finale di ringraziamenti e saluti.

5.1.1 Metriche

Con metriche si intendono sia i criteri con cui il task può essere dichiarata completata con successo o non completata, sia tutte le informazioni che si possono chiedere o che si vogliono raccogliere. Si possono dividere le metriche in due categorie: soggettive e quantitative.

Le metriche soggettive possono essere raccolte prima dello svolgimento del test, come per esempio informazioni sulla conoscenza del sistema, sull'età del partecipante e così via. Per esempio, se si vuole condurre un test su un'applicazione per smartphone, si vogliono utenti che abbiano dimestichezza con esso. Poi, dopo ogni task o dopo ogni gruppo di task si possono avere altre domande da fare al partecipante. Queste potrebbero essere domande a risposta chiusa o aperta o una serie di domande su cosa è stato fatto fino a quel momento. Alla fine del test possono essere fatte altre domande, come per esempio la facilità d'uso dell'applicazione o la soddisfazione dell'utente. Queste sono soggettive, in quanto il partecipante fornisce le sue impressioni o le sue idee sull'intera applicazione o su quello che è stato fatto fino a quel momento.

Le metriche quantitative invece vengono valutate sulle azioni compiute dal partecipante. Per esempio, in questo gruppo possiamo trovare il Completion Rate, ovvero il numero di persone che completano correttamente il singolo task, gli errori critici, gli errori non critici e l'Error-Free Rate, ovvero la percentuale di partecipanti che hanno condotto il test senza compiere nessun errore. Con errori critici si intendono tutti gli errori che non permettono all'utente di concludere il compito assegnato e l'utente potrebbe o non potrebbe sapere che il task è stata parzialmente completata o non è stata completata affatto. Gli errori non critici invece sono quegli errori che non prevengono il completamento del task, in quanto l'utente si accorge dell'errore che ha compiuto e lo corregge prima di completare il suo compito. Questo tipo di errori non risulta nel conteggio finale, in quanto l'utente riesce a completare correttamente il task assegnata.

Altre metriche quantitative sono il tempo impiegato per compiere il task o le misure soggettive. Potremmo volere, infatti, che l'utente compia il task entro un certo tempo massimo. Con misure soggettive si intendono delle domande a risposta chiusa con una scala da 1 a 5, in cui l'utente può valutare per esempio la facilità d'uso del task o la sua soddisfazione. Infine, si potrebbero fare anche delle domande aperte, in cui viene chiesto all'utente qualche consiglio su cosa potrebbe essere migliorato nel task appena eseguita.

5.1.2 Metodologie

Mentre i partecipanti compiono un determinato task, si potrebbero volere maggiori informazioni. Queste possono essere ottenute tramite le metodologie. La prima è chiamata *think-aloud*, nella quale al partecipante viene chiesto di dire ad alta voce tutti i ragionamenti che compie per andare a risolvere il compito che gli è stato assegnato. In questo modo, è possibile capire come il sistema viene utilizzato e se viene utilizzato nel modo pensato da chi lo ha creato. Uno svantaggio però è che è possibile che questo vada ad alterare il corretto svolgimento del task o alcune metriche. Per esempio, in questo caso è meglio non definire un tempo massimo in cui il task deve essere completata, in quanto raccontando tutti i passaggi compiuti l'utente potrebbe metterci più tempo del previsto. Questo può essere fatto durante il task oppure una volta finita il task, chiedendo all'utente di ripercorrere tutte le azioni che ha compiuto. La ricostruzione non sarà mai perfetta e precisa come potrebbe essere la spiegazione durante lo svolgimento del task, però potrebbe andare bene se si hanno delle limitazioni.

L'altra metodologia è chiamata *cooperative*, dove il partecipante collabora con il facilitatore facendosi domande a vicenda e lavorando insieme sullo svolgimento del task. Uno svantaggio potrebbe essere quello di non poter misurare il tempo in cui il compito assegnato viene concluso, come nel caso precedente. Un ulteriore vantaggio rispetto al think-aloud invece è quello di poter chiedere all'utente delucidazioni sul

perché compie determinate azioni. L'utente potrebbe essere incoraggiato a dire ciò che pensa veramente del sistema senza sentirsi osservato, in quanto viene instaurata una conversazione.

5.1.3 Questionari

Dopo aver effettuato il test o dopo una singola domanda, per avere alcune informazioni aggiuntive, si può far compilare un questionario all'utente.

Il primo tipo di questionario è il *SEQ*, ovvero un questionario da fare dopo ogni singolo task. Questo dev'essere corto e contenere circa 1-3 domande per cercare di interferire il meno possibile con la sessione del test. In questo caso, viene chiesto all'utente di valutare la difficoltà dell'attività appena svolta, tramite una scala che va da 1 a 7. Questo tipo di questionario viene usato raramente.

Un altro tipo di questionario è il *SUS*, che viene effettuato alla fine del test. Questo va a misurare l'usabilità del sistema percepita dall'utente. È un questionario composto da dieci domande e ognuno di esse ha cinque risposte che vanno da "completamente disaccordo" a "completamente d'accordo". Questo questionario va a produrre un punteggio che va da 0 a 100, ma non è da confondere con un punteggio percentuale, quindi non può essere confrontato con gli altri. Solitamente, un punteggio maggiore di 68 è considerato sopra la media. Le domande di questo questionario sono già stabilite e vengono alternate tra domande positive e negative. I vantaggi di questo questionario sono che è molto semplice e molto veloce da utilizzare, ed è applicabile a una vasta gamma di tecnologie e prodotti. Gli svantaggi sono che offre una misura soggettiva dell'usabilità del sistema e non è possibile andare a confrontare diversi sistemi basandosi sul punteggio ottenuto dal SUS.

Un altro tipo di questionario da fare alla fine del test è il *NASA-TLX* che può essere utilizzato in ambienti dove il sistema che si sviluppa può avere grandi responsabilità. Questo riguarda meno l'usabilità, e riguarda di più il carico di lavoro. Questo questionario è formato da sei domande che hanno un voto che va da 1 a 21. Andare a calcolare il punteggio di questo tipo di questionario può essere molto complicato.

5.2 Pianificazione

Per esigenze organizzative, si è scelto di effettuare il test online tramite videochiamata. Inoltre, il test è stato effettuato in lingua inglese, in quanto i partecipanti non erano italiani. All'interno del capitolo tutti i task e le domande a loro posti sono state tradotte in lingua italiana, ma è possibile trovare lo script del test di usabilità all'interno dell'Appendice A.

L'obiettivo del test è quello di sperimentare le nuove funzionalità aggiunte all'editor di p5 con lo scopo di rendere più semplice e intuitivo creare animazioni e visualizzazioni grafiche attraverso il creative coding.

Per il test di usabilità si è scelto di selezionare dei partecipanti che già conoscano il creative coding e sappiano cosa sia p5, in modo che abbiano già visto almeno una volta la sua interfaccia e possano in questo modo valutare le modifiche effettuate.

Ogni partecipante dovrebbe avere:

- Un computer con microfono e videocamera
- Una connessione a internet
- Delle conoscenze basiche di inglese

L'osservatore e chi prende appunti dovrebbero avere:

- Un computer con microfono e videocamera
- Un programma per registrare audio e video
- Qualcosa su cui prendere appunti
- Lo script e la lista di task che il facilitatore leggerà ai partecipanti

Il facilitatore dovrebbe avere:

- Un computer con microfono e webcam
- Lo script e la lista di task che leggerà ai partecipanti
- La possibilità di condividere lo schermo del proprio computer

5.2.1 Task

All'interno del test di usabilità si è scelto di inserire dieci diversi task, per andare a valutare le nuove funzionalità implementate all'interno dell'editor di p5$^+$.

Per ogni task, viene definito uno scenario, in modo che l'utente possa capire a pieno l'utilizzo della funzionalità da valutare. Si è, inoltre, deciso di seguire un determinato ordine nello svolgimento dei task, in modo da andare a creare un vero e proprio flusso di lavoro che passi attraverso tutte le funzionalità implementate in modo da simulare l'utilizzo dello struemento da parte di un artista.

I task sono brevemente riportate all'interno della Tabella 5.1, evidenziando per ognuna di esse il testo del task, i criteri di successo secondo i quali il partecipante supera il task e il tempo massimo oltre il quale il task viene dichiarata non superata. I task completi con tutte le loro metriche possono essere trovate nel test di usabilità riportato nell'Appendice A.

Task	**Testo del task**	**Criteri di successo**	**Tempo massimo**
1	Apri lo sketch chiamato *Random lines* e aggiungilo ai Building Blocks.	Aprire e aggiungere lo sketch ai Building Blocks.	3 minuti
2	Crea un nuovo sketch e importa al suo interno il Building Block del task precedente.	Creare uno sketch e importare il Building Block *Random lines*.	3 minuti
3	Scarica il dodicesimo frame sul tuo computer.	Scaricare il dodicesimo frame dello sketch.	//
4	Aggiungi un tag e salva lo sketch.	Aggiungere un tag.	3 minuti
5	Trova il tuo sketch nella lista cercando il tag inserito nel task precedente.	Trovare correttamente lo sketch usando la barra di ricerca.	3 minuti
6	Ripristina la penultima modifica di *Minimalism*.	Ripristinare la penultima modifica.	3 minuti
7	Aggiungi la seconda modifica ai preferiti e visualizzala nell'editor.	Aggiungere la seconda modifica ai preferiti ed aprirla.	//
8	Elimina definitivamente lo sketch utilizzato.	Eliminare lo sketch utilizzato.	3 minuti
9	Allarga la preview dello sketch nella lista delle modifiche.	Aprire le impostazioni e allargare la preview.	3 minuti
10	Attiva le reference e vedi la definizione di *draw()*.	Aprire le impostazioni e cliccare sulla parola *draw()*.	//

Tabella 5.1: Elenco dei task.

All'interno della Tabella è possibile notare come alcuni task, più specificatamente il 3, il 7 e il 10, non abbiamo limiti di tempo. Questo è stato fatto in quanto il task 3 utilizza la metodoligia think-aloud, mentre i task 7 e 10 la metodologia cooperative. Proprio per questo motivo, visto che i partecipanti devono nel primo caso spiegare ad alta voce i loro passaggi, e nel secondo interagire con il facilitatore, il tempo utilizzato per compiere il task non sarebbe una metrica affidabile.

Si è scelto di inserire tre diversi task che riguardano la funzionalità dei Building Blocks (1, 2, 8), uno che riguarda la possibilità di vedere e scaricare i frame dello sketch (3), due riguardo l'inserimento e la ricerca tramite tag (4, 5), tre riguardanti la funzionalità delle ultime modifiche (6, 7, 9) e uno riguardante il linguaggio trasparente (10). Questa scelta è stata fatta per andare a coprire tutti i diversi aspetti implementati all'interno delle singole funzionalità, così che il partecipante possa entrarci in confidenza e valutarli, sia durante lo svolgimento dei task, sia

successivamente rispondendo ad alcune domande post-test.

5.2.2 Domande post-test

Nel test di usabilità si è scelto di chiedere all'utente delle domande, sia chiuse che aperte, alla fine del test. Si è scelto di non utilizzare il SUS, in quanto all'utente viene chiesto di valutare l'usabilità delle funzionalità aggiunte all'editor di p5 e non tutta l'applicazione.

Per le domande chiuse si è scelto di creare un form che l'utente potesse compilare, così che potesse fornire la sua vera impressione sentendosi meno osservato. Successivamente, a voce, sono state chieste alcune domande a risposte aperta, che si sono concentrate sui pareri soggettivi riguardo l'usabilità di alcune funzionalità.

Le domande chiuse, a cui l'utente poteva rispondere con un punteggio che va da 1 a 5, ovvero da "fortemente in disaccordo" a "fortemente d'accordo", sono le seguenti:

- Consideri facile da utilizzare la funzionalità dei Building Blocks?
- Consideri facile da utilizzare la funzionalità che ti permette di vedere e scaricare i singoli Frame di uno sketch?
- Consideri facile da utilizzare la funzionalità che ti permette di aggiungere dei tag a uno sketch?
- Consideri facile da utilizzare la sezione delle ultime modifiche?
- Consideri facile da utilizzare la funzionalità che ti permette di vedere le definizioni delle funzioni utilizzate all'interno dello sketch?
- Ti piace l'interfaccia dell'applicazione con le nuove funzionalità?
- Consideri le nuove funzionalità dell'applicazione intuitive da utilizzare?

Le domande aperte chieste all'utente sono state ricollegate ai dubbi riscontrati durante lo svolgimento del test di usabilità. Nella maggior parte dei casi sono state chieste solamente alcune di queste domande, perché diverse risposte sono scaturite direttamente da alcune domande di approfondimento poste dall'utente alla fine del test.

- Quale task hai trovato più complicata da portare a termine?
- Useresti queste nuove funzionalità durante la creazione di uno sketch?
- Pensi che queste nuove funzionalità ti possano essere utili?

- Pensi che sia complicato importare un Building Block all'interno dello sketch corrente?
- Pensi che sia difficile capire come si fa a scaricare un singolo Frame dello sketch?
- Pensi che sia difficile capire la funzionalità che permette di aggiungere una modifica ai preferiti nella sezione delle ultime modifiche?
- Pensi che sia difficile capire cos'è una *reference* e il modo in cui queste lavorano?

5.3 Svolgimento dei test

Il test di usabilità è stato effettuato con sei partecipanti selezionando artisti che avessero già qualche esperienza con il creative coding e con l'editor online di p5. Questa decisione è stata presa in modo tale che i partecipanti potessero valutare correttamente le nuove funzionalità, conoscendo già quelle che l'editor offriva precedentemente. Tutti gli artisti intervistati hanno un'eta compresa tra i 25 e 35 anni. Alcuni di loro sono dottorandi, mentre altri lavorano da alcuni anni nel mondo del creative coding, sviluppando opere d'arte digitali. Tutti gli artisti intervistati hanno usato almeno una volta l'editor online di p5 e conoscono il linguaggio JavaScript utilizzato per poter creare opere d'arte statiche o animate con l'editor in questione.

Per evitare di dare informazioni differenti a ogni partecipante e per essere sicuri di definire in maniera chiara tutti i task, è stato creato uno script da seguire per ogni test di usabilità effettuato, disponibile nell'Appendice A.

I test sono stati svolti in modalità remota, tramite la piattaforma Zoom. Per i test si è deciso di avviare l'applicazione in locale e, tramite la condivisione dello schermo, abilitare il controllo remoto, in modo da permette ai partecipanti di poter controllare l'interfaccia e potersi muovere liberamente all'interno dell'applicazione.

Prima dell'inizio del test a ogni partecipante è stato fatto compilare un modulo per il consenso della registrazione del test tramite un modulo online. Alla fine del test, allo stesso modo, è stato fatto compilare a ogni partecipante un breve questionario a risposte chiuse per verificare l'usabilità delle nuove funzionalità.

5.4 Risultati

All'interno della sezione verranno elencati e discussi i risultati ottenuti dopo lo svolgimento dei test di usabilità.

Dopo aver elencato i risultati dei singoli task, evidenziando i problemi riscontrati dagli utenti in alcune di esse, verranno discusse le risposte date dopo i test, per

finire con un'analisi delle possibili modifiche da effettuare all'interfaccia, per rendere l'applicazione più usabile e intuitiva per gli artisti che andranno ad utilizzarla.

5.4.1 Task

Task	Totale errori critici	Totale errori non critici	Completion Rate	Error-Free Rate
1	0	2	1	0.833
2	1	1	0.833	0.667
3	0	3	1	0.667
4	0	0	1	1
5	0	0	1	1
6	0	2	1	0.833
7	2	6	0.667	0.333
8	0	0	1	1
9	2	7	0.667	0.333
10	2	8	0.667	0.333

Tabella 5.2: Misure ottenute dai risultati del test.

Nella Tabella 5.2 sono riportate le misure raccolte per ogni task. Sono stati evidenziati in rosso i valori più bassi, che caratterizzano i task in cui i partecipanti hanno avuto maggiori problemi.

I task con Completion Rate più basso sono quelle in cui gli artisti hanno commesso più errori critici, non riuscendo a completare correttamente l'obiettivo.

Nello specifico, nel Task 7 il problema principale che ha portato i partecipanti a non completare correttamente il task è legato alla visualizzazione della modifica nell'editor dopo averla salvata, in quanto non tutti gli utenti sono abituati a muoversi tra diversi file dello stesso progetto. Per questo motivo, non tutti erano a conoscenza dell'apposita sezione per aprire un nuovo file senza spostarsi in un progetto differente. In più, non è stato sempre chiaro il fatto che salvando una modifica ai preferiti, questa venisse aggiunta in automatico all'interno dei file del progetto.

Il problema dei Task 9 e 10 riguarda invece il pannello delle impostazioni, che alcuni partecipanti non sono riusciti a localizzare facilmente in quanto normalmente non lo utilizzano. Per questo motivo, è stato per loro complicato trovare e abilitare le impostazioni per modificare la grandezza della preview nella sezione dei cambiamenti o per attivare le references.

Un ulteriore problema legato al Task 10 riguarda la parola references, che secondo un partecipante non è adatta per definire le definizioni delle funzioni utilizzate all'interno dello sketch. Per questo motivo, non è stato ben compreso l'obiettivo del task.

I task con un Error-Free Rate basso sono invece quelle in cui i partecipanti hanno commesso almeno un errore, critico o non critico. Questi errori forniscono una visione delle possibili problematiche all'interno dell'applicazione, che, seppur non abbastanza gravi da non portare a termine l'obiettivo del task, hanno provocato incertezze e dubbi nei partecipanti.

Mentre alcuni task hanno riportato un punteggio più vicino a 1, come la 1 e la 6, altre hanno messo più in difficoltà i partecipanti, come la 7, 9 e 10.

Nel Task 1 e 2 il problema ha riguardato principalmente il fatto di non vedere immediatamente la sezione dei Building Blocks, andandola a cercare all'interno dei menu a tendina che si trovano sopra lo sketch.

Nel Task 3 c'è stata inizialmente dell'incomprensione nella riproduzione dei frame, in quanto non è stato immediatamente chiaro che prima che i singoli frame venissero caricati dall'applicazione dovesse essere riprodotto lo sketch.

Nel Task 6 e 7 il problema riguardava la sezione delle ultime modifiche che non è stata trovata subito da tutti i partecipanti. In più, per il Task 7, c'è stato anche un problema di salvataggio della vecchia modifica, in quanto non sempre è stata salvata la seconda modifica partendo dall'inizio, ma la penultima modifica. Il problema è però legato a un'incomprensione del task.

Nel Task 9 e 10 alcuni artisti sono andati a ricercare le impostazioni nei menu a tendina sopra lo sketch, invece che all'interno del menu delle impostazioni. Per un partecipante, il problema era legato alla visualizzazione della pagina, in quanto non visualizzava correttamente la parte destra dello schermo; questo potrebbe aver reso difficile trovare il menu delle impostazioni. Una volta abilitate le impostazioni, i task sono state svolte correttamente.

5.4.2 Domande post-test

Dopo aver completato il test, a ogni partecipante è stato fatto compilare un modulo online, contenente sette domande a risposta chiusa, con un punteggio da 1 a 5, ovvero da "fortemente in disaccordo" a "fortemente d'accordo". Mentre le prime domande si dedicavano a delle specifiche funzionalità dell'editor, le ultime erano più generali. Ogni domanda è stata accompagnata da una breve descrizione che riassumeva la funzionalità alla quale si riferiva, così da rinfrescare la mente all'utente.

Le risposte alle domande chiuse sono state mediamente positive, andando ad indicare che le nuove funzionalità sono state apprezzate e che possono essere utili per chi giornalmente utilizza l'editor di p5 per creare le proprie opere.

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
Building Blocks	4	5	3	4	5	4
Frames	4	5	5	4	4	5
Tag	4	5	4	4	4	5
Ultime modifiche	3	5	4	4	4	5
Linguaggio trasparente	4	5	4	4	4	5
Gradevolezza dell'applicazione	4	4	4	4	5	4
Intuitività dell'applicazione	4	3	3	4	4	4

Tabella 5.3: Punteggi delle domande a risposta chiusa dopo il test di usabilità.

Nella Tabella 5.3, per ogni riga, è stata indicata la funzionalità legata alla specifica domanda, e in ogni colonna è stato indicato il punteggio assegnato da ogni partecipante. I punteggi delle ultime due domande, riguardanti la gradevolezza dell'interfaccia e l'intuitività dell'applicazione con le nuove funzionalità fanno notare come le aggiunte all'interfaccia non pesino su di essa, facendola rimanere semplice e intuitiva.

Successivamente, a ogni artista sono state poste delle domande a risposta aperta, riguardanti i task su cui avevano avuto maggiori difficoltà e sulle funzionalità da loro apprezzate. Tramite le loro risposte, è stato possibile indagare sui principali problemi riscontrati nello svolgimento del test, evidenziando le principali carenze dell'applicazione. Infine, qualche artista ha ritenuto importante dare qualche ulteriore suggerimento per migliorare alcune parti dell'applicazione, in modo da renderle più intuitive e comprensibili.

Complessivamente, durante tutto lo svolgimento del test, si è potuto notare come i task 7, 9 e 10 siano stati le più complicate da svolgere, e questo è emerso anche nelle domande aperte poste alla fine del test. Infatti, le problematiche maggiormente evidenziate dagli utenti, sono andate a rispecchiare le principali incertezze e lacune riscontrate nello svolgimento delle task. Per esempio, alla domanda riguardo la facilità d'utilizzo della sezione degli ultimi cambiamenti, più specificamente riguardo alla possibilità di aggiungere una modifica ai preferiti, è stato riscontrato come fosse complicato capire che la modifica venisse anche aggiunta ai file del progetto, in quanto da nessuna parte viene spiegato in modo chiaro il suo funzionamento.

Riassumendo, tutte le principali problematiche riscontrate durante il test sono state le seguenti:

- Pensare di trovare la sezione dei Building Blocks nei menu a tendina in alto a

sinistra presenti nell'interfaccia, invece che nella sezione in basso a destra.

- Non comprendere la funzionalità dell'aggiunta di una modifica ai preferiti.
- Non comprendere come allargare l'anteprima nella sezione delle ultime modifiche.
- Avere dubbi su dove venissero visualizzati i tag una volta inseriti.
- Pensare che le *references* mostrassero tutti i posti dove una funzione è stata utilizzata all'interno dello sketch e non la sua definizione.
- Pensare che le *references* venissero mostrate in un posto diverso rispetto alla console.

5.4.3 Possibili modifiche all'interfaccia

I risultati ottenuti da questo test di usabilità hanno permesso di evidenziare alcuni possibili cambiamenti da implementare nell'interfaccia per renderla più intuitiva e facile da utilizzare.

La modifica più importante emersa da questi test è la necessità di avere più di un modo per accedere ad una funzionalità. Per esempio, è stato evidenziato come alcuni partecipanti non riuscissero facilmente a trovare il pulsante per aggiungere un nuovo sketch ai Building Blocks, cercando nei menu a tendina che si trovano in alto nello schermo, riportati in Figura 5.1. Una rapida soluzione potrebbe quindi essere quella di aggiungere dei collegamenti ai Building Blocks in modo che si possa accedere facilmente a questi anche tramite il menu.

Un'altra modifica suggerita dagli utenti è quella di non limitare il numero di tag a quattro per sketch, ma permettere di aggiungerne di più. Questo potrebbe dare l'opportunità di caratterizzare meglio uno sketch, ma potrebbe anche andare a peggiorare la ricerca di uno sketch, rendendo l'aggiunta di troppi tag molto caotica e poco precisa. Un ulteriore modifica riguarda lo stile dei tag, che secondo alcuni utenti non è molto chiaro per andare a comprendere la loro rimozione e il loro posizionamento; secondo alcuni partecipanti servirebbe uno stile che li vada a distinguere in modo netto dal resto dell'interfaccia.

Un altro aspetto che ha creato qualche problema durante il test riguarda la parola *references*. Nonostante la bibliografia della libreria p5 definisca in questo modo le definizioni delle proprie funzioni, secondo alcuni partecipanti questa parola non porterebbe a pensare alla definizione della funzione, ma ai suoi possibili usi all'interno dello sketch. Per questo motivo, potrebbe essere meglio utilizzare la parola *definitions*.

Qualche artista ha anche consigliato di dedicare una sezione solamente alle references, invece di mostrarle all'interno della console. Questo perché la console

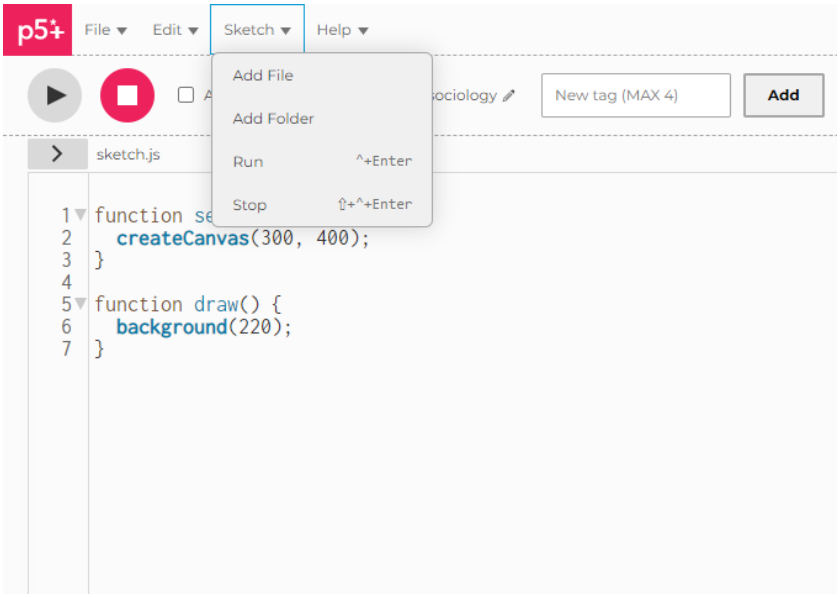


Figura 5.1: Uno dei menu a tendina presenti nell'interfaccia di p5$^+$.

viene spesso ricollegata ai vari errori che possono essere prodotti dallo sketch o ai log che un'artista potrebbe stampare per capire meglio l'evoluzione del proprio codice. Per questo motivo, è stato consigliato di inserire un ulteriore pannello per mostrare le varie definizioni. Questo nuovo pannello potrebbe essere inserito o a destra dell'editor oppure sopra la console, però rischiando di andare ad occupare parti dello schermo dedicate ad altri utilizzi, rendendo così l'interfaccia meno semplice e intuitiva.

Un'ultima modifica che si è ritenuta utile riguarda l'aggiunta di una spiegazione della funzionalità delle ultime modifiche, soprattutto per il salvataggio delle modifiche tra i preferiti. Infatti, non è stato sempre chiaro il fatto che salvando una modifica tra i preferiti, questa possa anche essere trovata tra i file dello sketch, con l'opportunità di lavorare in parallelo sull'attuale modifica e su una vecchia modifica senza perdere il lavoro di nessuno dei due sketch.

Capitolo 6

Conclusioni

Il lavoro svolto all'interno della tesi ha permesso di raggiungere gli obiettivi prefissati inizialmente, ovvero rendere più semplice e intuitivo andare a realizzare animazioni e visualizzazioni grafiche attraverso il creative coding, focalizzandosi sulle arti visive, statiche o animate, che si possono creare tramite questa disciplina. A tale scopo, è stato modificato l'esistente editor online di p5, rinominato p5$^+$, con l'aggiunta di nuove funzionalità, per aiutare gli artisti più e meno esperti nella creazione delle loro opere.

L'analisi della letteratura ha permesso di evidenziare come il creative coding si sta evolvendo ed espandendo sempre di più negli ultimi anni e quali sono i maggiori strumenti utilizzati dagli artisti. Infine, si sono potuti approfondire Processing e p5.js, indicando per ognuno di essi vantaggi e svantaggi e le loro differenze, così da comprendere meglio quali possono essere le esigenze degli artisti. Questo studio è proseguito anche nel Capitolo 3, dove si sono potute notare alcune lacune di Processing e p5 per poi decidere di proseguire l'analisi solamente sul secondo dei due. Tramite un'ulteriore studio della letteratura, sono emersi i principali problemi degli artisti nell'utilizzo di questi strumenti e nella creazione delle loro opere, così da poter scegliere e proporre alcune nuove funzionalità da aggiungere all'editor di p5.

Dopo aver discusso le principali tecnologie utilizzate, sono stati proposti dei mockup per le principali nuove funzionalità, così da vedere graficamente il loro impatto sull'esistente interfaccia di p5.

Successivamente, all'interno del Capitolo 4 si è discussa l'implementazione delle nuove funzionalità, soffermandosi inizialmente sull'attuale struttura dell'applicazione fosse strutturata e sulle principali librerie utilizzate per fare in modo che l'applicazione sia facilmente scalabile ed espandibile, come la gestione dello stato tramite Redux o il pattern con models, controllers e routes utilizzata nel backend. Per ogni funzionalità implementata, sono stati discusse le parti più importanti, mostrando anche i relativi Listati con il codice implementato.

All'interno del Capitolo 5 sono stati svolti dei test di usabilità con sei diversi partecipanti, così da poter comprendere eventuali problemi nelle nuove funzionalità implementate. Dopo aver discusso dei principali risultati di ogni task e delle risposte e osservazioni poste alla fine del test dai partecipanti, sono state riportate alcune modifiche all'interfaccia, che potrebbero essere implementate per rendere l'applicazione ancora più intuitiva e semplice da utilizzare.

6.1 Sviluppi futuri

Anche tramite i test di usabilità, sono state notate delle possibili modifiche da poter applicare all'applicazione proposta in tesi, per renderla ancora più intuitiva e utile per i vari artisti che la utilizzano.

Una prima modifica potrebbe essere quella di aggiungere automaticamente i tag ad uno sketch, a seconda delle funzioni utilizzate al suo interno. In questo modo, i vari tag all'interno dei diversi sketch diventerebbero parole chiave molto più caratterizzanti rispetto a quelle che potrebbe scegliere un utente. Infatti, un artista potrebbe utilizzare parole chiave diverse per definire lo stesso concetto, rendendo la funzionalità dei tag molto confusionaria e poco utile.

Un'ulteriore modifica proposta durante la fase di test, è stata quella di aggiungere i branch nella sezione delle ultime modifiche, con la possibilità di vedere se altri utenti hanno utilizzato lo stesso sketch come punto di partenza per sviluppare una propria opera. Questo renderebbe l'applicazione molto più interattiva e potrebbe essere utile anche per i nuovi utenti partire da opere già realizzare da altri artisti per comprendere meglio il proprio codice e per andare a creare animazioni più complesse. In questo modo, anche la community di p5 potrebbe essere molto più vicina ad ogni singolo artista.

Per rendere la community sempre più attiva, un'altra aggiunta potrebbe essere quella di proporre ad un utente non solo i propri Building Blocks, ma anche i Building Blocks creati da altri utenti, così da poter esplorare anche il loro codice. Potrebbe anche essere utile aggiungere alcuni Building Blocks predefiniti, creati dall'applicazione stessa, che vanno ad implementare qualche funzionalità poco utilizzata, così da proporre agli utenti degli esempi validi ed efficaci, in modo da fornire nuovi spunti per creare le proprie opere.

Un ultimo sviluppo futuro potrebbe essere quello di non mostrare agli utenti solamente la definizione della funzione che vanno a cliccare, ma spiegare anche nel dettaglio anche ogni parametro utilizzato all'interno di una funzione, indicando il suo scopo, così che l'utente possa comprendere meglio tutte le varie funzionalità offerte dalla libreria di p5. Un ulteriore passo in avanti potrebbe essere quello di aggiungere un piccolo esempio della funzione di cui si vuole leggere la definizione mostrando il

suo utilizzo all'interno di uno sketch, così che possano vedere chiaramente il suo utilizzo.

Tali aggiornamenti potrebbero rendere l'applicazione ancora più semplice e intuitiva da utilizzare, fornendo agli gli artisti più e meno esperti una maggiore facilità nell'ideazione e nella creazione delle proprie opere.

Appendice A

Script per il Test di Usabilità

A.1 Introduzione

[Give the user the link for participating in the video call and wait for him/her participating]

Hi, _____. My name is _____, and I'm going to be walking you through this session today. First of all, I want to thank you for participating in this study.

Before we begin, I have some information for you, and I'm going to read it to make sure that I cover everything. You probably already have an idea of why you are here, but let me go over it again briefly. We're asking people to try using an app that we're working on so we can see whether it works as intended. The session should take about 30 minutes.

This is an app that modify the online editor of p5.js adding some functionalities.

An important thing I want to make clear is that we're testing the app itself, not you. You can't do anything wrong here. We don't want you to worry about making mistakes. Also, please don't worry that you're going to hurt our feelings. We're doing this to improve it, so we need to hear your honest reactions. We appreciate your feedback and impression about the app.

If you have any questions as we go along, just ask them. I may not be able to answer them right away, since we're interested in seeing how people use the app when they are alone without someone next to them to help. Only in some tasks, I will ask you to describe your actions and decisions and in others, we will collaborate by asking each other questions. At the end of the session, I will ask

you some recap questions. And if you need to take a break at any point, just let me know.

With your permission, we're going to record what happens on the screen and our conversation. The recording will only be used to help us figure out how to improve the app, and it won't be seen by anyone except the people working on this project. Also, a few people are observing this session in the call.

Do you have any questions so far?

[Send him/her the link for recording permission form]

OK, great. Now we can start looking at the app.

[Share your screen and give him/her remote control]

First, I'm going to ask you if you see my screen and if you can control it with your mouse and keyboard.

[Start the screen recorder on your laptop]

This is the app you will test today.

Now, before you start doing anything, I'll give you some minute to familiarize with the app and the tabs.

You can scroll and navigate through the app if you want. I've already logged in with the account we're going to test with today.

[Let the user have a little tour in the app for 2-3 minutes. After that start the test]

A.2 Task

Task 1 You just created a sketch and you want to use it again inside another sketch without manually copy all your functions. Open the sketch called *Random lines* and add it to the Building Blocks.

Success Criteria: The participant is able to open the sketch and add it to the Building Blocks.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)
- Time on Task: 3 minutes

Task 2 You notice that the functions of the sketch of the previous task can be useful also for a new sketch as a starting point. Create a new sketch and import in

it the Building Block of the previous task.

Success Criteria: The participant is able to create a sketch and import in it the Building Block called *Random lines*.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)
- Time on Task: 3 minutes

Task 3 You like how 12th frame of the current sketch look. Download the frame on your computer.

Success criteria: The participant is able to reproduce the sketch and download the 12th frame inside the *Frames* section.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)

Methodology: Think-aloud

Task 4 At this point, you want to characterize the actual sketch. Add to it a tag and save it.

Success Criteria: The participant is able to add a tag that specify in a correct way the content of the sketch.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)
- Time on Task: 3 minutes

Task 5 You want to find your sketch among the others. Go on the list of all the sketches and find your sketch searching the tag you inserted previously.

Success Criteria: The participant is able to search and find correctly the sketch using the search bar.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)
- Time on Task: 3 minutes

Task 6 The sketch called *Minimalism* was modified a lot of times, but you like to work again on the previous change with respect to the actual sketch. Open the sketch and restore the previous change.

Success Criteria: The participant is able to restore the previous change clicking on the *Restore* button on the *Changes* section.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)
- Time on Task: 3 minutes

Task 7 At this point, you want to work on the current sketch, but also on its second change without losing all the work until now. Add the second change to your favourites and open it in the editor.

Success Criteria: The participant is able to save the second changes (starting from the bottom) to his favourites through the *Changes* section. Then he is able to open the sketch and see it in the editor.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)

Metodology: Cooperative

Task 8 Now you notice that the sketch you added before in the Building Blocks is no more useful. Delete it.

Success Criteria: The participant is able to click on the arrow near to the correct Building Block and click the button "Delete". After that, the elimination need to be confirmed.

Metrics:

- Successful Task Completion (T/F)
- Time on Task: 3 minutes

Task 9 You want a bigger preview of the sketch on the *Changes* section. Find a way to enlarge it.

Success Criteria: The participant is able to open the settings and click *Big* on the "Size of Changes' Preview".

Metrics:

- Successful Task Completion (T/F)
- Time on Task: 3 minutes

Task 10 Turn on the references, so you can see the definition of the function *draw()*.

Criteri di successo: The participant is able to open the settings and clicking *On* under "Show References". Then he/she has to click on the word *draw()*.

Metrics:

- Successful Task Completion (0-100% based on the correctness of the actions)

Metodology: Cooperative

A.3 Questionario

Thanks, that was very helpful. I'll send in chat a little questionnaire to have feedback about the test. Please answer to all the questions.
[Send him/her the link of the form with the questionnaire]

If you'll excuse me for a minute, I'm just going to see if my colleagues have any follow-up questions they'd like me to ask you.
[Ask the observers' questions, then prove anything you want to follow up on]

Some possible follow-up questions:

- Which task did you find more difficult?
- Would you use these new functionalities to create your sketches?
- Do you think these new functionalities are useful?
- Do you think is difficult import a Building Block inside the current sketch?

- Do you think is difficult understand how to download a frame?
- Do you think is difficult to understand the functionality of the heart button inside the "Changes" section?
- Do you think is difficult understand what is a reference and the way it works?

[Do other follow-up questions based on the result of the questionnaire]

Do you have any suggestions for us?
[Wait some minutes, so he/she can give some suggestions]

Thanks a lot for the help you give us today. Do you have any questions for me, now that we're done?
[Stop the screen recorder and save the file]

[Thank the participant and invite him/her to get out from the video call]

Bibliografia

[1] *How Technology is Changing the Art World*. visitato il 18/02/2023. 2020. URL: https://www.artdex.com/how-technology-is-changing-the-art-world-2/ (cit. a p. 1).

[2] M. Krasteva. «The Impact of Technology on the Modern Art». In: *Digital Presentation and Preservation of Cultural and Scientific Heritage* 6 (2016), pp. 247–254 (cit. a p. 1).

[3] A. Benedetti, T. Elli e M. Mauri. «"Drawing with code": the experience of teaching creative coding as a skill for communication designers». In: *Proceedings of EDULEARN20 Conference*. 12th International Conference on Education and New Learning Technologies. 2020, pp. 3478–3488 (cit. alle pp. 1, 7).

[4] *Programming for Art*. visitato il 17/02/2023. 2021. URL: https://www.codingforvisuallearners.com/blog/programming-for-art (cit. a p. 2).

[5] D. Xu, U. Wolz, D. Kumar e I. Greenburg. «Updating Introductory Computer Science with Creative Computation». In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE '18. 2018, pp. 167–172 (cit. alle pp. 2, 5).

[6] I. Bergstrom e R. Lotto. «Code Bending: A New Creative Coding Practice». In: *Leonardo* 48 (gen. 2015), pp. 25–31 (cit. a p. 2).

[7] M. C. Mitchell e O. Bown. «Towards a Creativity Support Tool in Processing: Understanding the Needs of Creative Coders». In: *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*. OzCHI '13. 2013, pp. 143–146 (cit. a p. 5).

[8] T. Dufva. «Creative Coding at the Arts and Crafts School Robotti (Käsityökoulu Robotti)». In: *Digital Humanities in the Nordic Countries Conference*. 2018 (cit. a p. 5).

[9] V. Fragapane e B. Standl. «Work in Progress: Creative Coding and Computer Science Education – From Approach to Concept». In: *2021 IEEE Global Engineering Education Conference (EDUCON)*. 2021, pp. 1233–1236 (cit. a p. 5).

[10] S. Suh, K. J. Lee, C. Latulipe, J. Zhao e E. Law. *Exploring Individual and Collaborative Storytelling in an Introductory Creative Coding Class*. 2021 (cit. a p. 6).

[11] B. Shneiderman. «Creating Creativity: User Interfaces for Supporting Innovation». In: *ACM Trans. Comput.-Hum. Interact.* 7.1 (mar. 2000), pp. 114–138 (cit. a p. 6).

[12] I. Greenberg, D. Kumar e D. Xu. «Creative Coding and Visual Portfolios for CS1». In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. SIGCSE '12. 2012, pp. 247–252 (cit. a p. 6).

[13] J. Li, S. Hashim e J. Jacobs. «What We Can Learn From Visual Artists About Software Development». In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21. 2021 (cit. alle pp. 8, 23, 24).

[14] B. Victor. *Learnable Programming - Designing a programming system for understanding programs*. Set. 2012. URL: `http://worrydream.com/LearnableProgramming/` (cit. alle pp. 9, 23).

[15] *PSLIVE*. visitato il 02/03/2023. URL: `https://github.com/ffd8/p5live` (cit. a p. 9).

[16] J. Hoffswell, A. Satyanarayan e J. Heer. «Augmenting Code with In Situ Visualizations to Aid Program Understanding». In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. 2018, pp. 1–12 (cit. alle pp. 10, 23).

[17] G. Sasson. *Tweak Mode*. visitato il 02/03/2023. URL: `http://galsasson.com/tweakmode/` (cit. alle pp. 10, 23).

[18] R. Chugh, B. Hempel, M. Spradlin e J. Albers. «Programmatic and Direct Manipulation, Together at Last». In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '16. 2016, pp. 341–354 (cit. a p. 10).

[19] Brian Hempel, Justin Lubin e Ravi Chugh. «Sketch-n-Sketch: Output-Directed Programming for SVG». In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. UIST '19. 2019, pp. 281–292 (cit. a p. 10).

[20] J. Li, J. Brandt, R. Mech, M. Agrawala e J. Jacobs. «Supporting Visual Artists in Programming through Direct Inspection and Control of Program Execution». In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20. 2020, pp. 1–12 (cit. alle pp. 12, 24).

[21] J. Jacobs, S. Gogia, R. Mundefinedch e J.R. Brandt. «Supporting Expressive Procedural Art Creation through Direct Manipulation». In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. 2017, pp. 6330–6341 (cit. a p. 12).

[22] *Awesome Creative Coding*. visitato il 15/02/2023. URL: `https://github.com/terkelg/awesome-creative-coding` (cit. a p. 12).

[23] *About: Cinder*. visitato il 15/02/2023. URL: `https://libcinder.org/about` (cit. a p. 13).

[24] *About openFrameworks*. visitato il 15/02/2023. URL: `https://openframeworks.cc/about/` (cit. a p. 13).

[25] A. Richardson. *Data-driven Graphic Design: Creative Coding for Visual Communication*. Fairchild Books, 2016 (cit. alle pp. 13, 15).

[26] *Features - Godot Engine*. visitato il 15/02/2023. URL: `https://godotengine.org/features/` (cit. a p. 14).

[27] *three.js*. visitato il 16/02/2023. URL: `https://github.com/mrdoob/three.js/` (cit. a p. 16).

[28] *Paper.js - About*. visitato il 16/02/2023. URL: `http://paperjs.org/about/` (cit. a p. 16).

[29] *Processing - FAQ*. visitato il 15/02/2023. 2022. URL: `https://github.com/processing/processing/wiki/FAQ` (cit. alle pp. 18, 19).

[30] *p5.js*. visitato il 15/02/2023. URL: `https://p5js.org/` (cit. a p. 20).

[31] *Add tags to sketches*. visitato il 02/03/2023. URL: `https://github.com/processing/p5.js-web-editor/issues/557` (cit. a p. 24).

[32] *Figma*. visitato il 10/03/2023. URL: `https://www.figma.com/` (cit. a p. 25).

[33] *Getting Started - React*. visitato il 02/03/2023. URL: `https://reactjs.org/docs/getting-started.html` (cit. a p. 30).

[34] *Installing Express*. visitato il 02/03/2023. URL: `https://expressjs.com/en/starter/installing.html` (cit. a p. 32).

[35] *MongoDB Documentation*. visitato il 02/03/2023. URL: `https://www.mongodb.com/docs/` (cit. a p. 33).

[36] *Docker Docs: how to build, share and run an application*. visitato il 02/03/2023. URL: `https://docs.docker.com/` (cit. a p. 34).

[37] *Redux Essentials*. visitato il 10/03/2023. URL: `https://redux.js.org/tutorials/essentials/part-1-overview-concepts` (cit. a p. 35).

[38] *reselect - npm*. visitato il 10/03/2023. URL: `https://www.npmjs.com/package/reselect` (cit. a p. 36).

[39] *Puppeteer*. visitato il 10/03/2023. URL: `https://pptr.dev/` (cit. a p. 48).

[40] J. Nielsen. *Why You Only Need to Test with 5 Users*. Mar. 2000. URL: `https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/` (cit. a p. 51).