POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Tracciabilità nelle Supply Chain e Serious Game: Sviluppo di un Serious Game per il supporto all'apprendimento dello standard EPCIS

Relatore:
Prof. Giovanni Malnati
Co-relatrice:
Dott. Flavia Moschetto

Candidato:
Gabriele Mariane

Luglio 2021

Introduzione

Le Supply Chain sono una realtà estesamente sviluppata e affermata nel mondo di oggi. La produzione industriale, sia essa di qualsivoglia natura, è spesso strutturata su numerose fasi potenzialmente molto distanti nello spazio, nel tempo e nelle caratteristiche.
Dal primo materiale raccolto fino alla vendita del prodotto finito in supermercato può passare una quantità arbitraria di settimane, mesi o anni, di chilometri e di diverse lavorazioni. Il prodotto, durante le diverse fasi di sviluppo, passa poi per una serie di aziende diverse, con politiche e dinamiche diverse, capitali diversi e alle volte culture diverse.
In questo tipo di contesto diventa sempre più importante la **visibilità** e la **tracciabilità**: la visibilità delle informazioni delle aziende che fanno parte della filiera e la tracciabilità del prodotto lungo tutta la sua vita produttiva, dal fattore che alleva i polli, al granaio che fornisce il mangime, al macello che seziona i capi, al supermercato che vende le vaschette preparate fino alla tavola del consumatore.

GS1, organizzazione no profit che sviluppa e mantiene standard globali per la comunicazione tra imprese, ha sviluppato lo standard di tracciabilità **EPCIS**.
Lo standard, in collaborazione con altri suoi simili sviluppati fa GS1, fornisce i metodi con cui dovrebbero essere catturate le informazioni e quelli con cui dovrebbero essere condivise. Fornisce la struttura dati generica con cui devono essere conservati. Fornisce un linguaggio strutturato per agevolare la comunicazione tra le parti.
Ma è uno standard profondamente articolato e complesso e, come chiunque lavori nel nostro campo sa bene, chi è del mestiere difficilmente investe energia e tempo, e quindi denaro, per apprendere qualcosa di complicato prima, e per implementarlo poi.

Qui si inserisce il mio lavoro di Tesi.
Ho lavorato allo sviluppo di un prodotto che potesse supportare la fase di apprendimento dello standard, così da poter avvicinare il cliente al mondo della tracciabilità strutturata. In questo elaborato del mio lavoro di Tesi racconto EPCIS e ciò che fa star lontano i clienti da esso; presento la scelta del **Serious Game** come mezzo per veicolare i contenuti e i dettagli che caratterizzano questo strumento; spiego il processo di pensiero e progettazione del prodotto, con riferimento al **Design Thinking**; e infine mostro nel dettaglio lo sviluppo, sia ad alto livello che a basso livello, di due iterazioni prototipali, la prima con il motore isometrico scritto in Javascript **Traviso.js** e la seconda col motore di gioco **Unity** e con **C#**.

Concludo infine raccontando quelli che sono i passi successivi al mio lavoro che non sono stati implementati e presentando le mie considerazioni finali rispetto ai risultati ottenuti.

1. Il Problema

Per capire il problema che si vuole affrontare è necessario comprendere anzitutto qual è il contesto tecnologico e commerciale in cui sorge.
In una dimensione di mercato globalizzato come quella in cui ormai da decenni ci troviamo è sempre più forte la realtà della Supply Chain.
La Supply Chain, o in italiano Filiera, è una realtà di business che comprende l'intera vita del prodotto finale e tutti gli step che lo hanno portato ad essere quello che è al momento della vendita al consumatore finale. Per sua natura quindi, la S.C. è costituita da più aziende legate da rapporti commerciali. Le modalità e la profondità di questi legami sono fortemente variabili.

Qui sorge il primo problema: la **scambio di informazioni** tra le parti. È necessario un linguaggio comune che sia semplice e immediato nonché privo di possibilità di fraintendimento.
Il secondo problema è strettamente legato al primo: la **tracciabilità**. Questa è infatti estremamente importante in quelle situazioni in cui sorgono dei problemi a valle della catena e si vuole risalire ai passaggi precedenti così da identificare, ove possibile, la fonte del problema e porre rimedio così alle conseguenze, siano esse evidenti o ancora celate.
A titolo di esempio, si consideri una situazione in cui, a fronte di una confezione di carne cattiva, si voglia riconoscere velocemente quale sia la fattoria da cui deriva e quale il singolo o i singoli capi di bestiame da cui è stata ottenuta.

In questo contesto nasce la necessità di standardizzare le interazioni tra i partner di una S.C. in modo da ottimizzare i vari aspetti che le caratterizzano e sfruttare al meglio le potenzialità del caso. Ma uno standard porta con se diverse problematiche, prima fra tutte la comprensione.

1.1 EPCIS

Nella stesura di questo paragrafo ho fatto riferimento a [4].

EPCIS è uno standard sviluppato da GS1, un'organizzazione no profit che sviluppa e mantiene standard globali per la comunicazione tra imprese. Gli standard e i servizi GS1 sono progettati per migliorare l'efficienza, la sicurezza e la visibilità delle supply chain, attraverso strumenti fisici e digitali, in un'ampia gamma di settori diversi. Il più noto tra questi standard è il codice a barre.
Le soluzioni GS1 compongono un linguaggio comune di business che identifica, "cattura" e permette di condividere informazioni sui prodotti, le sedi di produzione e distribuzione, gli asset logistici e molto altro. [1][2]

EPCIS (Electronic Product Code Information Services) è lo standard GS1 per fare tracciabilità in tempo reale. È uno strumento che consente di condividere le informazioni riguardanti la vita di un prodotto e il percorso che fa lungo tutta la filiera. EPCIS si applica a tutte quelle situazioni in cui è utile catturare e condividere "**visibility event**".
Infatti, esso permette di seguire uno specifico oggetto, o una partita di merce, e di registrare informazioni sulla sua storia, rendendole visibili agli altri attori della filiera. [3]

EPCIS fornisce interfacce standardizzate che permettono l'integrazione di servizi definiti in ambiente inter-aziendale o fra aziende senza soluzione di continuità.
Queste interfacce sono definite per abilitare la cattura e la consultazione di eventi di visibilità utilizzando un set di operazioni e dati, il tutto combinato con meccanismi di sicurezza appropriati alle necessità delle aziende.
Questo avviene con o senza il supporto di un DataBase persistente. A prescindere da questo, lo standard specifica solo un'interfaccia di cattura e condivisione di dati, ma non da informazioni sull'implementazione.

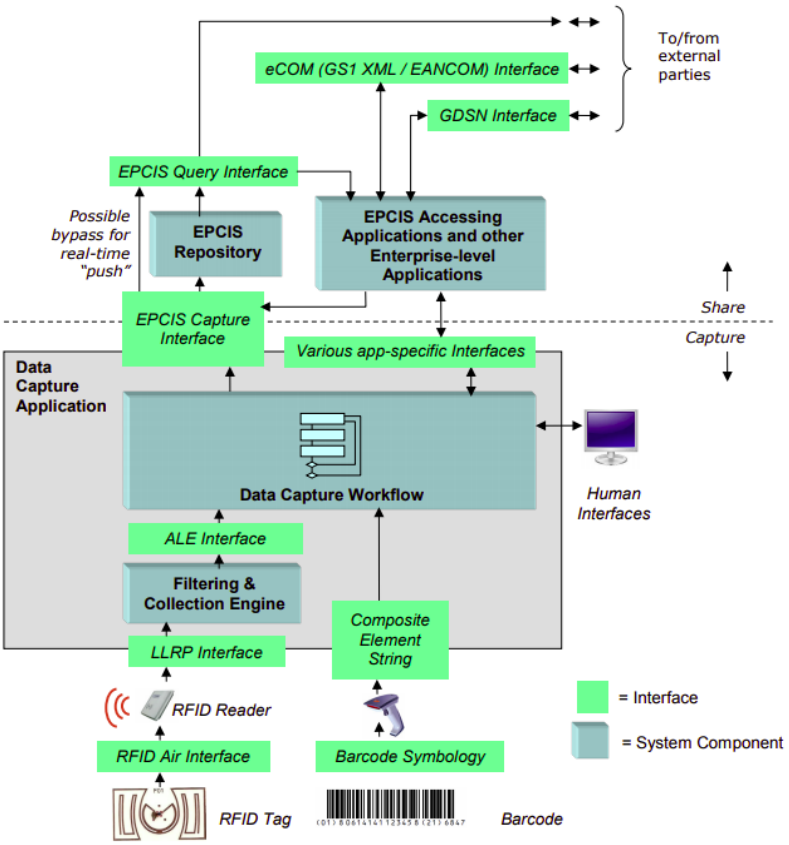
EPCIS deve essere usato assieme allo standard GS1 Core Business Vocabulary (CBV), il quale fornisce definizioni di dati che possono essere usati per popolare le strutture definite in EPCIS.
L'uso di CBV è critico per l'interoperabilità tra le aziende che vogliono condividere i dati tramite EPCIS.

Gli standard GS1 possono essere suddivisi in tre macro gruppi in base al loro ruolo nel supportare le necessità di informazione legate alle entità coinvolte nei business process di una supply chain.

- Gli standard che puntano ad **identificare** le entità.
- Gli standard volti alla **cattura** automatica dei dati e quindi alla traduzione degli oggetti fisici in informazioni elettroniche.
- Gli standard per la **condivisione** delle informazioni, sia tra aziende partner che all'interno di una stessa azienda.

Lo standard EPCIS si inserisce nell' ultimo di questi gruppi, fornendo lo standard dei dati per gli eventi di visibilità, per la loro cattura, utilizzando anche standard del secondo gruppo, e per la condivisione di queste informazioni.

Nella figura seguente è possibile vedere EPCIS inserito nel contesto di altri standard GS1.



La natura storica dei dati EPCIS implica che le interfacce necessitino di un più ricco set di tecniche di accesso rispetto ad altri standard GS1, come RFID e l'interfaccia per la lettura dei codici a barre. È necessario che sia in grado di gestire un più ricco set di tipi di dati e che sia molto più aperto alle estensioni, così da poter servire la grande varietà di business process esistenti nel mondo.
Infine è necessario che EPCIS sia attentamente stratificato in modo che, quando venisse usato tra sistemi esterni profondamente diversi, possa esserci consistenza e interoperabilità tra i dati e il loro significato nel mondo reale.

EPCIS è designato per essere stratificato, estensibile e modulare.

1.1.1 Gli Strati:

- Abstract Data Model Layer: specifica la struttura generica dei dati EPCIS. Questo è l'unico strato non estensibile senza modifiche allo standard.
- Data Definition Layer: specifica quali dati sono scambiati in EPCIS, la loro struttura astratta e il loro significato.
- Service Layer: definisce le interfacce ai servizi tramite le quali i client EPCIS possono interagire. Le definizioni di tali interfacce sono specificate in modo astratto con UML.
- Bindings: Specifica realizzazioni concrete dei due strati precedenti.
- GS1 Core Business Vocabulary Standard(CBV): companion di EPCIS, specifica elementi di vocabolario che possono essere usati per popolare le definizioni dei dati. EPCIS potrebbe essere implementato senza CBV ma è molto più beneficiario per questioni di compatibilità ed interoperabilità implementarlo con CBV.

1.1.2 Estensività:

La struttura a strati promuove l'estensibilità tramite la possibilità di riuso.
Apparte questo sono inclusi in EPCIS diversi meccanismi specifici per l'estensibilità:

- Subclassing: Una sottoclasse è un nuovo tipo che contiene tutti i campi di un tipo esistente e lo estende con nuovi campi.
- Extension Points: Possono essere usati dai vendor per fornire funzionalità estese senza fare subclassing.

1.1.3 Modularità:

EPCIS non consiste di una sola specifica ma di una collezione di specifiche individuali che sono correlate. In questo modo lo standard può evolvere in modo distribuito. Nonostante la modularità delle specifiche EPCIS, non è necessario che i confini dei moduli siano espliciti nelle implementazioni dello standard.

1.1.4 Abstract Data Model Layer

Generalmente, EPCIS ha a che fare con due tipi di dati: event data e master data. Gli Event data sorgono nel contesto dei business process e sono catturati tramite l'interfaccia di cattura e sono resi disponibili dalle Query Interface. I Master data sono dati addizionali che forniscono il contesto necessario per una corretta interpretazione degli Event data. Il modo in cui i Master data sono integrati nel sistema non è specificato dallo standard.

Gli ingredienti di questo strato sono:

- Event Data: Set di eventi
- Event: Struttura composta da un Event Type e uno o più Event Field
- Event Type: Un nome che indica a quale di molte possibili Event Structure un certo evento appartiene.
- Event Field: Un campo di un Evento. I ìl nome del campo fa riferimento a quanto definito nei vocabolari del Data Definition Layer o di una estensione.
- Master Data: Un set di vocabolari assieme agli attributi associati.
- Vocabulary: Set di identificativi. Questi possono essere usati per definire i campi di un Event.
- Vocabulary Element: Identificativo contenuto in un Vocabolario.
- Master Data Attributes: Set disordinato di coppie nome/valore associate a un elemento di vocabolario.

1.1.5 Data Definition Layer

Gli Eventi sono tipicamente generati da un'applicazione di cattura e sono forniti a EPCIS tramite specifiche operazioni. Questi eventi sono anche ritornati in risposta a operazioni di query.
Gli Event Type sono i seguenti:

- **EPCISEvent**: generica classe base per tutti gli altri tipi.
- **ObjectEvent**: rappresenta un evento riguardante uno o più oggetti fisici o digitali.
- **AggregationEvent**: rappresenta un evento accaduto a uno o più oggetti fisicamente aggregati insieme
- **QuantityEvent**: rappresenta un evento riguardante una specifica quantità di oggetti che condividono la stessa classe EPC ma in cui le entità individuali non sono specificate. (*DEPRECATO*)
- **TransactionEvent**: rappresenta un evento in cui uno o più oggetti diventa associato a o dissociato da una o più transazioni.
- **TransformationEvent**: rappresente un evento in cui oggetti input sono completamente o parzialmente consumati e oggetti output sono prodotti, di modo che ogni oggetto input possa aver contribuito a tutti gli oggetti output.

Ognuno dei tipi sopra descritti ha campi che rappresentano le 4 dimensioni chiave di un evento EPCIS.
- *What* : Gli identificatori dell'oggetto/degli oggetti o di altre entità che sono soggetto dell'evento
- *When* : La data e l'ora in cui l'evento è avvenuto e il fuso orario della zona
- *Where* : L'identificatore del luogo in cui è avvenuto l'evento, e l'identificatore del luogo in cui ci si aspetta che l'oggetto/gli oggetti si trovino in seguito all'evento.
- *Why* :Informazioni riguardo il contesto di business, inclusi: un identificatore che indichi il business step in corso(es., trasporto, ricezione, ecc.), identificatori delle parti che trasportano/ricevono (se l'evento è parte di un processo di trasferimento), collegamenti a documenti di business transactions rilevanti (es., un ordine d'acquisto, una ricevuta, ecc.), dati generali a livello di istanza di lotto, e/o altre informazioni definite su estensione dell'utente.

1.1.5.1 Action Field

Questo campo spiega come un evento si relaziona al lifecycle dell'entità descritta.
È un campo con tre possibili valori:

- ADD: L'entità è stata creata o aggiunta a.
- OBSERVE: L'entità non ha subito modifiche, non è stata creata, né aggiunta, né distrutta, né rimossa da.

- DELETE: L'entità è stata rimossa da o distrutta in toto.

1.1.5.2 What

Questa dimensione specifica quali oggetti fisici o digitali hanno preso parte all'evento.
EPCIS identifica gli oggetti in due modi:

- Instance-level: Un identificativo è detto Instance-level se viene assegnato in modo tale da essere univoco per un singolo oggetto.
- Class-level: Un identificativo è detto class-level se più oggetti possono esserne rappresentati.

Il secondo tipo è usato solo quando è poco pratico usare il primo.

1.1.5.3 Where

Vediamo 4 tipi che implementano la nozione di luogo in EPCIS.

- PhysicalReaderID: Identità unica o nome della specifica fonte di informazione (esempio lettore RFID, un altro standard EPCIS) che riporta i risultati di un EPC read event.
- LogicalReaderID: Identità o nome assegnato alla fonte di informazione del read event indipendentemente dal dispositivo fisico usato.
- ReadPointID: Luogo registrato discretamente che vuole identificare il posto più specifico in cui l'evento ha avuto luogo.
- BusinessLocationId: Luogo identificato univocamente e registrato discretamente atto a designare il punto specifico in cui si assume che l'oggetto si troverà in seguito all'evento.

1.1.5.4 Why

Tipi di dati:

- Business Step: Il BusinessStepId è un vocabolario i cui elementi denotano gli step in un business process.
- Disposition: Il DispositionID è un vocabolario i cui elementi denotano lo stato di business di un oggetto
- Business Transaction: Identifica una transazione
    - Business Transaction Type: Il BusinessTransactionTypeID è un vocabolario i cui elementi denotano uno specifico tipo di transazione.
    - Business Transaction ID: Il BusinessTransactionID è un vocabolario i cui elementi denotano specifiche transazioni.
- Source and Destination: Usate per fornire contesto di business quando un evento EPCIS è parte di un trasferimento, cioè un'operazione in cui c'è un cambio di proprietà, responabilità e(o custodia di un oggetto fisico o digitale.
    - Source and Destination Type: Il SourceDestTypeID è un vocabolario i cui elementi denotano uno specifico tipo di sorgente o destinazione di un trasferimento.
    - Source and Destination ID: Il SourceDestID è un vocabolario i cui elementi denotano specifiche sorgenti o destinazioni.
- Instance/Lot master Data: Dati che descrivono una specifica istanza di un oggetto fisico o digitale o di uno specifico lotto di oggetti prodotti in lotti. ILMD consiste di un set di attributi descrittivi che fornisce informazioni su uno o più oggetti specifici o lotti.

Dato questo complesso e strutturato sistema a eventi, che ho qui presentato solo in parte, il vero vantaggio dello standard sta nella condivisione dei dati. Infatti, nel momento in cui tutti gli eventi all'interno dell'azienda sono osservati e descritti in oggetti ben definiti, la condivisione delle informazioni è più semplice e immediata, così come la traduzione delle stesse da parte delle aziende partner lungo la Supply Chain.

Ecco che il rivenditore di carne, il quale vede gli eventi che avvengono nella fattoria in cui i suoi prodotti hanno origine, ha modo di prevedere con maggiore precisione i possibili risvolti della sua attività legati all'andamento della fattoria di cui sopra.
A titolo di esempio, una catastrofe che colpisce il bestiame del fattore oggi, se immediatamente condivisa con il venditore ultimo, permette a questo di prevedere che tra 6 mesi i suoi approvvigionamenti di materia prima scarseggeranno, permettendogli di correre ai ripari con largo anticipo.

Purtroppo nella realtà diventa poi molto difficile per chi ha effettivamente a che fare con lo standard comprenderne il funzionamento. Di fronte quindi a una nuova tecnologia che si presenta come estremamente complessa la disponibilità all'accoglienza tende a venir meno. E' pertanto necessario mostrare al cliente anche le potenzialità del prodotto e i possibili vantaggi che se ne potrebbero trarre.

Qui sorge una questione: chi è il cliente?
È chiaro che il cliente che userà lo standard e un'azienda, ma di preciso chi ne avrà a che fare in prima persona? Quali sono le persone fisiche a cui si rivolge la nostra attenzione e a cui vogliamo mostrare le potenzialità e il funzionamento di EPCIS?

Una volta identificato il cliente bisogna capire su cosa ci si vuole concentrare, cosa è importante esprimere.
Si è detto che l'intenzione è quella di rendere più facile la comprensione dello standard. Seppur estremamente utile, questo aspetto porta con sé una difficoltà intrinseca: semplificare un concetto complesso è a sua volta complesso e rischia di portare a risultati poco fruibili e quindi poco funzionali. È sensato concentrarsi sugli aspetti principali dello standard, tenendosi a un alto livello di astrazione e puntare sul 'colpo d'occhio'. Ciò a cui si auspica è che il cliente possa vedere con impatto immediato le dinamiche principali di EPCIS.
L'altro punto estremamente importante come ho accennato è quello del mostrare le potenzialità dello standard. Se il cliente comprende e prova interesse per le potenzialità allora è più facile che sviluppi la curiosità di comprenderne anche il funzionamento.

Quindi voglio realizzare un prodotto che sia capace di esprimere in modo semplice i concetti chiave del funzionamento di EPCIS e che ne esalti le potenzialità. Voglio che sia di facile fruizione e che esalti il colpo d'occhio, così da stimolare interesse e curiosità.

2. Il Serious Game

2.1 Perché il gioco?

Il dizionario Treccani definisce così il gioco:

"Qualsiasi attività liberamente scelta a cui si dedichino, singolarmente o in gruppo, bambini o adulti senza altri fini immediati che la ricreazione e lo svago, sviluppando ed esercitando nello stesso tempo capacità fisiche, manuali e intellettive." [5]

E' interessante in particolare l'aspetto dello *sviluppo* e delle capacità. Il gioco può essere strumento di apprendimento, di educazione, di formazione, questo senza rinnegare la sua natura principale, la natura della ricreazione e dello svago. La natura della leggerezza.

È però necessario fare uso di una forma evoluta del gioco che sia attentamente strutturata e finalizzata ai nostri scopi formativi. Pertanto il prodotto non potrà essere solo un gioco contestualizzato, col quale il cliente può divertirsi in un contesto specifico, ma piuttosto un'esperienza formativa veicolata in concomitanza a un contesto ludico di sfida e divertimento, in modo bilanciato.

2.2 Le Evoluzioni dell'utilizzo del gioco

Vediamo tre diverse evoluzioni del gioco:

2.2.1 Teoria dei giochi

Secondo il Dizionario Treccani:
"Modello matematico per lo studio delle 'situazioni competitive', in cui cioè sono presenti più persone (o gruppi di persone, o organizzazioni) dette appunto 'giocatori', con autonoma capacità di decisione e con interessi contrastanti (➔ gioco). Tali sono i giochi di società (come bridge, poker, tresette, scacchi) che hanno dato il nome alla teoria, e ne costituiscono il più tipico esempio di applicazione perché, essendo ben precisate le norme che li regolano, possono essere facilmente schematizzati in un modello matematico.(Treccani)"[6]

Secondo Wikipedia:
"È una disciplina della matematica applicata che studia e analizza le decisioni individuali di un soggetto in situazioni di conflitto o interazione strategica con altri soggetti rivali (due o più) finalizzate al massimo guadagno di ciascun soggetto
In tali situazioni le decisioni di uno possono influire sui risultati conseguibili dall'altro/i e viceversa secondo un meccanismo di retroazione, ricercandone soluzioni competitive e/o cooperative tramite modelli, che in particolare nel contesto economico si riferiscono al caso in cui due o più aziende interagiscono in concorrenza tra loro (wikipedia)"[7]

La Teoria dei Giochi è quindi un modello matematico, che prende dal mondo del gioco le dinamiche di competitività o collaborazione e di massimizzazione delle prestazioni per se stesso come parte giocante o per tutte le parti coinvolte.
Anche se questa evoluzione eredita dal gioco anche l'aspetto dello svago, è comunque caratterizzata da una difficoltà intrinsecamente elevata e non si presta quindi alla semplificazione di concetti complessi.

2.2.2 Gamification

Secondo il Dizionario Treccani:
"Utilizzo di meccanismi tipici del gioco e, in particolare, del videogioco (punti, livelli, premi, beni virtuali, classifiche), per rendere gli utenti o i potenziali clienti partecipi delle attività di un sito e interessarli ai servizi offerti."[8]

È l'applicazione di meccaniche ludiche ad attività che non hanno direttamente a che fare con il gioco con l'obiettivo di influenzare e modificare il comportamento delle persone, favorendo la nascita ed il consolidamento di interesse attivo da parte degli utenti coinvolti verso il messaggio che si è scelto di comunicare, sia questo relativo all'incremento di performance personali o più in generale alle performance d'impresa. [9]

Ci sono molti contesti nei quali è possibile applicare il "metodo" gamification: un sito, un servizio, una comunità, un contenuto o campagna sono tutti contesti che possono essere "gamificati" (da "to gamify") così da spingere l'interesse, il coinvolgimento e la partecipazione degli utenti.[10]

Questa evoluzione si avvicina di più rispetto alla precedente a quello che è il mio intento, sfruttando le dinamiche del gioco e la loro natura coinvolgente e divertente per supportare l'apprendimento o la formazione.
Tuttavia non è quello che si vuole usare in questo contesto in quanto non si vuole gamificare un contesto o un prodotto già esistente ma vogliamo fornire invece un prodotto a sé stante per cui la semplificazione del concetto di interesse e l'esaltazione del valore dello stesso siano gli unici scopi.

2.2.3 Serious Game

Secondo Wikipedia:
"Sono giochi in cui gli aspetti seri e ludici sono in equilibrio, infatti non hanno come scopo principale l'intrattenimento, ma sono progettati a fini educativi.
L'attenzione sta nella volontà di creare un'esperienza formativa efficace e piacevole, mentre il genere, la tecnologia, il supporto e il pubblico variano.
Anche la simulazione virtuale interattiva è spesso considerata serious game.
Lo scopo fondamentale è sviluppare abilità e competenze da applicare nel mondo reale attraverso l'esercizio in un ambiente simulato e protetto
Diversamente dalla Gamification, che contiene solo alcuni elementi mutuati dai giochi, quali l'assegnazione di punti o il raggiungimento di livelli, il Serious Game è gioco a tutti gli effetti."[11]

Questa evoluzione è quella che più soddisfa le nostre esigenze. Il Serious Game infatti può essere un prodotto finito e autonomo che si struttura in tutto e per tutto come un gioco (e ne possiede quindi per sua natura tutte le dinamiche) e che può essere costruito per servire allo scopo che si è prefissato. Si è scelto pertanto di proseguire sulla strada del Serious Game.

2.3 Serious Game

Nella stesura di questo paragrafo ho fatto riferimento a [12], [13] e [14].

2.3.1 Cos'è

Il termine serious game è stato utilizzato per la prima volta nel 1970 nel libro omonimo di Clark C. Abt, nel quale egli stabilisce il concetto di base del S.G..
Abt descrive i serious game come "giochi con un esplicito e ben definito scopo educativo, non pensati primariamente per il divertimento, senza però escluderlo".
La definizione di Abt non esclude pertanto l'aspetto del divertimento e dell'umorismo.

L'obiettivo del Serious Game pertanto si può dire essere quello di lasciare al giocatore qualcosa che vada oltre la semplice fruizione dell' S.G.. Ma in che modo fa questo?

L'approccio all'apprendimento messo in atto nei serious game si basa sul concetto del "learning by doing", il quale favorirebbe modifiche comportamentali profonde e durature.
Si sfrutta quindi un apprendimento attivo nella speranza di ottenere risultati qualitativamente migliori rispetto a quello passivo, in cui i contenuti sono veicolati attraverso lezioni frontali.
In aiuto a questo vi è anche una tipica dinamica dei giochi nota come trial-and-error, che permette, tramite ripetizioni di tentativi e fallimenti, un apprendimento più a tutto tondo delle dinamiche del gioco e, nel caso del S.G., delle tematiche portate.

Un articolo su *restorativeneurotechnologies*.com definisce il Serious Game come il bilanciamento tra Apprendimento, Intrattenimento e Simulazione.
Vediamo qui l'aggiunta di un ulteriore tassello: quello del mondo simulativo.
La simulazione, nei giochi in generale, permette un coinvolgimento maggiore del giocatore, il quale ha la percezione di un mondo realistico (coerente a se stesso e simile in una certa misura a quello che lui conosce) in cui avvengono automaticamente degli eventi possibili ai quali egli deve reagire facendo delle scelte.

2.3.2 Dove si applica il Serious Game

Il S.G. si presta a modellare in modo semplificato i sistemi complessi.
In un sistema complesso anche regole semplici associate a semplici interazioni possono portare a comportamenti difficili da prevedere. Questo li rende decisamente difficili da comprendere a fondo.
In quest'ottica il S.G. potrebbe essere utile nello sviluppo di nuove conoscenze scientifiche, nella formazione di competenze meccaniche o chirurgiche, nella modificazione di comportamenti disfunzionali o nello sviluppo di abilità cognitive.
Più in generale potrebbe essere utile in qualsiasi contesto di apprendimento di un sistema complesso. Può anche applicarsi alla didattica o alla formazione aziendale e professionale.

2.3.3 Apprendimento nel Serious Game

Per quanto riguarda il processo di apprendimento che ha luogo nell'utilizzo del Serious Game possiamo ipotizzare l'intervento di alcuni processi noti del mondo della neuropsicologia: il sistema attentivo, il modello motivazione/prestazione e il circuito del reward.

Il sistema attentivo sarebbe sicuramente il protagonista principale di questa storia: l'attenzione è infatti una funzione cognitiva trasversale che, con la sua presenza, può rafforzare o indebolire a cascata tutto il resto del sistema cognitivo. Un serious game ben costruito, attraverso dinamiche sfidanti, ludiche e graficamente appetibili, riesce a mantenere elevata la soglia di attenzione, assicurando un maggiore coinvolgimento cognitivo.

Il modello motivazione/prestazione descritto da Atkinson ipotizza che le performance migliori avvengano con una quota di motivazione equilibrata

Il circuito di reward è responsabile della nostra motivazione, ed è ciò che alimenta il sistema attentivo.

Possiamo aggiungere, come accennato in precedenza, che il tipo di apprendimento che avviene nell'utilizzo di un S.G è sicuramente l'Apprendimento Esperienziale, basato appunto sull'esperienza e sul trial-and-error.
Questa meccanica è detta learning-by-doing e sfrutta il coinvolgimento in prima persona del fruitore che impara dai suoi errori e vive il proprio miglioramento in prima persona, sviluppando la percezione, l'attenzione e la memoria.
Essenziale è la libertà del fruitore di esplorare diversi possibili scenari e usare le proprie risorse e competenze per dedurre quali sono le scelte migliori.
In questo modo si sviluppano le capacità di problem solving e la creatività e si acquisiscono comportamenti adattivi e autoconsapevolezza di sé.
Questo tipo di apprendimento, contestualizzato nel mondo ludico, abbatte le barriere del giudizio e lascia la libertà e la serenità di commettere errori.

2.3.4 Formazione col Serious Game

Auspicabilmente tutto ciò che viene appreso dal fruitore nel giocare il S.G. viene poi interiorizzato, sia in termini di situazioni vissute, si di sensazioni sperimentate.
In questo modo si attivano i processi che permettono di riflettere su tali esperienze e modificare il modo di approcciarsi alle stesse. L'ambiente simulativo permette quindi di portare nelle esperienze di vita reale quanto appreso nell'ambiente simulato.

2.3.5 Progettare un Serious Game

Nella progettazione di un S.G. è estremamente importante tener conto sia degli aspetti prettamente di **gaming** sia di quelli riguardanti la **pedagogia**.

2.3.5.1 Scopo

pedagogia

Capire quali sono gli obiettivi di formazione e apprendimento per i giocatori.

- Quale dominio affronta il gioco?
  - Quali sfide cognitive/sociali/operative questo dominio racchiude?
- Chi sono i giocatori?
  - Quali conoscenze pregresse del dominio hanno?
- Quale scopo vogliamo che raggiungano giocando?
  - Come possiamo misurarne il raggiungimento?

2.3.5.2 Contenuto

pedagogia

Scegliere i concetti specifici e le dinamiche specifiche che verranno spiegate (direttamente o indirettamente) all'interno dell'esperienza di gioco e capire come verranno rappresentati.

- Quali contenuti si intendono presentare?
  - Dove vengono reperiti?
  - Come vengono presentati?
  - Come si sposano con lo scopo generale del gioco?
- Quale parte del dominio viene simulata?
  - Che tipo di modello si utilizza?
  - E' abbastanza semplice da essere implementabile?
  - E' sufficientemente realistico?
  - Come si possono validare modello e simulatore?
  - Come aiuta a raggiungere gli scopi del gioco?

2.3.5.3 Contesto

pedagogia, gaming

Scegliere l'ambiente in cui ha luogo l'esperienza di gioco; dev'essere funzionale sia all'intrattenimento che all' apprendimento.

- Cosa (probabilmente) spinge i giocatori a giocare?
  - Che relazione ha con gli scopi dichiarati del gioco?
- Con quali atteggiamenti i giocatori si avvicinano al gioco?
  - Il contesto come condiziona tali atteggiamenti?
- Ci sono elementi del contesto che contrastano con lo scopo del gioco?
  - Obiettivo ben definito che orienta le azioni dell'utente
  - Insieme di regole che limitano lo spazio di azione dei giocatori
  - Presenza di conflitti Tra forze opposte, che si oppongono al raggiungimento dell'obiettivo e sono motivo di sfida per il giocatore (problem solving)
  - In grado di generare un valore intrinseco per cui i giocatori sono disposti ad investire le proprie risorse (tempo, denaro, …)

2.3.5.4 Meccanica

gaming

Come si gioca in pratica, tipo di gioco, tipo di azioni possibili, obbiettivo del gioco (per il giocatore), aspetti pratici.

- Quale spazio viene usato dal gioco?
  - E' discreto o continuo?
  - Quante dimensioni ha?
  - Quali sono i suoi confini?
  - Ci sono sotto-spazi? Come sono connessi?
- Quali oggetti sono presenti nel gioco?
  - Che attributi hanno?
  - Quali valori possono avere i singoli attributi?
  - Cosa determina un cambio di stato in un attributo?
  - Quali attributi sono noti a tutti, quali ai singoli giocatori e quali solo al motore di gioco?
- Quali verbi sono a disposizione dei giocatori?
  - A quali oggetti esse possono applicarsi?
  - In quanti modi è possibile raggiungere un obiettivo?
  - Quanti oggetti sono controllati da un giocatore?
  - Come possono alcuni effetti collaterali cambiare i vincoli di gioco?
- Quali sono le azioni dirette?
  - Quali quelle risultanti?
  - Il rapporto tra dirette e risultanti è equilibrato?
- Qual è l'obiettivo che i giocatori devono raggiungere?
  - E' comunicato in modo chiaro?
  - Se ce ne sono vari, sono correlati in modo sensato?
  - Sono concreti, raggiungibili e gratificanti?
- Quali sono le regole fondamentali (modello matematico) su cui si basa il gioco?
  - Quali le regole operative?
  - Sono comunicate in modo chiaro?
  - Serve un tutorial?

2.3.5.5 Storia e Narrativa

pedagogia, gaming

Costruire una storia che racconti il gioco in maniera coinvolgente e che leghi in modo più fruibile i diversi aspetti

- Quando due giocatori possono fare scelte diverse su come raggiungere un dato obiettivo, possono scaturire due storie diverse
    - Come aumentare tale possibilità?
- Le diverse possibilità di risolvere un conflitto portano a storie diverse?
    - Come aumentare il numero dei conflitti?
- Quando i giocatori possono personalizzare i loro personaggi, tendono a curarsi maggiormente degli esiti delle storie
    - Come permettere di personalizzare le storie?
- Che relazione c'è tra il personaggio principale e l'obiettivo che gli viene assegnato?
    - Perché dovrebbe essere di suo interesse?
- Che ostacoli ci sono tra il personaggio ed il suo obiettivo?
    - Ci sono antagonisti dietro gli ostacoli?
    - Che relazione c'è tra protagonista e antagonisti?
- Gli ostacoli hanno una progressività nel loro livello?
    - Sono sufficientemente complessi da essere interessanti?
- Per superare gli ostacoli, il protagonista deve trasformarsi in qualche modo?

2.3.5.6 Estetica e Grafica

gaming

Ideare l'aspetto del gioco, essenziale per catturare l'attenzione e la curiosità del fruitore. Puntare al colpo d'occhio.

- Questa componente si riferisce al linguaggio audiovisivo concettualizzato, scelto e utilizzato dai progettisti
    - Caratteristiche estetiche, immagini, stile, media artistici e tecniche grafiche computerizzate
- Definisce gli aspetti formali generali che inquadrano il contenuto (informazioni), la storia (il mondo e i personaggi del gioco), il contesto (gruppo target), l'impostazione e la meccanica (istruzioni, premi)
- Poiché l'estetica e la grafica presentano il gioco all'utente sin dalla prima immagine, ha un ruolo fondamentale nell'introduzione dello scopo del gioco e del suo impatto sul giocatore

2.3.6 Come si implementa

Il modello S.G. permette uno sviluppo flessibile in quanto è adattabile a diverse necessità.
In particolare è possibile sviluppare un S.G. con qualunque tecnologia si pensi essere adatta e perché funzioni su una qualsiasi piattaforma.

3. L’Implementazione

Quindi per spiegare lo standard in modo più semplice e per mostrarne il potenziale si è deciso di utilizzare un Serious Game. Come può essere strutturato?

3.1 Il Processo - Design Thinking

Il processo di lavoro che abbiamo portato avanti è riconducibile a quello del design thinking.
Il design thinking è un processo di progettazione centrato sulla persona e guidato dalla prototipazione. È strutturato in modo da far fronte alle problematiche che possono sorgere durante la fase di sviluppo di un prodotto.

E’ strutturato in 5 fasi che non sono strettamente lineari ma anzi le ultime servono anche a comprendere quali sono i problemi di quelle precedenti per poter aggiustare il tiro.

Le fasi del D.T. sono:

- Empathise
- Define
- Ideate
- Prototype
- Test

3.1.1 Empathise

La fase di empathise è volta alla conoscenza degli stakeholder così da individuarne i bisogni principali.
Tale processo di conoscenza e comprensione avviene tramite uno o più di quelli che sono i tipici strumenti della ricerca etnografica: intervista, osservazione questionario, focus group, ecc..

In questa fase abbiamo potuto vivere in prima persona le difficoltà con lo comprensione dello standard, che ci è apparso complesso e poco fruibile al cliente ultimo.

Abbiamo cercato di comprendere assieme a GS1 Italy quali potessero essere le funzioni di questo prodotto e quali fossero le nozioni più importanti da far arrivare all’utente. Alcuni incontri con potenziali clienti ci hanno permesso di approfondire quali fossero le difficoltà e i bisogni di chi poi avrebbe fruito del prodotto.

Abbiamo preso spunto da quelli che sono i corsi di formazione di GS1 in modo da orientarci sui contenuti del prodotto.

3.1.2 Define

In questa fase, grazie alle informazioni raccolte precedentemente si individuano i problemi degli utenti. Si vuole creare un resoconto dei bisogni e delle motivazioni degli utenti nell’uso di un prodotto. Gli strumenti sono Personas, Scenario, StoryBoard e Use cases.

Personas: Personaggi di fantasia creati sulla base delle ricerche effettuate. Lo scopo è rappresentare i diversi tipi di utenti e vedere le cose da un punto di vista diverso rispetto al proprio. Bisogni esplicitare dati stile carta d’identità, comportamenti abituali, esigenze o obiettivi, una storia che lo inserisca in una situazione in cui deve usare il prodotto o raggiungere un obiettivo.

Scenario: espansione della storia costruita per il personas che approfondisce i fattori umani e ambientali, fornisce il contesto in cui si muovono i personas, come si svolgono le loro esperienze e come manifestano i bisogni. Gli scenario fanno riferimento a esigenze specifiche e dettagliate che il personas cerca di soddisfare.
StoryBoard e Use Cases sono rappresentazioni particolari di scenari.

3.1.3 Ideate - brainstorming

Qui il team butta giù le prime idee su come sviluppare il prodotto.

- Brainstorming - In gruppo i partecipanti fanno emergere le proprie idee
- Braindumping - Le idee vengono scritte su post-it e poi discusse dal gruppo
- Brainwriting - Le idee vengono scritte su fogli, che verranno passati tra i partecipanti ed elaborate
- Brainwalking - I partecipanti si spostano tra diverse «ideation station»

Il nostro approccio, mio e dei relatori, in questa fase si è basato sul brainstorming:

3.1.3.1 Riferimenti

Una delle prime idee è stata quella di prendere spunto da un gioco di simulazione di tipo gestionale, **RollerCoaster Tycoon**, ma abbiamo subito compreso che le dinamiche dovevano essere differenti alla radice. Nel gioco in questione infatti, l'aspetto gestionale consiste nella costruzione pezzo per pezzo di un grande parco giochi. Si può quest'idea applicare alla nostra di gestione di una supply chain? Non ci è sembrato il caso di puntare su questo aspetto in quanto era ben lontano dal mondo degli eventi EPCIS, che prevedono strutture esistenti di aziende diverse.
Abbiamo però subito optato per un gioco che avesse un aspetto simile a quello del gioco di riferimento. **Un gioco quindi bidimensionale su mappa isometrica**.

3.1.3.2 Struttura

Il suddetto gioco di riferimento è strutturato a livelli, quali potrebbero essere i livelli per il nostro S.G.?
Da questo ragionamento sono nati i dubbi affrontati nella sessioni successive. Ci siamo chiesti se fosse meglio un gioco strutturato tutto su uno stesso livello e organizzato in obiettivi o se invece convenisse una organizzazione a livelli di difficoltà crescente.
Due idee principale sono venute fuori per la struttura livellare: la prima, prevedeva un'unica filiera che diventasse più articolata a ogni livello; la seconda invece verteva su filiere diverse (anche nel tipo di prodotti) ad ogni livello, eventualmente comunque più articolate con l'avanzare del gioco.
Comunque un sistema a obiettivi ci sembrava dovesse esserci a prescindere dalla struttura a livello singolo o a più livelli.

3.1.3.3 Gli Eventi

Un altro punto focale al centro delle prima discussioni è stato quello riguardo agli Eventi e al loro ruolo nel gioco.
Era chiaro dalle nostre idee di base che questi dovessero avere un ruolo centrale perché potessimo raggiungere il nostro scopo. Ma capire come inserirli nel gameplay si è rivelato tutt'altro che scontato.
Di tutte le possibili idee, due apparivano più azzeccate: gli eventi dovevano essere visibili ed espliciti e dovevano portare il giocatore a fare scelte decisive.

Inoltre abbiam deciso che, perché fossero utili alla comprensione dello standard, gli eventi dovessero essere strutturati in un modo che quanto meno richiamasse la loro struttura effettiva in EPCIS, invece che semplici oggetti testuali.

3.1.3.4 SinglePlayer o MultiPlayer?

Abbiamo considerato come avremmo potuto organizzare il gioco nei due casi. Sicuramente un gioco multiplayer si sarebbe prestato bene a due modalità: ogni giocatore gestisce una filiera e concorre per essere la migliore (tutti contro tutti) oppure, ogni giocatore gestisce una parte della filiera e deve quindi collaborare con gli altri per progredire insieme (tutti in squadra).
Abbiamo però optato per la modalità single player per due motivi: la realizzazione sarebbe sicuramente stata meno costosa in termini di lavoro e il gameplay sarebbe stato meno articolato, rendendo più facile per il fruitore vivere le esperienze formative e apprendere quanto volevamo apprendesse.

3.1.4 Prototype

In questa fase si creano approssimazioni delle idee migliori che possano essere testate con gli utenti e si definisce ogni singola interazione che l'utente deve compiere per raggiungere l'obiettivo desiderato

Un prototipo è una rappresentazione limitata di un progetto che consente agli stakeholder di interagire ed esplorare la soluzione proposta, allo scopo di comprendere più in profondità le implicazioni legate all'uso in contesti realistici. Il prototipo richiede la costante verifica fattuale delle scelte operate dal team. Il processo di prototipazione rapida a diversi livelli di fedeltà è essenziale per concretizzare il pensiero.

Il prototipo permette di discutere le idee tra stakeholder aiutando a cogliere i limiti e le potenzialità del prodotto, facilitare la comunicazioni all'interno del gruppo superando le barriere legate al linguaggio ed alla diversità culturale.

Serve a validare le proprie idee verificandone la sostenibilità/adeguatezza
Si possono distinguere due tipi principali di prototipo:

Prototype – low-fidelity

- Veloci da creare
- Basso costo
- Semplici

Esempi

- Paper prototype
- Sketch
- Stampe 3D
- Digital wireframe e
- mock-up

Prototype – high-fidelity

- Implementano la logica del funzionamento
- Utilizzano materiali simili a quelli del prodotto finale
- Hanno costi e tempi di sviluppo maggiori
- Tendono a polarizzare la discussione sugli aspetti

- superficiali
- Irrigidiscono la posizione degli sviluppatori
    - Che non vogliono più cambiarne le caratteristiche dopo lo sforzo necessario alla loro realizzazione
- Sono più rifiniti, spesso rappresentano l'aspetto del prodotto finale dal punto di vista grafico e funzionale
- Sono utili nei test di valutazione delle fasi avanzate per il feedback di clienti e utenti

Durante questa fase ho portato avanti due prototipi high-fidelity, usando due tecnologie differenti. Il primo si basa su un motore di gioco isometrico, Traviso.js, scritto in javascript. Il secondo è stato sviluppato su Unity, con C#.
Lo sviluppo del primo prototipo si è intrecciato con la fase precedente di ideazione e progettazione. Il secondo è stato cominciato quando il primo aveva raggiunto un buon livello di concretezza e plausibilità (nei sensi di come il prodotto sarebbe potuto essere).

È stato così messo appunto un progetto più definito sul quale si è basato lo sviluppo del secondo prototipo, il quale (grazie alla tecnologia usata) godeva di una maggiore stabilità e potenza, permettendo manipolazioni più precise e puntuali.
Sono spiegati nel dettaglio i progetti e lo sviluppo dei due prototipi nei paragrafi 3.2, 3.3, 3.4 e 3.5.

3.1.5 Testing

In questa fase i prototipi vengono mostrati e lasciati usare agli utenti, così da poter individuare i problemi di utilizzo. I Test consistono generalmente nell'osservazione di
una persona mentre interagisce col prodotto compiendo delle azioni.
Si distinguono in

- **Quantitativi**: richiedono che si eseguano misurazioni, come il calcolo della percentuale di successo o del tempo di risposta, sono molto rigorosi e necessitano di un campione di utenti ampio
- **Qualitativi**: sono molto più informali, vengono presi appunti sull'andamento del test e non è necessario avere un campione ampio, bastano anche solo 3 soggetti.

Strategie

- A/B test
- Scegliere i tester
- Scegliere le domande
- Neutralità
- Essere adattivi
- Tester attivi

A/B

Utile per sollecitare feedback critici (le persone tendono a trattenersi dal criticare apertamente i prototipi); presentando alternative, si consente di dire ciò che piace e non piace di ogni versione, ottenendo un feedback più onesto

I Tester
Chi testa i prototipi influenza l'utilità e la pertinenza del feedback. Nelle prime fasi un feedback semplice e approssimativo può essere efficace. Quando i prototipi diventano più dettagliati e più vicini al prodotto finale è utile considerare una gamma più ampia di utenti in modo da ottenere feedback migliori: colleghi, produttori, rivenditori, distributori e utenti finali avranno ciascuno i propri criteri per costruire o realizzare il prodotto e possono avere un impatto sull idea.

Le domande
Ogni test deve partire da domande fondamentali:

- Come è organizzata la navigazione globale?
- C’è un buon grado di usabilità?
- Il prodotto presenta una grafica accattivante?
- Come funziona il flusso di un determinato task?

Nonostante la presenza di domande chiave, spesso le sessioni di test possono rivelare problemi mai notati/pensati prima pertanto è importante mantenere la mente aperta durante il test.

Dopo il test è necessario decidere se è utile porre domande diverse durante le sessioni di test future

Neutralità
È essenziale essere il più oggettivi possibile ed evidenziare gli aspetti sia positivi che negativi della soluzione e non cercare di vendere l’idea.
Un feedback negativo è un’opportunità da cogliere esaminando ulteriormente la questione per capire cosa non va esattamente nella soluzione proposta, così da poter tornare indietro e migliorarla.

È importante evita di affezionarti troppo all’idea ed essere sempre pronti a smantellare, cambiare o persino abbandonare quando se ne presenta la necessità.

Essere adattivi
Importantissimo ancora è adottare una mentalità flessibile; se risulta che alcune componenti del prototipo distolgono l'attenzione dalle funzioni principali, bisogna rimuoverle o modificarle per riportare l'attenzione sugli elementi chiave dell’idea.

Se durante il test viene fuori che lo script pianificato non funziona bene è consigliato deviare e improvvisare per ottenere un miglior feedback dai tester

Tester attivi
Può essere molto utile spingere gli utenti a fornire idee, ad esempio, chiedendo come migliorare il prodotto o il servizio. Questo incoraggia gli utenti a fornire critiche utili e a contribuire per migliorare la soluzione.

Anche se non viene adottata l’idea proposta, il feedback probabilmente fornirebbe spunti sulle questioni che gli utenti hanno durante l'utilizzo del prodotto.

Test qualitativo «fai da te»
Vediamo un esempio di un semplice test qualitativo che si può mettere in atto.

Scopo:
individuare solo i problemi più gravi da risolvere prima del successivo ciclo di test

Partecipanti:
almeno 3

Task:
dipendono essenzialmente da ciò che è pronto da testare. Risulta utile prima pensare quali funzionalità siano più utili da indagare, per poi ideare una serie di compiti, di solito di difficoltà crescente, che l’utente dovrà svolgere (anche usando una task analisys)

Resoconto:
1-2 pagine sono più che sufficienti per riassumere i problemi riscontrati dagli utenti

Feedback capture grid

È un modo strutturato di organizzare il feedback raccolto dalle sessioni di test. Va usato durante il test come modo per prendere appunti oppure dopo il test per organizzare i vari feedback raccolti

I like, I wish, What if...
Simile al precedente, fornisce una struttura in cui è possibile raccogliere feedback.
Si invita l'utente a fornire un feedback aperto secondo tre tipi di dichiarazioni

- I Like
- I Wish
- What if

Sharing inspiring stories
Le storie sono strumenti potenti da utilizzare per ispirare il team a pensare nuove soluzioni
Dopo aver fatto una sessione di test sul prototipo, riunirsi in team per condividere le storie è un'attività molto utile per scoprire ciò che risuona tra i vari feedback
Ciò può aiutare a identificare idee e sentimenti su cui il team può lavorare quando pensa a nuove soluzioni

Rispetto al mio lavoro di Tesi, non sono arrivato alla fase di Test.

3.2 Il Primo Prototipo - Alto Livello

In base ai punti elencati nella sezione 2.3.5 sulla progettazione di un Serious Game qui elenco quali sono state le nostre scelte per quello che è il progetto del primo prototipo.
Sottolineo che lo sviluppo del prototipo e la messa appunto del progetto dello stesso sono avanzati in concomitanza e non in modo sequenziale, qui è riportato quello che è il risultato finale.

3.2.1 Scopo

Il dominio che il gioco vuole affrontare è quello delle **Supply Chain**, il mondo quindi della gestione di una filiera produttiva, sia nei rapporti tra le entità coinvolte, sia nell’organizzazione interna che ogni entità deve attuare in funzione del suo ruolo all’interno di una S.C..

Il fruitore si troverà in questo contesto a metter in gioco le sue abilità di **problem solving**, di **decision making** e di **apprendimento**.

Durante lo sviluppo di questo prototipo quale fosse il ruolo del giocatore non era ben definito. Poteva essere sia quello di manager responsabile della supply chain e del ruolo dell’azienda nella stessa, sia quello di un operatore che ha direttamente a che fare con le azioni che generano (o vorremmo generassero) eventi.

Lo **Scopo** che il giocatore deve raggiungere giocando è quello di aumentare la consapevolezza che la gestione digitale delle supply chain e lo scambio di dati lungo la filiera può aggiungere valore alla loro azienda. Questo è possibile tramite un aumento di tempestività, qualità ed elasticità, e una riduzione di sprechi, ri-lavorazioni e resi e ha come conseguenza una soddisfazione migliore del cliente.

3.2.2 Contenuto

Si vuole rappresentare il modello concettuale dello **standard EPCIS**. I riferimenti per riuscire in una rappresentazione puntuale sono stati reperiti dai documenti dello standard (casi d’uso).
Vengono rappresentati tramite l’ambiente simulato, in forma semplificata ma comunque sufficientemente esaustiva per quello che è lo scopo. il quale fa leva proprio sul valore che deriva dalla capacità di interpretare i dati.

Tutto ciò avviene nel contesto di una specifica supply chain tramite un modello a eventi discreti, abbastanza semplice da essere implementabile e sufficientemente realistico.

La traduzione in concreto di un concetto astratto aiuta al raggiungimento dello scopo.

3.2.3 Contesto

Il giocatore, probabilmente, si avvicina al gioco nell’ottica di una potenziale fonte di miglioramento per il proprio business, ma con un atteggiamento che potrebbe essere conservativo e restio alla novità. Probabilmente, rispetto allo scopo, il giocatore non parte con l’idea di avere bisogno di aumentare una propria consapevolezza ma pensa di saper far funzionare il suo lavoro senza bisogno di nuovi approcci, specialmente se complicati.
Il Serious Game dovrà agevolare un atteggiamento di curiosità verso il nuovo, tramite la leggerezza intrinseca dell’aspetto ludico. Il rischio è forse quello di cedere troppo spazio alla leggerezza per avvicinare il fruitore e perdere il fulcro dei contenuti, senza così raggiungere lo scopo.

In quest’ottica, l’obiettivo del giocatore sarà quello di mantenere efficiente la produzione di un impianto simulato, o di fare la sua parte all’interno di una realtà più ampia.

In questa fase non è sicuro in che modo sarà vissuto il conflitto. Potrebbe essere contro gli eventi negativi generati dalla simulazione o anche contro altri giocatori nel caso si prenda la strada del multiplayer.

3.2.4 Meccanica

Ci troviamo nel contesto di una filiera della carne, composta dalle fattorie, i siti di ingrasso, i macelli e i supermercati. Il giocatore ricopre il ruolo di una di queste entità, e osserva nel mondo simulato lo scorrere degli eventi.

La mappa di gioco è navigabile.

Ogni entità possiede un magazzino in uscita e uno in entrata. Questi magazzini non sono specifici del prodotto che trasportano ma servono solo come indicatori di disponibilità di spazio. Inoltre ogni entità ha un registro delle unità presenti all’interno del proprio complesso produttivo, anche queste generiche (vitelli, carne, ecc.)

Il giocatore può modificare le dinamiche esistenti tra la sua e le altre entità stipulando (o annullando) accordi commerciali
Questo avviene con semplici meccaniche di tipo punta-e-clicca, e delle finestre di interazione testuale.
In base a questi accordi la simulazione porta avanti in automatico diverse dinamiche commerciali tra il giocatore e i suoi partner. Il giocatore ha modo di vedere gli eventi generati da queste dinamiche. (es. il vitello trasportato dalla fattoria partner al mio macello, dove viene macellato dopo alcuni secondi).

Il giocatore può vedere la situazione della propria “azienda” sull’interfaccia utente. Questa è espressa sotto forma di percentuale dei magazzini occupata. È possibile vedere in modo analogo la situazione delle altre entità con cui il giocatore ha stipulato un accordo commerciale.

Gli accordi commerciali sono anche loro generici e servono solo ad avviare i processi di scambio merci, assolutamente automatici.

Sono presenti oggetti animati (i camion e i bovini) con cui il giocatore può interagire in modo molto blando. Entrambi sono completamente gestiti dal simulatore e danno un idea al giocatore del modo in cui le cose si svolgono e di come vengono generati gli eventi.

Alcune parti del gioco, non ben definite, avvengono all’interno di alcune delle entità, dove il giocatore può interagire con le dinamiche interne e comprendere più da vicino la generazione e la struttura degli eventi.

3.2.5 Storia e Narrativa

La storia è estremamente semplice e poco influenzabile (almeno nelle prime fasi). Più che una vera storia è un Soggetto.
La storia pertanto si riduce a questo:

Il giocatore si trova a ricoprire un certo ruolo legato al mondo della produzione carne. Deve stringere rapporti commerciali con altre entità in modo da costruire la filiera e far funzionare la sua azienda.

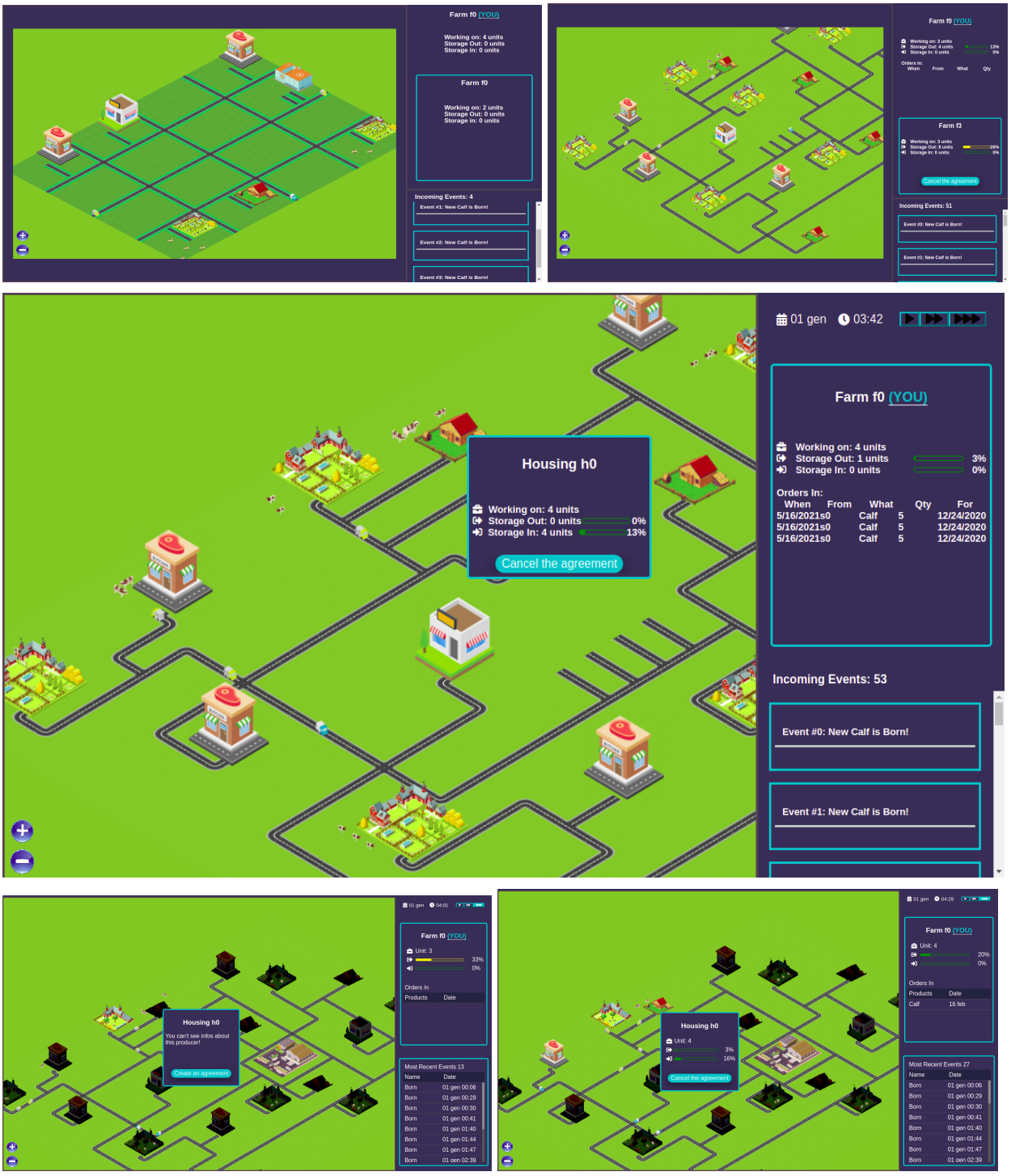
La narrazione avviene graficamente in modo implicito, tramite lo svolgersi delle dinamiche e lo scorrere degli eventi.

In mano al giocatore è solo la scelta sui rapporti commerciali che vuole creare, da cui derivano diversi sviluppi per le entità e i loro magazzini.

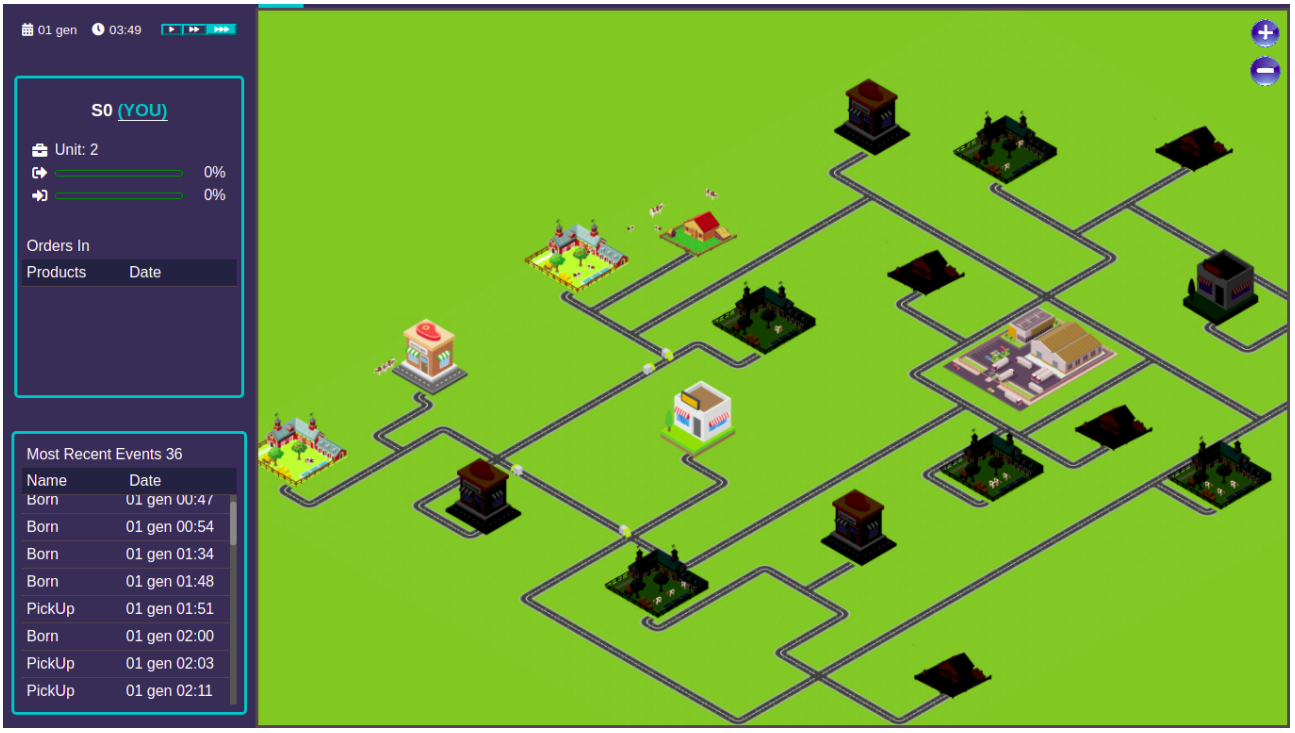
3.2.6 Estetica e Grafica

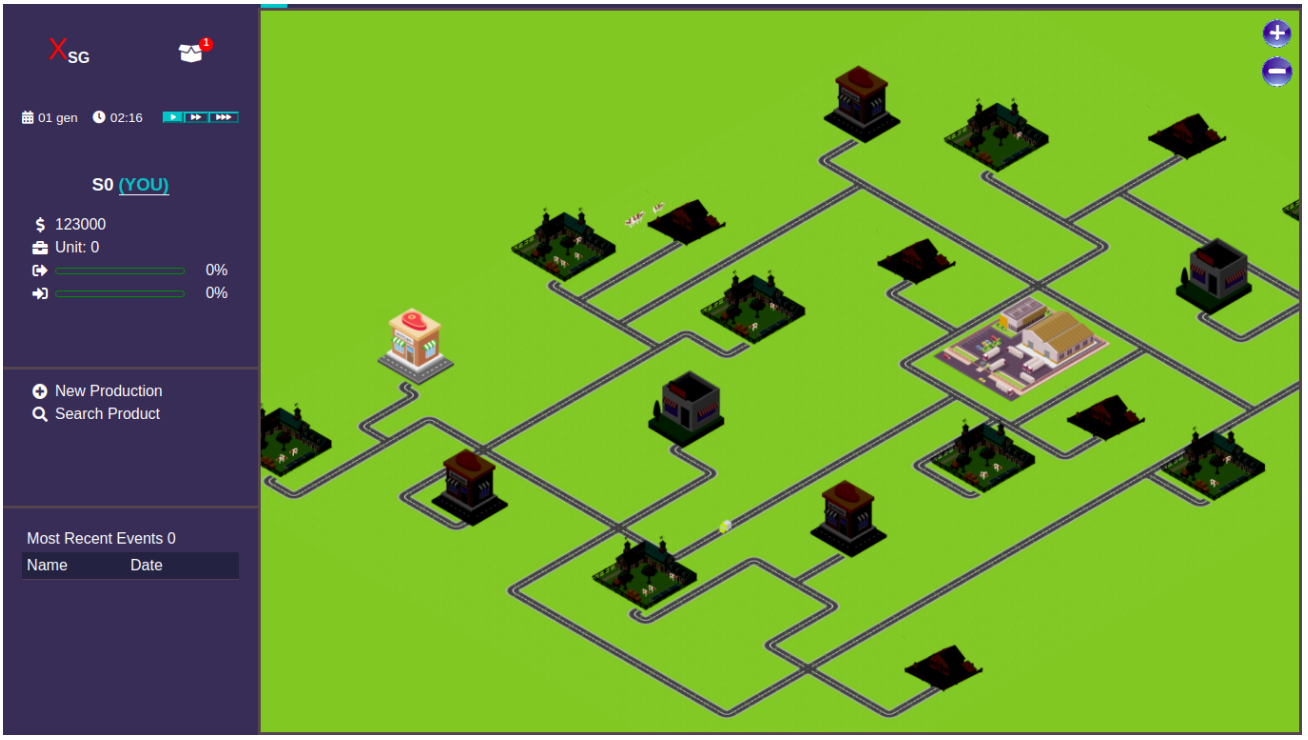
L’aspetto del gioco è bidimensionale isometrico, tipico dei giochi di tipo gestionale, e a tratti bidimensionale frontale, tipico dei punta e clicca. In entrambe le visualizzazioni è presente un’HUD in sovraimpressione sia per l’interazione che per il feedback. Questa gioca principalmente sui colori grigio, blu e azzurro e si colloca nei contorni dello schermo invadendo il centro solo in particolari situazioni. Non c’è comparto audio.

La posizione dell’interfaccia ha subito diverse modifiche nel corso dello sviluppo.
Ecco alcuni esempi:


Farm f0 (YOU)
Working on: 4 units
Storage Out: 0 units
Storage in: 0 units
Farm f0
Working on: 2 units
Storage Out: 0 units
Storage in: 0 units
Incoming Events: 4
Event #1: New Calf is Born!
Event #2: New Calf is Born!
Event #3: New Calf is Born!
Farm f0 (YOU)
Working on: 3 units
Storage Out: 4 units
Storage In: 0 units
Orders In:
When From What Qty
Farm f3
Working on: 3 units
Storage Out: 3 units
Storage In: 0 units
Cancel the agreement
Incoming Events: 51
Event #0: New Calf is Born!
Event #1: New Calf is Born!
01 gen 03:42
Housing h0
Working on: 4 units
Storage Out: 0 units 0%
Storage In: 4 units 13%
Cancel the agreement
Farm f0 (YOU)
Working on: 4 units
Storage Out: 1 units 3%
Storage In: 0 units 0%
Orders In:
When From What Qty For
5/16/2021s0 Calf 5 12/24/2020
5/16/2021s0 Calf 5 12/24/2020
5/16/2021s0 Calf 5 12/24/2020
Incoming Events: 53
Event #0: New Calf is Born!
Event #1: New Calf is Born!
Housing h0
You can't see infos about this produce!
Create an agreement
Farm f0 (YOU)
Unit: 3
33%
0%
Orders In
Products Date
Most Recent Events 13
Name Date
Born 01 gen 00:06
Born 01 gen 00:29
Born 01 gen 00:30
Born 01 gen 00:41
Born 01 gen 01:40
Born 01 gen 01:44
Born 01 gen 01:47
Born 01 gen 02:39
Housing h0
Unit: 4
3%
16%
Cancel the agreement
Farm f0 (YOU)
Unit: 4
20%
0%
Orders In
Products Date
Calf 16 feb
Most Recent Events 27
Name Date
Born 01 gen 00:06
Born 01 gen 00:29
Born 01 gen 00:30
Born 01 gen 00:41
Born 01 gen 01:40
Born 01 gen 01:44
Born 01 gen 01:47
Born 01 gen 02:39

Alla fine si è optato per un aspetto con interfaccia sulla sinistra.





L’interfaccia, molto snella, mostra i dettagli del giocatore (capitale, unità prodotte, situazione magazzini), gli eventi (semplificati con solo nome e data) e alcune possibili azioni (sui prodotti).

Le finestre di interazione escono a schermo nei punti in cui il giocatore clicca (a ridosso dell’oggetto di riferimento)

In alto una barra celeste mostra l’avanzare del tempo, mentre in alto sulla destra sono presenti due pulsanti per lo zoom.

Gli attori con cui non possiamo interagire sono mostrati oscurati (all’inizio tutti).
Quando si crea un accordo le entità si illuminano e diventai possibile vederne i dettagli.

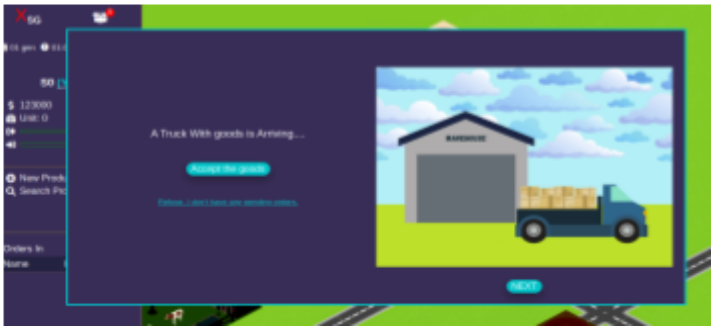
Housing h0 prima dell’accordo.

Housing h0 dopo l’accordo

Vediamo un esempio nel quale noi siamo un macello e, quando ci accordiamo con una fattoria, un vitello viene trasportato verso il nostro macello. All’arrivo si avvia l’interfaccia bidimensionale per iniziare una fase di gioco interna all’entità.

un carro bestiame trasporta un bovino al nostro macello

All’arrivo si avvia una fase frontale



Gli oggetti hanno un aspetto costituito da una o poche sprite, con texture molto semplici e colorate.
L’animazione è quasi del tutto assente, tranne qualche illusione della stessa data da alcuni cambi di texture o colore.
L’aspetto vuole essere leggero e di facile fruizione, con una grafica piacevole ma non pregnante.

3.3 Il Primo Prototipo - Basso livello: Traviso.js

Per l’implementazione del prototipo ho utilizzato Traviso.js, un motore isometrico open source scritto in JavaScript consigliatomi dal Professore.
La documentazione per questo motore è molto scarsa, il che lo rende di difficile comprensione.
E’ molto carente in termini di ottimizzazione, e le performance sono decisamente insostenibili.
Tuttavia nell’ambito di un primo prototipo si è rivelato funzionale.

3.3.1 Traviso.js

Nella stesura di questo paragrafo ho fatto riferimento a [15] e [16].

Traviso è un motore JS open source che vuole rendere di facile sviluppo le applicazioni isometriche che girano su un browser web. Punta alla massima flessibilità per permettere allo sviluppatore di implementare su di esso la propria logica. È costituito da una serie di algoritmi ottimizzati sulla base del motore di rendering **pixi.js**.

Basi sul Mondo Isometrico
La creazione del mondo isometrico su Traviso è abbastanza diretta; il motore infatti richiede soltanto le sprite per dare l'aspetto visivo e un file JSON che il motore usa per mettere ogni cosa al suo posto. Il rendering è supportato da pixi.

Una **`tile`** è definita da due parametri:
`Iso-Height`: Altezza della Tile. default: 64px

`Iso-Angle`: L'angolo tra il bordo isometrico and la diagonale (orizzontale) isometrica della Tile. default :30 gradi.

Nel creare l'aspetto degli oggetti è conveniente basarsi sulle misure delle tile in quanto il posizionamento dello stesso sarà espresso in un numero discreto di tile.

Questo è invece un esempio del file JSON

```
{
    "tiles": {
        "1":  {
            "movable": true,
            "path": "grassTile.png"
        }
    },
    "objects": {
        "1": {
            "movable": false,
            "interactive": false,
            "rowSpan": 1,
            "columnSpan": 1,
            "visuals": {
                "idle": {
                    "frames": [
                        { "path":
"house.png" }
                    ]
                }
            }
        }
    },
    "groundMap": [
        { "row": "1 ,1 ,1 ,1" },
        { "row": "1 ,0 ,0 ,1" },
        { "row": "1 ,0 ,0 ,1" },
        { "row": "1 ,1 ,1 ,1" }
    ],
    "objectsMap": [
        { "row": "0 ,0 ,0 ,0" },
        { "row": "0 ,1 ,0 ,0" },
        { "row": "0 ,0 ,1 ,0" },
        { "row": "0 ,0 ,0 ,0" }
    ]
}
```

La spiegazione della struttura del file è riservata a una sezione successiva.

Questo pezzo di codice inizializza il motore:

```
<script src="js/pixi.dev.js"></script>
<script src="js/traviso.dev.js"></script>

<script>

    ////// Here, we initialize the pixi application
    var pixiRoot = new PIXI.Application(800, 600, { backgroundColor : 0x6BACDE });

    // add the renderer view element to the DOM
    document.body.appendChild(pixiRoot.view);

    ////// Here, we create our traviso instance and add on top of pixi

    // engine-instance configuration object
    var instanceConfig = {
        mapDataPath: "mapData.json", // the path to the json file that defines map data,
required
        assetsToLoad: ["grassTile.png", "house.png"] // array of paths to the assets that
are desired to be loaded by traviso, no need to use if assets are already loaded to PIXI
cache, default null
    };

    var engine = TRAVISO.getEngineInstance(instanceConfig);
    pixiRoot.stage.addChild(engine);

</script>
```

Quanto definito qui sopra è il setting minimo di funzionamento, è possibile customizzare la propria applicazione.

Customizzare l'applicazione
Nella creazione di un'istanza di motore Traviso è necessario definire un oggetto configurazione.

```
var instanceConfig =
{
    mapDataPath : "mapData.json",
    assetsToLoad : ["grassTile.png", "waterTile.png", "house.png", "box.png"],
};

var engine = TRAVISO.getEngineInstance(instanceConfig);
```

`mapDataPath` è l'unico componente richiesto e specifica il path per il file JSON che definisce la mappa.
`assetsToLoad` è invece un array di path agli assets che si vuole siano caricati da Traviso.

Qui un elenco di tutti gli altri parametri personalizzabili.

```
mapDataPath {String} // the path to the json file that defines map data, required
assetsToLoad {Array(String)} // array of paths to the assets that are desired to be loaded
by traviso, no need to use if assets are already loaded to PIXI cache, default null

minScale {Number} // minimum scale that the DisplayObjectContainer for the map can get,
default 0.5
maxScale {Number} // maximum scale that the DisplayObjectContainer for the map can get,
default 1.5
numberOfZoomLevels {Number} // used to calculate zoom increment, default 5
initialZoomLevel {Number} // initial zoom level of the map, should be between -1 and 1,
default 0
instantCameraZoom {Number} // specifies whether to zoom instantly or with a tween
animation, default false

tileHeight {Number} // height of a single isometric tile, default 74
isoAngle {Number} // the angle between the top left edge and the horizontal diagonal of a
isometric quad, default 30

initialPositionFrame {Object} // frame to position the engine, default { x : 0, y : 0, w :
800, h : 600 }
initialPositionFrame.x {Number} // x position of the engine, default 0
initialPositionFrame.y {Number} // y position of the engine, default 0
initialPositionFrame.w {Number} // width of the engine, default 800
initialPositionFrame.h {Number} // height of the engine, default 600

pathFindingType {Number} // the type of path finding algorithm two use, default
TRAVISO.pfAlgorithms.ASTAR_ORTHOGONAL
pathFindingClosest {Boolean} // whether to return the path to the closest node if the
target is unreachable, default false

followCharacter {Boolean} // defines if the camera will follow the current controllable or
not, default true
instantCameraRelocation {Boolean} // specifies whether the camera moves instantly or with a
tween animation to the target location, default false
instantObjectRelocation {Boolean} // specifies whether the map-objects will be moved to
target location instantly or with an animation, default false

changeTransperancies {Boolean} // make objects transparent when the controllable is behind
them, default true

highlightPath {Boolean} // highlight the path when the current controllable moves on the
map, default true
highlightTargetTile {Boolean} // highlight the target tile when the current controllable
moves on the map, default true
tileHighlightAnimated {Boolean} // animate the tile highlights, default true
tileHighlightFillColor {Number(Hexadecimal)} // color code for the tile highlight fill
(this will be overridden if there is a highlight-image defined in the map data file),
default 0x80d7ff
tileHighlightFillAlpha {Number} // alpha value for the tile highlight fill (this will be
overridden if there is a highlight-image defined in the map data file), default 0.5
tileHighlightStrokeColor {Number(Hexadecimal)} // color code for the tile highlight stroke
(this will be overridden if there is a highlight-image defined in the map data file),
default 0xFFFFFF
tileHighlightStrokeAlpha {Number} // alpha value for the tile highlight stroke (this will
be overridden if a highlight-image is defined), default 1.0
```

```
dontAutoMoveToTile {Boolean} // when a tile selected don't move the controllable
immediately but still call 'tileSelectCallback', default false
checkPathOnEachTile {Boolean} // engine looks for a path every time an object moves to a
new tile on the path (set to false if you don't have moving objects other then your
controllable on your map), default true

mapDraggable {Boolean} // enable dragging the map with touch-and-touchmove or
mousedown-and-mousemove on the map, default true

backgroundColor {Number(Hexadecimal)} // background color, if defined the engine will
create a solid colored background for the map, default null
useMask {Boolean} // creates a mask using the position frame defined by
'initialPositionFrame' property or the 'posFrame' parameter that is passed to
'repositionContent' method, default false

engineInstanceReadyCallback {Function} // callback function that will be called once
everything is loaded and engine instance is ready, default null
tileSelectCallback {Function} // callback function that will be called when a tile is
selected, default null
objectSelectCallback {Function} // callback function that will be called when a tile with
an interactive map-object on it is selected, default null
objectReachedDestinationCallback {Function} // callback function that will be called when
any moving object reaches its destination, default null
otherObjectsOnTheNextTileCallback {Function} // callback function that will be called when
any moving object is in move and there are other objects on the next tile, default null
objectUpdateCallback {Function} // callback function that will be called every time an
objects direction or position changed, default null
```

Struttura del file Dati

Tiles

La sezione "tiles" definisce le possibili tile utilizzabili nella propria mappa.

```
"tiles": {
    "1":  { "movable": true, "path": "grassTile.png" },
    "2":  { "movable": false, "path": "waterTile.png" }
}
```

Un item tile possiede un id univoco. Questi id saranno usati in "groundMap" più avanti nel file per identificarne la posizione. Lo "0" rappresenta le tile vuote, non si può quindi usare come id per una tile.

Una tile dovrebbe includere i seguenti attributi:
movable: (true/false) Definisce se un character può muoversi dentro questa tile.
path: (String) path alla sprite

Objects

Similmente, la sezione "objects" definisce gli oggetti disponibili che possono essere utilizzati.

```
"objects": {
    "1": {
        "movable": false,
        "interactive": false,
        "rowSpan": 1,
        "columnSpan": 1,
        "noTransparency": true,
        "floor": false,
        "visuals": {
            "idle": {
                "frames": [
                    { "path": "boxes.png" }
                ]
            }
        }
    }
}
```

Anche l'object ha un id univoco che verrà però utilizzato dento "objectsMap" per identificarne la posizione. Ancora, "0" significa nessun oggetto.

I seguenti attributi dovrebbero essere inclusi nella definizione di un oggetto:

`movable`: (true/false) Definisce se altri personaggi possono muoversi nella tile su cui sta questo oggetto.

`interactive`: (true/false) Specifica se l'utente può selezionare o interagire con l'oggetto. II motore usa le callback per avvisare dell'interazione avvenuta.

`rowSpan`: (Integer 0<) Specifica la dimensione dell'oggetto in termini di righe.

`columnSpan`: (Integer 0<) Specifica la dimensione dell'oggetto in termini di colonne.

`noTransparency`: (true/false) Definisce se l'oggetto è trasparente o meno quando il controller principale è dietro di esso.

`floor`: (true/false) Specifica se l'oggetto è un oggetto del pavimento.

`visuals`: (Object) Contiene tutti gli sprite dell'oggetto (almeno una con id "idle")

Ecco un esempio delle sprite usate da un oggetto nella situazione idle, questo da il senso dell'animazione.

```
"idle": {
    "frames": [
        { "path": "hero_stand_1.png" },
        { "path": "hero_stand_2.png" },
        { "path": "hero_stand_3.png" },
        { "path": "hero_stand_4.png" },
        { "path": "hero_stand_5.png" },
        { "path": "hero_stand_6.png" }
    ]
}
```

Le texture e le animazioni possono essere create in due modi:

Usando la proprietà "frames" o in una singola linea sfruttando i nomi dei vari path, se questi sono formattati nel giusto modo. es.

```
"move_ne": {
    "frames": [
        { "path": "hero_move_ne_x.png" },
        { "path": "hero_move_ne_w.png" },
        { "path": "hero_move_ne_y.png" },
        { "path": "hero_move_ne_z.png" },
        { "path": "hero_move_ne_t.png" },
        { "path": "hero_move_ne_v.png" }
    ]
}
```

Oppure

```
"move_ne": { "path": "hero_move_ne_", "startIndex": 1, "numberOfFrames": 6, "extension":
"png" }
```

Gli attributi sono i seguenti:

`path`: path alla texture di riferimento.

`extension`: Estensione dell'immagine

`numberOfFrames`: (Integer 0<) Numero di frame in un'animazione.

`startIndex`: (Integer) Per la seconda modalità, indice da cui partire per scegliere la sequenza delle texture.

`ipoc`: (Integer) Punto della sprite per l'interazione (offset di colonne)

`ipor`: (Integer) Punto della sprite per l'interazione (offset di colonne)

Gli id di base per le animazioni sono:

`idle_s, idle_sw, idle_w, idle_nw, idle_n, idle_ne, idle_e, idle_se, move_s, move_sw, move_w, move_nw, move_n, move_ne, move_e, move_se`

Ma se ne possono creare di personalizzati.

Ground Map
Definisce lo strato di terreno della mappa:

```
"groundMap": [
    { "row": "1 ,1 ,1 ,1" },
    { "row": "1 ,0 ,0 ,1" },
    { "row": "1 ,0 ,0 ,1" },
    { "row": "1 ,1 ,1 ,1" }
]
```

In questo esempio le tile di tipo 1 sono il contorno mentre il centro non contiene alcuna tile.
Esiste anche la possibilità di definire il terreno con una singola immagine precaricata.

Object Map
Definisce lo strato di oggetti della mappa, i quali poggiano sul terreno in corrispondenza a goundMap:

```
"objectsMap": [
    { "row": "0 ,0 ,0 ,0" },
    { "row": "0 ,1 ,0 ,0" },
    { "row": "0 ,0 ,1 ,0" },
    { "row": "0 ,0 ,0 ,0" }
]
```

Le due map devono avere la stessa dimensione. "0" vuol dire nessuno oggetto.
Controllo della Camera
Traviso possiede dei metodi dedicati alla gestione della camera.

Questa è una lista dei suddetti metodi:

- `centralizeToCurrentExternalCenter( instantRelocate=false )` : Questo metodo modifica la camera perchè punti al centro della mappa.
- `centralizeToCurrentFocusLocation( instantRelocate=false )` : Centra la camera nella posizione di focus attuale.
- `centralizeToLocation( c, r, instantRelocate=false )` : Centra la camera sulla posizione dichiarata per colonna e riga.
- `centralizeToObject( obj )` : Centra la camera sull'oggetto.
- `focusMapToLocation( c, r, zoomAmount )` : Centra la camera alla posizione e zooma.
- `focusMapToObject( obj )` : Centra la camera sull'oggetto e zooma.
- `setZoomParameters( minScale=0.5, maxScale=1.5, numberOfZoomLevels=5, initialZoomLevel=0, instantCameraZoom=false )` : Setta i parametri di zoom.
- `zoomOut( instantZoom=false )` : Fa zoom-out di un livello.
- `zoomIn( instantZoom=false )` : Fa zoom-in di un livello.
- `zoomTo( zoomAmount, instantZoom=false )` : zooma direttamente a un certo livello

Callbacks
Traviso possiede una serie di metodi callbacks, definibili nell'oggetto configurazione che si passa all'istanza di motore traviso.
esempio:

```
// engine-instance configuration object
var instanceConfig = {
    mapDataPath : "mapData.json", // the path to the json file that defines map data,
required
    assetsToLoad : ["../assets/assets_map.json", "../assets/assets_characters.json"], //
array of paths to the assets that are desired to be loaded by traviso, no need to use if
assets are already loaded to PIXI cache, default null

    engineInstanceReadyCallback : onEngineInstanceReady, // callback function that will be
called once everything is loaded and engine instance is ready, default null
    tileSelectCallback : onTileSelect, // callback function that will be called when a tile
is selected, default null
    objectSelectCallback : onObjectSelect, // callback function that will be called when a
tile with an interactive map-object on it is selected, default null
    objectReachedDestinationCallback : onObjectReachedDestination, // callback function
that will be called when any moving object reaches its destination, default null
    otherObjectsOnTheNextTileCallback : onOtherObjectsOnTheNextTile // callback function
that will be called when any moving object is in move and there are other objects on the
next tile, default null
};

// create the engine
var engine = TRAVISO.getEngineInstance(instanceConfig);
```

Per ogni callback dichiarata deve poi essere definito il metodo.
esempio:

```
function onEngineInstanceReady()
{
    stage.addChild(engine);
}

function onObjectSelect(obj)
{
       engine.moveCurrentControllableToLocation(obj.mapPos);
}

function onObjectReachedDestination(obj)
{
    var objectsOnDestination = engine.getObjectsAtLocation(obj.mapPos);
    for (var i=0; i < objectsOnDestination.length; i++)
    {
        // check if there is a flag on the destination tile
        if(objectsOnDestination[i].type === 10)
        {
            obj.changeVisual("flip", false, true, flipAnimFinished);
            break;
        }
    }
}
// this method will be called when the custom flip anim finished
function flipAnimFinished (obj)
{
    // change the visual of the object so that it will face its last direction
    obj.changeVisualToDirection(obj.currentDirection, false);
}
```

```
function onOtherObjectsOnTheNextTile(movingObject, objectsOnNewTile)
{
    var boxAnim;
    for (var i=0; i < objectsOnNewTile.length; i++)
    {
        // check if there are boxes on the next tile
        if(objectsOnNewTile[i].type === 12)
        {
            boxAnim = createAndStartBoxAnim();
            engine.addCustomObjectToLocation(boxAnim, objectsOnNewTile[i].mapPos);
            engine.removeObjectFromLocation(objectsOnNewTile[i]);
        }
    }
}
```

Movimento

A supporto del movimento in Traviso è presente una classe, non istanziabile direttamente dallo sviluppatore, detta **MoveEngine**.

Tale Classe, istanziata all'interno dell'istanza del motore traviso, contiene tutte le proprietà di funzionamente del movimento nel gioco, come la velocità di default o la lista degli oggetti mobili, e possiede i metodi pubblici che supportano il movimento degli oggetti all'interno della mappa. Qui sono implementati anche gli algoritmi ottimizzati per il pathfinding all'interno della mappa.

I metodi:

- `addMovable`
- `addNewPathToObject`
- `addTween`
- `destroy`
- `easeInOutQuad`
- `easeInQuad`
- `easeOutQuad`
- `getEasingFunc`
- `killTweensOf`
- `linearTween`
- `prepareForMove`
- `removeAllMovables`
- `removeAllTweens`
- `removeMovable`
- `removeTween`
- `run`
- `setMoveParameters`

3.3.2 Il primo Prototipo con Traviso.js

L'approccio al motore è avvenuto anzitutto con dei tentativi semplici di definizione di una mappa e di movimento di oggetti al suo interno. Così ho iniziato a costruire quella che sarebbe stata la logica di gestione del traffico.

Ottenuti questi primi risultati di base ho provato a dar loro un contesto tramite l'integrazione di sprite inerenti al mondo della filiera della carne. In questo modo avevo un paio di fattorie collegate da strade. Nelle fattorie erano presenti dei bovini che si muovevano. I bovini venivano trasportati da una fattoria a un'altra tramite dei furgoncini che si muovevano su strada rispettando regole base di gestione del traffico.

Per dare vita all'idea della simulazione ho usato delle chiamate a tempo usando **setTimeout()**. In questo modo i vitelli potevano dare l'impressione di crescere tramite dei cambi di sprite temporali e potevano venir trasportati cons cadenza temporale.

Vediamo l'implementazione attraverso la suddivisione in file del progetto:

index.html
Questa è l'unica pagine web presente nel progetto ed è quella al cui interno si svolge il gioco nella sua interezza.
Al suo interno è definita la suddivisione del gioco in sezioni (con riferimento al foglio di stile) e anche la parte di script (js) che instanzia l'engine e chiama una serie di funzioni per settare il gioco.
styles.css
Questo file contiene tutti gli stili usati nel progetto. Per lo più ho usato flex per la disposizione delle sezioni nello spazio.
Alcuni stili definiscono l'aspetto delle sezioni, mentre altri sono specifici di singoli elementi dell'interfaccia.
setup.js
Questo file contiene i metodi necessari al setup del gioco.
Nel metodo getActors(), facendo riferimento alla mappa di gioco, prende nota di quelle che sono le entità presenti nel gioco salvandole all'interno di Maps on chiavi di riferimento. Per ogni oggetto che trova, in base all'id che possiede nel file JSON, ne definisce il tipo e alcune variabili specifiche del tipo Definisce anche una serie di variabili indipendenti dal tipo chiamando un'altra funzione detta getActor(obj).
La funzione replicateActor() crea una copia di una certa entità. Questo metodo è necessario quando il giocatore stringe accordi con un'entità e la texture di questa cambia.

interface.js
Questo file contiene una serie di metodi per la gestione dell'interfaccia.
Per lo più si tratta dell'implementazione di alcune callback di interazione definite nell'istanza del motore di gioco, quali la selezione di una tile o di un oggetto. ALl'interno di questi metodi ne vengono chiamati di altri che is occupano di mostrare/nascondere le finestre di interazione.
Il contenuto di queste finestre è riempito tramite le proprietà degli oggetti selezionati che sono definiti nelle Map globali.
Tramite la callback infatti si accede all'id dell'entità, e tramite questo si accede all'oggetto che la rappresenta tramite le Map e di conseguenza ai suoi valori di riferimento.

Nel settare ciò che deve essere rappresentato nelle finestre di interazione sono ssettate anche le callback dei bottoni, che vanno a tenere traccia dell'esistenza di un accordo commerciale tra il giocatore e le altre entità.

È presente il metodo per l’aggiunta e messa in funzione dei bottoni di zoom e quello che mostra l’avanzare del tempo. Qui è gestita anche rappresentazione degli eventi.

movement.js
Qui sono presenti i metodi per la gestione del movimento di tutti gli oggetti mobili, quindi i furgoncini e i vitelli, e sono implementate le callback legate al MoveEngine.

Le funzioni generiche contengono degli switch case che decidono come comportarsi in base al tipo di oggetto che si sta muovendo.

Il movimento delle vacche è totalmente casuale, e cadenzato dai timeout. È implementato un algoritmo che si assicura che il movimento sia relegato alle sole tile consentite alle vacche.

Quello dei furgoncini è invece dipendente dalla destinazione, ed è quindi necessario un algoritmo di ottimizzazione della scelta della direzione. Quest’ultimo è implementato nel file traffic.js.

Qui è implementato l’algoritmo che decide se è necessario (e possibile, in base al numero massimo di furgoncini presenti in contemporanea) creare un altro furgoncino.

Nella callback, nel caso del furgoncino, hanno luogo tutti i cambiamenti delle entità coinvolte conseguenti al completamento di una fase di trasporto.

Un trasporto consiste in:
Richiesta furgoncino: L’algoritmo controlla se ci sono furgoncini liberi al momento (tramite un vettore) e, se no, controlla se c’è spazio per generarne uno nuovo. Se la risposta è ancora no, viene settato un timeout per un nuovo tentativo, altrimenti viene creato un furgoncino (aggiunto poi al vettore) e gli vengono settati i parametri, in particolare la destinazione intermedia (carico merci) e quella finale (consegna merci). Viene settata la dest. intermedia come prossima destinazione e il furgoncino è dichiarato occupato.

Arrivo alla destinazione intermedia: Viene caricata la merce e aggiornata la situazione dei magazzini dell’entità raggiunta (storage out decrementato di uno). E viene settata la destinazione finale come prossima destinazione.

Arrivo alla destinazione finale: Viene consegnata la merce e quindi aggiornati i magazzini dell’entità raggiunta (storage In incrementato di uno) ed eventualmente creata la parte visiva necessaria a capire cosa avviene dopo la consegna. Il furgoncino viene poi dichiarato libero e viene settato il centro trasporti (punto di spawn) come prossima destinazione.

traffic.js
Qui è implementato l’algoritmo che gestisce il traffico.
All’inizio si prende nota della forma della mappa, in particolare degli incroci all’interno di un vettore e della loro situazione (liberi o meno) in una Map (inizialmente tutti settati su libero).
Quando un furgoncino deve entrare in un incrocio controlla se è occupato. Se lo è si mette in attesa che si liberi all’interno di un’altra Map.
Un furgoncino che è riuscito a entrare setta l’incrocio come occupato. Quando ne esce, lo setta come libero e, se qualcuno e in attesa,lo avvisa che è libero e lo ri setta come occupato.

time.js
In questo file sono implementate le funzioni di gestione del tempo, lo scorrere del tempo, le date degli eventi e degli ordini. Il tutto avviene tramite Timeout.
È gestita qui anche la velocità di playback del gioco.
La gestione di questo aspetto si è rivelata problematica in quanto non solo andava gestita la cadenza del timeout ma anche la velocità degli oggetti in movimento.
Per modificare la velocità di oggetti in movimento è necessario distruggerli e ri crearli.
Pertanto per modificare la velocità di playback viene settata una variabile di controllo (così tutti i timeout sono modificati) e poi viene chiamata una funzione di pausa che prende nota degli oggetti in movimento prima di distruggerli.
Dopodichè viene chiamata una funzione play che controlla se deve ri creare eventuali oggetti e li setta alla velocità giusta in base alla variabile di controllo.

foreground.js
Questo file implementa (solo in parte) le funzioni di passaggio dalla modalità di gioco con vista sulla mappa a quella di zoom al'interno di un'entità. Al momento del passaggio al prototipo successivo non eseguiva nessuna funzione particolare eccetto quella visiva di simulare un passaggio da un livello all'altro.

actors_events.js
Infine qui sono modellate tutte le dinamiche di simulazione delle varie entità, che riguardano la nascita, crescita e macello dei vitelli (tutto con timeout), più la richiesta di tutti i trasporti tra le entità. In corrispondenza di ciò sono modellati gli eventi corrispondenti.
È importante che per garantire la continuità degli oggetti che vengono trasportati, e quindi di fatto distrutti e poi ricreati, ho dovuto tenere traccia degli attributi. Questo è fatto passandoli al furgoncino responsabile del trasporto.

3.3.3 Conclusioni sul Primo Prototipo

Il primo prototipo è stato estremamente utile per capire in che direzione ci dovevamo muovere e per capire se le nostre idee avevano senso una volta messe in pratica. La tecnologia usata è assolutamente inadatta allo sviluppo di un grosso progetto, in quanto il supporto di base è scarso, la personalizzazione minima e l'ottimizzazione insufficiente.
Il primo prototipo pur essendo un prodotto con poche funzionalità eragià troppo pesante per essere supportato.

3.4 Secondo Prototipo - Alto Livello

Qui elenchiamo in cosa il nuovo prototipo differisce da quello vecchio. Viene omesso ciò che è rimasto pressoché identico.

3.4.1 Scopo

Il dominio del gioco è più specifico rispetto a quello ideato nel primo prototipo: ci concentriamo sul “**Supply Chain Management**”.

Si è deciso che il ruolo del giocatore dovesse essere solo quello del **Manager**, in particolare di “**supply chain manager**”, il responsabile quindi del ruolo dell’azienda in filiera.

Nello specifico, il gioco si rivolge a persone che svolgono nella propria azienda il ruolo di “logistic manager” (per quanto riguarda la supply chain) e sono quindi responsabili della pianificazione e gestione dei beni in ingresso e in uscita, l’uso del magazzino, del trasporto presso i punti produttivi e della spedizione. Inoltre curano la gestione delle scadenze e le giacenze, ottimizzano i costi e l’efficienza dei processi nel rispetto dei budget assegnati e delle norme e dei regolamenti vigenti.

Ciò che il giocatore già conosce del dominio affrontato è la realtà della supply chain di cui l’azienda fa parte ma solo dal punto di vista del proprio ruolo, quindi i clienti e i fornitori prossimi alla loro posizione nella catena.

Lo **Scopo** che il giocatore deve raggiungere è rimasto pressoché invariato.

3.4.2 Contenuto & Contesto

Il contenuto non è variato da come era definito nel precedente prototipo.

Per quanto riguarda il contesto è stato specificato l’aspetto del conflitto: considerato che, al punto dello sviluppo di questo secondo prototipo, siamo più indirizzati verso l’approccio single player, il conflitto è vissuto dal giocatore nei confronti del simulatore, che genera eventi potenzialmente avversi.

3.4.3 Meccanica

Iniziamo specificando ancora che quanto non dichiarato apertamente è rimasto invariato dal precedente prototipo.

L’**obiettivo principale** del giocatore è mantenere efficiente la produzione di un impianto simulato attraverso la raccolta e l’interpretazione dei dati relativi al processo e le conseguenti scelte operative.
Bisognerà pertanto, garantire la presenza di input e la consegna dei beni in uscita, garantire l'approvvigionamento dei punti produttivi, ridurre gli sprechi (gestendo i beni relativi alla catena del freddo e quelli in scadenza), gestire richiami di prodotti e lamentele e gestire le risorse (tempo e denaro).

È stata data una definizione più specifica alle entità.

Si fa importante distinzione tra quelli che sono gli attori e quelle che sono le unità produttive. Gli attori vogliono modellare le aziende o attività commerciali, le quali sono dotate di una o più unità produttive.
Gli attori hanno un capitale, mentre le unità produttive hanno i magazzini in entrata e in uscita. Le unità produttive hanno pertanto lo stesso indicatore di salute che avevano le entità nel prototipo precedente, non vale lo stesso per gli attori, i quali non sono renderizzati.

Per questo prototipo si è optato per una filiera differente, ovvero quella della **produzione di biscotti**. Gli attori coinvolti sono la **catena del freddo**, che si occupa dei prodotti caseari, la **catena dei secchi**, che si occupa di frumento e farine, la **catena del cacao** e la **catena di distribuzione**, Il giocatore ricopre il ruolo poi del produttore dei biscotti che si interfaccia con gli altri attori.

Viene mantenuto l'aspetto simulativo che indica l'esistenza di un dato rapporto commerciale (con il trasporto di merci tra unità produttive se i rispettivi attori sono in accordo) ma è presente qui, nelle fasi più avanzate di gioco, la possibilità di avere interazione più profonde, come la possibilità di fare e ricevere ordini e acquisti.

Rispetto alla propria azienda sono state aggiunte diverse possibilità di azione inerenti l'interazione con gli altri attori e la gestione della produzione della propria azienda.

In funzione della meccanica di gioco (e non solo) il gioco può essere suddiviso in 3 fasi:

Fase 1
La fase 1 è quella introduttiva allo standard e alla realtà simulata dal gioco.
L'ingresso in questa fase avviene come uno zoom all'interno di una unità produttiva nel momento dell'arrivo di un camioncino.
A questo punto la struttura è a livelli bidimensionali di tipo punta e clicca. Questi livelli sono atti alla comprensione degli eventi EPCIS, come vanno letti e come vanno interpretati.
Non è presente un vero e proprio tutorial ma sono presenti dei tip all'inizio di ogni livello che permettono la comprensione immediata delle meccaniche.

Questa fase è composta da 4 livelli il cui scopo è conoscere gli eventi EPCIS.
Livello 0: Object Event →Scarica i pallet dal furgoncino
Livello 1: Object event → Stampa i codici degli oggetti prodotti con scadenze
Livello 2: Aggregation event → Aggrega / disaggrega pallet
Livello 3: Transformation event → Lavora gli ingredienti per creare prodotti

Livello 4: Transaction event → Acquista materie prime / Vende i prodotti
Quest'ultimo livello in realtà da inizio alla fase 2 e si svolge nel contesto del gioco vero e proprio.

Fase 2
In questa fase il giocatore deve vendere i prodotti che ha ottenuto dalla fase precedente creando così le prime basi per gli accordi commerciali con gli altri attori. Qui ha modo di esplorare meglio la mappa e l'interfaccia di gioco e iniziare a conoscere le prime dinamiche, ma non tutte. Una volta venduti tutti i prodotti si ritorna alla visione zoomata 2D dove il giocatore vedrà gli eventi generati dalle transazioni.
A questo punto è pronto per cominciare la fase 3.

Fase 3 (non implementata)
Finalmente il giocatore potrà occuparsi della sua unità produttiva e dell’intera supply chain.
In particolare dovrà gestire i magazzini in entrata e uscita, i prodotti del freddo e le scadenze e schedulare i trasporti della sua unità produttiva e, al contempo, dovrà gestire le relazioni con gli altri attori, prevedendo le tempistiche, garantendo i tempi di consegna, tracciando la provenienza delle merci e gestendo i richiami.

In questa fase il simulatore proporrà al giocatore una serie di sfide legate a quanto appena indicato che il giocatore dovrà affrontare per raggiungere lo scopo.

3.4.4 Storia e Narrativa

Su questo punto non ci sono modifiche rispetto al prototipo precedente, a parte per il fatto che il giocatore si trova all’interno di una filiera differente.

L’aspetto narrativo inoltre è veicolato anche tramite la suddivisione in fasi, che accompagnano il giocatore a una consapevolezza graduale dell’ambiente di gioco e delle sue dinamiche.

3.4.5 Estetica e Grafica

L’aspetto del gioco in generale è rimasto abbastanza invariato dal precedente.
L’interfaccia utente si sviluppa non più solo a sinistra dello schermo ma anche nella parte superiore. Non vi è più una sezione dedicata agli eventi, ma vi si può accedere da uno specifico comando nella sezione delle azioni, che è allargata per via del numero maggiore di interazioni possibili.

Le finestre di interazione escono in una posizione fissa al centro della mappa di gioco e non più nel punto in cui il giocatore ha cliccato.

Alla barra superiore è dedicata la zona riguardante il tempo quindi, oltre alla barra temporale, qui si trovano anche i bottoni di playback.

Le dinamiche grafiche che mostrano lo stato di interazione con gli attori sono rimaste uguali al prototipo precedente.

I livelli della fase 1 si svolgono a schermo intero, con interfaccia semitrasparente posta ai lati dello schermo. L’interfaccia nei livelli non è quasi mai interattiva, tranne alcune piccole eccezioni.
Qui tramite la grafica avviene anche la narrazione relativa al livello di gioco in stile fumetto.

3.5 Secondo Prototipo - Basso Livello

Per l’implementazione del gioco vero è proprio ho optato per Unity e C#.

3.5.1 Unity

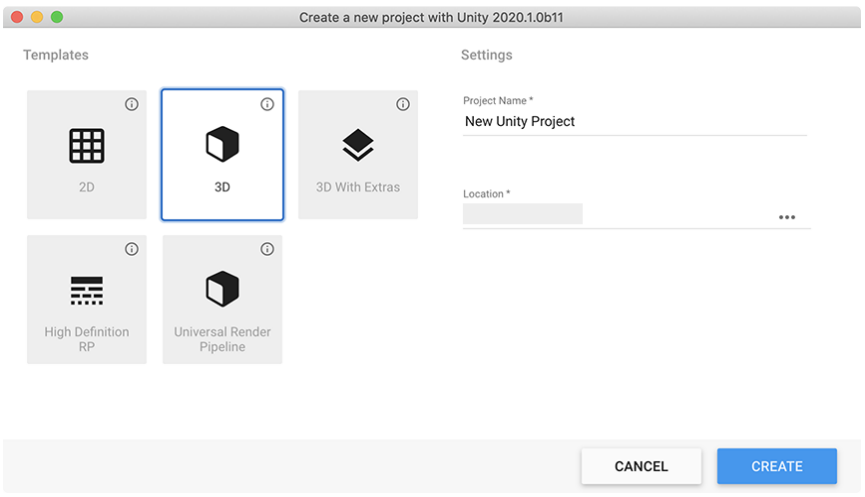
Nella stesura di questo paragrafo ho fatto riferimento a [18] e [19].

3.5.1.1 Generali

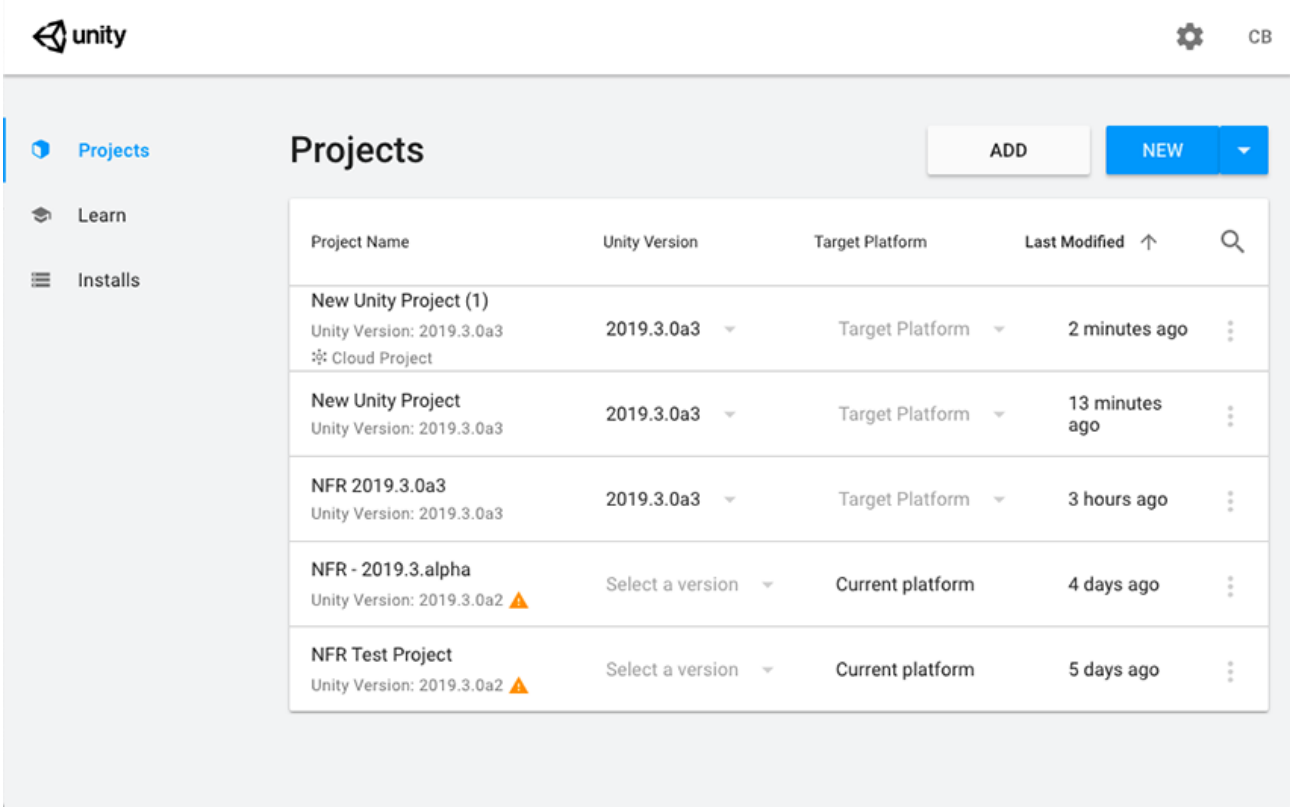
Unity è un motore di gioco e un framework 2D/3D che fornisce un sistema per il design di giochi in 2D, 2.5D e 3D. Permette di interagire col mondo 2D e 3D non solo tramite codice ma anche attraverso componenti visivi, nonché di esportare il prodotto su ogni principale piattaforma mobile e non. Per questo Unity può essere strumento per lo sviluppo di app che non sono prettamente dei giochi.
Supporta tutte le principali applicazioni 3D e numerosi formati audio e legge i file .psd di Photoshop.
Unity permette di importare ed assemblare “assets”, scrivere codice per interagire con gli oggetti e creare o anche importare animazioni (supportate da un potente sistema dedicato).
Assicura il supporto cross-piattaforma di fatto permettendo di sviluppare il proprio gioco/app per una qualsiasi delle principali piattaforme con sforzi minimi: mobile (Android, iOS), Desktop (Windows, Mac, Linux), Web (WebGL), Console(Ps4, Ps5, Xbox One, Xbox Serie X/S, Nintendo Switch, Stadia), Realtà Virtuale/Aumentata (Oculus, PS VR) e altre.
Forse lo strumento più potente di Unity e l’Asset Store, nel quale si possono trovare numerosissimi componenti per il proprio gioco come artwork, modelli 3D, file di animazione per modelli 3D, effetti audio, tracce audio, plug-in (inclusi quelli per supporto multi piattaforma), shader, texture ecc..
Unity fornisci una grande quantità di progetti già pronti e utilizzabili per i tutorial, che accompagnano all’apprendimento delle dinamiche di funzionamento, dell’uso dell’editor e dello scripting.

3.5.1.2 L’Hub

La gestione dei progetti di Unity avviene tramite lo **Unity Hub**, un’applicazione standalone necessaria per il funzionamento di Unity. Da qui è possibile creare un nuovo progetto o avviarne uno precedentemente salvato.
Per ogni progetto è possibile scegliere la versione di Unity che si vuole utilizzare, e ogni progetto è creato a partire da un template:



Questa è la schermata da cui si avvia un progetto esistente:



3.5.1.3 Architettura e compilazione

Unity è basato su C++. Lo sviluppatore Unity scrive però il codice con un linguaggio di script. Le possibilità sono C#, JavaScript (UnityScript) e Boo.
Il codice scritto dall'utilizzatore gira su Mono o .NET, compilato Just-in-Time (eccetto per iOS).
Unity permette di testare il proprio gioco direttamente nell'IDE senza bisogno di eseguire alcuna build.
Per il debug e la scrittura del codice Unity si appoggia a MonoDevelop ma può essere configurato per funzionare con Visual Studio (o VS Code), in questo caso serve utilizzare UnityVS (plug-in per VS) per fare debug, perchè il debugging avviene nell'ambiente virtuale di Unity (non su Unity stesso).

3.5.1.4 CrossPlatform

La maggior parte dell'API e della struttura del progetto resta identica per tutte le piattaforme supportate. Esistono però fondamentali differenze di hardware e metodi di messa in pratica, perciò parti di progetto potrebbero risultare non portabili se non vengono modificate appositamente..

3.5.1.5 Le Scene

Tutto ciò che c'è in un gioco di Unity esiste all'interno di una scena. Quando il gioco è impacchettato per una certa piattaforma, questo consiste in una collezione di una o più scene (più eventuale codice platform-dependent aggiunto dallo sviluppatore).
Il numero di scene in un progetto è arbitrario. Una scena può essere pensata come livello di gioco oppure può contenere più livelli (questo è possibile spostando la camera e gli oggetti in diversi punti della scena).
Un file "scene" è un file che contiene tutti i metadati che riguardano le risorse usate nel progetto per la scena corrente e le sue proprietà.

Nella creazione di una nuova scena è possibile fare riferimento a dei template.
In una scena nulla è visibile senza una Camera e nulla è udibile senza un component Audio Listener attaccato a un qualsiasi GameObject. Di base quando Unity crea una scena questa è già dotata di una Camera cui è attaccato un Audio Listener.

Il passaggio da una scena all'altra è gestito via script usando la classe UnityEngine.SceneManager mentre il passaggio di parametri tra le stesse può avvenire in più modalità.
Un esempio dalla classe gamePlay in cui si usano le PlayerPrefs:

```
public void startPhase1(){
      PlayerPrefs.SetInt("score", 0);
      PlayerPrefs.SetFloat("time", 0f);
      SceneManager.LoadScene("Level0", LoadSceneMode.Single);
  }
```

3.5.1.6 Struttura del progetto e assets

Un progetto in Unity non è altro che una struttura di cartelle. Le cartelle principali sono Assets, Library, ProjectSettings e Temp, ma l'unica visibile nell'IDE è Assets.
Questa contiene tutti gli asset del progetto: sprite, codice, tracce audio e qualsiasi altro file venga importato. Nell'IDE questa cartella è sempre la cartella top-level.

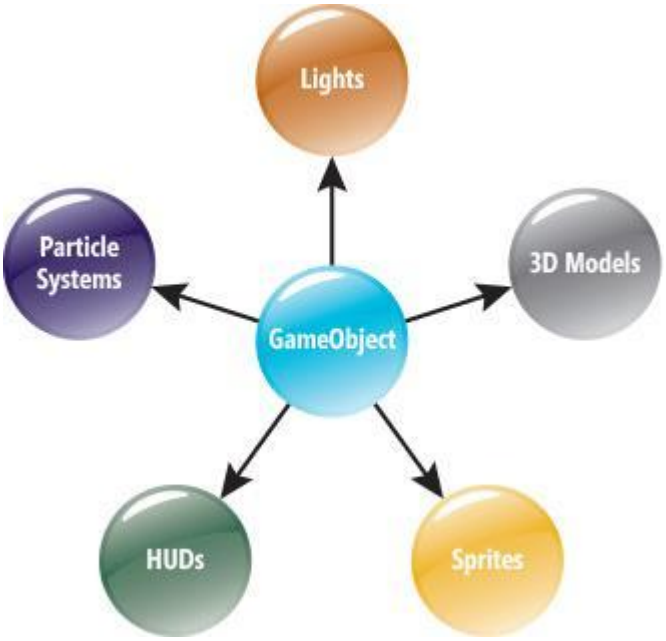
La cartella Library è la cache locale per gli asset importati. Contiene tutti i metadati degli asset.

la cartella ProjectSettings contiene le impostazioni del progetto, configurabili tramite l'Editor.
Temp contiene file temporanei creati da Mono e Unity durante il processo di Build.

3.5.1.7 GameObject

Di fatto, tutto ciò che è presente nella scena è un GameObject. Questa è la classe base di tutti gli oggetti all'interno della scena (simile alla classe Object in Java).

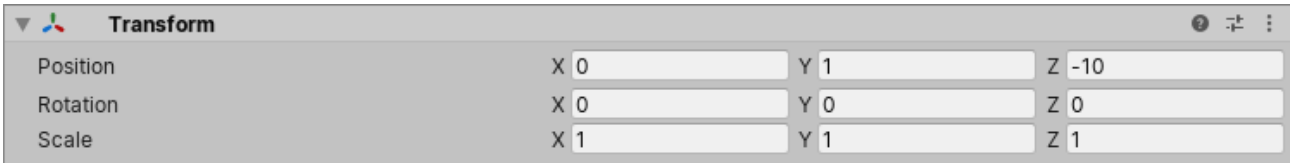
Oggetti che derivano da GameObject:



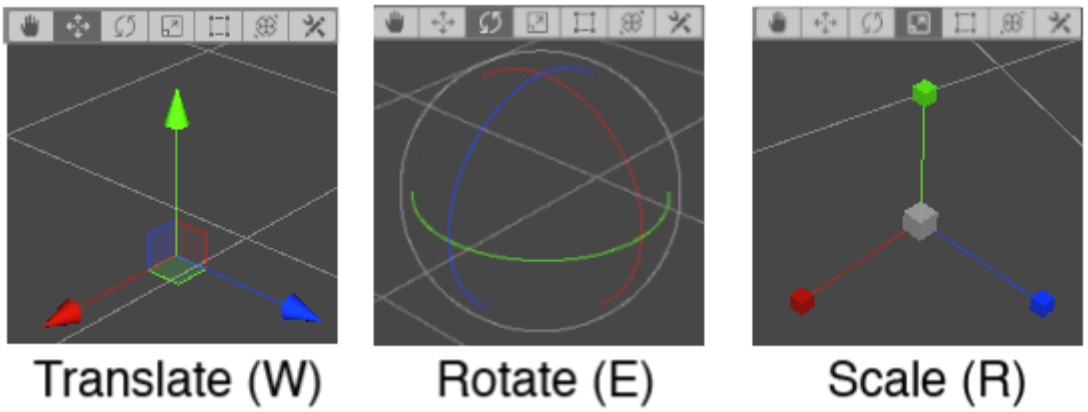
Un GameObject di per sé è un oggetto piuttosto semplice. Non è dotato di alcuna proprietà visiva, eccetto quella che l'Editor mostra quando il GameObject è evidenziato (detta **gizmo**).
Possiede un nome, un tag, un Layer e il Transform (essenziale).

Transform

Non è possibile creare un GameObject senza un Transform, né rimuovere il Transform da un GameObject già creato.
Il Transform è semplicemente la posizione, rotazione e scala del GameObject.
Unity usa un sistema di coordinate sinistrorso. La posizione di un oggetto in Unity (accessibile semplicemente via codice tramite Transform.position) è un vettore con tre valori: X, Y e Z (classe Vector3). Anche nei giochi 2D la posizione è espressa su 3 dimensioni.



Il transform può essere modificato tramite l'inspector View (vedi sopra) o tramite la Scene View. Questo può avvenire usando una serie di tool di manipolazione:



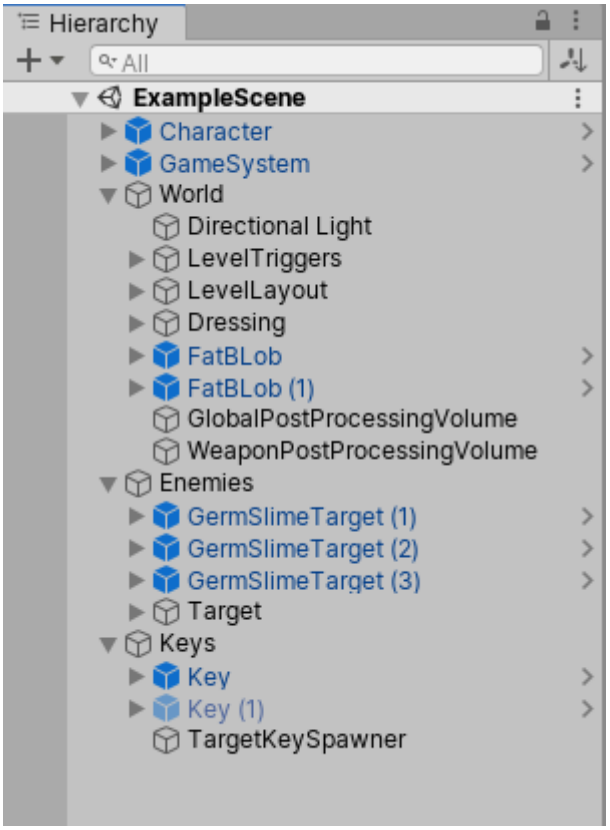
In questa immagine è visibile il **gizmo** associato al GameObject di cui stiamo modificando il Transform.

Il Parenting (ereditarietà) è uno dei concetti più importanti da comprendere quando si vuole usare Unity. Quando un GameObject è Parent di un altro GameObject, il GameObject Child si muoverà, ruoterà e verrà scalato esattamente allo stesso modo del Parent.

Un child può avere oggetti figli a sua volta, e così via. Ogni oggetto può avere più children ma un solo parent. La gerarchia dei Transform è data da questa struttura di Parenting. L'oggetto in cima alla gerarchia è detto root.

Nella vista gerarchica (vedremo poi) è possibile creare una relazione di parenting semplicemente trascinando un oggetto dentro un altro.

Esempio. I GameObject con la freccia a sinistra sono parent.



È importante notare che i valori del Transform di un GameObject child sono relativi al Transform del parent.
La posizione relativa al parent è detta “local”, differente da quella “global”. Se anche per la struttura della scena è sufficiente lavorare con le coordinate locali, in fase di gameplay è meglio far riferimento a quelle globali. L’API di scripting fornisce metodi diversi per ottenere coordinate diverse, e anche metodi di conversione da local a global.

Lo scaling può essere uniforme o non uniforme (è uniforme se i valori di x y e z dello scale sono tutti uguali.
Lo scaling non uniforme può essere utile in alcuni contesti specifici ma porta con se delle problematiche di cui bisogna tenere conto.

- Alcuni Component non supportano completamente tale forma di scaling. Ad esempio i Component dotati di elementi sferici o circolari non riescono a scalare in modo non uniforme, pertanto non diventano ellittici come ci si aspetterebbe ma restano circolari.(Esempio fonte di luce o fonte audio).
- Quando un oggetto child ha un parent scalato in modo non uniforme ed è ruotato rispetto ad esso potrebbe apparire distorto o tranciato ed eventualmente non funzionare propriamente.
- Per ragioni di performance, un child di un parent scalato in modo non uniforme non avrà lo scale aggiornato automaticamente quando ruota. Perciò l’aspetto del child potrebbe cambiare improvvisamente quando dovesse essere aggiornato lo scale (esempio detach dal aprent)

Lo scale del Transform è particolarmente importante per determinare la differenza tra la dimensione di una mesh quando viene modellata e la dimensione della stessa dentro la scena di Unity.

Ci sono tre fattori che possono influenzare lo scale dell'object:
- La dimensione della mesh quando è modellata nella applicazione scelta
- Il Mesh Scale Factor setting nelle impostazioni di import.
- I valori dello scale nel Transform

Idealmente, lo scale del Transform dovrebbe essere l'ultimo che si vuole modificare, o addirittura da non modificare affatto.

I GameObject che vengono creati possono essere salvati come Prefab.

Prefab
Il sistema dei prefab di Unity permette di creare, configurare e salvare un GameObject completo di tutti i suoi component, valori proprietari e child come Asset riutilizzabile.
Il Prefab agisce come template dal quale è possibile creare nuove istanze del Prefab all'interno della scena, sia via Editor sia via Script.

Quando si vuole ri utilizzare un GameObject configurato in un modo particolare in più posti della scena, o tra più scene, è buona pratica usare un Prefab.
Il vantaggio rispetto al semplice copiare e incollare il GameObject sta nel fatto che il sistema dei Prefab permette di avere tutte le istanze sincronizzate. Ogni modifica al Prefab si riflette automaticamente su tutte le sue istanze.

I prefab possono essere annidati l'uno nell'altro creando strutture complesse ma facilmente modificabili.

Comunque è possibile sovrascrivere le diverse istanze del Prefab modificandone qualsiasi attributo. In questo modo non tutte le istanze dello stesso Prefab devono necessariamente essere identiche. È anche possibile creare Varianti dello stesso Prefab per raccogliere le sovrascrizioni che si vogliono ripetere più volte.

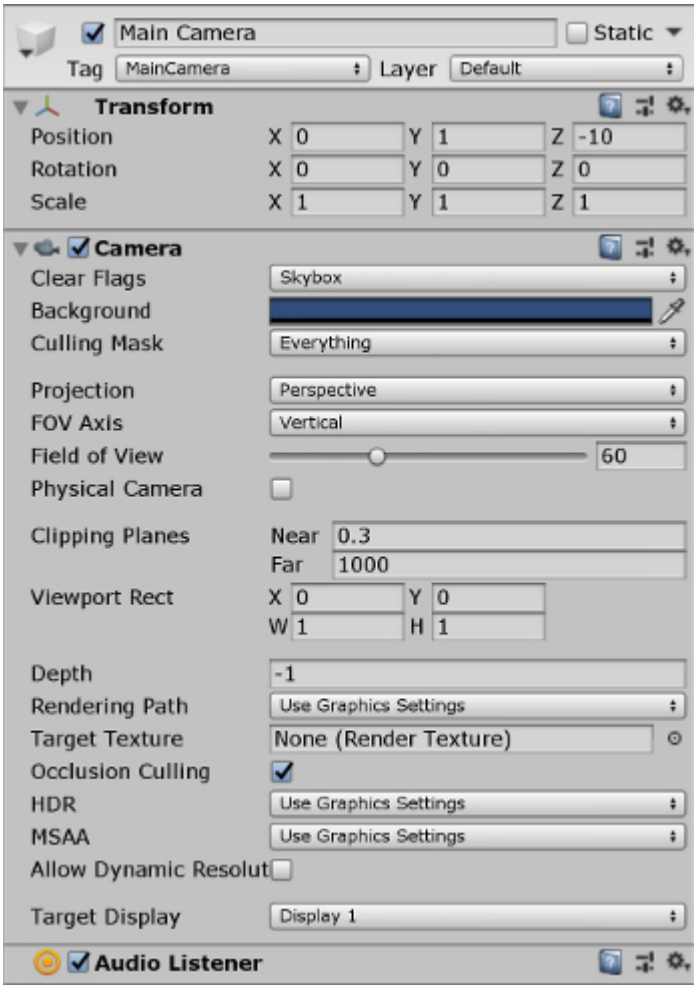
I prefab Sono utili anche per istanziare a runtime oggetti che non sono inizialmente presenti nella scena.

Quest avviene tramite la funzione Instantiate(GameObject, Transform). La posizione di "spawn" è quella definita dalla transform passata. La funzione esiste anche in diverse signature, il cui GameObject diventa automaticamente parent dell'oggetto istanziato.

3.5.1.8 Components
Un GameObject contiene i Components. Questi definiscono il comportamento del GameObject.

La vista dei Component di un GameObject avviene tramite l'Inspector Vie (che vedremo poi).
L'interazione con questi può avvenire direttamente nell'Editor o via Script (il metodo è GameObject.GetComponent<Type>()).

Vediamo come esempio i component del GameObject Main Camera.



Questo oggetto, configurato per comportarsi come Camera principale all’interno della scena, contiene (di base) un Transform component (come ogni GameObject), un Camera component e un Audio Listener component.

Si possono aggiungere funzionalità a un GameObject aggiungendo ulteriori Components.

I Component sono di fatto i pezzi funzionali dei GameObject.

Un GameObject è contenitore di diversi Component. Se ne possono attaccare con qualsiasi combinazione, ma alcune combinazioni funzionano meglio di altre (ad esempio un RigidBody funziona accompagnato a un Collider).

Un importante aspetto dei component è la loro flessibilità. Un Component ha diversi valori e proprietà che possono essere modificati, si via Editor che via Script (runtime).
Le proprietà possono essere valori o riferimenti. I riferimenti possono essere ad Assets, ad altri GameObject o ad altri Component.

Sia un GameObject che un suo component può essere attivato o disattivato. In base a questo il GameObject o un suo component sarà visibile/funzionante o meno all’inizio del gioco. A runtime può essere poi modificata questa proprietà via script (setActive(true/false)).

3.5.1.9 Layers

I Layer in Unity definiscono quali GameObject possono interagire con diverse features e anche tra loro. Sono più comunemente usati dalle Cameras per renderizzare solo parte della scena, e dalle Lights per illuminare parte della scena. Possono essere usate anche dal **raycasting** per ignorare in maniera selettiva alcuni collider o per creare collisioni.

I layer possono essere creati e assegnati a uno o più GameObject.
La selezione di cosa renderizzare è effettuato tramite la **culling mask** della Camera, la quale seleziona solo gli oggetti presenti in certi layer.

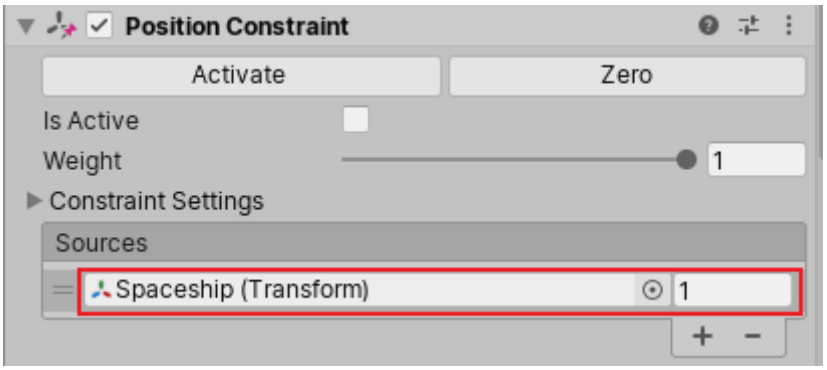
3.5.1.10 Constraints

Il constraint component è un particolare component che serve a legare dei vincoli tra il GameObject e un altro GameObject.
UN GameObject con constraint si muove, ruota o scala come il GameObject cui è collegato.

Unity supporta i seguenti tipi di constraint:

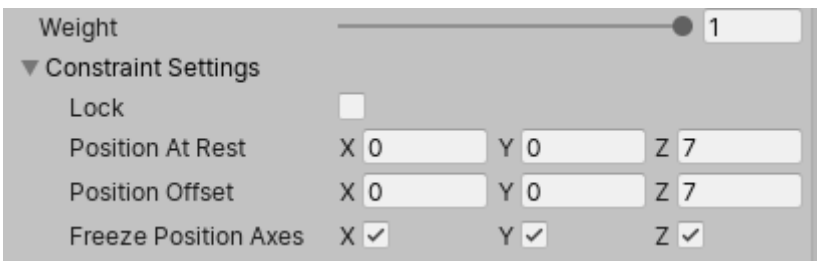
- Aim: Ruota il GO con constraint perché sia di fronte al linked GO
- Look At: Ruota il GO con constraint verso linked GO. (come il primo ma semplificato)
- Parent: Muove e ruota il GO con constraint assieme al linked GO.
- Position: Muove il GO con constraint come il linked GO.
- Rotation: Ruota il GO con constraint come il linked GO.
- Scale: Scala il GO con constraint come il linked GO.

Nella Sources List del constraint component vengono specificati i GameObject a cui collegarsi (Reference).



Il constraint può infatti riferirsi a più di un GameObject. In questo caso il collegamento è fatto con la media delle posizioni, rotazione e scale.
Meglio evitare di generare constraint circolari, le ripercussioni non sono prevedibili.ù

Le proprietà dei constraint:



Il peso serve a variare l'influenza del constraint. Peso 1 significa che la variazione è di entità identica all'originale. Peso 0 rimuove l'effetto del constraint.
In questo caso Unity fa riferimento alla proprietà "A riposo".

La proprietà di offset specifica la posizione relativa del GO, mentre freeze axis identifica quali assi sono modificabili dal constraint.

Un constraint deve essere attivato e "locked" per avere effetto. Se il constraint è locked non è possibile modificare il Transform del GO. PEr poterlo fare è necessario sbloccare il constraint. Inizialmente un nuovo è constraint è inattivo e sbloccato.

Al constraint può essere legata anche un'animazione.

3.5.1.11 Rotation

Le Rotazioni nelle applicazioni 3D sono spesso rappresentate in due modi: Quaternions o Euler angles.
Unity usa internamente i Quaternion ma mostra i valori degli angoli di Eulero equivalenti per semplificare il lavoro dello sviluppatore.

Gli angoli di Eulero sono rappresentati da tre valori angolo per X, Y e Z che sono applicati in modo sequenziale. Per applicare una rotazione di Eulero a un particolare GO ogni valore della rotazione è applicato a turno, come rotazione attorno all'asse corrispondente.
Sono intuitivi e intelligibili e possono rappresentare un cambio di orientamento usando una rotazione di 180°. Soffre del cosiddetto Gimbal Lock: quando si applica la rotazione su 3 turni separati può capitare che per il terzo asse il primo o il secondo risulti nella stessa posizione che aveva all'inizio, perdendo così un grado di libertà.

I Quaternions possono essere usati per rappresentare l'orientamento o la rotazione di un GameObject. Tale rappresentazione consiste internamente in 4 numeri (x, y, z e w in Unity). Comunque questi numeri non rappresentano né angoli né assi è non è quasi mai necessario accedervi direttamente. È sufficiente sapere che il Quaternion rappresenta una rotazione in 3 dimensioni, il funzionamento interno è matematicamente complesso.

In the same way that a Vector can represent either a position or a direction (where the direction is measured from the origin), a Quaternion can represent either an orientation or a rotation - where the rotation is measured from the rotational "origin" or "Identity". It is because the rotation is measured in this way - from one orientation to another - that a quaternion can't represent a rotation beyond 180 degrees.
Differentemente dagli angoli di Eulero non soffrono del Gimbal Lock. Non è però possibile rappresentare una rotazione maggiore di 180° in nessuna direzione ed è meno intuitivo.

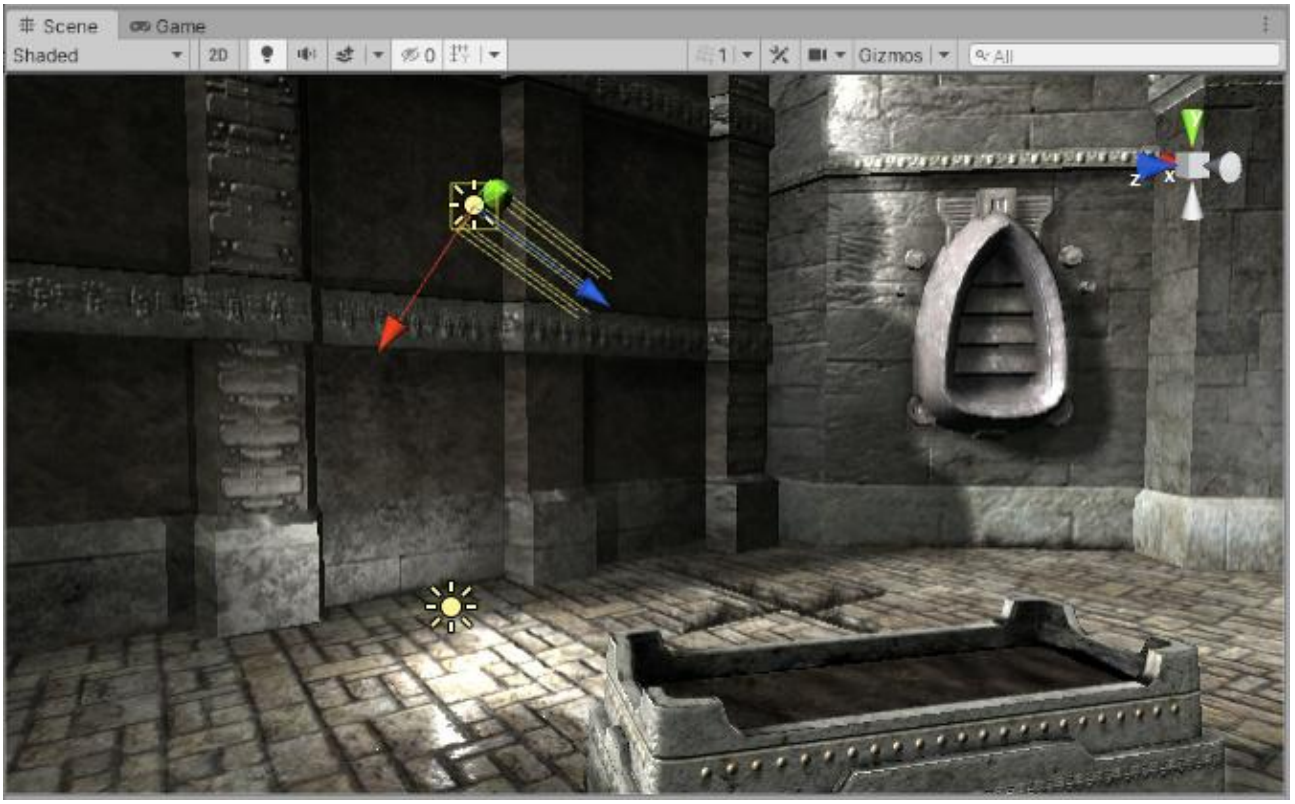
3.5.1.12 Camera

Allo stesso modo in cui le camere sono usate nei film per rappresentare una storia agli spettatori, le Camera in Unity sono usate per rappresentare il mondo di gioco al giocatore. Ogni scena avrà sempre almeno una Camera.
Le Camere multiple possono dare vita a split screen per il multi-giocatore, o creare effetti personalizzati avanzati.
Le Camere possono essere animate o controllate con la fisica.

3.5.1.13 Lights

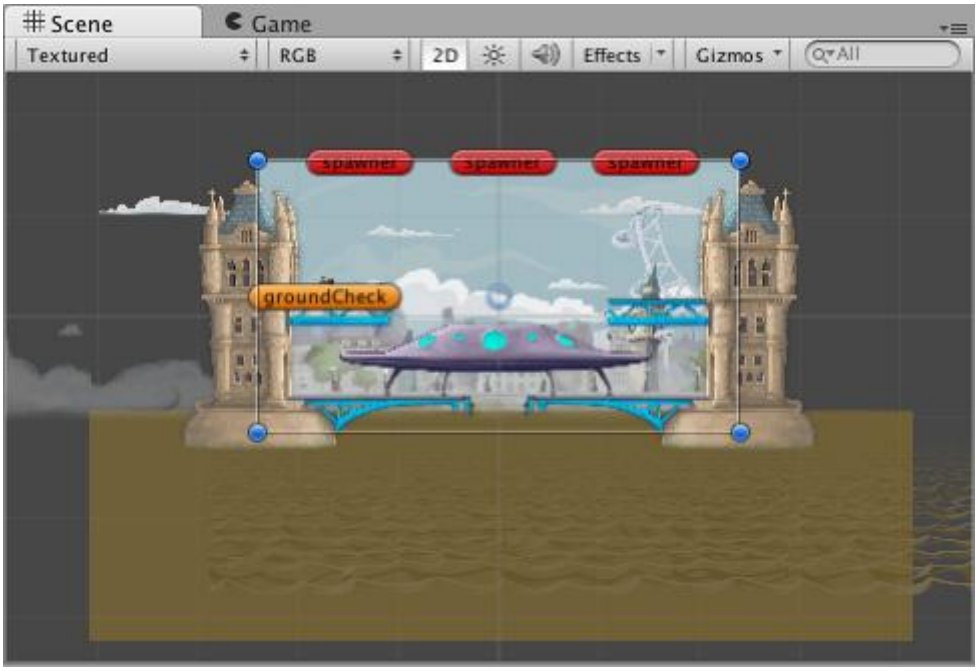
Le luci sono parte essenziale di ogni scena. Mentre le mesh e le texture definiscono forma e aspetto della scena, le luci ne definiscono il colore e il mood dell'ambiente 3D.

Solitamente in una scena ci si ritrova ad usare almeno due fonti di luce:



3.5.1.14 2D

Anche se è principalmente noto per il mondo 3D, Unity può essere usato anche per lo sviluppo di giochi 2D. Le funzioni di base sono le stesse ma accompagnate da diverse semplificazioni.



La prima cosa importante è la modalità di vista 2D, che permette di vedere la scene view e la game view in 2D.
Quando è attiva la modalità 2D la Camera è settata in modalità Orthographic.

Gli oggetti grafici in 2D sono detti Sprites. Essenzialmente sono texture standard. Unity fornisce uno sprite editor built-in.
Le Sprite sono renderizzate da un component detto Sprite Renderer (invece del Mesh Renderer del mondo 3D).

Unity ha poi un motore fisico separato per gestire la fisica 2D. I component corrispondono a quelli standard (Collider, RigidBody…) ma possiedono il suffisso 2D.
La maggior parte di questi componenti sono semplicemente la versione 3D appiattita su due dimensioni, ma ci sono delle eccezioni.

Legato all'ambiente 2D vediamo in particolare le Tilemap.
Il Tilemap component è un sistema che immagazzina e gestisce i Tile Asset per creare livelli 2D. Trasferisce le informazioni richieste dalle Tiles posizionate su di essa tu altri component come il Tilemap Renderer o il Tilemap Collider2D.

Quando si crea una Tilemap questa è automaticamente inserita come child di un GameObject con Grid component. Per creare, modificare e posizionare le Tiles si usa una Tile Palette, che è un tool dedicato.

Per quanto riguarda la fisica 2D vediamo rapidamente due component: il Collider2D e il Rigidbody2D.

Un Rigidbody2D component posiziona un oggetto sotto il controllo del motore fisico.
Un RB2D appare differentemente in base al tipo di Body che viene selezionato.

Il RB2D sovrascrive il Transform e lo aggiorna a una posizione/rotazione definita dallo stesso RB2D. È possibile per lo sviluppatore sovrascrivere il transform ma potrebbe portare a malfunzionamenti.

Ogni Collider 2D component aggiunto allo stesso GO è implicitamente attaccato al RigidBody e si muove con esso.

Il Body Type di un RB2D può essere Dynamic, Kinematic o Static. In base al tipo cambiano il comportamento nel movimento e l'interazione tra i collider.
Dynamic: è il più interattivo tra i tipi, rende l'oggetto affetto dalla gravità e può collidere con ogni altro body. È il tipo di default.
Kinematic: anche questo tipo si muove in simulazione ma solo solo su espliciti controlli dell'utente. Non è affetto dalla gravità.
Static: Non si muove in simulazione, e si comporta come un oggetto inamovibile. Non può collidere con un altro static rigidbody.

Il collider2D component definisce la forma dell'oggetto 2D per quanto riguarda le collisioni fisiche. Un collider, che è invisibile, non necessita di avere la stessa forma dell'oggetto di riferimento.
I tipi di Collider 2D sono:

- Circle Collider 2D per aree circolari

- Box Collider 2D per quadrati e rettangoli
- Polygon Collider 2D per qualsiasi forma
- Edge Collider 2D per aree convesse
- Capsule Collider 2D per aree circolare o losanghe..
- Composite Collider 2D per unire box collider e polygon collider.

3.5.1.15 UI

Unity UI è un toolkit per sviluppare interfacce utente. È basato sui GameObject e Components. Vediamo alcuni aspetti importanti.

Canvas

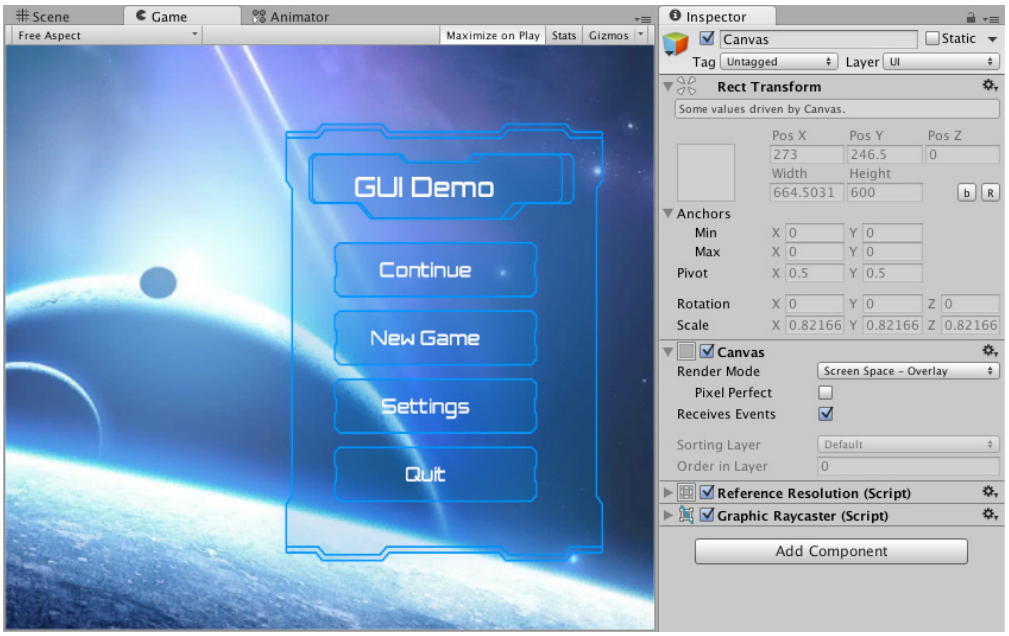
La Canvas è l'area in cui tutti gli elementi di UI risiedono. Nel creare un elemento di UI viene creata anche la Canvas, che fa da Parent all'elemento in questione.
La Canvas è mostrata come un semplice rettangolo nella Scene View. Posizionare gli elementi al suo interno è molto facile e intuitivo.
Perchè una Canvas sia interattiva è necessaria la presenza di un GameObject detto EventSystem.

Gli elementi di UI sono disegnati nello stesso ordine in cui appaiono nella gerarchia (prima il Parent, poi i Child a partire dal primo). Da questo è possibile capire il comportamento in caso di sovrapposizioni.

La Canvas ha un'impostazione detta Render Mode. Cion questa è possibile selezionare uno tra **Screen Space e World Space**.

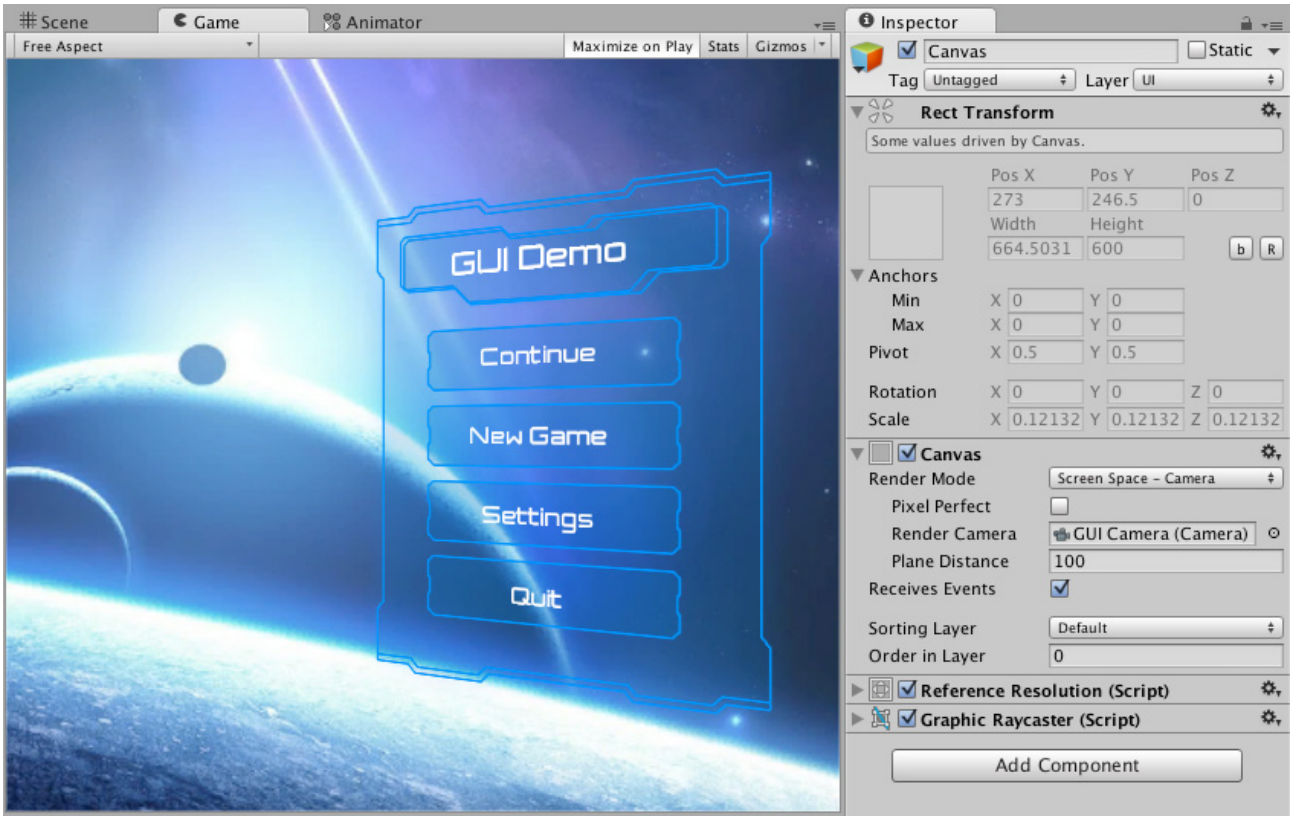
- Screen Space - Overlay: Questa modalità di render posiziona gli elementi di UI sullo schermo, al di sopra della scena. Questo è particolarmente utile per costruire le parti di HUD o i menu. In questa modalità la Canvas si ridimensiona in funzione della risoluzione dello schermo. Far funzionare il tutto in modo fluido non è automatico.

esempio:



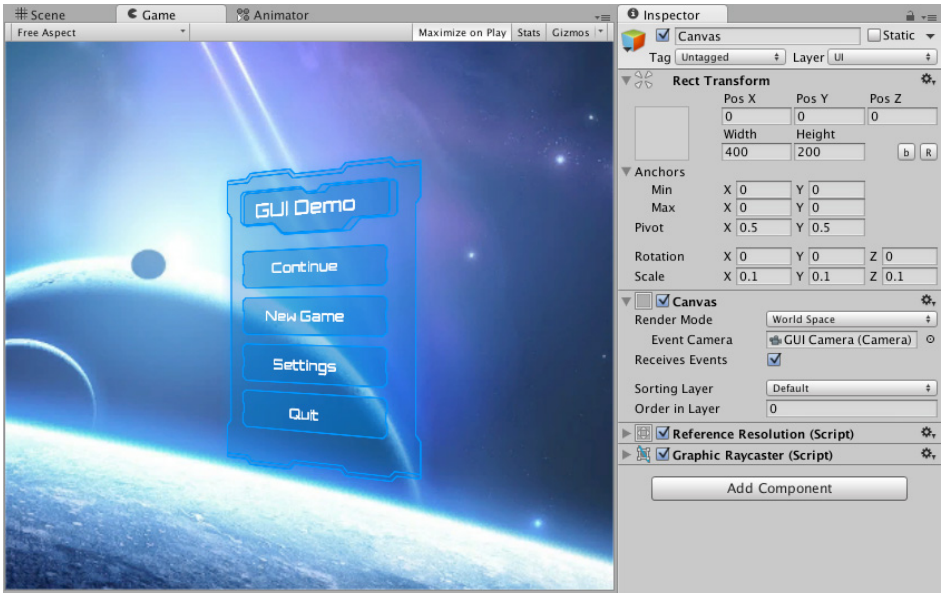
- Screen Space - Camera: Simile al precedente, ma in questo caso la camera di riferimento non è necessariamente quella principale. Anche questo si comporta diversamente in funzione della risoluzione dello schermo e anche delle impostazioni della camera di riferimento.

esempio:



- World Space: In questa modalità la Canvas si comporterà come un qualsiasi altro oggetto della scena. La dimensione è settata manualmente, e gli elementi saranno renderizzati davanti o dietro altri oggetti in base al posizionamento. Questa modalità è utile per quelle parti di UI che lo sviluppatore vuole siano parte del mondo di gioco, come una barra di salute sopra un personaggio.

esempio:



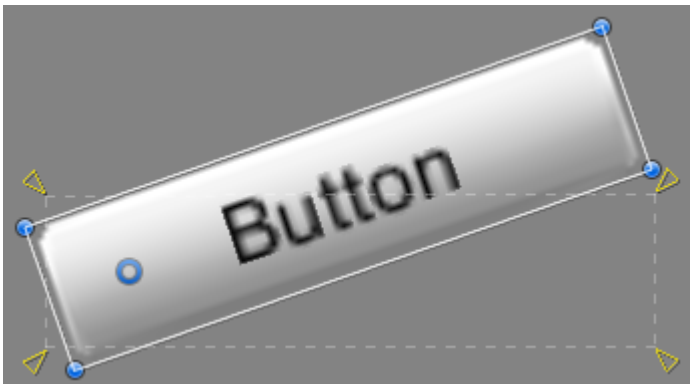
Basic Layout

Vediamo come possono essere posizionati gli elementi rispetto alla Canvas.

Ogni elemento è rappresentato come rettangolo. Questo può essere manipolato in Scene View usando il Rect Tool.

Con questo tool è molto semplice muovere, ridimensionare e ruotare gli elementi di UI.
Il Rect Transform è il Transform component nel mondo dei GameObject di UI. Possiede in più rispetto a un normale Transform anche le proprietà di larghezza e altezza.

Il Pivot è un punto rispetto al quale avvengono i ridimensionamenti, le rotazioni e le scalature di un elemento UI. Ogni elemento ha il suo pivot, che di default si trova al centro del rettangolo.
esempio di rotazione rispetto al pivot:



Le Anchors invece servono ad ancorare un elemento di UI al suo elemento Parent (se anche questo è un Rect Transform). Per esempio il child può ancorarsi al centro del Parent o a un suo angolo. Ci sono 9 modalità di ancoraggio preimpostate, ma è possibile anche l'ancoraggio manuale.

Visual Components

- **Text**: Il component Text (o Label) contiene una text area per l'inserimento di testo. È possibile selezionare dimensione, font e formato. È possibile anche definire l'allineamento del testo e il colore, oltre a impostazioni che regolano l'adattamento del testo al rettangolo.
- **Image**: Un Image component richiede un riferimento a una sprite (jpg, png o altro) la quale verrà renderizzata all'interno del Rect Transform. È possibile settare il colore dell'immagine e anche il material. Il tipo può essere *simple*, che scala l'intera immagine equamente, *sliced*, che nel ridimensionare l'immagine allunga solo il centro e preserva gli angoli, *tiled*, simile a sliced ma che invece di allungare il centro lo ripete e *filled*, che si comporta come simple ma mostra l'immagine riempita di una certa percentuale in una certa direzione a partire da un certo punto.
- **Mask**: Non è un elemento visibile ma serve a mascherare gli elementi child e restringerli alla dimensione della mask.

Interaction Components
I component interattivi non sono di per sé visibili e sono quindi accompagnati a ad altri componenti.

La maggior parte di questi component sono selezionabili, hanno cioè funzionalità built-in per visualizzare la transizione da uno stato a un altro.

- **Button**: Contiene uno Unity Event di tipo OnClick cui deve essere associata una funzione.
- **Toggle**: Questo contiene una checkbox di tipo IsOn che indica se il toggle è acceso o no. Possiede uno Unity Event di tipo OnValueChanged.
- **Toggle Group**: Da usare per raggruppare più toggle mutualmente esclusivi.
- **Slider**: Contiene un valore decimale che può essere fatto scivolare dal minimo al massimo (entrambi personalizzabili). Può essere orizzontale o verticale. Contiene un OnValueChanged.
- **Scrollbar**: Questa contiene un numero decimale tra 0 e 1. Spesso si accompagnano agli Scroll Rect e a una Mask in modo da creare una Scroll View.
- **Dropdown**: Lista di opzioni da cui è possibile scegliere. Possiede un OnValueChanged.
- **Input Field**: Per rendere il testo di un Text editabile dal giocatore. È possibile settarlo in modo che abbia dei limiti sul tipo di testo inserito.
- **Scroll Rect (Scroll View)**: Come si diceva è composto da diversi component che collaborano. Nel suo insieme serve a rappresentare molte informazioni in uno spazio ristretto navigabile tramite scrolling.

Event System
Come accennato primo, Questo component è necessario affinché Unity possa caturare gli input del giocatore (mouse, tastiera, touch altro).

I ruoli dell'Event System sono:

- Gestire quale GameObject si può considerare selezionato
- Gestire quale ,modulo input è in uso
- Gestire il Raycasting
- Aggiornare i moduli di Input

Moduli di Input
Un modulo di input è dove risiede la logica principale con cui l'Event System deve comportarsi.
Sono usati per:

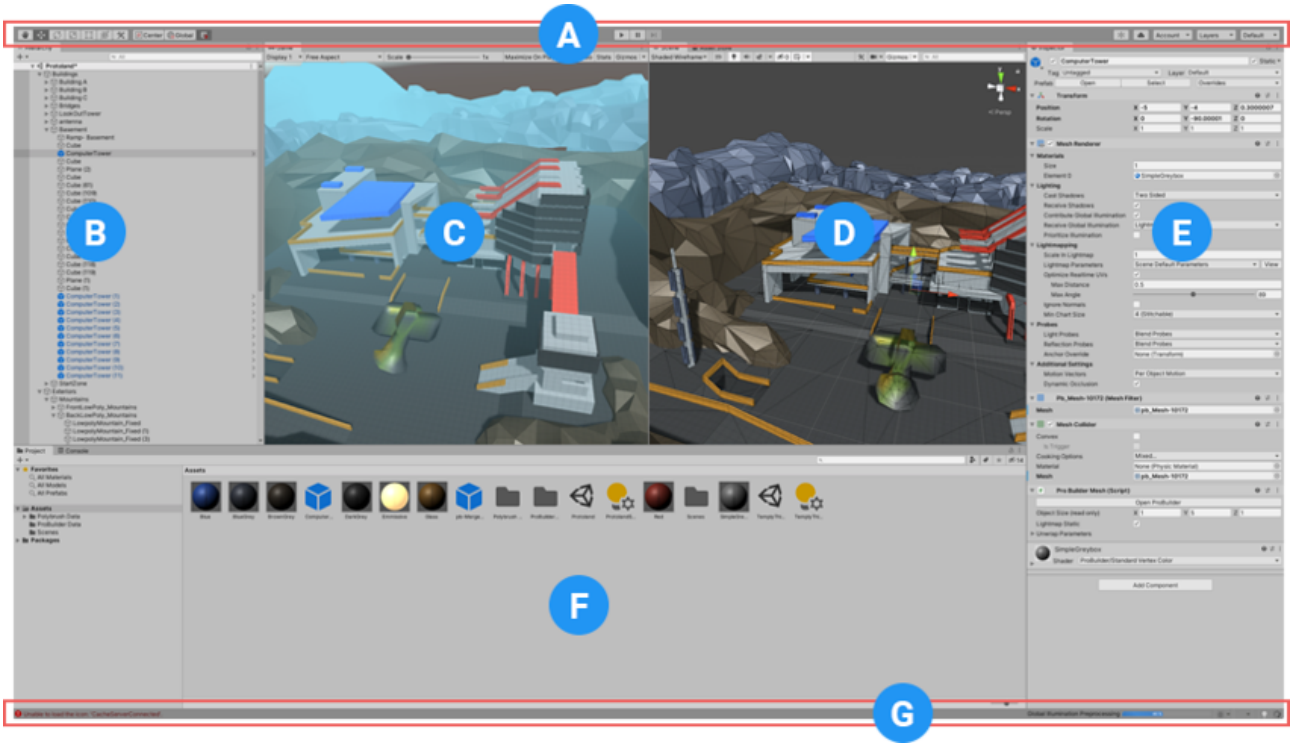
- Gestire gli input
- Gestire lo stato degli eventi
- Mandare gli eventi alla scena

I Moduli Input possono essere attivi uno per volta.

Raycasters
I Raycaster sono usati per comprendere a cosa punta il cursore.
Esistono di default 3 Raycaster:

- Graphic Raycaster - Per elementi UI
- Physics 2D Raycaster - Per elementi di fisica 2D
- Physics Raycaster - Per elementi di fisica 3D

3.5.1.16 L'Editor



Sono qui rappresentate le principali finestre dell’editor nella loro posizione di default.

(A) La Toolbar fornisce accesso agli strumenti di lavoro essenziali. Sulla sinistra contiene i tool per la manipolazione di base della Scene view e dei GameObject al suo interno. Al centro sono presenti i controlli del playback. Sulla destra ci sono i bottoni per l’accesso a Unity Collaborate, Unity Cloud Services e all’account Unity. A seguire troviamo il menù di visibilità dei layer e infine il menù per il Layout dell’Editor.
(B) La finestra “Hierarchy” è una rappresentazione testuale gerarchica di ogni GameObject nella Scene. Ogni item della Scene ha una sua entry nella gerarchia.
La gerarchia rivela la struttura di come i GameObject sono legati tra loro.
(C) La Game View simula l’aspetto di quello che sarà il gioco finale già renderizzato. La simulazione inizia quando viene cliccato il tasto play.
(D) La Scene View permette di navigare visivamente la scena e di editarla. La scena può essere mostrata in prospettiva 2D o in 3D, a seconda del progetto su cui si sta lavorando.
(E) La finestra “Inspector” permette di vedere ed editare tutte le proprietà (e i componenti) del GameObject selezionato. Il layout di questa finestra cambia leggermente a seconda del tipo di GameObject selezionato.
(F) La finestra “Project” mostra la libreria di assets disponibili al progetto. Qui appaiono gli assets quando vengono importati.
(G) La barra di status mostra le notifiche riguardanti i vari processi di Unity e fornisce accesso rapido ai relativi tool e impostazioni.

3.5.1.17 Scripting

Ogni classe in Unity deriva dalla classe base MonoBehaviour, la quale permette di assegnare la nuova classe a un GameObject (oltre a diverse altre funzionalità).

Qui di seguito vediamo alcuni dei metodi principali forniti da MonoBehaviour, che si riferiscono alla sequenza degli eventi: (quando parlerò "dell'oggetto" mi riferirò all'oggetto cui è stata attaccata la classe)

`Awake`: Questo metodo è chiamato una volta per oggetto quando questo è inizializzato per le prima volta. Qui va inizializzato l'oggetto stesso, senza riferirsi ad altri oggetti che potrebbero essere non inizializzati.

`Start`: Questo metodo è chiamato durante il primo frame della vita dell'oggetto ma prima di qualsiasi chiamata ad Update. Differentemente da Awake, si è certi che ogni altro oggetto è già stato inizializzato dal suo Awake.

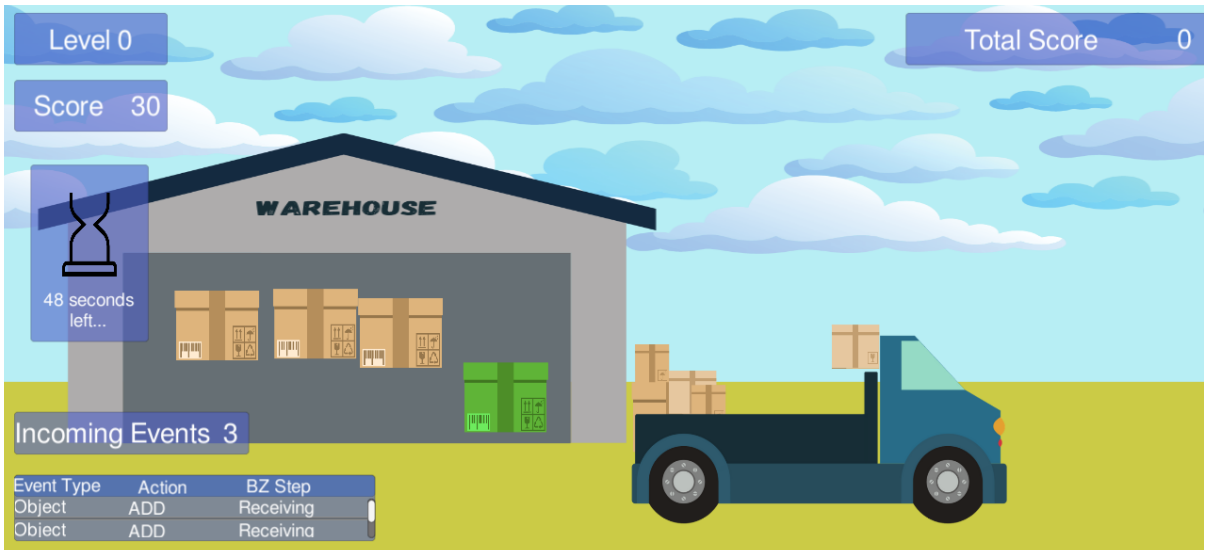
`Update`: Questo metodo è chiamato una volta per frame, il che significa un numero variabile di volte al secondo in quanto è completamente dipendente dal calcolatore.

`FixedUpdate`: Questo metodo è chiamato un numero prefissato di volte al secondo, indipendentemente dal frame rate. Siccome Update è chiamato un numero variabile di volte al secondo e non è in sync col motore fisico, è tipicamente meglio usare FixedUpdate quando si vuole fornire delle funzioni relative alla fisica a un oggetto. (esempio, gestione del tempo). FixedUpdate di default è chiamato ogni .02 secondi, il che significa che Unity esegue anche i calcoli fisici ogni .02 seconds (questo intervallo può essere modificato dallo sviluppatore).

3.5.2 Il Prototipo in Unity: Le scene e il codice

Il gioco è organizzato in 5 scene.
Il flusso di gioco è il seguente:
map_game → Level0 → Level1 → Level2 → Level3 → Level4 → map_game → Level4 → map_game
Il giocatore non può saltare tra le diverse scene ma può muoversi solo all'interno di quella in cui si trova, il gioco passa tra le scene quando è il momento opportuno tramite il UnityEngine.SceneManager.
Il passaggio di parametri tra le diverse scene avviene con l'utilizzo di PlayerPrefs, che permette di salvare su file dati importanti.

3.5.2.1 Level0



Arriva un furgoncino e il giocatore deve trascinare i pallet che trasporta all'interno del magazzino. Ogni scatola inserita nel magazzino fa guadagnare dei punti. Questo è un livello a tempo, il giocatore deve completarlo entro lo scadere.

All'avvio della scena sono presenti alcuni gameObject che introducono al livello e alle sue regole per completarlo. Questi oggetti vengono distrutti nel momento in cui il giocatore preme il bottone di avvio. (questa dinamica si ripete in tutti i livelli successivi)
Gli Oggetti principali sono lo sfondo, il furgoncino, il magazzino e le scatole.
Le scatole sono dotate di un collider così come la parte interna del magazzino.

Lo spostamento delle scatole così come il riconoscimento del loro ingresso in magazzino avviene tramite le callback `OnMouseDrag()`, `OnMouseDown()` e `OnMouseUp()`.

Quando una scatola è rilasciata nel magazzino viene generato un evento EPCIS visibile in un pannello dedicato dell'interfaccia.

È presente un GameObject non visibile chiamato level cui è attaccato uno script atto a gestire il livello nel suo insieme, tramite variabili di controllo del tempo, del punteggio e dell' interazione.

Oggetti e relative classi:

- **Box** - Oggetto scatola, con attaccato uno script contenente la classe Box. Questa classe contiene i metodi gestire il trascinamento e il rilascio delle scatole, nonché il relativo punteggio. Ve ne è più di uno nel livello.

- **`HUD`** - Questo oggetto è una Canvas in modalità screen space - overlay. contiene dei pannelli per mostrare il punteggio, un pannello per mostrare il tempo che scorre (che si modifica nel tempo) e un pannello che mostra gli eventi. Questi sono gestiti tramite una ScrollView che mostra degli oggetti di tipo Event. Questi Event sono dei Prefab che sono passati alla ScrollView come parametro. ScrollView possiede anche un metodo (chiamato da Level 0) che istanzia gli event usando il Prefab: Instantiate(GameObject, Transform).
- **`Level`** - Questo oggetto non è visibile a schermo ma è essenziale per il funzionamento del livello. Ha attaccato a se uno script contenente la classe Level 0. Questa gestisce lo scorrere del tempo all'interno di FixedUpdate(), aggiorna l'interfaccia utente e tiene il punteggio. Contiene un metodo chiamato startLevel0() che viene chiamato quando il giocatore clicca il bottone di inizio livello.

`Prefab`:
Tutti gli oggetti creati per questo livello sono stati salvati come Prefab e usati come base per creare gli oggetti simili nei livelli successivi.

3.5.2.2 Level1

Ci spostiamo all'interno dell'unità produttiva. Qui il giocatore ha a che fare con la lettura dei codici dei pallet e con la gestione delle scadenze.

Il giocatore deve cliccare sullo scanner presente a lato dello schermo. In questo modo il cursore cambia texture e diventa lo scanner. Cliccando sulle scatole si generano degli eventi EPCIS di tipo Object Event. Questi eventi mostrano la data di scadenza dei vari prodotti.. Il giocatore dovrà selezionare le scatole contenenti i prodotti la cui scadenza è più vicina nel tempo e spostare le stesse in una zona dedicata.
Il giocatore guadagna punti da tutte le azioni che compie.

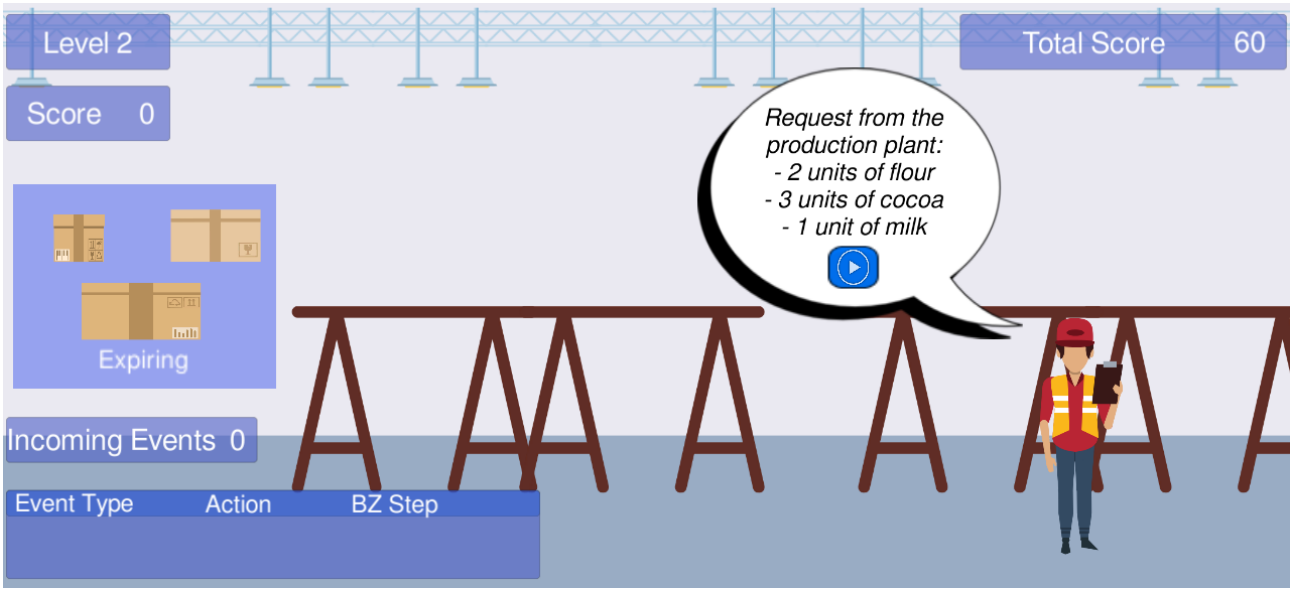
Anche qui sia le scatole che la zona dedicata hanno un collider e sono gestiti come nel livello precedente (così come ogni trascinabile anche nei livelli successivi)
Vale lo stesso per il GameObject level che gestisce il livello nel suo insieme.
Oggetti e relative classi:

- **`BoxLevel1`** - Questo oggetto è costruito a partire dal Prefab del Box del livello precedente. ha infatti gli stessi metodi per trascinamento e rilascio ma gestisce in modo un po' diverso il punteggio e possiede un riferimento a Level1 (diverso da Level0).
- **`HUD`** - Anche questo creato a partire dal prefab di HUD del livello 0. Al posto del pannello del tempo lascia uno spazio vuoto.

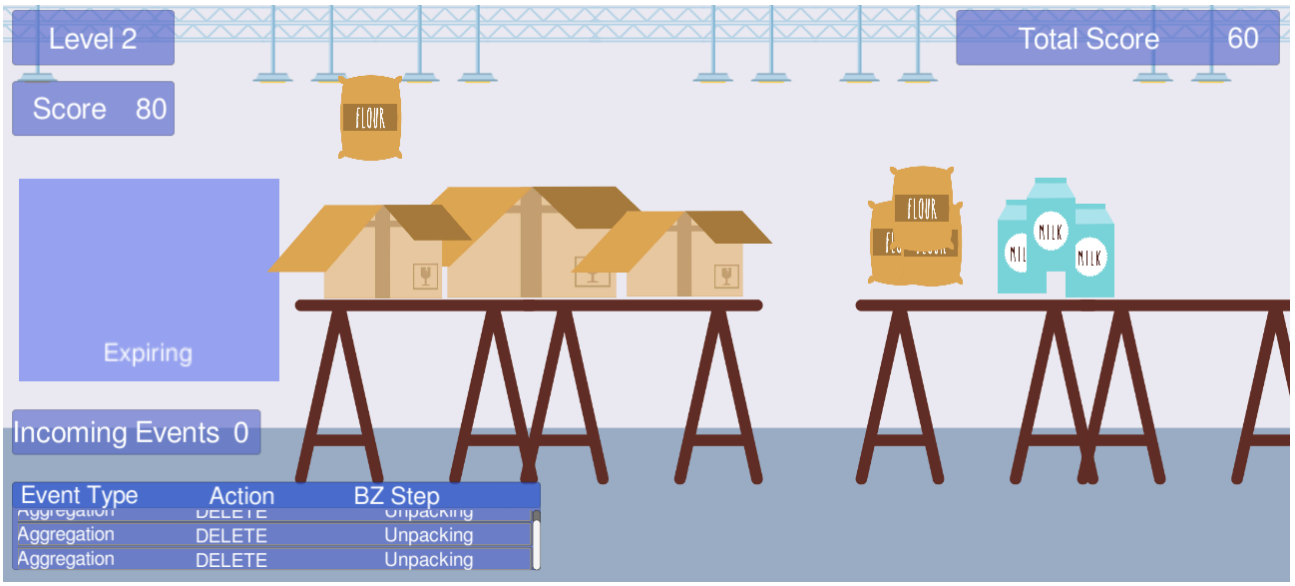
- **`Level`** - A partire dal Prefab del Level0, ha uno script diverso in quanto deve gestire dinamiche leggermente diverse. Ad esempio questo oggetto non deve gestire il tempo.
- **`expiringPlace`** - Questo oggetto serve ad identificare la zona dedicata al rilascio delle scatole. Contiene una Canvas in modalità World Space che lo confonde come un elemento di interfaccia e lo rende visibile. Inoltre ha un Collider che permette di riconoscere quando vi è rilasciato qualcosa all'interno. Questo è il motivo per cui non è un semplice oggetto Canvas, che non può avere un Component di tipo Collider. È passato come parametro al level.

Prefab:
Molti oggetti del livello 0 come spunto, un nuovo Event da istanziare nella ScrollView (anch'esso costruito a partire da quello precedente).

3.5.2.3 Level2



Ora il giocatore deve spacchettare le scatole selezionate nel livello precedente. Deve prenderle nella zona dedicata in cui si trovano e trascinarle sul primo dei due tavoli presenti. Dopodichè cliccandoci sopra le scatole rivelano il loro contenuto. Il giocatore dovrà trascinare i prodotti estratti nel secondo tavolo. Tutte le dinamiche di livello sono gestite allo stesso modo del livello precedente.

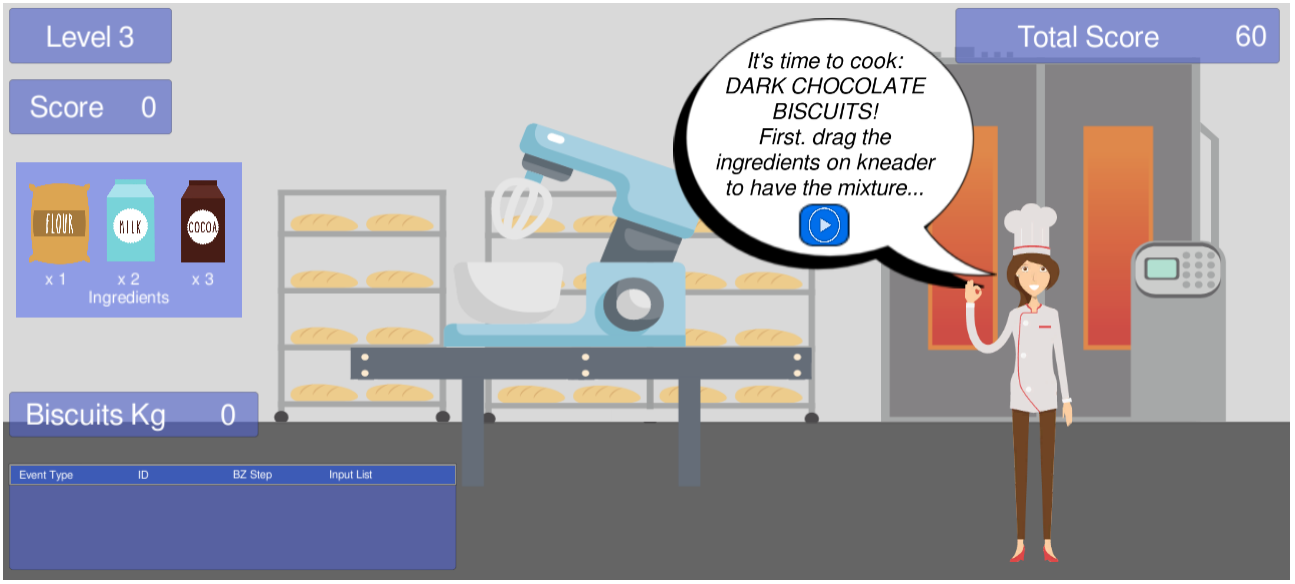


Oggetti e relative classi:
Oltre a quelli derivati e adattati a partire dai livelli precedenti (Level, HUD e box) qui si aggiungono:

- **`OpenedBox`** - del tutto simile a expiringPlace, rappresenta una zona nello spazio dove vanno rilasciate le box (situato sopra il primo tavolo). Ha un collider per riconoscere il rilascio e contiene come child tre oggetti che rappresentano le tre scatole aperte. Questi sono inizialmente inattivi e vengono attivati al rilascio delle box.
- **`opened1, opened2 e opened3`** - le tre scatole aperte attivate la rilascio delle box. Hanno anche loro un collider che consente l'interazione tramite click. Possiedono un riferimento a dei prefab che rappresentano i prodotti. Al click istanziano il prodotto cui fanno riferimento appena sopra di loro.
- **`goodsArea`** - come OpenedBox ma per i prodotti istanziati dagli oggetti opened.

Prefab:
`flour, cocoa e milk` sono i prodotti istanziati dalle box opened. Hanno un collider per permettere il trascinamento. Oltre a quelli derivati dai livelli precedenti.

3.5.2.4 Level3



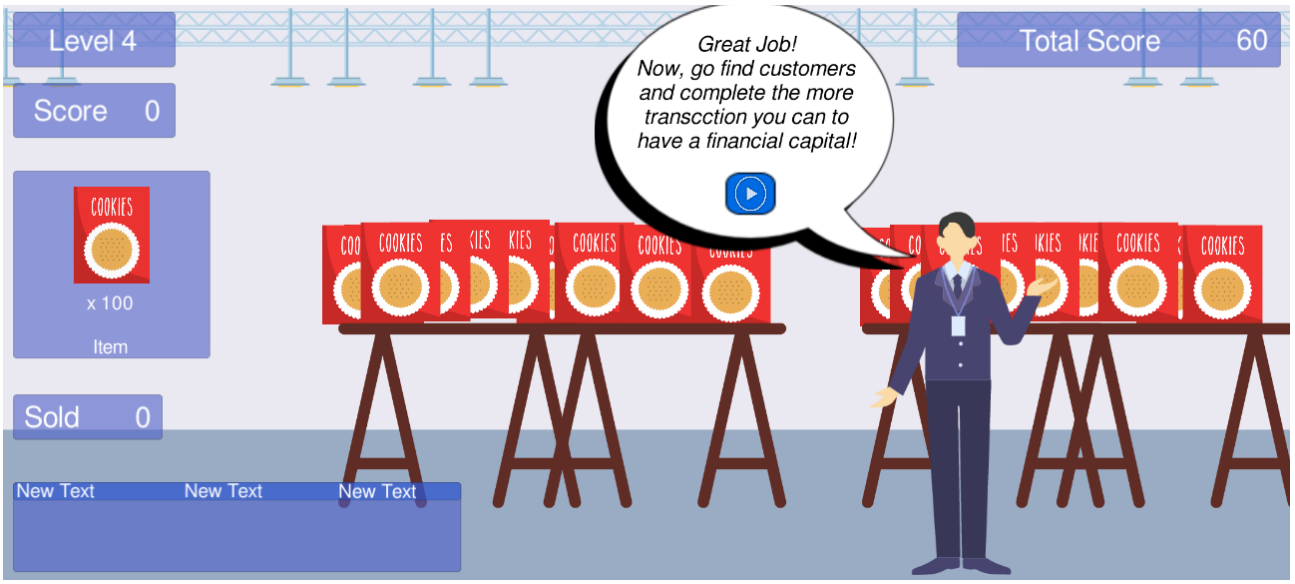
I Prodotti estratti nei livelli precedenti vengono qui combinati per la produzione di biscotti. Anche qui questo si fa tramite trascinamento. Gli eventi si generano dalla creazione dei prodotti. (Transformation Event).
Una volta generati tutti i biscotti generabili il livello si conclude

Oggetti e relative classi:
Oltre a quelli derivati e adattati a partire dai livelli precedenti (Level, HUD e box) qui si aggiungono:

- **`Kneader`** - simile a tutti quelli presenti precedentemente, per il rilascio dei prodotti

Prefab:
Gli stessi dei livelli precedenti, evoluti per il livello di riferimento.

3.5.2.5 Level4



Ora che tutti i prodotti ci sono è il momento di venderli. Questo livello, una volta avviato nel modo di sempre, avvia la scena map_game. È infatti una scena molto scarna in quanto non avviene null’altro che il passaggio a quella successiva.

Oggetti e relative classi:
Oltre a quelli derivati e adattati a partire dai livelli precedenti (Level, HUD e box) qui si aggiungono:

- **`biscuits`** - oggetto solo visivo che contiene numerosi oggetti (anch’essi puramente visivi) rappresentanti i prodotti creati nel livello precedente.

- **`Level`** - menzione speciale va fatta per il Level di Level4 in quanto esso possiede un riferimento alla fase di gioco per capire se la scena è chiamata in seguito al completamento del livello 3 o al completamento del livello 4. Per capirlo usa PlayerPrefs.

Prefab:
Nessuno in più rispetto ai precedenti.

3.5.2.6 map_game

Questa scena è la principale del gioco. Il gioco comincia e (nelle fasi 2 e 3) si svolge in questa scena.
Abbiamo qui una griglia isometrica contenente la TileMap. Questa è la base su cui poggia il gioco. E' navigabile tramite trascinamento.
Sulla mappa sono presenti le unità produttive, caratterizzati da una struttura visible. Le unità produttive sono raggruppate (con un legame di tipo parent-child) in GameObject chiamati Actor. Questi non sono entità visibili ma sono presentate al giocatore nell'interagire con le unità produttive. Vi è un centro trasporti che gestisce la circolazione dei furgoncini.

Inoltre sono presenti un simulatore a eventi discreti, un gestore delle fasi di gioco e un gestore dell'interazione e dell'interfaccia utente.

All'inizio della fase 2 la maggior parte delle funzionalità sono bloccate. Con l'avvio della fase 3 (quando il magazzino in uscita del giocatore è vuoto) si sbloccano tutte le funzionalità e il simulatore assume il ruolo di generatore di eventi discreti.

Nell'implementazione attuale, al termine della fase 2 il gioco si ferma in quanto la fase 3 non è implementata.

Oggetti e relative classi:

- **`world`** - Questo oggetto serve solo a fare da parent a tutti gli oggetti (visibili) che fanno parte della mappa.
    - **`Grid`** - Oggetto griglia che fa da parent alle TileMap. Ha layout Isometrico.
        - **`ground`** - Tilemap contenente il terreno, separata da roads.

Questa contiene anche le funzioni dedicate alla navigazione nella mappa. L'ultima avviene tramite le funzioni del tipo Input.Get all'interno di Update e cambiando la posizione della camera.

- **roads -** Tilemap contenente le strade, separata da ground.

  Questa distinzione serve a gestire più facilmente gli oggetti che si muovono su strada o sul terreno.

- **transports** - Oggetto per la gestione dei trasporti. Contiene un riferimento al Prefab del furgoncino. Si occupa di far partire i mezzi dando loro le destinazioni e fornisce metodi per la schedulazione dei trasporti.
- **productionUnits** - Sono tutte le strutture fisiche presenti nella mappa. Hanno una sprite diversa in funzione del loro ruolo. La classe contiene dei valori riguardanti i magazzini in entrata e in uscita (valore massimo e valore attuale), un riferimento all'Actor proprietario, un riferimento all'interactionUnit e un Collider. Contiene dei vettori di gameObject generici che rappresentano gli oggetti presenti nei magazzini (utile alla fase 3). Contiene dei metodi per la modifica dell'aspetto (grigi o colorati a seconda che siano interagibili per il giocatore, e in concomitanza healthBar visibile o meno).
  - **healthBar** - questa è un oggetto Canvas in modalità world space che serve a rappresentare lo "stato di salute" della productionUnit. È di fatto una semplice barra, costituita da contorno e riempimento, il cui colore e la cui percentuale di riempimento sono gestite da productionUnit all'interno di Update().
- **Actors** - Questi oggetti non sono direttamente visibili nel gioco ma sono percepibili dal fruitore tramite il contesto (es. finestre di interazione che mostrano che una productionUnit appartiene a un certo Actor). Fanno da parent a tutte le productionUnit di cui sono proprietari. All'interno della Classe contengono un vettore delle proprie productionUnit che viene riempito all'interno di Start(). Tramite la funzione makeInteractive(bool mode) l'attore diventa o meno interattivo. Questo modifica l'aspetto di tutte le sue productionUnit nonché ciò che succede quando queste vengono cliccate.
  Sono presenti due vettori di GameObject generici: products e RawMaterials, che indicano quali sono i prodotti in output da quella azienda e quli quelli in input.
  La classe contiene anche il capitale e un vettore di oggetti Order, utile alla fase 3.
- **Player** - Questo oggetto rappresenta il giocatore. La classe Player eredita da Actor e possiede in più un vettore di Actor chiamato partners (entità partner con quella del giocatore, utile in fase 3) e alcuni riferimenti alla parte dell'interfaccia utente riguardante i dati del giocatore.

- **HUD** - Oggetto di tipo Canvas in modalità screen space - overlay. Questo oggetto costituisce l'interfaccia utente e comprende sia le parti atte a informare il giocatore sulla situazione generale del gioco (simulazione, statistiche e valori importanti) sia quelle di interazione, quindi i bottoni che il giocatore può cliccare o gli input che può inserire. Per lo più è sempre visibile ma ha delle parti inattive che vengono renderizzate all'occorrenza. L'HUD è divisa in pannelli.
  - **top** - barra superiore
    - **timeHandle** - pulsanti per gestire la velocità di playback
    - **ID** - Parte dedicata all'identità del personaggio interpretato dal giocatore

        - **TimeBar** - Barra superiore che indica l'avanzare del tempo, gestita dal simulatore. Di fatto un'immagine di cui viene modificato il valore di fill amount.
    - **left** - barra laterale
        - **AboutMe** - Pannello superiore, contiene il logo del gioco, le statistiche del giocatore e un titolo. Interessanti sono le statistiche, gestite tramite un componente GridLayout Group che ne rende l'aspetto ordinato. Ogni riga delle "stats" ha un'icona, delle info ed un valore. Le info possono essere testuali o, nel caso dei magazzini, delle "barre di salute" gestite allo stesso modo delle healthBar delle productionUnit. Queste stats sono aggiornate all'interno dell'Update() del Player.
        - **ActionBar** - Pannello inferiore, contiene tutte le possibili azioni che il giocatore può compiere nella fase 3. Questo contiene un titolo (testuale) e la lista delle opzioni organizzate anche queste in un GridLayout Group. Ogni riga contiene icona e nome. Nell'implementazione della fase 3 tutte queste righe avranno un Component di tipo Button cui sarà associata una funzione di InteractionUnit. Le azioni possibili sono: Search Product, New Production, Schedule a Transport, Add an Order, Orders In, Expiring products ed Event flow. (spiegozzo, forse non qui).
    - **zoom** - pulsanti di zoom (in basso a sinistra)
        - **ZoomIn** - Button collegato alla funzione di zoom in avanti. Lo zoom avviene decrementando il valore dell'orthographicSize della Camera.
        - **ZoomOut** - Button collegato alla funzione di zoom indietro. Lo zoom avviene incrementando il valore dell'orthographicSize della Camera.
    - **interactionPanel** - centro schermo (leggermente tendente a destra per fare in modo che sia centrale rispetto alla schermata di gioco.
- **Simulator** - Questo oggetto, non renderizzato, è il gestore della simulazione in senso ampio. Si occupa dello scorrere del tempo (e relativo aggiornamento dell'HUD), delle simulazioni prettamente visive e di contorno (oggetti che si muovono solo per animare la scena), della generazione di eventi discreti (fase 3) e delle notifiche al player (fase 3). Ha un riferimento a gamePlay, uno a InteractionUnit e un vettore di tutte le productionUnit presenti.
  Per la gestione della velocità di scorrimento del tempo ho fatto uso di una variabile che ho chiamato timeScaler. Questa è utilizzata in tutti i calcoli dipendenti dal tempo. Pertanto la modifica della velocità consiste nella semplice modifica di questa variabile.
  esempio di uso della variabile nella funzione handleTime() :

```
void handleTime(){
       seconds += (int)Time.timeScale * timeScaler;
       float percentage = (float)seconds/(float)secondsToEnd;

HUD.transform.Find("top").Find("TopBar").Find("TimeBar").GetComponent<Image>().fillA
mount = percentage;
        }
```

- **gamePlay** - Questo oggetto, non renderizzato, gestisce il procedere delle diverse fasi di gioco. Il simulatore, così come ogni altra entità del gioco, fa riferimento a questo oggetto per sapere in che fase di gioco ci troviamo. Questo avviene tramite alcune variabili di controllo booleane. Ha un riferimento al simulatore che serve per inizializzare alcune variabili all'inizio delle varie fasi.
- **InteractionUnit** - Questo oggetto, non renderizzato, si occupa di gestire quasi ogni interazione che l'utente ha con gli oggetti del gioco (in particolare quelli dell'HUD). Ha un

riferimento al Player, al gamePlay all'HUD e a world (per poter accedere a qualunque oggetto del gioco). Contiene i metodi productionUnitSelected(), makeOffer(), e closeInteraction() che gestiscono l'interazione con le productionUnit. Per la fase 3 conterrà tutte le funzioni di interazione necessarie per fare le azioni presenti a sinistra nell'HUD..

- **`car`** - oggetto istanziato da transports all'occorrenza. Contiene la logica per il movimento su strada. L'algoritmo per la guida calcola la direzione ottimale a partire dalle posizioni (Vector3) di partenza e arrivo. Dopodichè controlla tra le 4 possibili direzioni (a partire da quella ottimale) quali sono possibili (facendo riferimento alla tilemap roads e alla funzione hasTile(position) e le valuta con un punteggio di ottimalità. Quando ha scelto il passo lo traduce di modo che sia coerente con la natura isometrica della mappa:

  Codice:
  ```
  step = chooseStep(roads.WorldToCell(me), roads.WorldToCell(dest));
            dest.x = me.x - step.y*roads.cellSize.x/2 + step.x*roads.cellSize.x/2;
            dest.y = me.y + step.x*roads.cellSize.y/2 + step.y*roads.cellSize.y/2;
  ```
  Lo spostamento su spazio isometrico necessita di una traduzione dal semplice destra e sinistra e avanti indietro. Questo perchè uno spostamento in x implica anche uno spostamento in y (la y aumenta nella stessa direzione di x) e uno spostamento in y implica anche uno spostamento in x (la x aumenta nella direzione a opposta a quella della y).

  lungo x -> se x aumenta lo fa anche y, se x diminuisce lo fa anche y
  lungo y -> se y aumenta x diminuisce, se y diminuisce x aumenta

  La destinazione lungo l'asse x (y), per essere tradotta sullo spazio isometrico, deve essere calcolata sottraendo (aggiungendo) dalla posizione iniziale in x (y) il passo lungo y (x) moltiplicato per la metà della dimensione della cella lungo l'asse x (y) e aggiungendo il passo lungo x (y) moltiplicato per la metà della dimensione della cella lungo l'asse x (y).

  In più ci sono alcune classi che esistono ma non sono implementate ne assegnate a oggetti perchè fanno parte della fase 3:

- **`Order`** - per rappresentare gli ordini sia in entrata che in uscita
- **`Event`** - per rappresentare gli ordini con una struttura simile a EPCIS

Prefab:
`car e ProductionUnit`

4. Conclusioni

4.1 Gli step successivi

In questo elaborato ho mostrato il processo (parziale) di design e progettazione di un Serious Game per il supporto all'apprendimento di EPCIS.

Dico processo parziale in quanto non sono arrivato a un prodotto finito (commercializzabile) ma mi sono fermato a una versione prototipale.
Non è stato fatto il testing sul primo prototipo ma si è passati direttamente al secondo: questo ha sicuramente inficiato sui risultati ottenuti nel secondo prototipo che non ha avuto il supporto dei feedback di test.

Le fasi immediatamente successive sarebbero quella di completamento del secondo prototipo con l'implementazione della fase 3 e quella di testing.
Dopodichè la quantità di lavoro prima di arrivare a un prodotto commercializzabile non è quantificabile a priori in quanto è fortemente dipendente dagli esiti della fase di testing, che potrebbero portare a modifiche anche profonde.

4.2 Considerazioni finali

Dal punto di vista tecnico, sono soddisfatto di essermi potuto cimentare nell'utilizzo di strumenti che non conoscevo, in particolare di Unity, che ho trovato molto fruibile e di cui ho apprezzato enormemente le potenzialità e la potenza. Nel passaggio dal primo al secondo prototipo ho infatti potuto constatare quanto è importante la tecnologia utilizzata quando si tratta di sviluppo, sia per quanto riguarda appunto la potenza della stessa, sia per quanto riguarda la documentazione a disposizione, che rimane sempre e comunque un importante strumento per lo sviluppatore.
I risultati ottenuti usando Unity sono stati di gran lunga migliori rispetto a quelli ottenuti con Traviso.js, e la mole di lavoro per uno stesso traguardo decisamente inferiore.
Sono contento anche di aver potuto sviluppare del codice usando C#, in quanto non avevo avuto occasione durante il mio percorso universitario.

Questo lavoro di Tesi mi ha permesso di assumere nuove skills sia in campo tecnico (sviluppo con Unity e C# e lo sviluppo di giochi isometrici), sia in ambito di relazione professionale, in quanto per la prima volta ho sperimentato un ambiente di lavoro, con scadenze e richieste. Auspico che tali skills si rivelino utili nel mio prossimo futuro nel mondo del lavoro.

Le mie considerazioni rispetto ai risultati sono certamente manchevoli del supporto dell'evidenza, per via dell'assenza dei test. Tuttavia credo che il Serious Game abbia un enorme potenziale in numerosi ambiti.
In ambito formativo o riabilitativo non può certamente essere una entità autonoma per portare a dei risultati ma deve piuttosto essere strumento di accompagnamento che alleggerisca e supporti l'apprendimento senza la pretesa di poterlo veicolare in autonomia.
Se utilizzato in questa dimensione credo possa affermarsi sempre meglio nel prossimo futuro.

Un'altra considerazione che sento di dover aggiungere è la seguente: certamente la rappresentazione di un concetto così complesso, specie se vuole essere semplice e immediata, è caratterizzata da un'enorme complessità intrinseca.
Non si tratta solo di sviluppare del codice che risolva un problema di logica, ma piuttosto di riuscire ad andare incontro alle vere esigenze dell'utente finale. Ciò significa dover spendere una quantità di tempo ed energia nella ricerca e progettazione non inferiore a quella spesa nello sviluppo tecnologico.

Ho potuto provare come sia essenziale, al termine di un percorso di studi ingegneristici e/o informatici, riconoscere quali sono i limiti che porta con sé tale formazione. Ciò che impariamo a fare in modo approfondito è sviluppare con precisione e puntualità la richiesta del professore (e poi del cliente) mettendo in pratica le nostre "Hard Skills", nel modo più pragmatico possibile.
Ciò che invece appare meno evidente e che più fatichiamo ad assimilare è l'idea della nostra limitatezza e specificità; la consapevolezza cioè che in quanto ingegneri la quantità di sapere a noi nascosto è enorme e che essere aperti a tali conoscenze ed esperienze, in particolare nelle materie umane, è non solo utile ma direi necessario perché le nostre abilità possano essere realmente beneficio per la comunità.

Ambisco a poter portare con me nel mondo del lavoro questa attenzione.

5. Riferimenti

[1]URL: https://en.wikipedia.org/wiki/EPCIS

[2]URL: https://en.wikipedia.org/wiki/GS1

[3]URL:https://gs1it.org/assistenza/standard-specifiche/epcis-tracciabilita-prodotti-tempo-reale/

[4]URL:https://www.gs1.org/sites/default/files/docs/epc/EPCIS-Standard-1.2-r-2016-09-29.pdf

[5]URL: https://www.treccani.it/vocabolario/gioco/

[6]URL: https://www.treccani.it/enciclopedia/teoria-dei-giochi

[7]URL: https://it.wikipedia.org/wiki/Teoria_dei_giochi

[8]URL:https://www.treccani.it/vocabolario/gamification_%28Neologismi%29/#:~:text=gamification%20s.%20f.%20inv.,e%20interessarli%20ai%20servizi%20offerti.&text=gamification%2C%20derivato%20dal%20s.

[9]URL:https://www.dispoc.unisi.it/sites/st10/files/allegatiparagrafo/20-12-2017/noname-gamification.docx#:~:text=La%20gamification%20(Figura%201)%20viene,comportamento%20delle%20persone%2C%20favorendo%20la

[10]URL:https://www.gamification.it/gamification/introduzione-alla-gamification/

[11] URL: https://en.wikipedia.org/wiki/Serious_game

[12]URL:https://www.diversity-management.it/2015/12/24/serious-game-definizione-vantaggi/

[13]URL:https://www.restorativeneurotechnologies.com/articoli-serious-games/serious-games-cosa-sono-e-perche-utilizzarli-saranno-efficaci

[14]URL:https://www.stateofmind.it/2021/01/serious-games-applicazioni/

[15]URL:https://www.travisojs.com/

[16]URL:https://github.com/axaq/traviso.js

[17]URL:https://docs.microsoft.com/en-us/archive/msdn-magazine/2014/august/unity-developing-your-first-game-with-unity-and-csharp#:~:text=Unity%20is%20a%202D%2F3D,2D%2C%202.5D%20and%203D.&text=Unity%20allows%20you%20to%20import,animation%20system%2C%20and%20much%20more.

[18]URL:https://unity.com/

[19]URL:https://docs.unity3d.com/Manual