UNIVERSITÀ POLITECNICA DELLE MARCHE

Facoltà di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Sviluppo di funzionalità aggiuntive in micropolisJS per la realizzazione di un serious game nell'ambito del water-reuse

Development of features in micropolisJS for a serious-game in the water-reuse context

Relatore:
Ch.mo Prof. Mancini Adriano

Laureando:
Merla Massimo
1061333

Anno Accademico 2019-2020

"Non insegnare le discipline
con la costrizione, ma come giocando;
potrai così scoprire le tendenze
individuali di ciascuno."

PLATONE

Indice

Introduzione

Serious Games e Gamification

I **serious games**, letteralmente *"giochi seri"*, sono giochi che contengono elementi educativi il cui scopo non è il mero intrattenimento. Dalla definizione sembra che "gioco serio" sia una nozione paradossale, cioè "serio" contro "nessun obiettivo ma svago e divertimento". Probabilmente a causa di questa contraddizione, le definizioni sono numerose e differiscono in una certa misura.

Una caratteristica comunemente accettata è che i giochi seri non solo sono divertenti, ma hanno anche uno scopo specifico [1].
La prima apparizione ed espressione del termine "serious game" viene attribuito al ricercatore e sviluppatore americano di origine tedesca Carl Abt, che per primo ne discusse nel suo libro *Serious Games(1970)* [2]. Questa fu la sua definizione:

"ridotto alla sua essenza formale, un gioco è un'attività tra due o più decisori indipendenti che cercano di raggiungere i propri obiettivi in un contesto limitante. Una definizione più convenzionale direbbe che un gioco è un contesto con regole tra avversari che cercano di raggiungere degli obiettivi. Siamo interessati ai giochi seri, nel senso che questi giochi hanno uno scopo educativo esplicito ed attentamente pensato e non sono destinati ad essere giocati principalmente per divertimento."

Ovviamente, nonostante la sua definizione fosse rivolta ai giochi di carte e da tavolo, questa definizione può essere facilmente estesa anche ai giochi dell'era digitale [3].
La natura di un serious game deriva in parte dalla tecnica della **Gamification**; come si può intuire deriva dalla parola "game", cioè gioco, e può essere italianizzato in "ludicizzazione". In letteratura esistono varie definizioni di gamification, una universalmente accettata è quella di Deterding che definisce la gamification come "l'uso di elementi di game design in contesti non di gioco" [4], ovvero in contesti reali, al fine di incoraggiare un comportamento o la fidelizzazione mediante partecipazione attiva impiegando elementi di gioco come punteggi, classifiche, avatar, badge, livelli, medaglie, [5] per creare coinvolgimento, interazione, sfide ed immersione che caratterizzano l'esperienza come una dimensione distante dal mondo abituale che impegna il giocatore in modo pervasivo e completo. Trasformare una attività in qualcosa di appagante, stimolante e che sia capace di mantenere chi la svolge in uno stato di *"flow"*, concetto introdotto

dallo psicologo ungherese Mihály Csikszentmihalyi e definito come "stato mentale in cui una persona è concentrata così profondamente su un determinato compito che perde il senso del tempo e smette di preoccuparsi di altre cose.

È un'esperienza ottimale a livello di mente e corpo quando una persona si immerge in un'attività e prova un godimento più profondo" [6], causato da un'attività in cui le sfide percepite corrispondono ed ampliano le capacità dell'individuo, producendo un'esperienza di pieno coinvolgimento nel compito e di agire all'altezza delle sua capacità [7][8]. Sono sempre più i settori che impiegano la gamification: si spazia dal mondo dell'intrattenimento a quello del business, dall'istruzione alla salute [4]. Il maggiore impiego e l'evoluzione del fenomeno è dovuto alla volontà di far emergere nel proprio target la loro motivazione intrinseca, considerata come la forza più produttiva dietro il comportamento delle persone; ad esempio come accade nelle aziende, avere un dipendente a cui piace lavorare, ed è di conseguenza più produttivo, è motivo di orgoglio.

Poiché è evidente che nei giochi siamo spesso coinvolti in modo avvincente e intrinsecamente motivati, le esperienze positive sono in grado di trasmettere benefici cognitivi, emotivi e sociali [9]. Una rassegna su studi empirici [10] ha fatto notare come in effetti la gamification produca effetti positivi in termini psicologici e motivazionali in relazione alle offerte motivazionali impiegate (caratteristiche gamificate del servizio) ed ai contesti. L'uso della gamification non è affatto lontana dalla vita quotidiana; basti pensare ad applicazioni come Duolingo che permette di poter imparare lingue straniere progredendo di livello in livello grazie a sistemi di ricompense [11]; un esempio 'offline' possono essere le Escape Rooms che possono approcciare scenari storici, artistici e culturali [12]. Insomma, la gamification è, e sarà un utile strumento in grado di pilotare messaggi e stimolare interessi e/o comportamenti per il raggiungimento di specifici obiettivi.
Come esposto precedentemente in questa introduzione, il **serious game** eredita alcune peculiarità della gamification, come si può notare dall'infografica (fig.1) anche se in misure e modalità differenti. Inoltre i serious games comprendono altri contenuti tecnici e pedagogici rilevanti che li rendono concettualmente diversi dalle applicazioni gamificate [13].
Un tratto importante nel serious game è la presenza di un gameplay: la componente di intrattenimento non deve essere sottovalutata. A differenza delle applicazioni gamificate, qui non si parla più solo dell'applicazione di elementi ludici di carattere informativo/educativo ma dell'aggiunta di una storyline, definita come una sequenza di cicli (fig.2) di gioco che ne costituiscono la trama [14] che fanno dei serious games dei giochi veri e propri nei quali progredire. La sua forma più elementare è affrontare una sfida. Ciò significa che un giocatore deve intraprendere un'azione in conformità con le regole definite dal gioco. Quindi, questo reagisce e se l'azione intrapresa è corretta, il giocatore può riprodurre il loop successivo, altrimenti se sbagliata, deve ripetere il loop oppure il gioco può reagire mostrando l'evoluzione del problema generato dalla scelta errata: con la nuova situazione il problema deve essere ridefinito e devono essere intraprese nuove serie di azioni [1]. In un contesto del genere ad esempio, il giocatore può essere ricompensato o punito in base a un sistema di punteggio.

Figura 1: *Relazione Serious Game e Gamification* [15].

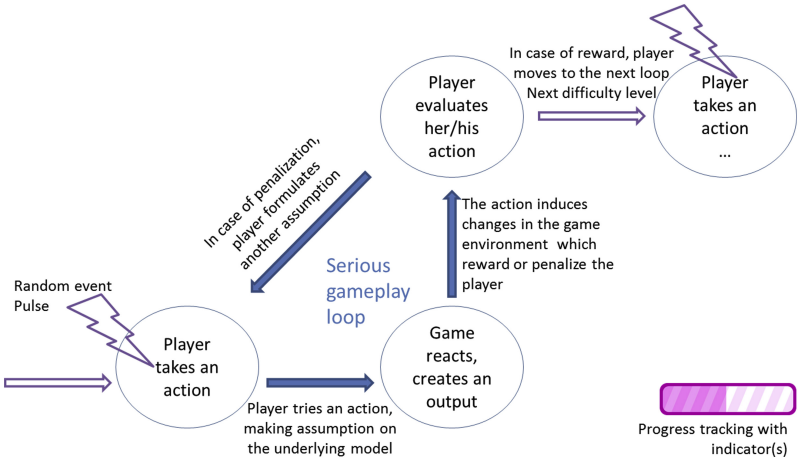

Figura 2: *Un modello di gameplay loop di un serious game* [1].

Nella vita, le lezioni più profonde vengono dai fallimenti. Allo stesso modo quando un giocatore infrange una regola in un gioco serio, incorrerà in una penalità. Questo li porta a cambiare il loro comportamento per evitare di infrangere nuovamente quella regola.
I serious game rappresentano un approccio incentrato sull'apprendimento in cui l'utente autonomamente sperimenta, interagendo in modo attivo, le alternative in un ambiente di prova e privo di ansietà, e grazie alla sensazione di controllo derivante dalla libertà di azione lo rende padrone, permettendogli di aumentare le esperienze e le prove trasformando gradualmente le sue potenzialità in abilità e conoscenze. L'operare in un mondo fittizio cambia lo stato dell'errore riducendo l'ansia associata al processo di apprendimento e consente di imparare facilmente anche senza essere consapevoli [16]. È chiaro che la funzione di apprendimento, nonché scopo del serious game, è mista alla natura di gioco dell'applicazione. Fare un gioco serio deve emozionare e coinvolgere l'utente per garantire l'acquisizione di conoscenze [3]. L'obiettivo di questi giochi, chiamati anche *meaningful games*, è intrinsecamente motivato nelle attività del gioco

stesso. In particolare, i serious games (ma questo vale anche le applicazioni gamificate) possono essere classificati secondo il *"Purpose"*, lo scopo, che può essere diviso in tre classi: trasmissione di un messaggio, formazione e scambio di dati.
Nell'ultimo decennio, sono stati sviluppati giochi seri che combinano simulazione al computer e giochi di ruolo nell'istruzione, nell'insegnamento della gestione dell'acqua e di altre risorse comuni; questa tipologia di serious game cerca di far passare un messaggio significativo e se possibile, promuovere il cambiamento [1].

Inquadratura del contesto

Ora sappiamo che i serious game sono utilizzati principalmente nel cercare di far emergere nel giocatore comportamenti responsabili poi nella vita reale.
Tutti sappiamo che *l'acqua*, il nostro "oro blu", è la risorsa più preziosa sulla Terra per cui va gestita e preservata, e diciamocela tutta: chi non ha mai visto o sentito qualche messaggio istituzionale o campagna di sensibilizzazione a riguardo; ecco, in questo senso esistono già decine di giochi seri [10] che in differenti modi e diversi ambiti cercano di sensibilizzare l'adozione di misure precauzionali per evitare sprechi d'acqua.
La compagnia europea "***digital-water.city***", leader urbano nella gestione delle risorse idriche che fornisce diverse soluzioni tecnologiche, mira a promuovere la gestione idrica integrata nelle aree urbane e periurbane di cinque grandi città dell'UE ed in particolare adotta anche un serious game connesso al riutilizzo dell'acqua a fini di irrigazione.
L'azienda stessa stima che in tutta Europa, allo stato attuale, circa il 2,4% delle acque reflue urbane trattate viene riutilizzato. A seconda della disponibilità di acqua, questo tasso può raggiungere il 70%. Il riutilizzo delle acque reflue trattate a fini di irrigazione è una misura per ridurre lo stress idrico, il sovra sfruttamento delle risorse di acqua dolce e fornisce una soluzione sostenibile per affrontare la carenza d'acqua nelle regioni vulnerabili. Tuttavia, il riutilizzo potrebbe aumentare il costo della produzione di acqua a causa dell'impiego di trattamenti aggiuntivi e del consumo di energia presso l'impianto di trattamento.
Per la produzione alimentare, il riutilizzo dell'acqua può supportare la riduzione dell'uso di fertilizzanti, migliorare lo stato nutrizionale delle colture e ridurre il consumo di energia per l'estrazione delle acque sotterranee. Gli impatti e i benefici del riutilizzo dell'acqua sono complessi e interconnessi. A livello europeo è stata approvata una proposta sui requisiti minimi per il riutilizzo dell'acqua che è entrata in vigore nell'aprile del 2020. Lo scopo di questo strumento giuridico è quello di facilitare la diffusione del riutilizzo dell'acqua ogniqualvolta sia appropriato ed efficiente in termini di costi [17]. Nell'attuale contesto del cambiamento climatico e della minaccia della carenza idrica, l'uso di acqua riciclata nelle aziende agricole può diventare sempre più necessario [18].

Struttura della tesi

In questa tesi si descrive il lavoro svolto in unione tra me ed il collega Tommaso Coricelli per l'integrazione all'interno del software 'micropolisJS', di un sistema di riutilizzo delle acque reflue per l'irrigazione dei campi. La tesi viene strutturata come segue:

- nel primo capitolo vengono introdotti informazioni sul codice sorgente, le principali tecnologie software e i linguaggi di programmazione utilizzati;
- nel secondo capitolo si presentano alcune nozioni per la configurazione del progetto, dopodiché verranno presentate la gestione e la logica implementate per il sistema di riutilizzo delle acque reflue per l'irrigazione;
- nel terzo capitolo si espongono le conclusioni e le limitazioni del progetto. Infine sono introdotti gli sviluppi futuri.

Capitolo 1

Strumenti utilizzati

Di seguito viene presentata la risorsa impiegata ed introdotte le tecnologie e i software adottati per il debugging e lo sviluppo delle funzionalità aggiuntive al progetto. Altre tecnologie e concetti per lo sviluppo de progetto verranno presentati nel capitolo successivo.

1.1 Codice sorgente

Bisogna ricordare che l'integrazione delle funzionalità aggiuntive che verranno descritte nel seguito, sono stato compiuto su una versione open source scritta in JS/HTML5, indicataci dal prof. Mancini, di uno dei più vecchi giochi di simulazione, Micropolis, che è a sua volta è la versione open source del gioco SimCity della Electronic Arts che ha reso disponibili i sorgenti sotto licenza GPL 3 a partire dal 2008 [19].
il sorgente **micropolisJS** è praticamente una applicazione web, ciò significa che è scritto con i linguaggi inerenti il mondo del web e si serve dei browser per poter essere eseguito. In particolare il file si trova nella repository `https://github.com/graememcc/micropolisJS` di *GitHub*.

1.1.1 Git e GitHub

Per dare una specifica del servizio GitHub è inevitabile l'introduzione di **Git**. Sviluppato inizialmente da Linus Torvalds, Git è un software di controllo versione distribuito totalmente gratuito pensato per gestire progetti ad alte prestazioni, utilizzabile da linea di comando. Esso è caratterizzato da alcuni elementi tra i quali [20][21]:

- supporto allo sviluppo non lineare: Git supporta ramificazione e fusione (*branching* e *merging*), e comprende strumenti specifici per visualizzare e navigare una cronologia di sviluppo non lineare;

- supporto allo sviluppo distribuito: Git fornisce ad ogni sviluppatore una copia locale, dell'intera cronologia di sviluppo, detta *repository*. Eventuali modifiche effettuate alla repository possono essere trascritte, tramite opportuni comandi, nella copia di ogni sviluppatore, che le importa come

diramazioni di sviluppo, e le fonde allo stesso modo di una diramazione sviluppata localmente;

- autenticazione crittografica della cronologia: la cronologia di Git viene memorizzata in modo tale che il nome di una revisione particolare, chiamata *commit*, dipenda dalla completa cronologia di sviluppo che conduce a tale modifica. Una volta che una revisione è stata pubblicata, non è più possibile modificare una vecchia versione del progetto senza che ciò venga notato;
- modalità memorizzazione dati: Git considera i propri dati più come una sequenza di istantanee *(snapshot)* di un mini filesystem. Ogni volta che registri (commit), o salvi lo stato del tuo progetto, Git fa un'immagine di tutti i file in quel momento, salvando un riferimento allo snapshot. Per essere efficiente, se alcuni file non sono cambiati, Git non li risalva, ma crea semplicemente un collegamento al file precedente già salvato. Git considera i propri dati più come un flusso di istantanee;
- operazioni solitamente locali: la maggior parte delle operazioni in Git necessitano solo di file e risorse locali per operare e poiché l'intera storia del progetto è sul disco locale molte operazioni sembrano quasi immediate.

Segue che **GitHub** è una piattaforma web che offre un servizio di hosting di repository Git basato su cloud. In sostanza, rende molto più facile per individui e team utilizzare Git per il controllo delle versioni e la collaborazione interagendo con una GUI [22].
Attualmente la community di GitHub vanta più di 50 milioni di utenti e oltre 100 milioni di repository [23].
Si è usufruito di GitHub non solo perché è la piattaforma che ospita il sorgente, anche perché offre piani gratuiti di hosting per piccoli gruppi; in particolare ci si è serviti di **GitHub Desktop**, client per W10 scaricabile da `https://desktop.github.com/` permettendo di operare velocemente tramite interfaccia grafica. Il progetto trattato in questa tesi comprensivo delle funzionalità aggiunte è consultabile sulla repository online. [24].
GitHub è stato un oggetto fondamentale in questo lavoro perché il lavoro era condiviso tra me e un collega, per cui le modifiche che venivano apportare da entrambi potevano essere scaricate velocemente permettendo anche in caso di modifiche contrastanti di poter recuperarle una volta scaricato il *push del commit* del collega da GitHub.

Figura 1.1: *Logo di GitHub* [25]

1.2 Tecnologie per lo sviluppo

Come già specificato, micropolisJS è un'applicazione web; si andranno ad introdurre le principali tecnologie (fig.1.2) che vengono direttamente interpretate dai browser web, fornendo struttura, stile e logica e dunque sono utilizzate generalmente per la realizzazione della parte *front-end* (la parte visibile e interattiva) di un sito o una applicazione web.

Figura 1.2: *Tecnologie necessarie a realizzare un sito o una applicazione web: HTML, CSS, JS* [26]

1.2.1 HTML

HTML acronimo di *Hyper Text Markup Language* ("Linguaggio di contrassegno per gli Ipertesti") non è un linguaggio di programmazione. Si tratta invece di un linguaggio di markup (di 'contrassegno' o 'di marcatura'), che ha lo scopo di indicare i contenuti e specificandone allo stesso tempo la struttura grafica. Queste indicazioni vengono date attraverso l'inserimento nel testo di marcatori o etichette, detti *tag*. Per dichiarare un documento che utilizzerà lo standard dell'Html5 si andrà ad utilizzare il tag: <!DOCTYPE html>. Dopo la dichiarazione del tipo di documento, esso presenta una struttura ad albero annidato, composta da sezioni delimitate da tag opportuni che al loro interno contengono a loro volta sottosezioni più piccole, sempre delimitate da tag. La struttura più esterna è quella che delimita l'intero documento, ed è compresa tra i tag <html> e </html>. All'interno dei tag <html> lo standard prevede sempre la definizione di due sezioni distinte e disposte in sequenza ordinata: la sezione di intestazione o header, marcata tra i tag <head> e </head>, e la sezione del corpo delimitata tra i tag <body> e </body>, che contiene la parte informativa vera e propria, ossia il testo, le immagini e i collegamenti che costituiscono la parte visualizzata dal browser. Al di sotto di questa suddivisione generale, lo standard non prevede particolari obblighi per quanto riguarda l'ordine e il posizionamento delle ulteriori sottosezioni all'interno dell'header o del body, a parte l'indicazione del rispetto dei corretti annidamenti [27].

1.2.2 CSS

CSS sta per *Cascading Style Sheets* ("fogli di stile a cascata"). Come l'HTML, anche il CSS non è un vero e proprio linguaggio di programmazione. Si tratta di un "linguaggio di stile", che consente di applicare, selettivamente, uno stile

agli elementi dei documenti HTML per modellare la formattazione delle pagine web. I CSS istruiscono un browser o un altro programma utente su come il documento debba essere presentato all'utente, per esempio definendone i font, i colori, le immagini di sfondo, il layout, il posizionamento delle colonne o di altri elementi sulla pagina. L'emanazione delle specifiche CSS 1 nel 1996 da parte del W3C *(World Wide Web Consortium)* fu il primo passo di un sistema atto a perseguire la separazione tra il contenuto e la presentazione [28]. L'introduzione del CSS si è resa necessaria per separare i contenuti delle pagine HTML dalla loro formattazione e permettere una programmazione più chiara e semplice da utilizzare, garantendo allo stesso tempo il riutilizzo di codice ed una più facile manutenzione.
L'inserimento di codice CSS nelle pagine web può essere effettuato in tre modi:

- inserendo nel tag <head> della pagina in codice HTML un collegamento ad un foglio di stile esterno, cioè un file contrassegnato dall'estensione .css tramite il tag link: soluzione che assicura il massimo del riuso del codice;
- inserendo, sempre all'interno dell'<head> tra gli specifici tag <style> e </style> le dichiarazioni css;
- in linea all'interno degli elementi HTML: questo metodo non offre nessuna garanzia sul riuso del codice [29].

La soluzione utilizzata nel codice sorgente è la prima: avremo dunque un file .css importato nel documento HTML che istruisce sullo stile.

1.2.3 JavaScript

JavaScript (spesso abbreviato in JS) è un linguaggio di programmazione leggero, interpretato ed orientato agli oggetti e agli eventi per lo più utilizzato come linguaggio di scripting all'interno di altro codice per automatizzare delle operazioni; grazie alle funzionalità di manipolazione dei documenti, che è possibile effettuare direttamente lato client senza coinvolgere il server, JS aggiunge alle pagine HTML la possibilità di essere modificate in modo dinamico, creando, in siti e applicazioni web, effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso del browser (mouse, tastiera, caricamento della pagina, etc). Le funzioni di script, utilizzati nella logica di presentazione, possono essere opportunamente inserite in file HTML, in pagine JSP o in appositi file con estensione .js poi richiamati nella logica di business [30].
Le caratteristiche principali di JavaScript sono:

- essere un linguaggio interpretato: il codice non viene compilato, ma eseguito direttamente; in JavaScript lato client, il codice viene eseguito dall'interprete contenuto nel browser dell'utente;
- la sintassi è relativamente simile a quella dei linguaggi C, C++ e Java;
- definisce le funzionalità tipiche dei linguaggi di programmazione ad alto livello (strutture di controllo, cicli, ecc.) e consente l'utilizzo del paradigma object oriented.

- è un linguaggio debolmente tipizzato: quando si dichiara una variabile non c'è bisogno di specificarne il tipo;

- è un linguaggio debolmente orientato agli oggetti: gli oggetti stessi ricordano più gli array associativi del linguaggio Perl che gli oggetti di Java o C++ [30]. [31].

Attualmente quasi tutte le pagine web contengono JavaScript, e se per qualche ragione è disabilitato, i contenuti e le funzionalità delle pagine possono risultare limitati o addirittura non disponibili [32].
Come si può notare dal riquadro qui di seguito (fig.1.3), nell'anno appena trascorso JavaScript può essere annoverato nella top 10 dei linguaggi più popolari in base all'indicatore di popolarità dei linguaggi di programmazione *"TIOBE Programming Community index"* [33].

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	15.95%	+0.74%
2	1	↓	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	↑	PHP	2.49%	+0.62%
9	19	⇈	R	2.37%	+1.33%
10	8	↓	SQL	1.76%	-0.19%

Figura 1.3: *Classifica popolarità in base al "TIOBE Programming Community Index"* [33]

1.3 Ambiente

1.3.1 Visual Studio Code

Si tratta di un editor di codice sorgente sviluppato dalla Microsoft, cross-platform compatibile con Windows, Linux e macOS che permette di evidenziare la sintassi di ciascun linguaggio di programmazione (Syntax highlighting), integra il supporto per il debugging, un terminale a riga di comando, il controllo Git, IntelliSense (il completamento automatico delle istruzioni), la possibilità di mantenere aperti più file affiancandone il contenuto in più schede e molto altro ancora. Punto di forza di Visual Studio Code sono le estensioni grazie alle quali è possibile ampliare notevolmente le funzionalità del programma. Sebbene il codice sorgente sia coperto da licenza MIT, il sito della Microsoft distribuisce l'applicazione come freeware [34][35].

Figura 1.4: *Logo Visual Studio Code* [36]

1.3.2 Web browser

Un Browser è un programma pensato in maniera specifica per poter recuperare, presentare e navigare risorse (testi, video, canzoni, immagini e svariate altre tipologie di contenuti) sul web (che si appoggia sull'infrastruttura di rete Internet) che vengono ciascuna identificate attraverso un apposito URL *"Uniform Resource Locator"*. Tra i browser più utilizzati vi sono *Google Chrome, Internet Explorer, Mozilla Firefox, Microsoft Edge (uscito con Windows 10), Safari, Opera.* Il browser può essere utilizzato da computer ma anche da smartphone o tablet (sotto forma di app). In entrambi i casi non vi sono sostanziali differenze, il funzionamento resta praticamente lo stesso a prescindere dal dispositivo impiegato. Il primo browser fu sviluppato da *Tim Berners-Lee* (tra i primi precursori del concetto di WWW e fondatore del W3C), e venne chiamato WorldWideWeb: serviva a scopi dimostrativi. Una caratteristica imprescindibile di un browser è che si appoggia sempre ad un motore di ricerca per raggiungere i siti web interessati. Nell'architettura di rete "client-server" di Internet il browser rappresenta il client che fa richieste di risorse web ai vari web server e application server ospitanti rispettivamente siti web e applicazioni web. Esso rappresenta dunque il sistema software di interfacciamento dell'utente con la rete che rende la navigazione dell'utente tipicamente user-friendly [37].
Ho utilizzato Google Chrome per poter connettermi al server locale all'indirizzo di loopback (127.0.0.1) risolto dal SO con l'utilizzo di un nome associato, ovvero localhost, e raggiungibile su una porta HTTP che risulta libera, a partire dalla 8080, e tutto ciò viene messo a disposizione da webpack-dev-server, uno strumento che verrà trattato in seguito.

Capitolo 2

Impostazione e Presentazione

In questo capitolo inizialmente si introdurranno i passaggi per una corretta impostazione per lo sviluppo, seguirà una prima presentazione del programma micropolisJS e poi l'esposizione dell'implementazione delle funzionalità aggiuntive al software.

2.1 Cosa c'è da sapere

Nel paragrafo seguente verranno introdotte oltre che la preparazione dell'ambiente al debugging anche una piccola guida essenziale per la corretta configurazione del programma per far in modo che questo sia eseguibile senza problemi, poiché nel nostro lavoro abbiamo avuto qualche problema. Intanto qui espongo alcune informazioni che sono doverose sapere per avere piena coscienza delle tecnologie 'nascoste' che ci serviranno in seguito. Vediamo cosa sono.

2.1.1 NPM

NPM è il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js. Consiste in un client da linea di comando, chiamato anch'esso npm, e un database online di pacchetti pubblici e privati, chiamato npm registry. L'NPM o Node Package Manager è la più grande libreria opensource al mondo di pacchetti Javascript, è uno strumento essenziale che permette di scaricare facilmente tutti i moduli sviluppati dalla community Node.js. Tutti questi moduli sono pubblicati nel sito web: `http://npmjs.org`. in più l'NPM regola le dipendenze tra i vari moduli: questo significa che, se un modulo ha bisogno di un altro modulo per funzionare, l'NPM lo scaricherà automaticamente. Un modulo è facilmente installabile, basta posizionarti nel terminale all'interno della cartella del tuo progetto e scrivere:

```
1     npm install nomemodulo
```

Il modulo verrà installato localmente, ovvero soltanto all'interno di quel progetto specifico, verrà scaricato automaticamente l'ultima versione del modulo e lo posizionerà in una sotto-cartella node_modules: NPM per default installa i moduli localmente per ogni progetto, motivo per cui crea sottocartelle in node_modules [38];

2.1.2 YARN

YARN, acronimo di "Yet-Another-Resource-Negotiator", è anch'esso un package manager che è parallelo e deriva, dalla versione del 2016, da NPM (all'epoca alla sua quarta versione) ma con svariate funzionalità in più, nello specifico per sopperire al sistema non deterministico di gestione dei pacchetti di NPM, per questo è stato introdotto il file *"yarn.lock"* accanto al *"package.json"*(che definiremo di seguito). Quanto troviamo scritto nel yarn.lock non è semplicemente una lista di dipendenze, ma una lista che, per prima cosa non punta ai server npm, e che contiene anche un hash per controllare che il package scaricato sia precisamente quello che vogliamo [39];

2.1.3 package.json

Tutti i pacchetti npm contengono un file, di solito nella directory principale del progetto, chiamato *package.json*. Questo file contiene vari metadati rilevanti per il progetto. Viene utilizzato per fornire informazioni ad npm che gli consentono di identificare il progetto e gestire le sue dipendenze: per dipendenze si intendono altri pacchetti utili al suo funzionamento. Può anche contenere altri metadati come una descrizione del progetto, la versione del progetto in una particolare distribuzione, informazioni sulla licenza, persino dati di configurazione, che possono essere vitali sia npm sia per gli utenti finali del pacchetto [40];

2.1.4 Webpack e webpack-dev-server

webpack, è un pacchetto node per questo installabile con NPM e in un progetto si trova nella cartella node_modules; la sua funzione è quella di essere un bundler di moduli. Lo scopo principale è raggruppare i file JavaScript per l'utilizzo in un browser, ma è anche in grado di trasformare, raggruppare o impacchettare praticamente qualsiasi risorsa. Di grande importanza anche il *webpack-dev-server* che sostanzialmente permette di creare un server locale e fornisce il ricaricamento della pagina web in tempo reale, quest'ultima funzione dovrebbe essere usata solo per lo sviluppo.
Nel progetto è presente anche il file di configurazione, webpack.config.js, nel quale si possono definire i comportamenti dei pacchetti webpack come vedremo in seguito [41].

2.2 Messa in opera

Prima di tutto bisogna trasferire il progetto dalla repository di GitHub alla propria macchina, utilizzando i comandi per terminale che Git mette a disposizione; si può utilizzare il terminale del SO o anche dell'editor di testo, in questo caso Visual Studio Code(o VS Code), in cui scrivere:

```
1 C:\workfolder> git clone https://github.com/graememcc/micropolisJS
2
```

Listing 2.1: git clone da repository a cartella di lavoro locale

per cui creerà una cartella 'micropolisJS' nel percorso C:\workfolder.

Una volta copiato andremo ad aprire la cartella del progetto 'micropolisJS' dall'editor VS Code dal comando sulla barra dei comandi 'File' e poi 'Add folder to Workspace'; il progetto si presenta con diversi file e sottocartelle. Per far in modo che possa essere eseguibile correttamente raggiungendolo sulla propria macchina (facendo riferimento a localhost:porta) è inevitabile dover seguire alcuni passaggi. Prima di tutto avremo bisogno di installare su VS Code una estensione di Yarn; abbiamo installato l'estensione di nome 'yarn' con identificatore 'gamunu.vscode-yarn' di Gamunu Balagalla [42]. Questo plugin ci serve per poter installare le dipendenze tracciate nel file package.json. Una volta installato, l'avvenuta installazione di yarn apparirà nella sezione 'Extensions' di VS Code. Successivamente cliccando col tasto destro su package.json si ha la possibilità di mandare in esecuzione il comando 'Install yarn Packages' che permette di installare i pacchetti che compaiono in un formato chiave/valore ("nome": "versione") nella sezione 'devDependencies' di package.json e verrà di conseguenza prodotto automaticamente il file yarn.lock, sotto la directory principale del progetto, che non dovrà essere modificato. Dopo l'installazione delle dipendenze andiamo a modificare alcuni file:

- tsconfig.json: in questo file, che istruisce il compilatore TS dei file .ts (TS sta per TypeScript, è un linguaggio front-end robusto e adatto per applicazioni JavaScript complesse, compilato a runtime in JavaScript), andiamo a modificare il "target" da "es5" a "es6";

- config.js: questo file si trova sotto src/, ci permetterà di visualizzare più opzioni per il debugging. Ci troveremo davanti ad un oggetto con tre proprietà settate tutte a false, basterà cambiarle in true;

- index.html: questa è la modifica più grande. Per avere la versione funzionante fare riferimento al file nella repository su GitHub `https://github.com/capatommy/micropolisJS/blob/prova1/index.html`.

A questo punto per far partire il web server in locale ci dirigiamo nella sezione "scripts" del file package.json, cliccando sulla scritta "▷ Debug" si aprirà la finestra dei comandi, scegliendo start (o comunque una modalità che sia in development) partirà webpack-dev-server per la creazione del server locale(fig.2.1). Diversamente è possibile farlo partire da terminale integrato in VS Code col comando "npm run 'nomescript'" dove 'nomescript' può essere build/start/watch/test, anche in questo caso scegliere sempre una modalità che sia in development. Dal browser adesso potremo raggiungere il programma che in questo caso è in ascolto sulla porta HTTP 8080, per cui lo si può raggiungere direttamente digitando "localhost:8080" nella barra degli indirizzi e vedremo micropolisJS in esecuzione.

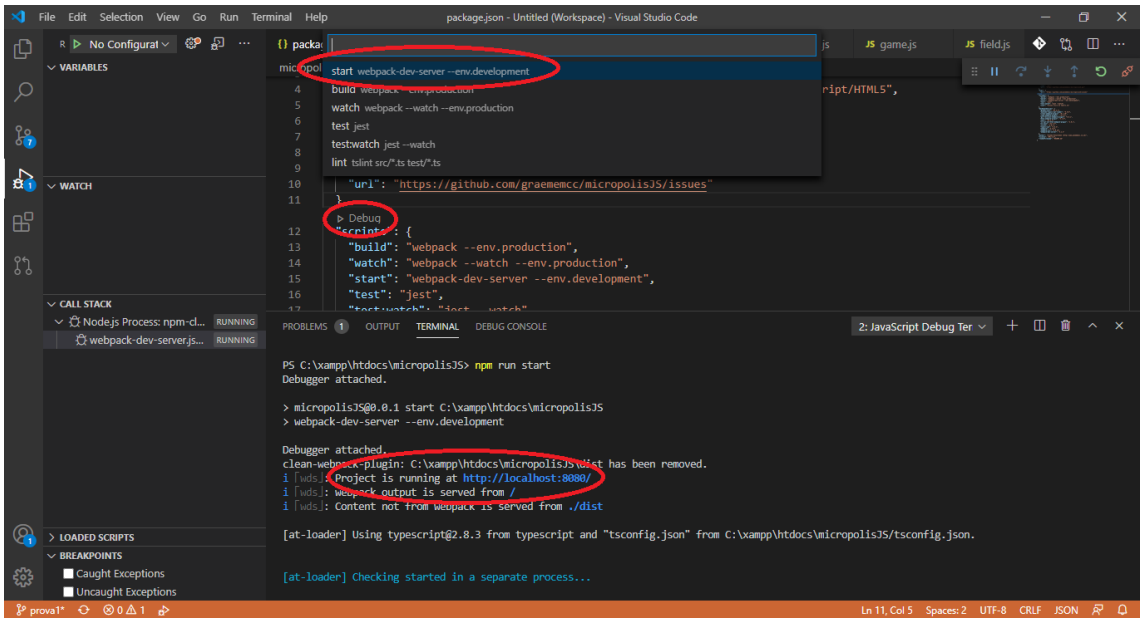

Figura 2.1: *Avvio server locale*

2.3 Panoramica su micropolisJS

Dopo un tempo di caricamento, l'applicazione web si presenta subito con una schermata in cui viene proposta una mappa che è possibile scartare generandone un'altra oppure confermare quella presentata; una volta scelta verrà chiesto il nome della città e la difficoltà con cui si desidera giocare. Infine si avvierà la sessione di gioco.

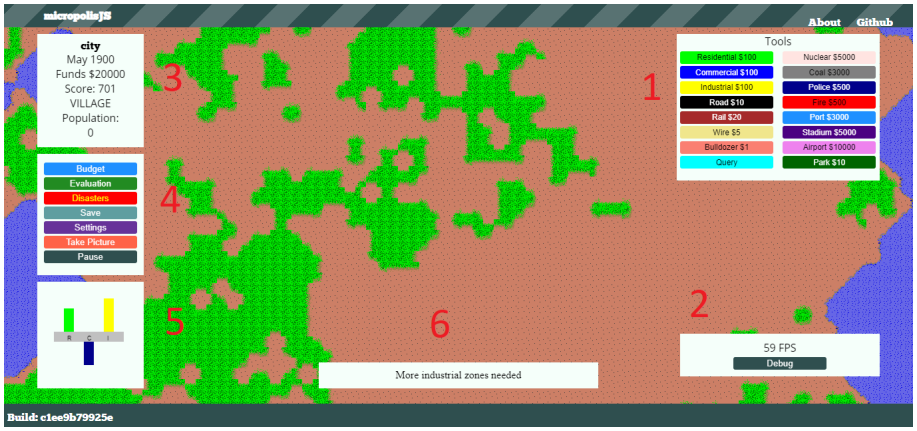

Figura 2.2: *Schermata di gioco*

La schermata che si presenta contiene:

1. La finestra degli strumenti in alto a destra racchiude i bottoni per poter posizionare i blocchi sulla mappa; abbiamo il blocco residenziale, commerciale, industriale, della stazione di polizia, dei vigili del fuoco, impianti di

energia elettrica, lo stadio, il porto, l'aeroporto, per la rete elettrica, le strade e le ferrovie, un blocco per il parco ed infine per la distruzione e per l'interrogazione di una singola mattonella della mappa (query) (fig. 2.3);

2. la finestra in basso a destra consente di usufruire di altri fondi; viene visualizzata solo in modalità di sviluppo per portare avanti il debugging;

3. la finestra in alto a sinistra espone quella che è la situazione attuale, ovvero il mese in cui ci si trova, i fondi rimasti, il punteggio e la popolazione che determina lo status della città;

4. la finestra centrata sulla sinistra contiene gli strumenti di opzione, dunque permette di salvare la partita, di cambiare alcune impostazioni, dal bottone 'Budget' si possono considerare i flussi di denaro potendo cambiare anche i tassi per le spese, dal tasto 'Evaluation' viene mostrato un riepilogo dell'opinione dei cittadini sul lavoro svolto e delle statistiche, mentre dal bottone 'Disasters' si può avviare un disastro (inondazione, fuoco, tornado, ecc);

5. la finestra RCI indica la situazione di richiesta delle tre aree: residenziale, commerciale e industriale;

6. la finestrella al centro in basso mostra i messaggi che appaiono per comunicare eventuali problemi e per richiedere una determinata risorsa per migliorare la situazione attuale in cui versa la città.

Il gameplay è di semplice utilizzo poiché è strutturato come un gioco gestionale, in cui il giocatore è il sindaco di una città che prende le decisioni con a disposizione un budget iniziale. L'obiettivo è sviluppare e gestire la città in modo da attrarre nuovi potenziali abitanti. Il giocatore sceglie come agire e può osservare la reazione e le conseguenze delle decisioni, in termini di popolazione, di questioni economiche e sociali tenendo conto delle finanze e della 'felicità' della popolazione.

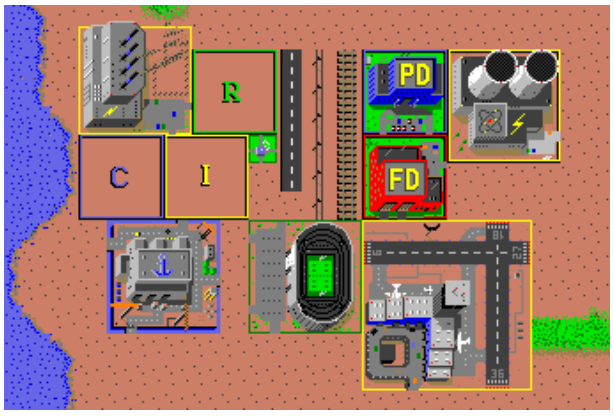

Figura 2.3: *Elementi del gioco*

Capitolo 3

Progettazione funzionalità

3.1 Elementi del sistema di riutilizzo delle acque

In questo capitolo verranno presentate le implementazioni degli elementi che compongono il nostro sistema per il riutilizzo delle acque reflue a scopo di irrigazione. Incominceremo subito col dire che il funzionamento del sistema è del tutto indipendente dalle variabili e dall'evoluzione degli elementi che costituiscono il gioco iniziale.
Il sistema è costituito da:

- **WWTP**: sta per "Waste Water Treatment Plant" ovvero dell'impianto per il trattamento e la sanificazione delle acque reflue, rappresenta il primo estremo attivo del sistema, ovviamente è indispensabile affinché abbia origine l'irrigazione;

- **Field**: ovvero il campo coltivato, costituisce il secondo estremo del sistema, in questo caso passivo, nel quale viene riversata l'acqua che proviene dall'impianto di trattamento;

- **Channel**: elemento di congiunzione tra l'impianto e il campo, destinato dunque alla canalizzazione dell'acqua per far confluire le acque sanificate nel terreno coltivabile;

- **IndField**: che sta per 'campo indipendente', è un oggetto "bonus" in quanto questo elemento costituisce di per sé un sistema autosufficiente comportando uno sviluppo che si distacca dall'effettivo sistema di irrigazione. Come verrà mostrato in seguito nei codici, questo elemento non sortisce alcun effetto di irrigazione cosiddetto "a cascata" nel caso in cui dovesse essere adiacente ad un campo normale che ha invece necessità di essere irrigato.

Di seguito verranno spiegate passo passo come sono stati implementati tali elementi e verranno forniti i codici utilizzati per una maggiore chiarezza.

3.2 Implementazione elementi

3.2.1 Utilizzo dei tiles

Essendo micropolisJS un gioco basato sull'interazione con la mappa, alla base del gioco esiste una logica che permette di utilizzare un *TileSet*, ovvero di un'immagine che contiene tutti gli elementi grafici, da quelli che formeranno la mappa a quelli che si potranno inserire giocando. Il TileSet (fig.3.1) chiamato `tiles.png`, fornita già dal gioco e appositamente modificato per le nostre esigenze grazie ad un programma di elaborazione delle immagini come *GIMP*, è una immagine 512x512 pixel ed ogni elemento è costituito da un insieme di *tile* "piastrella" di dimensione 16x16, per cui sono rappresentati 1024 tile. Per motivi di spazio abbiamo utilizzato i tile dei binari ferroviari per rappresentare il canale di irrigazione. D'ora in poi i binari saranno il nostro "channel".

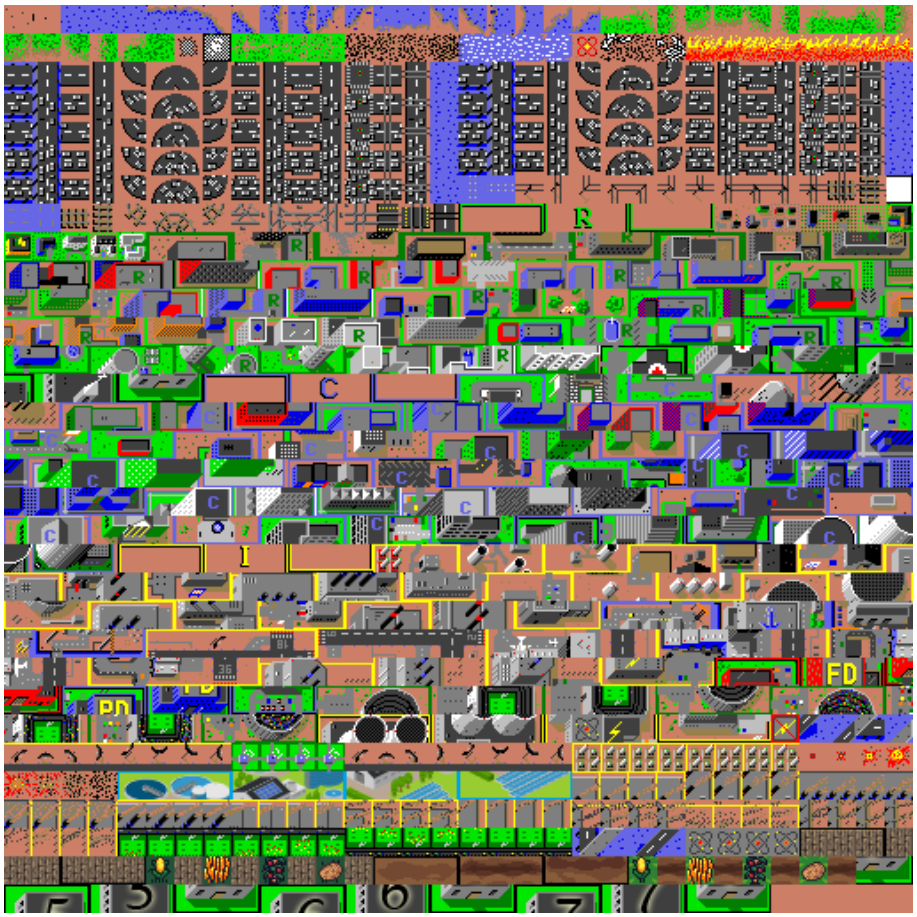

Figura 3.1: `tile.png`

Il file `tile.js` è utile nella gestione di questi tile consentendo di identificare il singolo tile e di riconoscerne le caratteristiche nel gioco mediante un *Flag*. Questo flag viene di volta in volta confrontato con una *BitMask*, un numero binario in cui ad ogni bit corrisponde un significato. L'assegnazione di un nome per ogni singolo tile è molto utile, perché consente di riferirci a quel tile

senza dover ricordare il numero che lo identifica in questa specie di matrice grafica. Per quanto riguarda il canale, abbiamo utilizzato dei nomi che contengono "tube" perché Tile.CHANNEL era già presente e per evitare ogni ambiguità ci siamo adattati. Considerando il listato 3.1 qui sotto, ad esempio la funzione `isHydraulic()` ritorna il valore `'true'` per tutti quei tile che sono settati come "conduttori di acqua", cioè con il bit 17(`HYDRABIT`) posto a 1, come il WWTP, il campo e il canale.

```
1  Tile.prototype.isHydraulic = function() {
2    return (this._value & Tile.HYDRABIT) > 0;
3  };
4
5  Tile.prototype.isIrrigated = function() {
6    return (this._value & Tile.IRRIGBIT) > 0;
7
8  // Bit-masks for statusBits
9  Tile.HYDRABIT = 0x20000; // bit 17, tile can run water.
10 Tile.IRRIGBIT = 0x10000; // bit 16, tile is irrigated.
11 Tile.POWERBIT  = 0x8000; // bit 15, tile has power.
12 Tile.CONDBIT = 0x4000; // bit 14. tile can conduct electricity.
13 Tile.BURNBIT = 0x2000; // bit 13, tile can be lit.
14 Tile.BULLBIT = 0x1000; // bit 12, tile is bulldozable.
15 Tile.ANIMBIT = 0x0800; // bit 11, tile is animated.
16 Tile.ZONEBIT = 0x0400; // bit 10, tile is center tile of the zone.
17 Tile.BLBNBIT   = Tile.BULLBIT | Tile.BURNBIT;
18 Tile.BLBNCNBIT = Tile.BULLBIT | Tile.BURNBIT | Tile.CONDBIT;
19 Tile.BNCNBIT   = Tile.BURNBIT | Tile.CONDBIT;
20 Tile.ASCBIT   = Tile.ANIMBIT | Tile.CONDBIT | Tile.BURNBIT;
21 Tile.BNHYBIT   = Tile.BURNBIT | Tile.HYDRABIT;
22 Tile.BLBNHYBIT = Tile.BULLBIT | Tile.BURNBIT | Tile.HYDRABIT;
23 Tile.ALLBITS = Tile.HYDRABIT | Tile.IRRIGBIT | Tile.POWERBIT | Tile.
       CONDBIT | Tile.BURNBIT | Tile.BULLBIT | Tile.ANIMBIT | Tile.
       ZONEBIT ;
24 Tile.BIT_START = 0x400;
25 Tile.BIT_END = 0x20000;
26 Tile.BIT_MASK = Tile.BIT_START - 1;
27
28 /* tubes lines */
29 Tile.VTUBE         = 224;
30 Tile.HTUBE         = 225;
31 Tile.LHTUBE        = 226;
32 Tile.LVTUBE        = 227;
33 Tile.LVTUBE2       = 228;
34 Tile.LVTUBE3       = 229;
35 Tile.LVTUBE4       = 230;
36 Tile.LVTUBE5       = 231;
37 Tile.LVTUBE6       = 232;
38 Tile.LVTUBE7       = 233;
39 Tile.LVTUBE8       = 234;
40 Tile.LVTUBE9       = 235;
41 Tile.LVTUBE10      = 236;
42 Tile.HTUBEROAD     = 237;
43 Tile.VTUBEROAD     = 238;
44 Tile.TUBEBASE      = Tile.HTUBE;
45 Tile.LASTTUBE      = 238;
46
47 //wwtp zone tiles
```

```
48 Tile.WWTPBASE       = 868;
49 Tile.WWTP           = 873;
50 Tile.LASTWWTP       = 883;
51
52 //field zone tiles
53 Tile.FIELDBASE      = 956;
54 Tile.FREEF          = 960;
55 Tile.CORN           = 965;
56 Tile.FCORN          = 966;
57 Tile.WHEAT          = 967;
58 Tile.FWHEAT         = 968;
59 Tile.ORCHARD        = 969;
60 Tile.FORCHARD       = 970;
61 Tile.POTATO         = 971;
62 Tile.FPOTATO        = 972;
63 Tile.FZB            = 975;
64
65 //INDIE field zone tiles
66 Tile.INDFIELDBASE    = 973;
67 Tile.FREEINDF        = 977;
68 Tile.INDFZB          = 975;
69 Tile.INDCORN         = 982;
70 Tile.INDFCORN        = 983;
71 Tile.INDWHEAT        = 984;
72 Tile.INDFWHEAT       = 985;
73 Tile.INDORCHARD      = 986;
74 Tile.INDFORCHARD     = 987;
75 Tile.INDPOTATO       = 988;
76 Tile.INDFPOTATO      = 989;
77
```

Listing 3.1: `tile.js`

Il file `tileUtils.js` invece permette di recuperare se un dato tile è impostato come "zona". Il tile centrale di ogni elemento, quindi di WWTP, residential, commercial, etc.. sono settati in questo modo per distinguerli l'uno dall'altro.

```
1  var isField = unwrapTile(function(tile) {
2    return (tile >= Tile.FIELDBASE && tile <= Tile.FPOTATO);
3  });
4
5  var isFieldZone = function(tile) {
6    return tile.isZone() && isField(tile);
7  };
8  var isIndField = unwrapTile(function(tile) {
9    return (tile >= Tile.INDFIELDBASE && tile <= Tile.INDFPOTATO)
10 });
11
12 var isIndFieldZone = function(tile) {
13   return tile.isZone() && isIndField(tile);
14 };
15
```

Listing 3.2: `tileUtils.js`

3.2.2 WWTP

WWTP, acronimo di *"Waste Water Treatment Plant"*, ovvero 'impianto di trattamento delle acque reflue è una struttura in cui vengono utilizzati una serie di processi (ad esempio, fisico, chimico e biologico) per trattare le acque reflue industriali e rimuovere gli inquinanti. In questo contesto viene utilizzato al fine di irrigare i campi con acqua pulita e riutilizzabile. L'implementazione di questo elemento viene gestito esattamente come le centrali di energia elettrica già presenti, con la sola differenza che l'idea è stata rivista per esprimere il concetto di erogazione idrica. L'implementazione del WWTP è stata realizzata in `PowerManager.js`: qui sono presenti anche le proprietà e metodi per le centrali elettriche. Allo stesso modo sono state definite analogie per il nostro `WWTP`. Inizialmente è fissata una costante che determina il limite massimo che una sola WWTP può supportare, in termini di tile; inoltre a titolo esemplificativo se consideriamo la griglia che mappa quali `tile` sono posti come irrigati `irrigateGridMap`, viene impiegata nello scan una prima volta per essere riempita, la seconda volta viene ripresa nel metodo `setTileIrrigate` aiutandoci ad individuare eventuali altri tile che devono essere definiti di tipo irrigato. La `wwtpPowerFound` non fa altro che contare quante WWTP ci sono nella mappa.

```
1  var COAL_POWER_STRENGTH = 700;
2  var WWTP_POWER_STRENGTH = 700;
3  var NUCLEAR_POWER_STRENGTH = 2000;
4
5
6  var PowerManager = EventEmitter(function(map) {
7    this._map = map;
8    this._powerStack = [];
9    this._irrigateStack = [];
10   this._setCropStack = [];
11   this.powerGridMap = new BlockMap(this._map.width, this._map.height,
       1);
12   this.irrigateGridMap = new BlockMap(this._map.width, this._map.
      height, 1);
13   this.costFieldMap = new BlockMap(this._map.width, this._map.height,
       1);
14 });
15
16 PowerManager.prototype.setTileIrrigate = function(x, y) {
17   var tile = this._map.getTile(x, y);
18   var tileValue = tile.getValue();
19
20   if (((tileValue >= Tile.WWTPBASE && tileValue <= Tile.LASTWWTP) ||
21       this.irrigateGridMap.worldGet(x, y) > 0) &&
22       (tileValue < Tile.INDFIELDBASE || tileValue > Tile.INDFPOTATO))
       {
23     tile.addFlags(Tile.IRRIGBIT);
24     return;
25   }
26
27   tile.removeFlags(Tile.IRRIGBIT);
28
29   PowerManager.prototype.wwtpPowerFound = function(map, x, y, simData
      ) {
```

```
30    simData.census.wwtpPowerPop += 1;
31
32    this._irrigateStack.push(new map.Position(x, y));
33    }
34  };
```

Listing 3.3: `PowerManager.js`

3.2.3 Channel

Il canale di irrigazione è l'elemento di congiunzione tra il campo e l'impianto di trattamento acque. Potremmo però dire che il canale non è strettamente necessario; nella figura qui sotto (fig.3.2) possiamo vedere come un campo, adiacente ad un altro campo già irrigato o al WWTP, risulta anch'esso irrigato.

Figura 3.2: *Utilizzo del* `channel`

Per l'implementazione del canale di irrigazione ci siamo ispirati al `wire`, già esistente, che permette di condurre corrente elettrica per far funzionare blocchi come `residential`, ma anche in questo caso `residential` risulterebbe powerizzata se adiacente ad una centrale elettrica.
Quando si vuole posare un `tile` del canale il metodo chiamato è `layChannel` in `channelTool.js`. Questo si preoccupa solo di capire su quale `tile` si va a costruire e il costo relativo; il resto del lavoro, cioè di guardarsi intorno per vedere che tipi di tile si sono, viene derogato all'oggetto `connector`.

```
1  ChannelTool.prototype.layChannel = function(x, y) {
2    this.doAutoBulldoze(x, y);
3    var cost = this.toolCost;
4    var tile = this._worldEffects.getTileValue(x, y);
5    tile = TileUtils.normalizeRoad(tile);
6    switch (tile) {
7      case Tile.DIRT:
8        this._worldEffects.setTile(x, y, Tile.LHTUBE, Tile.HYDRABIT |
     Tile.BURNBIT | Tile.BULLBIT);
```

```
9       break;
10
11    case Tile.RIVER:
12    case Tile.REDGE:
13    case Tile.CHANNEL:
14      cost = 25;
15
16      if (x < this._map.width - 1) {
17        tile = this._worldEffects.getTile(x + 1, y);
18        if (tile.isHydraulic()) {
19          tile = tile.getValue();
20          tile = TileUtils.normalizeRoad(tile);
21          if (tile != Tile.HTUBEROAD && tile != Tile.TUBEHPOWERV &&
    tile != Tile.HTUBE) {
22            this._worldEffects.setTile(x, y, Tile.VTUBE, Tile.
    HYDRABIT | Tile.BULLBIT);
23            break;
24          }
25        }
26      }
27
28      if (x > 0) {
29        tile = this._worldEffects.getTile(x - 1, y);
30        if (tile.isHydraulic()) {
31          tile = tile.getValue();
32          tile = TileUtils.normalizeRoad(tile);
33          if (tile != Tile.HTUBEROAD && tile != Tile.TUBEHPOWERV &&
    tile != Tile.HTUBE) {
34            this._worldEffects.setTile(x, y, Tile.VTUBE, Tile.
    HYDRABIT | Tile.BULLBIT);
35            break;
36          }
37        }
38      }
39
40      if (y < this._map.height - 1) {
41        tile = this._worldEffects.getTile(x, y + 1);
42        if (tile.isHydraulic()) {
43          tile = tile.getValue();
44          tile = TileUtils.normalizeRoad(tile);
45          if (tile != Tile.VTUBEROAD && tile != Tile.TUBEVPOWERH &&
    tile != Tile.VTUBE) {
46            this._worldEffects.setTile(x, y, Tile.HTUBE, Tile.
    HYDRABIT | Tile.BULLBIT);
47            break;
48          }
49        }
50      }
51
52      if (y > 0) {
53        tile = this._worldEffects.getTile(x, y - 1);
54        if (tile.isHydraulic()) {
55          tile = tile.getValue();
56          tile = TileUtils.normalizeRoad(tile);
57          if (tile != Tile.VTUBEROAD && tile != Tile.TUBEVPOWERH &&
    tile != Tile.VTUBE) {
58            this._worldEffects.setTile(x, y, Tile.HTUBE, Tile.
```

```
    HYDRABIT | Tile.BULLBIT);
59              break;
60            }
61          }
62        }
63        return this.TOOLRESULT_FAILED;
64
65      case Tile.ROADS:
66        this._worldEffects.setTile(x, y, Tile.HTUBEROAD, Tile.HYDRABIT
    | Tile.BURNBIT | Tile.BULLBIT);
67        break;
68
69      case Tile.ROADS2:
70        this._worldEffects.setTile(x, y, Tile.VTUBEROAD, Tile.HYDRABIT
    | Tile.BURNBIT | Tile.BULLBIT);
71        break;
72
73      case Tile.LHPOWER:
74        this._worldEffects.setTile(x, y, Tile.TUBEVPOWERH, Tile.
    HYDRABIT | Tile.BURNBIT | Tile.BULLBIT);
75        break;
76
77      case Tile.LVPOWER:
78        this._worldEffects.setTile(x, y, Tile.TUBEHPOWERV, Tile.
    HYDRABIT | Tile.BURNBIT | Tile.BULLBIT);
79        break;
80
81      default:
82        return this.TOOLRESULT_FAILED;
83    }
84
85    this.addCost(cost);
86    this.checkZoneConnections(x, y);
87    return this.TOOLRESULT_OK;
88  };
```

Listing 3.4: `channelTool.js`

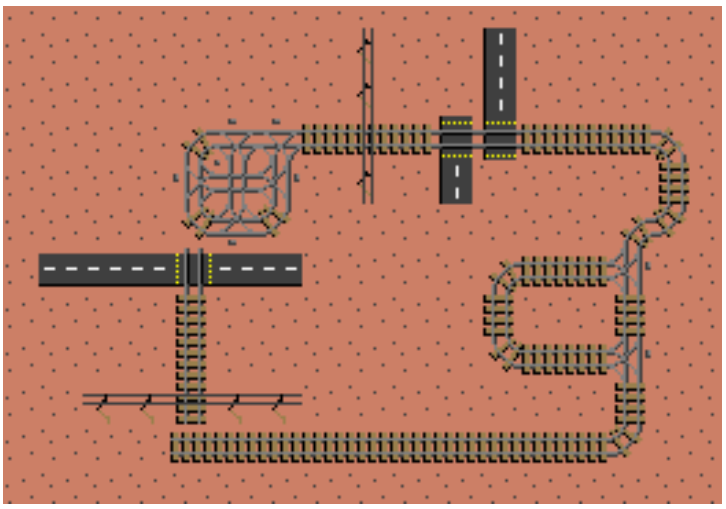

Figura 3.3: *Tile di* `channel` *e sue intersezioni*

3.2.4 Field

Inizialmente la logica dell'oggetto di tipo field è stata copiata dal residential, in seguito sono state apportate delle modifiche. In primo luogo rispetto al residential, il nostro campo non è più subordinato all'allaccio della rete elettrica, ma è dipendente dalla presenza e dalla connessione con il WWTP. Il campo ha dunque la necessità di essere irrigato tramite il canale, se questa operazione non avviene, il campo verrà si posizionato sulla mappa ma non si vedrà crescere la coltivazione. Per come è organizzato il gioco, è possibile irrigare un campo anche se il WWTP è posto adiacente ad esso, perciò un campo risulterà irrigato se un tile al suo fianco risulta irrigato.

```
1  var fieldFound = function(map, x, y, simData) {
2    var lpValue;
3    var tile = map.getTileValue(x, y);
4    var zoneIrrigate = map.getTile(x, y).isIrrigated();
5    var prevTile = map.getTileValue(x-1, y);
6    var cost=0;
7    if( prevTile !== (Tile.FREEINDF-1))
8    {
9    if(zoneIrrigate) {
10     simData.census.fieldZonePop += 1;
11     cost = simData.powerManager.costFieldMap.get(x, y);
12     switch(cost){
13       case BaseTool.CORN_COST:
14         tile = Tile.CORN;
15         break;
16
17       case BaseTool.WHEAT_COST:
18         tile = Tile.WHEAT;
19         break;
20
21       case BaseTool.ORCHARD_COST:
22         tile = Tile.ORCHARD;
23         break;
24
25       case BaseTool.POTATO_COST:
26         tile  = Tile.POTATO;
27         break;
28
29       default: break;
30     }
31   }
32   else{
33     tile = Tile.FREEF;
34 map.setTile(x, y, tile, Tile.BLBNHYBIT | Tile.ZONEBIT);
35
```

Listing 3.5: `field.js`

La funzione richiamata per gestire lo sviluppo di un field si chiama FieldFound, cioè "campo trovato", perché nella dinamica ciclica consistente anche nello scansionare la mappa, viene

3.2.5 Independent field

Il progetto prevede anche la creazione di un campo indipendente dall'impianto WWTP, questo perché almeno nelle prime fasi del gioco il giocatore vorrà sicuramente costruire una città e per evitare un gran esborso di fondi dovuto al dover comprare un WWTP, è stato introdotto questo campo autosufficiente che non dipende dalla condizione di irrigazione, perciò quando viene utilizzato verrà sin da subito visualizzata la coltivazione che si è scelta. L'introduzione di questo campo ha sollevato alcune problematiche; non era possibile utilizzare le stesse tessere utilizzare per il field poiché il gioco è fortemente basato sui loro valori di *FLAG*, si è scelto dunque di utilizzare altre tessere, con altre immagini, in modo da poter facilmente differenziare l'uno dall'altro. Il problema che si riscontrava era che se entrambi facevano riferimento alle stesse tessere, quelle stesse tessere venivano riconosciute sia come field che come indField, per cui quando veniva richiamata la `action`, il `"criterion"`, che nel caso del campo indipendente è `isIndFieldZone()`, permetteva di entrare nella `current.action FieldFound` anche se il tile su cui si stava operando era del campo indipendente.

```
for (var i = 0, l = this._actions.length; i < l; i++) {
    var current = this._actions[i];
    var callable =isCallable(current.criterion);
    if (callable && current.criterion.call(null, tile)) {
        current.action.call(null, this._map, x, y, simData);
        break;
    } else if (!callable && current.criterion === tileValue) {
        current.action.call(null, this._map, x, y, simData);
        break;
        }
      };
```

Listing 3.6: parte di `mapScan` di `mapScanner.js`

3.2.6 Scelta del campo e della coltura

Abbiamo deciso di dare la possibilità di scegliere uno e l'altro tramite un unico bottone `field` con l'onere di implementare un'unica finestra nel quale vengono richieste al giocatore alcune informazioni per capire la sua scelta.

3.2.6.1 Gestione della fieldWindow

Nel listato qui sotto c'è contenuto di `fieldWindow.js` che insieme ad altri metodi definiti in `game.js` e in `baseTool.js` permettono un giusto scambio di dati che ci porta poi a poter utilizzare il componente `buildingTool` con i dati inseriti dal giocatore. Per maggiori dettagli sul passaggio dei dati potete consultare la tesi del collega con cui ho lavorato al progetto, Coricelli Tommaso, al paragrafo 4.3 "Scelta del campo e del tipo di coltura".

```
var FieldWindow = ModalWindow(function() {
  $(fieldFormID).on('submit', submit.bind(this));
},);

var cropCornID = '#cropCorn';
```

```
6  var cropPotatoID = '#cropPotato';
7  var cropWheatID = '#cropWheat';
8  var cropOrchardID = '#cropOrchard';
9  var fieldFormID = '#fieldForm';
10 var fieldOKID = '#fieldOK';
11 var WWTPYesID = '#WWTPYes';
12 var WWTPNoID = '#WWTPNo';
13
14
15 FieldWindow.prototype.close = function(actions) {
16   actions = actions || [];
17   this._emitEvent(Messages.FIELD_WINDOW_CLOSED, actions);
18   this._toggleDisplay();
19 };
20
21
22 var submit = function(e) {
23   e.preventDefault();
24
25   var actions = [];
26   var shouldWWTP =$('.WWTPField:checked').val();
27   if (shouldWWTP === 'true')
28     shouldWWTP = true;
29   else
30     shouldWWTP = false;
31   actions.push({action: FieldWindow.WWTP, data: shouldWWTP});
32
33   var cropSelect = $('.cropSetting:checked').val() - 0;
34   actions.push({action: FieldWindow.CROP, data: cropSelect});
35   this.close(actions);
36 };
37
38
39 FieldWindow.prototype.open = function(fieldData) {
40     $(WWTPYesID).prop('checked', true);
41     $(cropCornID).prop('checked', true);
42      this._toggleDisplay();
43 };
44
45
46 var defineAction = (function() {
47   var uid = 0;
48
49   return function(name) {
50     Object.defineProperty(FieldWindow, name, MiscUtils.
     makeConstantDescriptor(uid));
51     uid += 1;
52   };
53 })();
54
55 defineAction('WWTP');
56 defineAction('CROP');
57
58 export { FieldWindow };
```

Listing 3.7: `fieldWindow.js`

3.2.6.2 BuildingTool

A prescindere dell'importanza di questo oggetto riguardante la catena di passaggi dei valori per consentire al giocatore di inserire il giusto `field` o `indiefield`, è importante perché viene richiamato ogni qual volta si vuole installare una struttura sulla mappa con i giusti valori di `FLAG` per ogni `tile` di ogni tipologia di struttura. Il contenuto di questo file sarà ripreso successivamente per mostrare la modalità di come vengono differenziati già dalla fase iniziale i campi.

La distinzione tra il `Field` e l'`indieField` ha richiesto l'utilizzo di `tile` diverse nonostante facessero comunque parte di una stessa entità. É stata una azione obbligata perché, e lo ribadisco, la logica è altamente dipendente dai valori di `FLAG` dei `tile`, che durante lo `scan` fa la differenza: infatti come già anticipato nella presentazione degli elementi è proprio in `mapScan`, con la complicità del valore del `FLAG`, che evitiamo la possibilità a `indiefield` di poter irrigare per "contatto"(fig. 3.4).

```
1  MapScanner.prototype.mapScan = function(startX, maxX, simData) {
2    for (var y = 0; y < this._map.height; y++) {
3      for (var x = startX; x < maxX; x++) {
4        this._map.getTile(x, y, tile);
5        var tileValue = tile.getValue();
6
7        if (tileValue < Tile.FLOOD)
8          continue;
9
10       if (tile.isConductive())
11         simData.powerManager.setTilePower(x, y);
12
13       if (tile.isHydraulic())
14         simData.powerManager.setTileIrrigate(x, y);
15
16       if(tileutils.isFieldZone(tile) || tileutils.isIndFieldZone(tile
     ))
17         simData.powerManager.setCostCrop(x,y);
18
19       if (tile.isZone()) {
20         simData.repairManager.checkTile(x, y, simData.cityTime);
21         var powered = tile.isPowered();
22         if (powered)
23           simData.census.poweredZoneCount += 1;
24         else
25           simData.census.unpoweredZoneCount += 1;
26       }
```

Listing 3.8: parte del metodo di `mapScan` in `mapScanner.js`

Figura 3.4: `indiefield` *non permette a* `field` *di essere irrigato per conduzione*

Gli argomenti fin qui trattati sono comunque definiti in maniera più esplicita e dettagliata nella tesi del collega col quale ho condiviso questo progetto, Coricelli Tommaso.

3.3 Mantenimento: stabilità e degrado

Come è stato anticipato nella sezione dedicata ai `field` e `indiefield` oltre al costo iniziale che cambia rispetto al tipo di coltivazione che si sceglie c'è da sostenere un costo di manutenzione. Perché questa decisione? Perché proprio come succede però nella vita reale, se un campo agricolo non viene curato, mantenuto e viene trascurato, il terreno inizierà a non essere più fruttuoso e perdere i suoi frutti. Quindi per simulare questo fenomeno di degrado e abbandono nel gioco, si è deciso di ricreare una logica che penalizzasse uno o più campi nel caso in cui il giocatore non avesse i fondi a disposizione per poterli mantenere: se il giocatore si dovesse trovare in questa situazione vedrà il decadimento dei campi. Il degrado, da un punto di vista grafico, non è altro che il ritorno ad una situazione per cui il campo non presenta più il `tile` della coltivazione centrale nonostante il terreno possa risultare ancora irrigato. Ma andiamo con ordine.

3.3.1 Similitudini `field` e `indfield`

Il `field` e il Il `indiefield` sono perciò, tra gli elementi da noi integrati in questo gioco, quelli che hanno bisogno di essere caratterizzati da un ciclo di vita attivo e visibile. É stato pensato, oltre alla dipendenza dal WWTP, di rendere possibile la scelta tra 4 colture, sia per il primo che per il secondo tipo di campo, ognuna con un costo diverso, che può in effetti essere sostenuto in diverse fasi del gioco. Questa scelta viene domandata praticamente quando si decide di piazzare un terreno col meccanismo della `fieldWindow` sul quale ci siamo soffermati in un paragrafo precedente. Per quanto riguarda il mantenimento e dunque la tassazione non esistono differenze: nonostante l'`indfield` sia indipendente dal WWTP, anch'esso è subordinato al pagamento annuale del mantenimento.

Altre analogie esistono tra le due tipologie: in primis si possono certamente valorizzare alcune differenze che li contraddistinguono dal resto delle strutture quali `residential`, `commercial` e `industrial` perché non prevedono né in fase di costruzione, né nel proseguo del gioco, di una logica che prenda in considerazione l'aumento della popolazione, e conseguentemente a tale caratteristica statica che non contribuisce alla progressione della città non è previsto uno sviluppo ulteriore a quello della vista della coltivazione.

3.3.2 Crescita e stabilità dei campi

Come già visto, il `field` e l'`indfield` inizialmente presentano un comportamento diverso l'uno dall'altro. La principale distinzione è ottenuta nel metodo `putBuilding` nel file `buildingTool.js` nel quale, relativamente alla scelta del campo ed in base al valore identificativo del `tile` (definito in `tile.js`), vengono direttamente inizializzati con valori diversi per avere una netta separazione. Inizialmente un `indfield` rispetto ad un `field` verrà visualizzato subito con la coltivazione nonostante come si può notare dal listato 3.9 qui sotto, ad esempio per quanto riguarda la coltivazione del mais (corn), i valori di `INDFCORN` e di `FCORN` sono valori che identificano nell'immagine dei tile 2 tessere vuote con due valori di riferimento diversi.

```
1  BuildingTool.prototype.putBuilding = function(leftX, topY) {
2    var posX, posY, tileValue, tileFlags;
3   // var baseTile = this.centreTile - this.size - 1;
4    var baseTile;
5    var b = BaseTool.getWWTP();
6    var c = BaseTool.getCropCost();
7
8    if(this.centreTile == Tile.FREEF || this.centreTile == Tile.
       FREEINDF) {
9      if(b) {this.centreTile = Tile.FREEF;}
10     else this.centreTile = Tile.FREEINDF;
11     this.addCost(c);
12   }
13
14   baseTile = this.centreTile - this.size - 1;
15
16   for (var dy = 0; dy < this.size; dy++) {
17     posY = topY + dy;
18
19     for (var dx = 0; dx < this.size; dx++) {
20       posX = leftX + dx;
21       tileValue = baseTile;
22
23       if (TileUtils.isIndField(tileValue))
24       {
25         if (dx === 1 && dy === 1 && (tileValue === Tile.FREEINDF))
26         {
27           switch (c){
28             case BaseTool.CORN_COST:
29               tileValue = Tile.INDFCORN;
30               break;
31
32             case BaseTool.WHEAT_COST:
```

```
33              tileValue = Tile.INDFWHEAT;
34              break;
35
36            case BaseTool.ORCHARD_COST:
37              tileValue = Tile.INDFORCHARD;
38              break;
39
40            case BaseTool.POTATO_COST:
41              tileValue = Tile.INDFPOTATO;
42              break;
43
44            default: break;
45          }
46        }
47      tileFlags = Tile.BURNBIT;
48      }
49        else if (TileUtils.isField(tileValue) ||(tileValue>=Tile.
    WWTPBASE && tileValue<=Tile.LASTWWTP))
50        {
51        if (dx === 1 && dy === 1 && (tileValue === Tile.FREEF))
52        {
53          switch (c){
54            case BaseTool.CORN_COST:
55              tileValue = Tile.FCORN;
56              break;
57
58            case BaseTool.WHEAT_COST:
59              tileValue = Tile.FWHEAT;
60              break;
61
62            case BaseTool.ORCHARD_COST:
63              tileValue = Tile.FORCHARD;
64              break;
65
66            case BaseTool.POTATO_COST:
67              tileValue = Tile.FPOTATO;
68              break;
69
70            default: break;
71             }
72           }
73        tileFlags = Tile.BNHYBIT;
74      }
75      else
76        tileFlags = Tile.BNCNBIT;
77
78      if (dx === 1) {
79        if (dy === 1){
80          tileFlags |= Tile.ZONEBIT;
81        }
82        else if (dy === 2 && this.animated)
83          tileFlags |= Tile.ANIMBIT;
84      }
85
86      this._worldEffects.setTile(posX, posY, tileValue, tileFlags);
87
88      baseTile++;
```

```
89      }
90    }
91  };
```

Listing 3.9: metodo `putBuilding` in `buildingTool.js`

L'iniziale crescita delle coltivazioni in effetti non avviene qui, ma in una delle funzioni dei file che determinano il comportamento dei relativi campi. Il tempo di latenza di comparsa della coltivazione a partire dall'apposizione effettiva del campo che riguarda il `indiefield`, ed in parte anche per `field` a condizione che venga aggiunto vicino a un tile che irrighi, è dovuto al tempo di arrivo della simulazione alla funzione `mapScan` dell'oggetto `mapScanner`. Qui sotto il listato di una sezione della funzione di scan dove viene riconosciuto e avviato il metodo di crescita e vedremo anche che serve per la stabilità.

```
1
2   for (var i = 0, l = this._actions.length; i < l; i++) {
3         var current = this._actions[i];
4         var callable = isCallable(current.criterion);
5
6         if (callable && current.criterion.call(null, tile)) {
7           current.action.call(null, this._map, x, y, simData);
8           break;
9         } else if (!callable && current.criterion === tileValue) {
10          current.action.call(null, this._map, x, y, simData);
11          break;
12        }
13      }
```

Listing 3.10: metodo `mapScan`

3.3.2.1 field.js

L'implementazione della possibile crescita e possibile stabilità del `field` è affidata alla funzione `fieldFound`. Qui prima di tutto vengono passate le coordinate `x,y` del tile `ZONE`, cioè della tessera centrale del campo, in seguito viene determinato se questo `tile` nel corso del gioco è stato settato come irrigato assegnano alla variabile `zoneIrrigate` un valore che a questo punto è booleano, per cui un valore `true` corrisponde alla presenza di irrigazione; in tal caso si deve cambiare il `tile`, ad esempio da `Tile.FWHEAT` a `Tile.WHEAT`, ciò viene fatto in base alla condizione di `switch` confrontando il valore del costo estratto dall'oggetto `costFieldMap` nella posizione corrispondente al `tile x, y`, con il costo sostenuto nella costruzione della coltivazione iniziale.

```
1  var fieldFound = function(map, x, y, simData) {
2    var lpValue;
3    var tile = map.getTileValue(x, y);
4    var zoneIrrigate = map.getTile(x, y).isIrrigated();
5    var cost = 0;
6    if(zoneIrrigate) {
7      // Notify the census
8      simData.census.fieldZonePop += 1;
9      cost = simData.powerManager.costFieldMap.get(x, y);
10     switch(cost){
```

```
11        case BaseTool.CORN_COST:
12          tile = Tile.CORN;
13          break;
14
15        case BaseTool.WHEAT_COST:
16          tile = Tile.WHEAT;
17          break;
18
19        case BaseTool.ORCHARD_COST:
20          tile = Tile.ORCHARD;
21          break;
22
23        case BaseTool.POTATO_COST:
24          tile  = Tile.POTATO;
25          break;
26
27        default: break;
28      }
29    }
30    else{
31      tile = Tile.FREEF;
32      }
33      map.setTile(x, y, tile, Tile.BLBNHYBIT | Tile.ZONEBIT);
```

Listing 3.11: parte del metodo `fieldFound` per la crescita e stabilizzazione della coltura

3.3.2.2 indfield.js

Il comportamento dell'`indiefield` è gestito nel metodo `indfieldFound`. Non sussistono alcune differenze a quanto esposto per field.js, l'unica differenza è implicita nella definizione della loro differente natura: in `indfieldFound` il cambio dei `tile` non è preceduto dalla condizione di vedere se il campo è irrigato.

3.3.3 Condizione di mantenimento e degrado

Perché ho definito tali funzioni anche come responsabile della stabilità nel tempo delle colture? Semplicemente perché ogni volta che viene trovata sulla mappa un campo, le funzioni `fieldFound e indfieldFound` e le relative verifiche vengono eseguite sempre, per cui si ha un controllo e un cambio continuo e repentino dei `tile`, ma essendo immediatamente seguite dalla funzione di `degradeZone`, quest'ultima assume una "priorità" maggiore con la capacità di ribaltare i cambiamenti che sono stati fatti precedentemente, in effetti `degradeZone` fa il contrario di quanto detto in precedenza. Vedremo però che la chiamata di questa funzione è subordinata alla condizione di un'altra funzione di cui spiegherò i dettagli; si tratta del metodo `shouldDegradeField` contenuto in `budget.js`.

3.3.3.1 budget.js

Il metodo `shouldDegradeField` è scritto in questo file il quale, e come suggerisce il nome stesso, è responsabile del calcolo dei fondi, dei costi, delle tasse e del flusso di denaro che si genera una volta che si iniziano a costruire i blocchi, le infrastrutture e quando inizia a crescere l'affluenza di nuovi cittadini. Anche

in questo caso abbiamo sfruttato una logica di pagamento già presente nel programma e nello specifico abbiamo preso spunto per quanto fatto per la `road`.
In realtà `shoudlDegradeField` presenta un corpo semplice e snello, rappresentano nel listato 3.12 qui sotto, nel quale viene ritornato un valore determinato dal confronto di `this.fieldEffect` con un numero sempre costante (pari a 600) determinato a partire dalla costante `MAX_FIELD_EFFECT` definita nell'oggetto `budget`. Il valore ritornato può quindi essere `false` nel caso in cui `this.effectField` sia minore della costante, `true` nell'altro caso; è in quest'ultima evenienza che si avrà accesso alla funzione `degradeZone` in `field.js`.

```
1  Budget.prototype.shouldDegradeField = function() {
2    return this.fieldEffect < Math.floor(15 * this.MAX_FIELD_EFFECT /
       25);
3  };
```

Listing 3.12: funzione `shouldDegradeField` in `budget.js`

Il valore di `fieldEffect` esprime l'effetto che il livello della spesa stabilito nella finestra di budget ha sul deterioramento dei campi. Di seguito verranno presentati diversi estratti di `budget.js` per spiegare in che modo viene calcolato `fieldEffect`.
Il primo metodo che viene richiamato dal `simulation` è `budget.collectTax`; la chiamata viene effettuata ogni Gennaio del gioco, quindi possiamo dire che i calcoli sulle spese e sulle tasse e il conseguente aumento o detrazione dei fondi viene fatto all'inizio di ogni anno.

```
1  Budget.prototype.collectTax = function(gameLevel, census) {
2    this.cashFlow = 0;
3
4    // How much would it cost to fully fund every service?
5    [...]
6    this.fieldMaintenanceBudget = ( census.fieldZonePop + census.
       indfieldZonePop ) * fieldMaintenanceCost;
7
8    [...]
9
10   this.taxFund = Math.floor(Math.floor(census.totalPop * census.
       landValueAverage / 120) * this.cityTax * FLevels[gameLevel]);
11
12   if (census.totalPop > 0) {
13     this.cashFlow = this.taxFund - (this.policeMaintenanceBudget +
       this.fireMaintenanceBudget + this.roadMaintenanceBudget + this.
       fieldMaintenanceBudget);
14     this.doBudgetNow(false);
15   } else {
16     // We don't want roads etc deteriorating when population hasn't
       yet been established
17     // (particularly early game)
18     this.roadEffect   = this.MAX_ROAD_EFFECT;
19     this.policeEffect = this.MAX_POLICESTATION_EFFECT;
20     this.fireEffect   = this.MAX_FIRESTATION_EFFECT;
21     this.fieldEffect  = this.MAX_FIELD_EFFECT;
22   }
```

```
};
```

Listing 3.13: metodo `collectTax` di `budget.js`

In `collectTax` innanzitutto viene calcolato il costo totale per la manutenzione annuale; il calcolo viene fatto su una quota fissa pari a 100$ per terreno che viene dunque moltiplicata per il numero totale di campi. Viene poi calcolata la tassa (`taxFund`) che la popolazione deve versare: se non si è insediata ancora nessuna popolazione il `fieldEffect` viene posto al massimo per evitare condizione di deterioramento (ricordando che la condizione è verificata se minore di una costante), se dovesse esserci qualcuno si calcola il flusso(`cashFlow`) di denaro che può determinare entrate, o al contrario determinare delle perdite; dopo questo calcolo si entra nel metodo `doBudgetNow` che come prima azione richiama il metodo `_calculateBestPercentages`.

```
Budget.prototype._calculateBestPercentages = function() {
  // How much would we be spending based on current percentages?
  // Note: the *Budget items are updated every January by collectTax
  this.fieldSpend = Math.round(this.fieldMaintenanceBudget * this.
    fieldPercent)
  this.roadSpend = Math.round(this.roadMaintenanceBudget * this.
    roadPercent);
  this.fireSpend = Math.round(this.fireMaintenanceBudget * this.
    firePercent);
  this.policeSpend = Math.round(this.policeMaintenanceBudget * this.
    policePercent);
  var total = this.roadSpend + this.fireSpend + this.policeSpend +
    this.fieldSpend;

  // If we don't have any services on the map, we can bail early
  if (total === 0) {
    this.roadPercent = 1;
    this.firePercent = 1;
    this.policePercent = 1;
    this.fieldPercent = 1;
    return {road: 1, fire: 1, police: 1, field: 1};
  }

  // How much are we actually going to spend?
  var roadCost = 0;
  var fireCost = 0;
  var policeCost = 0;
  var fieldCost = 0;

  var cashRemaining = this.totalFunds + this.taxFund;

  // Spending priorities: road, fire, police, field
  1.[...]

  if (cashRemaining >= this.fieldSpend)
    fieldCost = this.fieldSpend;
  else
    fieldCost = cashRemaining;
  cashRemaining -= fieldCost;


  2.[...]
```

```
38
39    if (this.fieldMaintenanceBudget > 0)
40      this.fieldPercent = (fieldCost / this.fieldMaintenanceBudget).
        toPrecision(2) - 0;
41    else
42      this.fieldPercent = 1;
43
44    return {road: roadCost, police: policeCost, fire: fireCost, field:
        fieldCost};
45  };
```

Listing 3.14: metodo `_calculateBestPercentages` di `budget.js`

`this.fieldSpend` rappresenta il pagamento messo da parte per i campi, calcolato come valore di mantenimento totale, moltiplicato `this.fieldPercent` che esprime la percentuale sul costo di mantenimento unitario, questo valore può essere cambiato nello *slider* della finestra del budget, quindi ad esempio se vogliamo destinare solo il 75% `this.fieldPercent` sarà uguale a 0.75. Successivamente viene determinata la spesa totale; se questo totale è nullo (non ci sono terreni, strade, stazioni di polizia e dei pompieri) il percentuale viene rimesso al 100%. Poi viene calcolato il denaro totale comprensivo di tasse riscosse: se tale valore è maggiore del costo, `fieldCost` assume il valore della spesa dei terreni, altrimenti ci troveremmo a voler spendere più del denaro che abbiamo (`cashRemaining` sarebbe minore della spesa `fieldSpend`), per cui il costo dei campi è posto uguale al denaro rimasto; in entrambi i casi la spesa, `fieldCost`, viene sottratta ai fondi disponibili. Successivamente viene verificato se il budget di mantenimento totale è maggio di 0 (ovvero se abbiamo dei campi attivi), in questo caso il `fieldPercent` viene calcolato in relazione al costo che possiamo supportare (che può essere anche pari a 1 se, come visto prima, il `cashRemaining` è maggiore di tutta la spesa). Dopodicché ritorna i valori che verranno ripresi in `doBudgetNow`.

```
1  Budget.prototype.doBudgetNow = function(fromWindow) {
2    var costs = this._calculateBestPercentages();
3
4    if (!this.autoBudget && !fromWindow) {
5      this.autoBudget = false;
6      this.awaitingValues = true;
7      this._emitEvent(Messages.BUDGET_NEEDED);
8      return;
9    }
10
11   var fieldCost = costs.field;
12   var roadCost = costs.road;
13   var policeCost = costs.police;
14   var fireCost = costs.fire;
15   var totalCost = roadCost + policeCost + fireCost + fieldCost;
16   var cashRemaining = this.totalFunds + this.taxFund - totalCost;
17
18   // Autobudget
19   if ((cashRemaining > 0 && this.autoBudget) || fromWindow) {
20     // Either we were able to fully fund services, or we have just
       normalised user input. Go ahead and spend.
21     this.awaitingValues = false;
22     this.doBudgetSpend(roadCost, fireCost, policeCost, fieldCost);
```

```
23      return;
24    }
25
26    // Uh-oh. Not enough money. Make this the user's problem.
27    // They don't know it yet, but they're about to get a budget window
        .
28    this.setAutoBudget(false);
29    this.awaitingValues = true;
30    this._emitEvent(Messages.BUDGET_NEEDED);
31    this._emitEvent(Messages.NO_MONEY);
32  };
```

Listing 3.15: metodo doBudgetNow di budget.js

La variabile `costs` contiene i valori dei costi che vengono poi assegnati a singole variabili. Viene dunque calcolato il costo totale `totalCost` di tutti i servizi e calcolato l'effettivo denaro che si avrebbe a disposizione `cashRemaining`. Se questo valore è positivo allora viene eseguito il metodo `doBudgetSpend`

```
1   Budget.prototype.doBudgetSpend = function(roadValue, fireValue,
        policeValue, fieldValue) {
2     this.fieldSpend = fieldValue;
3     this.roadSpend = roadValue;
4     this.fireSpend = fireValue;
5     this.policeSpend = policeValue;
6     var total = this.roadSpend + this.fireSpend + this.policeSpend +
        this.fieldSpend;
7
8     this.spend(-(this.taxFund - total));
9     this.updateFundEffects();
10  };
```

Listing 3.16: metodo doBudgetSpend di budget.js

che a sua volta richiama i metodi `this.spend` per detrarre effettivamente i costi calcolati e `this.updateFundEffects`

```
1   Budget.prototype.updateFundEffects = function() {
2     // The caller is assumed to have correctly set the percentage spend
3     this.roadSpend = Math.round(this.roadMaintenanceBudget * this.
        roadPercent);
4     this.fireSpend = Math.round(this.fireMaintenanceBudget * this.
        firePercent);
5     this.policeSpend = Math.round(this.policeMaintenanceBudget * this.
        policePercent);
6     this.fieldSpend = Math.round(this.fieldMaintenanceBudget * this.
        fieldPercent);
7
8     // Update the effect this level of spending will have on
        infrastructure deterioration
9     this.roadEffect = this.MAX_ROAD_EFFECT;
10    this.policeEffect = this.MAX_POLICESTATION_EFFECT;
11    this.fireEffect = this.MAX_FIRESTATION_EFFECT;
12    this.fieldEffect = this.MAX_FIELD_EFFECT;
13
14    if (this.roadMaintenanceBudget > 0)
15      this.roadEffect = Math.floor(this.roadEffect * this.roadSpend /
        this.roadMaintenanceBudget);
16
```

```
17    if (this.fireMaintenanceBudget > 0)
18      this.fireEffect = Math.floor(this.fireEffect * this.fireSpend /
        this.fireMaintenanceBudget);
19
20    if (this.policeMaintenanceBudget > 0)
21      this.policeEffect = Math.floor(this.policeEffect * this.
        policeSpend / this.policeMaintenanceBudget);
22
23    if (this.fieldMaintenanceBudget > 0)
24      this.fieldEffect = Math.floor(this.fieldEffect * this.fieldSpend
        / this.fieldMaintenanceBudget);
25
26  };
```

Listing 3.17: metodo `updateFundEffects` di `budget.js`

che assegna i valori degli effetti in relazione alle spese che concretamente possiamo supportare.
Dopo tutte queste chiamate si ritorna al metodo `collectTax` che, essendo stata chiamata dal `simulate` dell'oggetto `simulation`, aggiorna periodicamente i valori delle spese.
A questo punto nella continua simulazione verrà eseguita `fieldFound` che, oltre al cambio del `tile` centralein condizioni di irrigamento, racchiude la verifica del listato 3.18, tale per cui, il valore di ritorno di `shouldDegradeField` (listato 3.12) che dipende dal `fieldEffect` aggiornato, determina l'esecuzione di `degradeZone`. Se `fieldEffect` dovesse essere maggiore della costante saremo in condizioni di **STABILITA'** della coltivazione perchè non verrà eseguita `degradeZone`, altrimenti `shouldDegradeField` assume valore `false` e si entra nella struttura condizionale del listato qui sotto che ne determina il **DEGRADO**.

```
1
2   if(simData.budget.shouldDegradeField()){
3       lpValue = ZoneUtils.getLandPollutionValue(simData.blockMaps, x, y
        );
4       degradeZone(map, x, y, simData.blockMaps, population, lpValue,
        zoneIrrigate);
5       return;
```

Listing 3.18: verifica per il degrado in `field.js`

Siamo arrivati al metodo `degradeZone`, che considera lo stesso `tile` in termini posizionali considerato in precedenza in `fieldFound`, quindi si valutano le coordinate, e direttamente il valore viene confrontato e sostituito con un `tile` che mantiene comunque l'informazione sulla tipologia della coltura ma, graficamente, vuoto.

```
1   var degradeZone = function(map, x, y, blockMaps, population, lpValue)
        {
2
3     var tileValue = map.getTileValue(x, y);
4
5     switch(tileValue){
6       case Tile.CORN:
7         tileValue = Tile.FCORN;
8         map.setTile(x, y, tileValue, Tile.BLBNHYBIT | Tile.ZONEBIT);
9         break;
```

```
10
11      case Tile.WHEAT:
12        tileValue = Tile.FWHEAT;
13        map.setTile(x, y, tileValue, Tile.BLBNHYBIT | Tile.ZONEBIT);
14        break;
15
16      case Tile.ORCHARD:
17        tileValue = Tile.FORCHARD;
18        map.setTile(x, y, tileValue, Tile.BLBNHYBIT | Tile.ZONEBIT);
19        break;
20
21      case Tile.POTATO:
22        tileValue = Tile.FPOTATO;
23        map.setTile(x, y, tileValue, Tile.BLBNHYBIT | Tile.ZONEBIT);
24        break;
25    }
26  return;
27  };
```

Listing 3.19: funzione `degradeZone` presente in `field.js`

3.3.3.2 Finestra del budget

Tutto ciò che è stato detto nel paragrafo precedente è gestibile dalla `budgetWindow` che si apre cliccando sul bottone del `budget`. Abbiamo inserito in questa finestra una sezione in HTML nel file `index.html` prendendo sempre spunto dagli elementi presenti, introducendo un nuovo *slider* col quale si può controllare la quantità di denaro da destinare ai campi attualmente presenti in gioco.

```
1  <fieldset>
2      <legend>Field</legend>
3      <input type="range" id="fieldRate" min="0" max="100" step="1"
       data-source="fieldMaintenanceBudget">
4      <div>
5          <label for="fieldRate" class="elided budgetData" id="
       fieldRateLabel">100% of $100 = $100</label>
6          </div>
7  </fieldset>
```

Listing 3.20: introduzione dello slider in `index.html`

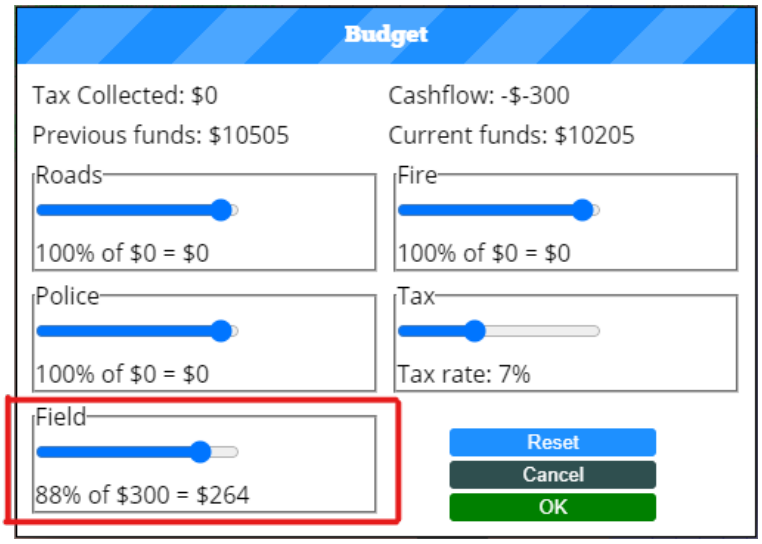

Figura 3.5: *finestra per gestione costi e riepilogo spese ed entrate*

Nella figura della finestra che mostra un riepilogo dei fondi, delle tasse, del flusso di denaro, e permette di gestire i costi, nel riquadro specifico di `field` il valore:

- 88%: è la percentuale di spesa sostenuta, che indicativamente rappresenta il `fieldPercent` in termine percentuale;
- 300$: è il `fieldMaintenanceBudget`, spese totali che bisognerebbe sostenere per tutti i campi che sono stati piazzati (3 in questo caso);
- 264$: indica il costo reale che si deve sostenere, cioè il `fieldSpend`.

Capitolo 4

Conclusioni e Sviluppi Futuri

4.1 Conclusioni e limiti

Lo sviluppo delle funzionalità in questo progetto è stato condiviso in tutte le sue parti tra me ed il collega Tommaso Coricelli. Nonostante il lavoro fosse suddiviso è stato comunque arduo a causa del fatto che di per sé il progetto iniziale era abbastanza articolato.
Bisogna comunque evidenziare che la modalità di mantenimento dei campi basata sul pagamento annuale di una tassa collettiva può sicuramente essere implementata in modo differente per evitare che tutti i campi vengano degradati nello stesso anno, infatti la condizione che porrà il campo in situazione di degrado viene ripresa da tutti i campi sulla mappa; si necessità quindi di una caratterizzazione di pagamento che consideri sul totale un numero inferiore o uguale, nel peggiore dei casi, di campi che non può essere manutenuto.
Malgrado le difficoltà legate alla complessità del progetto, la collaborazione ha prodotto una base da cui partire per l'introduzione di organi della sfera ecologica che possano anche dipendere ad esempio dall'impianto integrato in questo progetto; tutto questo potrebbe evolversi nella costruzione di una città "utopica" che possa abbattere definitivamente qualsivoglia emissione inquinante, che con i tempi che corrono, possono certamente generare spunti per una realtà a minore impatto ambientale. In fondo il ruolo che gioca un Serious Game è proprio questo, sensibilizzare, ispirare idee e indurre le persone a pensare in maniera utile per andare incontro agli interessi della società.

4.2 Sviluppi futuri

L'inizio di uno sviluppo indirizzato verso strutture sostenibili di questo lavoro offre innumerevoli spunti per eventuali prossimi progressi da poter integrare in questo progetto, soprattutto richiamando le tematiche ambientali che si stanno sviluppando in questi ultimi anni.
Possibili sviluppi, in questo senso, che possono essere aggiunte allo stato attuale del progetto sono ad esempio:

- la produzione di energia pulita con l'utilizzo della cinetica dei canali idrici che alimentando una centrale idroelettrica possa permettere la produzio-

ne di energia elettrica rinnovabile in una forma decisamente eco-friendly se messa a confronto con le attuali alternative ne gioco quali centrali di carbone e centrali nucleari, permettendo un abbattimento dei valori di inquinamento nei centri residenziali;

- lo sviluppo di una popolazione nelle zone rurali, in modo da decentralizzare il lavoro, riducendo la popolazione concentrata nelle zone residenziali, di conseguenza lo smog, e quindi attuabile con l'introduzione di una logica di popolamento coincidente alla crescita dei campi

Bibliografia

[1] Alice H.Aubert, René Bauer, and Judit Lienert. A review of water-related serious games to specify use in environmental multi-criteria decision analysis. *Environmental Modelling & Software*, 105:64–78, 2018.

[2] Wikipedia contributors. Carl abt. `https://en.wikipedia.org/wiki/Clark_C._Abt`, 2020.

[3] Francesco Ricciardi and Lucio Tommaso De Paolis. A comprehensive review of serious games in health professions. *International Journal of Computer Games Technology*, 2014.

[4] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: defining "gamification". *MindTrek '11: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media*, pages 9–15, 2011.

[5] Heinz Mandl Michael Sailer, Jan Hense and Markus Klevers. Psychological perspectives on motivation through gamification. *Interaction Design and Architecture(s) Journal - IxD&A*, 19:28–37, 2013.

[6] Martin Sillaots. Achieving flow through gamification: A study on re-designing research methods courses. 2014.

[7] Juho Hamari and Jonna Koivisto. Measuring flow in gamification: Dispositional flow scale-2. *Computers in Human Behavior*, 40:133–143, 2014.

[8] Mihaly Csikszentmihalyi. Flow: The psychology of optimal experience. *Harper and Row*, pages 133–143, 1990.

[9] Nannan Xi and Juho Hamari. Does gamification satisfy needs? a study on the relationship between gamification features and intrinsic need satisfaction. *International Journal of Information Management*, 46:210–221, 2019.

[10] J. Hamari, J. Koivisto, and H. Sarsa. Does gamification work? – a literature review of empirical studies on gamification. In *2014 47th Hawaii International Conference on System Sciences*, pages 3025–3034, 2014.

[11] 4 esempi di gamification ben riusciti. `https://www.kallo.it/4-esempi-di-gamification-ben-riusciti/`, 2019.

[12] Valerio Bamberga. Le escape room anche nel mondo della cultura. `https://www.tuomuseo.it/gamification/le-escape-room-anche-nel-mondo-della-cultura`, 2017.

[13] Stéphane Gobron Nicolas Wenk. Reinforcing the difference between simulation, gamification, and serious game. `http://www.stephane-gobron.net/Core/Publications/Papers/2017_GSGS17-1.pdf`, 2017.

[14] Alessio La Forgia. Serious games vs. gamification. http://gamithing.altervista.org/serious-games-vs-gamification, 2017.

[15] Serious game e gamification. http://onseriousgames.com, 2020.

[16] Houda Mouaheb, Ahmed Fahlib, Mohammed Moussetad, and Said Eljamali. The serious game: What educational benefits? *Procedia - Social and Behavioral Sciences*, 105:5502–5508, 2012.

[17] digital-water.city. https://www.digital-water.city, 2020.

[18] Eu introduces standards for use of reclaimed water for irrigation. https://www.oliveoiltimes.com/production/eu-introduces-standards-for-use-of-reclaimed-water-for-irrigation/81119, 2020.

[19] Micropolis. https://it.wikipedia.org/wiki/Micropolis, 2020.

[20] Wikipedia contributors. Git. https://it.wikipedia.org/wiki/Git_(software), 2020.

[21] Scott Chacon and Ben Straub. *Pro Git*. 2014.

[22] Kinsta.com. Cosa è github? introduzione a github per principianti. https://kinsta.com/it/knowledgebase/cosa-e-github/, 2020.

[23] Github home page. https://github.com, 2020.

[24] Repository progetto github. https://github.com/capatommy/micropolisJS, 2020.

[25] IconBunny. Github libero icona. https://icon-icons.com/it/icona/Github/102542.

[26] Julien Monty and Roberto Huertas. Libero icona. https://icon-icons.com/it/icona/html5-originale-wordmark-logo/146478,https://icon-icons.com/it/icona/file-di-tipo-di-css/130661,https://icon-icons.com/it/icona/file-di-tipo-di-js-ufficiale-di/130509.

[27] Wikipedia contributors. Html. https://it.wikipedia.org/wiki/HTML, 2020.

[28] Cesare Lamanna. Guida css di base. https://www.html.it/pag/14209/introduzione1/, 2013.

[29] Wikipedia Contributors. Css. https://it.wikipedia.org/wiki/CSS, 2020.

[30] Wikipedia Contributors. Javascript. https://it.wikipedia.org/wiki/JavaScript, 2020.

[31] David Flanagan. *JavaScript - La guida*. 2000.

[32] Toni Podmanicki. Come abilitare javascript nel tuo browser. https://www.enable-javascript.com/it/, 2020.

[33] Tiobe.com. Tiobe index for september 2020. https://www.tiobe.com/tiobe-index/, 2020.

[34] Michele Nasi. Visual studio code: cos'è e come funziona. https://www.ilsoftware.it/articoli.asp?tag=Visual-Studio-Code-cos-e-e-come-funziona_19189, 2019.

[35] su code.visualstudio.com. Microsoft visual studio code terms. https://code.visualstudio.com.

[36] Papirus Development Team. Visual, studio, codice libero icona. https://icon-icons.com/it/icona/visual-studio-codice/93981.

[37] Wikipedia contributors. Browser. https://it.wikipedia.org/wiki/Browser.

[38] Node Academy. Cos'è l'npm e come usarlo. https://www.nodeacademy.it/cose-npm-installazione-locale-globale-aggiornamento/, 2018.

[39] Giacomo Cerquone. Yarn vs npm – in cosa si differenziano oggi? https://italiancoders.it/yarn-vs-npm/, 2018.

[40] nodejs.org. What is the file 'package.json'? https://nodejs.org/en/knowledge/getting-started/npm/what-is-the-file-package-json/, 2011.

[41] GitHub Contributors. webpack. https://github.com/webpack, 2020.

[42] Gamunu Balagalla. yarn. https://marketplace.visualstudio.com/items?itemName=gamunu.vscode-yarn, 2020.

Elenco delle figure