UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Edile

SVILUPPO DI UN AMBIENTE DI SIMULAZIONE SPAZIALE PER UN SISTEMA OLONICO DI GESTIONE "LEAN" DEI PROCESSI DI COSTRUZIONE

Development of a spatial simulation environment within a holonic system for lean construction management.

Relatore:

Prof. Ing. Carbonari Alessandro

Correlatori:
Prof. Ing. Massimo Vaccarini
Prof. Ing. Francesco Spegni

Candidato:

Gliaschera Enrico

A.A. 2020 / 2021

SOMMARIO

1. INTRODUZIONE 1

2. LO STATO DELL'ARTE SULLA MODELLAZIONE DEL PROCESSO DI COSTRUZIONE 4

2.1 INDUSTRIA DELLE COSTRUZIONI 4

2.1.1 Industria delle costruzioni come sistema complesso 4

2.1.2 Limiti dei metodi basati sulla trasformazione 5

2.2 LEAN MANAGEMENT 7

2.2.1 Lista dei flussi produttivi nelle costruzioni 7

2.2.2 Flusso dello spazio e della forza lavoro 7

2.2.3 Forme di spreco secondo il Lean Management 8

2.3 LIMITI DEGLI ATTUALI METODI DI PIANIFICAZIONE LEAN 9

2.3.1 Difficoltà nella gestione dello spazio di lavoro 9

2.3.2 Principi del System Last Planner (LPS) 10

2.3.3 Limiti di LPS e diagramma di Gantt nel comprendere il flusso dello spazio 10

2.3.4 Percent Plan Complete (PPC) e Principi del sistema Location Based Management System: LBM 12

2.4 LA VARIABILITÀ NEL FLUSSO DI PRODUZIONE 14

2.4.1 Definizione di variabilità e perché è necessario analizzarla 14

2.4.2 Struttura di un workflow variabile 14

2.4.3 Considerazioni sulla variabilità 15

2.5 METRICHE PER ANALIZZARE E QUANTIFICARE LA QUALITÀ DEL FLUSSO 16

2.5.1 Construction flow index: CFI 16

2.5.2 Definizione dei valori che può assumere ed esempi di diagrammi di flusso 16

2.5.3 Misure e calibrazione del flusso che compongono il CFI 18

2.6 MODELLI DETERMINISTICI PER LA GESTIONE DELLO SPAZIO DI LAVORO 20

2.6.1 Fasi della gestione degli spazi 20

2.6.2 Pre-Collision: Modell e In-put per classificare gli spazi di lavoro 20

2.6.3 Simulate: Caratteristiche e limiti della simulazione ad eventi discreti 22

2.6.4 Collision detection: Metodi di rilevazione delle collisioni 22

2.6.5 Tecniche per valutare l'entità delle collisioni e limiti 23
2.6.6 Limiti dei modelli deterministici di gestione dello spazio di lavoro 24

3. APPROCCIO OLONICO PER LA PIANIFICAZIONE E GESTIONE DELLO SPAZIO 25

3.1 PIANIFICAZIONE GERARCHICA OFF-LINE IN SISTEMI COMPLESSI 25
3.1.1 Limiti delle architetture gerachiche 25
3.2 HOLONIC MANUFACTURING SYSTEMS (HMS) 27
3.2.1 Gestione decentralizzata 27
3.2.2 Concetto di olone 27
3.2.3 Architettura di sistema PROSA e tipologie di olone 28
3.2.4 Interazione fra oloni 29
3.3 PATTERN ARCHITETTONICO DMAS 31
3.3.1 Caratteristiche del pattern architettonico DMAS 31
3.3.2 Ruolo dell'agente formica esplorante 32
3.4 SIMULAZIONI ONLINE AD EVENTI DISCRETI IN AMBIENTI OLONICI 33
3.4.1 Vantaggi nell'utilizzo di simulazioni online ad eventi discreti 33

4. ELABORAZIONE DI UN NUOVO WORKFLOW NEL CASO DI LAVORI DI RIQUALIFICAZIONE 34

4.1 STRUTTURA DEI SISTEMI DI GESTIONE DELLA COMUNICAZIONE 34
4.1.1 Principi dello scambio di informazioni: standart O-MI 34
4.1.2 Struttura del sistema KanBim per gestire l'informazione 36
4.1.3 Stabilità del piano dei lavori 41
4.2 WORKFLOW NEL CASO DI LAVORI DI RIQUALIFICAZIONE 43
4.2.1 Struttura del workflow nel caso di lavoro di riqualificazione 43

5. IL CASO STUDIO: SVILUPPO DI UN SISTEMA OLONICO PER LA GESTIONE DEI LAVORI IN TEMPO REALE 50

5.1 STRUTTURA CASO STUDIO 50
5.1.1 Mappatura fra caso studio e architettura PROSA e DMAS 50
5.1.2 Necessità di un motore grafico per visualizzare conflitti spaziali 52
5.2 STRUMENTI UTILIZZATI 54
5.2.1 Ambiente simulazione: Unity 3D 54
5.2.2 Piglet: glTF Importer 57
5.3 OBBIETTIVO DEL SISTEMA SVILUPPATO 61

5.4 STRUTTURA DEL SISTEMA SVILUPPATO PER LA GESTIONE DEI LAVORI IN TEMPO REALE 62

5.4.1 Workflow realizzativo del sistema 62

5.4.2 Fase 1: Definizione di una gerachia di elementi nell'ambiente di simulazione 63

5.4.3 Fase 2: Importazione del piano dei lavori e del modello 3D 65

5.4.4 Fase 3: Generazione degli spazi di lavoro 84

5.4.5 Generazione dei colori associati a spazi di lavoro 89

5.4.6 Fase 4: Output simulazione: lista delle sovrapposizioni di spazi di lavoro 93

5.4.7 Sviluppi futuri: misurazione della sovrapposizione volumetrica delle collisioni 100

6. DISCUSSIONE DEI RISULTATI E CONCLUSIONI 104

7. APPENDICE 108

7.1 "reader_json_simulation" script 108

7.2" merge_workspace" script 119

7.3 "overlapbox" script 121

7.4 Porzione baseline di "ACOIN_XXXX_BASELINE.JSON" 124

8. BIBLIOGRAFIA 139

9. INDICE DELLE FIGURE 140

1. INTRODUZIONE

In sistemi di produzione industriale, dove la generazione di più prodotti avviene in condizioni prestabilite, difficilmente soggette a variazioni attraverso un processo identico a se stesso per un grande numero di iterazioni, la produzione può essere ottimizzata attraverso piani redatti in anticipo non essendo questa soggetta a deviazioni imprevedibili.
In termini assoluti, il processo produttivo raggiunge la massima efficienza quando ogni risorsa è processata senza interruzioni e genera del valore.

Nell'industria delle costruzioni, il prodotto realizzato è definito *"one-of-a-kind product"* poichè ogni elemento differisce in termini progettuali e realizzativi rispetto quello precedente e quello successivo.
Difficilmente, la stessa costruzione viene ripetuta in serie, in luoghi differenti.
Anche se ciò avvenisse, non sarebbe possibile replicare in ogni processo di realizzazione le stesse condizioni al contorno: il meteo, la temperatura che influisce nella resa di specifiche applicazioni, forniture di materiali di grosse dimensioni possono tardare a causa di ingorghi stradali, le prestazioni dei macchinari possono variare in funzione di guasti o condizioni del terreno.
In aggiunta, nei processi produttivi di costruzione la manodopera che influisce nella realizzazione dell'opera è estramente eterogenea.
Al processo di produzione partecipano più individui organizzati in squadre con differenti know-how, la cui comunicazione risulta frammentata e disorganizzata.
La mancanza di coordinamento tra i vari partecipanti al progetto, interrompe il flusso regolare del lavoro attraverso discontinuità che si manifestano in tempi di attesa per le squadre, rilavorazioni, spostamenti e trasformazioni che non generano valore.

Tutto ciò, permette di considerare il processo delle costruzioni come un sistema complesso il cui comportamento non può essere definito a priori.
Significa che la stessa scelta progettuale, non genera lo stesso effetto se applicata in processi produttivi differenti, perché questa verrà influenzata da fattori difficilmente identici in processi differenti.

Lo strumento attraverso cui migliorare i processi di costruzione sono le tecniche di pianificazione, le quali hanno il compito di organizzare nel tempo le attività.
Al momento i sistemi di pianificazione dispongono di metodi redatti relativamente troppo in anticipo per rispondere alle esigenze dinamiche delle costruzioni, dove è necessario un controllo quotidiano del processo realizzativo.
La maggior di questi risulta gerarchico, poco flessibile ed inaccurato in termini di risorse necessarie al completamento delle azioni.
Generare un piano dei lavori completo ad alto livello di dettaglio, prima dell'inizio della costruzione può essere uno strumento di pianificazione di riferimento che, però, non ha la capacità di simulare il reale avanzamento dei lavori, essendo il processo altamente variabile.
Inoltre, quando variazioni si presentano durante il processo, spesso gli strumenti di pianificazione attuali non hanno implementati metodi decisionali capaci di vagliare più ipotesi progettuali (spesso l'unico parametro decisionale risulta essere la data ultima entro cui completare l'attività).

Il processo di costruzione può essere ottimizato se le risorse, ad esempio quella spaziale, vengono continuamente processate.
In altri termini, si ottiene ottimizzazione del processo se negli spazi dell'area di costruzione avviene in continuo un processo di produzione.
In termini assoluti questo porta ad una diminuzione delle tempistiche, dei costi ed un aumento della qualità, intesa come la capacità di produrre in maniera conforme al pianificato.
Risulta perciò indispensabile diminuire l'orizzonte temporale del programma dei lavori al fine di ottimizzare il processo in orizzonti temporali più brevi, capaci di considerare almeno i principali fattori al contorno di un sistema complesso.
Ogni azione per poter essere portata a termine ha bisogno di manodopora, forniture di materiali, attrezzature e spazio di lavoro libero.
Quindi, è indispensabile ai fini dell'ottimizzazione del processo definire strumenti in grado di tracciare e misurare le risorse per ogni attività.

Il lavoro proposto, mira a definire uno strumento capace di sensibilizzare il cronoprogramma in termini della risorsa spaziale.
Ovvero, associare ad ogni attività uno spazio di lavoro specifico all'interno dell'area di cantiere.
La risorsa spaziale include lo spazio necessario alla movimentazione dei tecnici, allo stoccaggio del materiale e delle attrezzature.
Al momento, i classici diagrammi di Gantt non riconoscono ad un'attività i termini spaziali nei quali avviene.
Questo comporta un assoluta miopia in termini di sovrapposizioni degli spazi di lavoro di attività differenti che di fatto ne impediscono l'esecuzione contemporanea.

Allo stesso tempo, il lavoro si pone come obbiettivo quello di introdurre lo strumento all'interno di un sistema in grado compiere simulazioni del processo costruttivo in orizzonti temporali abbastanza brevi, dell'ordine di qualche giorno.
Essendo l'orizzonte temporale di simulazione breve, è giustificabile escludere eventi eccezionali come guasti di macchinari o condizioni meteo sfavorevoli la cui provabilità di accadere in ordini temporali così brevi è minima.

Lo scopo principale degli strumenti di simulazione è quello di di introdurre flessibilità nella gestione del piano dei lavori ed aumentare la qualità del pianificato.
Intercettare una collisione fra più spazi di lavoro prima che l'attività abbia inizio vale a dire diminuire il numero di variazioni da attuare in corso d'opera.
Allo stesso tempo, la simulazione costituisce uno strumento di previsione capace di prendere in considerazione i risultati effettivi di più processi decisionali.
In questo modo, è possibile vagliare più ipotesi ed attuare quella che ne determina la maggiore ottimizzazione.

2. LO STATO DELL'ARTE SULLA MODELLAZIONE DEL PROCESSO DI COSTRUZIONE

2.1 INDUSTRIA DELLE COSTRUZIONI

2.1.1 INDUSTRIA DELLE COSTRUZIONI COME SISTEMA COMPLESSO

Per comprendere l'industria delle costruzioni si deve riconoscere che, a differenza di quella manufatturiera, genera un prodotto unico (identico solo a sé stesso quindi mai realizzato prima *"one-of-a-kind product*"), complesso ed articolato, generato ed assemblato principalmente in sito attraverso la cooperazione di squadre multi-qualificate.

Il prodotto unico rende il flusso di comunicazione importante quanto il flusso di materiali e attrezzature, aggiungendo sostanzialmente complessità al progetto, specialmente nelle fasi del processo più dense di flussi produttivi.

Nelle costruzioni il flusso di produzione è meno evidente rispetto a quello manufatturiero; quindi, risulta più difficile controllarlo indagando i legami e gli effetti delle risorse che concorrono allo stesso progetto.

Inoltre nei processi di produzione delle costruzioni il sistema di produzione è temporaneo e di solito ci sono diversi progetti in competizione per le stesse risorse. Così, alterazioni in un progetto sono facilmente trasmesse ad altri progetti.

La difficoltà di percepire in maniera rapida i legami e i vincoli fra le risorse che collaborano è il frutto delle seguenti considerazioni:

1. le costruzioni sono caratterizzate da supply-chain eterogenee composte da individui indipendenti che forniscono manodopera, servizi o materiali.

Tutto ciò si traduce in difficoltà di comunicazione fra squadre e trasferimento di responsabilità, che causa la possibile perdita di sensibilità del pianificato e del controllo in fase di costruzione.

2. le squadre di produzione si muovono mentre il prodotto rimane fermo.

Non esiste apparentemente un flusso fisico misurabile o risulta altamente spezzettato.

3. il processo di costruzione è basato fortemente sul progetto, il quale non descrive solamente il prodotto ma indica la pianificazione, l'organizzazione delle squadre di cantiere e delle risorse volte alla costruzione (materiali da utilizzare, posizionamento

delle attrezzature in cantiere, tempo per completare azioni ecc.) che tradotto in termini manufatturieri assomiglia a dover progettare e demolire ogni volta la catena di montaggio.
Risulta perciò un sistema variegato dove anche la variazione di singole variabili (esempio un ritardo di consegna di un materiale, un danno ad un attrezzatura o un cambio meteorologico) può modificare l'intero flusso di produzione.

Un sistema complesso mostrerà comportamenti che non possono essere dedotti a priori studiando la forma dei suoi elementi.
Poiché i sistemi complessi sono per loro natura imprevedibili, la gestione non può essere basata su istruzioni o piani dettagliati.
L'idea di base è che le costruzioni non dovrebbero essere intese come trasformazioni da portare a termine ma come flussi di produzione che generano valore.

2.1.2 LIMITI DEI METODI BASATI SULLA TRASFORMAZIONE

I metodi attuali di pianificazione e controllo sono basati sul concetto di trasformazione, intesa come il completamento di specifiche azioni *"task"* vincolate fra loro.
Costruire un'intera pianificazione basandosi esclusivamente sul completamento di azioni è improvabile che determini la miglior efficienza perché vengono escluse dal ragionamento tutte le risorse che contribuiscono alla realizzazione dell'azione stessa.

Le risorse necessarie alla realizzazione del task possono essere in termini di spazio, di manodopera, di materiali, di attrezzature, di spazio disponibile per lo stoccaggio del materiale ecc., quindi con alta capacità di influenzare il completamento dell'azione e la durata o il costo del progetto.
Ad esempio, risulta insufficiente non considerare il legame fra azione da svolgere e spazio nel quale deve essere svolta.
I metodi attuali concentrandosi sulla sola realizzazione di attività non possono notare se alcuni spazi sono già occupati da altre lavorazioni in corso o una lavorazione occupa più spazi generando di fatto interferenze che causano rallentamenti e attese della forza lavoro.
Questi metodi non considerano se più squadre si sovrappongono nello stesso spazio magari insufficientemente grande per compiere entrambe le lavorazioni assieme.

I metodi attuali non hanno la possibilità di determinare l'indisponibilità di una squadra già occupata in altre lavorazioni, né tanto meno della indisponibilità della risorsa materiali.
In sistemi complessi come quello delle costruzioni è indispensabile organizzare le lavorazioni in funzione delle risorse che contribuiscono alla realizzazione della stessa e come queste sono relazionate fra loro, così da poter evidenziare le interferenze.
Inoltre, i sistemi attuali di pianificazione e controllo sono fondati su metodi con alto livello di dettaglio redatti relativamente troppo in anticipo per rispondere alle esigenze di produzione dinamica delle costruzioni.
A questo schema rigido e all'esclusione delle risorse dal metodo di pianificazione corrispondono ampi margini di miglioramento in termini di durata e costo del progetto.

2.2 LEAN MANAGEMENT

2.2.1 LISTA DEI FLUSSI PRODUTTIVI NELLE COSTRUZIONI

A fronte del fatto che il completamento delle attività, su cui si basano i metodi attuali, è fortemente influenzato dai flussi che le generano, il *"Lean Construction"* suggerisce di concentrare le attenzioni sui flussi che contribuiscono alla realizzazione delle costruzioni e comprendere le interazioni che si generano fra loro per ridurre la durata e i costi del progetto.

In ottica Lean, la realizzazione delle costruzioni è definita da una serie di flussi composti da trasformazioni, ispezioni, movimentazioni e tempi di attesa, i quali possono essere divisi nelle seguenti tipologie:

- flusso dello spazio di lavoro,
- flusso della forza lavoro,
- flusso delle attrezzature,
- flusso dei componenti,
- flusso delle informazioni e progettazione,
- flusso delle condizioni esterne,
- flusso dei lavori preliminari.

2.2.2 FLUSSO DELLO SPAZIO E DELLA FORZA LAVORO

Un'interpretazione delle costruzioni è quella di definire la produzione in funzione del flusso dello spazio.

Sebbene lo spazio sia una risorsa e non si muova, può essere paragonato ad un prodotto lungo una linea di produzione.

Questo perché nello spazio (come lotti, porzioni di edificio, stanze ecc.) si genera produzione e tracciando che cosa succede al suo interno è possibile fornire indicazioni della produttività.

A differenza delle linee di produzione dell'industria manufatturiera stazionarie, nelle costruzioni le squadre di lavoro si muovono all'interno del progetto e il loro flusso di produzione può essere osservato in funzione del luogo in cui operano.

Anche altri flussi ausiliari devono essere considerati perché essi non determinino attese, scorte insufficienti, interferenze e altre forme di spreco.

2.2.3 FORME DI SPRECO SECONDO IL LEAN MANAGEMENT

L'obbiettivo del lean management è quello di minimizzare gli sprechi nel flusso produttivo, considerandoli elementi che nascondono margini di miglioramento nel processo.
Uno spreco è inteso come l'utilizzo della risorsa in termini di tempo, forza lavoro, materiali ecc., a cui non è associata una generazione del valore.

Uno spazio, quindi, una zona dell'edificio in cui non è presente un'attività produttiva ed è in attesa che una squadra avvii una lavorazione al suo interno risulta uno spreco ed è ipoteticamente considerabile come ad un prodotto che si accumula di fronte al macchinario in attesa di essere processato.
Allo stesso modo, risultano uno spreco le squadre di operatori che non lavorano in attesa che vengano terminate lavorazioni quindi liberato lo spazio necessario alla lavorazione o che del materiale arrivi in cantiere.
Anche la movimentazione delle squadre di lavoro, come il ritorno sullo stesso posto della stessa squadra per differenti aspetti legati alla lavorazione, è considerata uno spreco che oltretutto avendo interrotto la lavorazione ha lasciato per un determinato periodo lo spazio non processato.

In riassunto, un valido flusso degli spazi e delle forze lavoro si raggiunge se negli spazi e nei loro sottoinsiemi avvengono delle lavorazioni in continuo con una produttività stabile, minimizzando i cicli di lavoro delle squadre nello stesso spazio con *"handoff"* fluidi e privi di attese.
Ovvero quando una lavorazione passa dall'essere eseguita da una squadra di lavoro ad un'altra con specializzazione differente senza che avvengano attese o perdita di informazioni.

2.3 LIMITI DEGLI ATTUALI METODI DI PIANIFICAZIONE LEAN

2.3.1 DIFFICOLTÀ NELLA GESTIONE DELLO SPAZIO DI LAVORO

La gestione dello spazio di lavoro è definita come il processo di generazione e assegnazione degli spazi di lavoro, il loro collegamento alle attività di pianificazione, il rilevamento dei conflitti spazio-temporali tra gli spazi di lavoro e la risoluzione dei conflitti identificati.

L'obiettivo principale è quello di garantire l'assenza di conflitti spaziali e congestioni degli spazi di lavoro diminuendo l'impatto negativo su costi, tempi e sicurezza.

I conflitti fra gli spazi di lavoro (chiamati anche *"operational clashes", "4D clash", "workspace interference")*, sono sovrapposizioni inaspettate tra le risorse spazio durante l'esecuzione delle attività, le quali possono determinare una perdita di produttività e un ritardo dell'intero progetto.

Incertezze come le variazioni di produttività, durate di esecuzione più lunghe o più brevi rispetto quelle pianificate o l'occupazione di spazi di lavoro più grandi del previsto, possono portare a collisioni.

Gli spazi di lavoro, necessari per eseguire le attività, sono una risorsa limitata che influisce sulla consegna delle azioni.

La gestione degli spazi di lavoro in cantiere è impegnativa per le seguenti ragioni:

i) le posizioni e i volumi degli spazi di lavoro cambiano in tre dimensioni e nel tempo in funzione delle proprietà dell'attività;

ii) le attività in cantiere sono di solito eseguite da più squadre di lavoro con specializzazione differente, le quali richiedono aree di lavoro, stoccaggio di materiali, attrezzature e infrastrutture di supporto.

Questo si traduce spesso in conflitti spazio-temporali in cui due attività condotte dalla stessa squadra o da due squadre diverse competono per lo stesso spazio;

iii) le tecniche tradizionali di pianificazione della costruzione come i diagrammi di Gantt sono inadeguati per gestire gli spazi di lavoro a causa della loro mancanza di rappresentazione spaziale.

I conflitti spazio-temporali sono riconosciuti come una delle principali cause di perdita di produttività.

2.3.2 PRINCIPI DEL SYSTEM LAST PLANNER (LPS)

Il System Last Planner (LPS) è un sistema collaborativo di pianificazione che mira a ridurre le variazioni del workflow aumentando l'affidabilità del pianificato attraverso quattro fasi di pianificazione e il rispetto di due principi:

1. il processo di *"make ready"* nella fase di pianificazione a medio termine (*lookahead)*, dove vengono identificati i vincoli per ogni compito con l'attenzione di impostare l'inizio delle lavorazioni affinché il flusso di produzione che si genera sia privo di interruzioni.

Questo significa che una lavorazione viene avviata solo se il suo completamento corrisponde con l'inizio di una successiva lavorazione favorendo un handoff fluido.
Se la lavorazione terminasse prima, lo spazio rimarrebbe non processato per un tempo diverso da zero, quindi si generebbe spreco.

2. inglobando nella fase di maggior dettaglio del pianificato (*week workly planning)* i responsabili delle squadre di lavoro coinvolte nelle lavorazioni dell'orizzonte temporale di pianificazione.

In questa fase:

- si verifica che i vincoli del pianificato siano realmente stati adempiti, quindi: spazi liberati, lavorazioni terminate, disponibilità di materiali, attrezzature presenti e pronte ad essere utilizzate.
- i responsabili delle lavorazioni successive definiscono in maniera puntuale quali lavorazioni hanno bisogno che siano completate affinché possano avviare la loro.

2.3.3 LIMITI DI LPS E DIAGRAMMA DI GANTT NEL COMPRENDERE IL FLUSSO DELLO SPAZIO

Il Last Planner System, in particolar modo nella fase di pianificazione *"lookahead",* è strettamente legato all'identificazione e il monitoraggio dello stato dei vincoli delle attività, il tutto per rispettare il concetto di *"make ready"* (esposto nei paragrafi precedenti).

Lo stato dei vincoli a sua volta è strettamente dipendente dai flussi delle risorse (spazio, materiali, forza lavoro, attrezzatura ecc.) e dunque dalla loro gestione.

In questo contesto, il metodo LPS presenta due grandi limitazioni:

1. spesso i vincoli che vengono identificati sono quelli esterni al sistema di costruzione (ad esempio l'arrivo in cantiere di materiali, disponibilità di informazioni dal

progettista), mentre vengono trascurati quelli interni al processo (ad esempio se due attività si svolgono nello stesso spazio oppure utilizzano la stessa attrezzatura);

2. non modella esplicitamente i flussi tra le attività, dunque risulta difficile definire quale sarà l'impatto della variazione di un flusso su altre attività che dipendono dal flusso stesso.

Inoltre, il metodo LPS utilizza come strumento di pianificazione e rappresentazione delle attività il diagramma di Gantt, generato dai vincoli che uniscono le attività.

Lo strumento a causa della propria struttura presenta tre problematiche legate alla gestione delle risorse e dello spazio di lavoro, nello specifico:

1. ad alti livelli di dettaglio della pianificazione (esempio Lookahead o Weekly work plan) corrispondono diagrammi complessi e disordinati a causa dell'alto numero di attività e dei loro vincoli. Non restituiscono una visione chiara della situazione generando il rischio di non riuscire ad identificare le interferenze;

2. le relazioni spaziali tra le attività non sono facili da cogliere e non si capisce come le risorse si trasferiscano e varino attraverso il passaggio di spazi di lavoro o azioni nello stesso spazio di lavoro. Per esempio, nella figura seguente, non è immediatamente chiaro da quali azioni venga occupato lo spazio liberato dal completamento dell'azione *"concrete orizzontal"* nella zona 3 e come le risorse (lavoro, attrezzature e materiali) variano all'interno della zona 3 (Fig. 1):

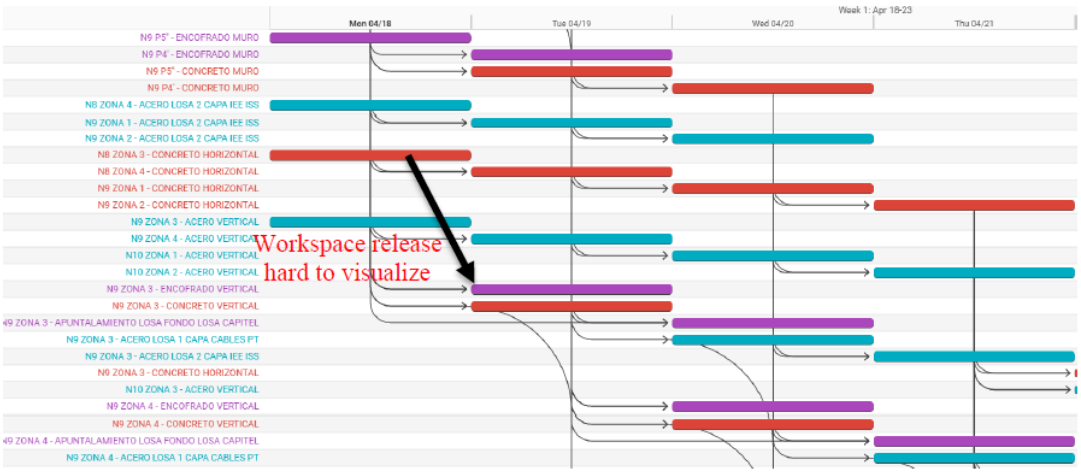

Figura 1 Il rilascio dello spazio di lavoro è difficile da cogliere attraverso la rappresentazione Gantt

3. non viene considerata la possibilità che due azioni nella stessa zona dell'edificio possano necessitare di due spazi di lavoro differenti fra loro a causa, ad esempio, delle attrezzature legate alla lavorazione.

Dunque, i modelli 4D che uniscono al modello 3D la variabile temporale sono preferiti nella gestione delle risorse ed in particolare degli spazi di lavoro, in quanto:
- hanno una rappresentazione esplicita delle relazioni spaziali permettendo di individuare i conflitti anche in maniera automatica;
- definiscono gli spazi di lavoro come nuova variabile di pianificazione,
- rispettano la natura dinamica degli spazi di lavoro in termini di posizione e forma geometrica che possono variare nel tempo,
- consentire l'associazione fra spazi di lavoro e attività in più tipi di relazione (cioè 1 a 1, 1 a n e n a n).

2.3.4 PERCENT PLAN COMPLETE (PPC) E PRINCIPI DEL SISTEMA LOCATION BASED MANAGEMENT SYSTEM: LBM

Nel metodo LPS l'affidabilità del pianificato è misurabile attraverso un parametro chiamato Percent Plan Complete (PPC) che viene calcolato come il numero di incarichi completati rispetto al numero totale di incarichi assegnati.
Tuttavia, il PPC risulta ancora una misura insufficiente del flusso di produzione per i seguenti motivi:
- misura eventuali variazioni solo quando l'attività verrà completata,
- non quantifica l'impatto della variazione e quali conseguenze questa esercita sulle altre attività,
- misurando la variazione solo alla data di completamento tralascia la variazione alla data di inizio e sulla durata.

Si è dimostrato che in progetti di grandi dimensioni il metodo LPS, focalizzato sulla gestione delle attività e non sulle variazioni del workflow, porta ad un miglioramento del flusso ma, ciò nonostante, il PPC risulta debolmente correlato ad una produttività fluida e rimane una buona misura per rappresentare la qualità del pianificato ma non della produzione.

Altro metodo è il Location Based Management System (LBM) che visualizza attraverso

diagrammi di linee di flusso *(flowline chart)* i flussi di produzione.
In sistemi complessi è altamente improbabile che il raggiungimento di un flusso di produzione ottimale coincida con il raggiungimento della massima produttività di tutte le squadre di lavoro.

2.3.6 CARATTERISTICHE DI UN FLUSSO OTTIMALE DI PRODUZIONE

Difatti, anche solo osservando la rappresentazione dei diagrammi di flusso è riconoscibile la qualità indicativa del flusso che risulta ottimale quando sono rispettate le caratteristiche seguenti:

- balanced work: tempi simili per completare le lavorazioni in ogni spazio, traducibile nel
  diagramma di flusso con linee dalla pendenza simile (raggiungibile bilanciando la produttività).
- il numero di spazi occupati dalla squadra di lavoro deve essere costantemente pari ad uno.
- il tempo di attesa fra ogni lavorazione è zero per ogni spazio.
- il numero di operatori della squadra di lavoro deve essere ridotto al minimo.
- non devono essere ripetute più volte le stesse lavorazioni nello stesso spazio.
- eseguire solo le azioni i cui vincoli sono stati rimossi.
- prediligi dove possibile just-in-time delivery, ovvero produci solo quando serve.

Si nota come le prescrizioni riguardano sia la risorsa spazio che quella della forza lavoro.

2.4 LA VARIABILITÀ NEL FLUSSO DI PRODUZIONE

2.4.1 DEFINIZIONE DI VARIABILITÀ E PERCHÉ È NECESSARIO ANALIZZARLA

Per variabilità si intende un discostamento dalla qualità o quantità delle azioni pianificate; quali possano riguardare: ritardo in consegna dei materiali, produttività inferiore di una determinata squadra o resa maggiore di attrezzature rispetto quella attesa ecc.

In ambienti complessi è necessario gestire e pianificare mentre si produce, direzionando la pianificazione in funzione delle variabilità che si presentano.

Avere delle regole per comprende come la variabilità viene trasmessa fra le azioni e come questa può evolvere nel workflow permette di anticipare il suo avvenire oppure favorirlo nel caso in cui determini uno scostamento vantaggioso dal pianificato.

Le variabilità e le influenze indotte sulle azioni non sono tutte uguali, a volte nemmeno correlabili fra loro. Ad esempio, la mancanza di un determinato materiale non influenza sulle attività che non lo utilizzano.

2.4.2 STRUTTURA DI UN WORKFLOW VARIABILE

Una schematizzazione della variabilità in un worflow può essere quella mostrata nella figura successiva. Vengono introdotti, nei sette flussi esposti nel paragrafo *2.1.1* due meccanismi di variabilità:

(i) "Variability Factors" che influiscono sull'intero flusso delle attività;

(ii) "Late release of flows" dunque il rilascio del flusso di un'attività fuori tempo che interagiscono con un solo flusso.

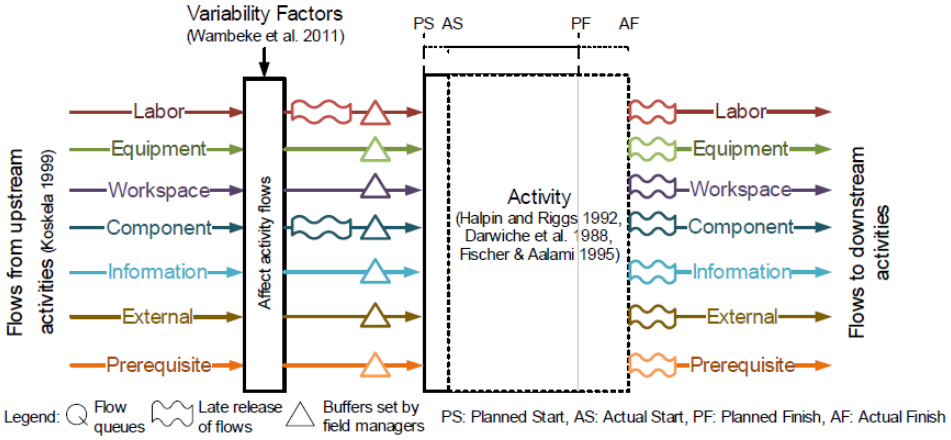

Figura 2 La struttura di un workflow variabile prevede due tipologie di variabilità

Due meccanismi regolano il modello:

1. Se le attività a monte terminano in ritardo, allora anche l'avvio delle azioni successive vincolate alla prima presenterà ritardo, causando variazione del flusso delle attività che susseguono.

Al contrario se un'attività finisce prima ci sarà un rilascio dei vincoli prematuro che causa tempi di attesa.

2. È possibile implementare buffers in termini di scorte o tempo mirati ad assorbire variazioni dei singoli flussi di azioni.

Se i buffers risultano insufficienti allora si presenta una variazione nell'esecuzione.

Attraverso l'analisi delle seguenti tre grandezze per ogni azione si analizza l'evoluzione della variabilità nel worflow:

(1) activity start= inizio reale – inizio pianificato,

(2) activity duration= durata reale – durata pianificata,

(3) total variability= fine reale – fine pianificata.

2.4.3 CONSIDERAZIONI SULLA VARIABILITÀ

La raccolta e l'analisi delle grandezze ha dimostrato che:

- la variabilità è trasmessa dalle attività a monte a quelle a valle, ma non necessariamente quelle a valle mostrano una maggiore variabilità rispetto alle attività a monte.
- attività con alto numero di vincoli come, ad esempio, quelle del gruppo impiantisco non necessariamente presentano variabilità maggiori rispetto ad attività che devono rispettare un numero inferiore di vincoli.
- è più probabile che un'attività sia influenzata da un ritardo piuttosto che essere avviata o completata prima del previsto.
- la maggior parte delle attività è influenzata da una piccola variabilità mentre solo alcune attività (in numero molto minore rispetto le prime) sono gravemente influenzate dalla variabilità.
- la variabilità media è estremamente instabile all'inizio del progetto e si stabilizza con l'avanzamento della costruzione, in particolar modo per i sub-contractor con molte attività.

2.5 METRICHE PER ANALIZZARE E QUANTIFICARE LA QUALITÀ DEL FLUSSO

2.5.1 CONSTRUCTION FLOW INDEX: CFI

Il construction flow index (CFI) ha l'intento di misurare la qualità del flusso nei progetti di costruzione e comprendere se scelte pianificate migliorano o peggiorano il flusso.
Il suo calcolo è realizzabile anche rispetto intervalli temporali brevi come giorni o settimane ed è basato su misure relativamente semplici e veloci da conoscere.
Le misure sono ricavabili dal diagramma di flusso pianificato o as-built ed è calcolabile attraverso una funzione polinomiale parametrica di primo o secondo grado il cui valore può variare fra 1 e 10.
Maggiore sarà il valore, maggiore sarà la qualità del flusso.
Di seguito è mostrata la formula:

$$CFI(t) = 10 \sum_{i=1}^{7} W_i \, P_i^{x_i} \;\; con \, X_i \;\in [1,2]$$

Figura 3 Formula del Construction Flow Index

Il valore delle grandezze che compongo la formula dell'indice CFI viene esplicato nei paragrafi successivi.

2.5.2 DEFINIZIONE DEI VALORI CHE PUÒ ASSUMERE ED ESEMPI DI DIAGRAMMI DI FLUSSO

Nella fase di generazione della formula sono stati analizzati differenti scenari di progetti reali e simulati, ognuno dei quali rappresentato da un diagramma di flusso al quale è stato assegnato un certo valore di CFI.
Al valore 10 corrisponde la massima qualità del flusso di produzione, al contrario al valore 1 la minima.Seguentemente è riportato un esempio pratico della valutazione delle principali caratteristiche che classificano le linee di flusso.
I principi utilizzati sono quelli già elencati nel paragrafo 2.3.6.
Ogni colore rappresenta una lavorazione: impianti elettrici, impianti idraulici, partizioni

interne, intonaci, impermeabilizzazione, sistemi antincendio, piastrellatura.
Ad ogni lavorazione corrisponde una squadra di lavoro con specializzazione differente.
Quando la linea viene tracciata significa che la lavorazione è in corso.

In ordinata leggiamo dove avviene e in ascissa il conteggio dei giorni dall'inizio del progetto, da cui è deducibile la durata.

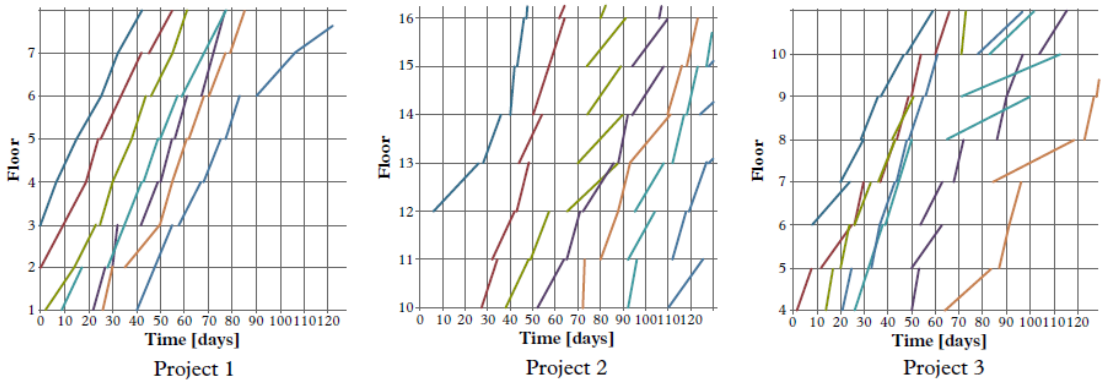

Figura 4 Flowline charts di tre differenti progetti che mostrano il reale progresso di sette lavorazioni in sette spazi

Il progetto 1 ha il flusso migliore perché:
- la massima attesa della singola lavorazione è di circa 10 giorni (lavorazione viola al giorno 30). Linee continue corrispondono a squadre che completato un piano e passano a quello successivo senza attese, quindi sprechi.
- ha inclinazioni delle linee lineari e simili fra loro *"balance worked"*, traducibili in produttività simile priva di variazioni, completano lavorazioni differenti nello stesso piano con la stessa durata.

Il progetto 2 ha un flusso di qualità media perché:
- ha il maggior numero di lavorazioni in corso per squadra.

Infatti, la lavorazione color ocra al giorno 80 è presente in quattro differenti spazi contemporaneamente (piani 12,13,14,15).
- ha la durata maggiore delle singole lavorazioni, deducibile dalla bassa inclinazione media delle linee.

Il progetto 3 ha la peggior qualità del flusso perchè:
- ha alto numero di interferenze fra lavorazioni deducibile dal fatto che più linee si

intersecano.
La lavorazione ocra e quella fucsia lavorano contemporaneamente nello stesso spazio (piani 5,6,7,8).
- ha la maggior variazione nella sequenza.
Quasi nessuna lavorazione presenta un diagramma continuo. La lavorazione rosa inizia a lavorare nel piano 7 prima ancora di avviare le lavorazioni nel piano 5.

In nessun caso mostrato la stessa squadra si presenta sullo stesso piano per più di una volta (avvia la lavorazione, poi la interrompe e poi torna per concluderla) deducibile dal fatto che nessuna linea dello stesso colore viene tracciata per più di una volta nello stesso piano.

2.5.3 MISURE E CALIBRAZIONE DEL FLUSSO CHE COMPONGONO IL CFI

Successivamente si è calcolato il CFI di ogni diagramma usato nell'indagine attraverso la formula indicata nel paragrafo 2.5.1.
Per determinare il parametro (P_i) sono state compiute le misure del flusso di produzione. Queste indicano da che cosa dipende il valore del CFI e sono di seguito riportate:

Symbol	Description	Unit of measure
RS_i	The square of the Pearson product moment correlation coefficient, for each trade *i*. This reflects the smoothness of the flowline	–
P_i	Duration per location for each trade *i*	Days/location
STD	Standard deviation of duration per location	Days/location
ND	Days of break for all tasks	Days
NB	Number of breaks for all tasks	–
TFP	Sum of all locations produced by all trades	Locations
NT	Number of tasks considered (active tasks)	Tasks
BP	Number of times a task is performed before its predecessor in a location (work out of sequence)	Tasks
BF	Number of times a crew works on location *X* before location *X* – 1 (location out of sequence)	Tasks
WIP	Work in Process (WIP) – number of locations with work in progress	Number of locations

Figura 5 Indicazione delle grandezze che influiscono nel calcolo di CFI

In funzione dei valori di CFI associati ai diagrammi di flusso nella fase precedente si è calibrata la formula. Utilizzando un algoritmo di ricerca a forza bruta selettiva si è definito il peso dei valori mancanti (Wi).
Il risultato è stato ottenuto facendo tendere a zero la differenza fra il CFI rilevato nella fase di campagna e il CFI calcolato dalla formula.
Nella risoluzione sono stati imposti i seguenti vincoli: la somma dei pesi deve essere

uguale ad 1, non possono esserci pesi negativi ed il valore di calcolo CFI è compreso fra 1 e 10.

Attraverso due funzioni obbiettivo differenti si è minimizzato con una l'errore medio totale, mentre con l'altra si è minimizzato l'errore singolo. Giungendo così alle seguenti formulazioni:

(A) $CFI(t) = 10[0.28P_2^2 + 0.38P_3^2 + 0.20P_5^2 + 0.14P_6]$

(B) $CFI(t) = 10[0.30P_2^2 + 0.32P_3^2 + 0.06P_4^2 + 0.24P_5^2 + 0.03P_6 + 0.05P_7^2]$

2.6 MODELLI DETERMINISTICI PER LA GESTIONE DELLO SPAZIO DI LAVORO

2.6.1 FASI DELLA GESTIONE DEGLI SPAZI

Come mostrato in figura successiva, i problemi di gestione dello spazio di lavoro sono suddivisibili in tre fasi:

1) pre-collision:

si effettua la pianificazione degli spazi di lavoro, in cui vengono inseriti i dati riguardo la dimensione, la forma, il legame e il comportamento delle risorse.

Il passo seguente è il processo di simulazione ad eventi discreti per visualizzare l'evoluzione degli spazi di lavoro;

2) collision detection:

avviene una sovrapposizione di risorse ed il sistema è in grado di rilevarlo;

3) post-collision:

si valuta la gravità dell'impatto attraverso la dimensione della collisione, la sua durata e la criticità dell'attività.

Di conseguenza, viene determinata l'azione di risoluzione, cioè se procedere come pianificato, cambiare le proprietà dello spazio di lavoro o alterare la pianificazione.

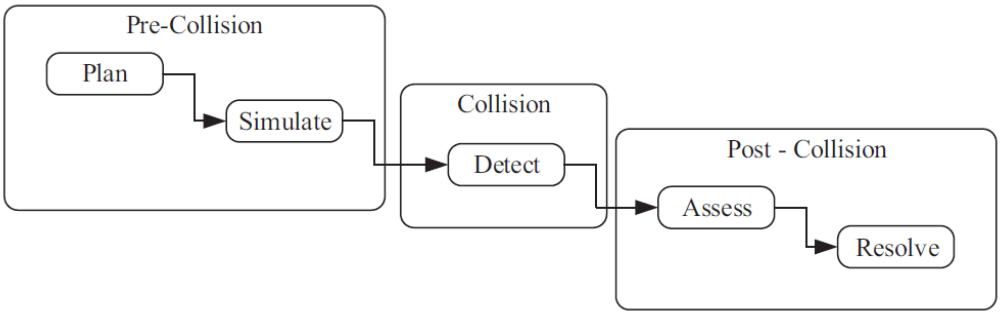

Figura 6 Il modello deterministico di gestione dello spazio è composto da tre fasi differenti: Pre-Collision, Collision, Post-Collision

2.6.2 PRE-COLLISION: MODELL E IN-PUT PER CLASSIFICARE GLI SPAZI DI LAVORO

La prima fase è la pianificazione *(Plan)*.

In questa fase si genera un modello degli gli spazi di lavoro corrispondenti a quelli del progetto e mettendoli in relazione alle attività associate.

Il modello viene preferito ad un modello 3D disconnesso dalla pianificazione per la

gestione dello spazio di lavoro perché:

1. un elemento della costruzione (una colonna od una trave ad esempio) spesso è il prodotto di più attività; quindi, deve essere associato a vari spazi di lavoro in diverse fasi della costruzione;

2. gli spazi di lavoro devono essere suddivisi in varie fasi, ognuna delle quali ha proprietà spazio-temporali uniche per tutta la durata dell'attività e differenti da quelle delle successive fasi;

3. le collisioni devono essere controllate in ogni fase dello spazio di lavoro con altri rispettivi spazi di lavoro o elementi che condividono le stesse proprietà temporali.

Per mettere in relazione gli spazi di lavoro e controllare le collisioni sono necessari dei parametri di input con cui classificare gli spazi, i quali sono di tre tipologie:

1) spaziali:

riguardo *dimensione, forma e posizione*.

Le geometrie dello spazio nel modello sono spesso una scala rispetto quelli dell'elemento edilizio e le forme geometriche degli spazi sono delle scatole rettangolari o dalla forma cilindrica;

2) temporali:

riguardano *tempo* e *produttività*;

3) comportamentali:

riguardano il *tipo* dello spazio di lavoro in base alle risorse da cui è occupato, *la natura* dello spazio che descrive la sua evoluzione nel processo (*statica o dinamica*) e i movimenti.

La natura statica dello spazio implica che occupi uno spazio fisso senza variazioni durante l'intera durata dell'attività, al contrario la natura dinamica implica una posizione e dimensioni variabili nel tempo (possono sia diminuire che aumentare).

Il movimento descrive: la direzione di costruzione dell'elemento (indicando la direzione di evoluzione degli spazi dinamici rispetto alla coordinate degli elementi) e il movimento dell'attività (rappresenta la priorità e le relazioni tra più oggetti coinvolti nella stessa attività).

Esempio: se è positivo rispetto ad un elemento verticale come una colonna, significa che questa verrà processata dal basso verso l'alto.

2.6.3 SIMULATE: CARATTERISTICHE E LIMITI DELLA SIMULAZIONE AD EVENTI DISCRETI

L'output della fase *"Plan"* è un modello spaziale contenente le informazioni richieste per la simulazione.

La maggior parte degli approcci attuali utilizza tecniche di simulazione ad eventi discreti ovvero con variabili che cambiano istantaneamente il loro valore in definiti istanti di tempo.

Gli spazi di lavoro sono suddivisi in parti più piccole in funzione degli input inseriti nella fase precedente.

Questi metodi scompongono gli spazi di lavoro assumendo due ipotesi principali:

i) un tasso di produzione uniforme per tutta la durata dell'attività;

ii) le attività occupano lo spazio di lavoro pianificato (in termini di dimensioni e posizione), senza contemplare l'ipotesi che possano subire variazioni.

Considerando che le attività di costruzione di solito non progrediscono mantenendo la stessa produttività e sono altamente soggette a variazioni, un approccio deterministico che non considera le incertezze esclude dalla simulazione alcuni scenari che hanno possibilità di accadere.

2.6.4 COLLISION DETECTION: METODI DI RILEVAZIONE DELLE COLLISIONI

L'output della fase *"Simulate"* è una disposizione spazio-temporale degli spazi di lavoro per tutta la durata del progetto che può essere esaminata per rilevare le sovrapposizioni. Questa rappresenta lo scenario base con cui avviare l'esecuzione.

I metodi di rilevamento adottati sono i seguenti:

1) Temporal detection:

si identificano le aree con maggiore potenziale di collisione analizzando il pianificato e individuando i periodi con più attività simultanee nella stessa posizione.

2) *Approximation detection:*

si confronta la lunghezza del raggio dello spazio di lavoro con la lunghezza della linea che connette il centro della zona di lavoro di un attività e la zona di lavoro dell'attività che opera in spazi adiacenti.

Nel caso in cui, il raggio risulti insufficiente significa che c'è collisione.

3) *Topographical detection:*
ad ogni spazio di lavoro viene assegnata una matrice spaziale in cui l'operatore binario entry-wise definisce la collisione.
4*) Geometrical intersection tests:*
ogni spazio di lavoro è controllato (attraverso il confronto a coppie) sovrapponendolo a tutti gli altri spazi di lavoro con cui può avere collisioni.

2.6.5 TECNICHE PER VALUTARE L'ENTITÀ DELLE COLLISIONI E LIMITI

L'output della fase *"Detect"* prevede una lista di informazioni sulle collisioni, compresa la dimensione, la durata delle collisioni e il numero di risorse coinvolte in ogni incidente, senza valutarne però la gravità.
Le tecniche attuali per valutare l'entità delle collisioni possono essere classificate in tre livelli principali:
1) site (S):
calcola ad una data prefissata il rapporto tra spazi richiesti e disponibili in quella data. Un valore superiore a uno indica una sovrapposizione.
Questo metodo non indica quali sono le attività che interferiscono.
2) workspace (W):
utilizza due parametri del conflitto: il volume degli spazi sovrapposti *(volume di collisione)* e la durata della sovrapposizione *(durata della collisione).*
I rapporti tra ciascuno di questi valori e le loro controparti pianificate sono utilizzati per valutare l'intensità della collisione.
3) attività (A):
si usano funzioni parametriche per valutare la gravità di una collisione indagando misure relative all'attività come la criticità del pianificato, il numero di spazi di lavoro per attività, il numero di conflitti per attività.
Tutti i parametri utilizzati nei tre livelli non prendono in considerazione lo stato degli spazi di lavoro e la tipologia delle squadre coinvolte nelle collisioni.
Presuppongono quindi che, indipendentemente da quando avviene un'interferenza nello spazio di lavoro o da chi coinvolge, l'impatto rimane lo stesso, il che è un'ipotesi poco

realistica.

Inoltre, la valutazione non presuppone effetti di propagazione che le collisioni potrebbero avere sulle attività future.

2.6.6 LIMITI DEI MODELLI DETERMINISTICI DI GESTIONE DELLO SPAZIO DI LAVORO

I parametri che vengono attualmente utilizzati nei modelli usano grandezze individuali e non intervalli di valori o distribuzioni di provabilità.

Pertanto, la prima lacuna evidente osservata nei modelli deterministici è l'impossibilità di considerare le possibili incertezze che potrebbero verificarsi in fase di esecuzione, che disturberebbero il piano e influenzerebbero negativamente i risultati della pianificazione.

Inoltre, si ipotizza che le collisioni possono generare effetti di propagazione e ci vorrebbe del tempo perché le squadre di lavoro tornino alle loro prestazioni ottimali.

Infine, squadre di lavoro diverse possono reagire in modo diverso, anche alla stessa collisione.

In aggiunta alle precedenti, si possono citare le seguenti lacune dei modelli deterministici:

1) la mancanza di modellazione non deterministica durante la pianificazione dello spazio di lavoro;

2) non cogliere i possibili effetti compositivi derivanti dalle molteplici fonti di incertezza;

3) non considerare gli effetti di propagazione delle collisioni sulle attività future;

4) non considerare l'influenza delle squadre di lavoro sulla strategia di risoluzione.

5) l'assenza di un approccio decisionale per la gestione delle collisioni.

3. APPROCCIO OLONICO PER LA PIANIFICAZIONE E GESTIONE DELLO SPAZIO

3.1 PIANIFICAZIONE GERARCHICA OFF-LINE IN SISTEMI COMPLESSI

3.1.1 LIMITI DELLE ARCHITETTURE GERACHICHE

Nei processi di pianificazione di sistemi complessi come quello delle costruzioni vengono utilizzate architetture di controllo della produzione gerarchiche.

Tuttavia, la pianificazione attraverso architetture gerarchiche ha le seguenti caratteristiche:

1. è generata prima dell'inizio del processo, quindi prima dell'inizio dei lavori;

2. basata sulle prestazioni delle risorse e condizioni operative approssimative e stimate;

3. si basa principilamente su:

- durata delle attività,
- vincoli di precedenza delle attività.

In questa maniera, la soluzione di pianificazione rimane valida per un certo scenario operativo predeterminato e presenta i principali difetti:

A. non è in grado di fornire la flessibilità necessaria in contesti soggetti a incertezze e variazioni come quello delle costruzioni.

Nel processo realizzativo delle costruzioni possono variare le prestazioni di una risorsa o le condizioni operative al contorno più volte nello stesso processo realizzativo.

B. non hanno la capacità di determinare se una decisione alternativa è migliore di un'altra.

Ricordiamo che in un sistema complesso, le stesse condizioni di partenza possono produrre risultati diversi, a seconda della sequenza delle interazioni tra i componenti del sistema.

In sistemi complessi è evidente che i progettisti non possono imporre vincoli arbitrari o almeno devono rendere facile la loro revisione.

Dunque, le variazioni registrate in corso d'opera, come ad esempio la variazione della prestazione di una risorsa, spesso rendono la pianificazione iniziale inutilizzabile nella pratica.

L'incremento della produttività è possibile attraverso la trasparenza in runtime delle variazioni, ovvero elaborare la pianificazione in funzione dei dati raccolti durante il processo.
Aggiornando la pianificazione con lo stato reale del progetto permette anche di azzerrare con ad esempio frequenze settimanale il gap che si genera fra pianificato e realtà.
Con più informazioni in runtime, diventa più facile identificare le fonti di problemi o opportunità e prendere decisioni efficaci.
A più informazioni disponibili deve essere associato uno strumento capace di elaborarle ad uno scopo preciso volto all'ottimizzazione del processo.

3.2 HOLONIC MANUFACTURING SYSTEMS (HMS)

3.2.1 GESTIONE DECENTRALIZZATA

E' ormai consolidato che in ambienti complessi, le possibilità di gestire le variazioni, quindi prevederle e risolverle, è direttamente proporzionale alla presenza di:

- sottosistemi autonomi;
- sottoinsiemi composti da *elementi base* stabili;
- *elementi base* autosufficienti rispetto le parti subordinate;
- *elementi base* capace di risolvere le contingenze senza chiedere istruzioni alle autorità superiori;

In questa maniera, il concetto di autonomia permette agli *elementi base* di reagire ai cambiamenti, all'incertezza e alle variazioni dell'ambiente.
La coordinazione e il controllo non centralizzato devono definire la struttura dell'ambiente e non i meccanismi decisionali che determinano le prestazioni e i loro metodi di regolazione.

3.2.2 CONCETTO DI OLONE

Il concetto olonico ha come obbiettivo quello di gestire il comportamento dei sistemi complessi considerando le entità che lo compongono come se fossero sia interi che parti allo stesso tempo.
L'unità base dell'organizzazione è chiamata *"olone"*.
Un sistema olonico o olarchia è una gerarchia di oloni autoregolanti che funzionano:
A. come insiemi autonomi in sovraordinazione delle loro parti;
B. come parti dipendenti in subordinazione al controllo ai livelli superiori;
C. in coordinazione con il loro ambiente locale.
L'olone:

- è un elemento autonomo e cooperativo che trasforma, trasporta, immagazzina o convalida informazioni e oggetti fisici;
- è indipendente perche gestisce le contingenze senza chiedere istruzioni alle autorità superiori.
- allo stesso tempo, èsoggetto al controllo delle autorità superiori.

3.2.3 ARCHITETTURA DI SISTEMA PROSA E TIPOLOGIE DI OLONE

L' architettura di un sistema è un insieme di regole e linee guida per gestire lo sviluppo e il funzionamento di sistemi complessi.

Un'architettura olonica che permette un buon compromesso tra gerarchia ed eterarchia si chiama PROSA *(Product Resource Order Staff Approach)*.

Si basa sul concetto di olone e l'architettura ne descrive le responsabilità e le loro interazioni.

Di seguito vengono elencati i vari tipi di olone:

A.

Product Holons (PH):

L'olone del prodotto ha le seguenti responsabilità:

A.1 Conoscere come le operazioni devono essere eseguite dalle risorse, cioè, quali sono quelle necessarie per realizzare il compito correttamente e qualitativamente;

A.2 Contenere le informazioni sui vincoli che legano la sequenza di operazioni;

A.3 Dopo il completamento di un'operazione, l'olone dell'ordine ha bisogno di informazioni sulla sua prossima operazione. Quindi è suo compito informare su tutte le possibilità per la prossima operazione.

B.

Resource Holons (RH):

L'olone di risorsa ha le seguenti responsabilità:

B.1 Contenere e fornire informazioni sullo stato attuale della risorsa e gli stati futuri previsti;

B.2 Tenere traccia della disponibilità della risorsa nel tempo, con particolare attenzione alla disponibilità futura;

B.3 Avere autorità locale su come organizzare la sequenza di pianificazione delle varie operazioni richieste dall'olone ordine.

C.

Order Holons (OH):

Ogni olone dell'ordine è strettamente legato all'olone del prodotto che rappresenta il

compito corrispondente.

La sua principale responsabilità è:

C.1 Contenere le informazioni aggiornate fra stato attuale e lo stato reale dell'ordine come la posizione dell'ordine, l'operazione in corso di elaborazione, la risorsa che esegue questa operazione, ecc.

D. Staff Holons (SH):

Sono consulenti chiamati in causa da oloni OH o RH per risolvere uno specifico problema per prendere una decisione, applicando regole generali fisse del sistema, o una variante derivante dall'applicazione delle strategie locali.

3.2.4 INTERAZIONE FRA OLONI

Solo due tipi di oloni prendono decisioni in un HMS: gli oloni di ordine (OH) e di risorse (RH).

Il tipo di decisioni che prendono sono diverse.

Gli RH prendono decisioni locali, mentre gli OH affrontano obiettivi globali del sistema.

Di seguito è rappresentato uno schema esemplificativo di come i vari oloni relazionano fra loro:

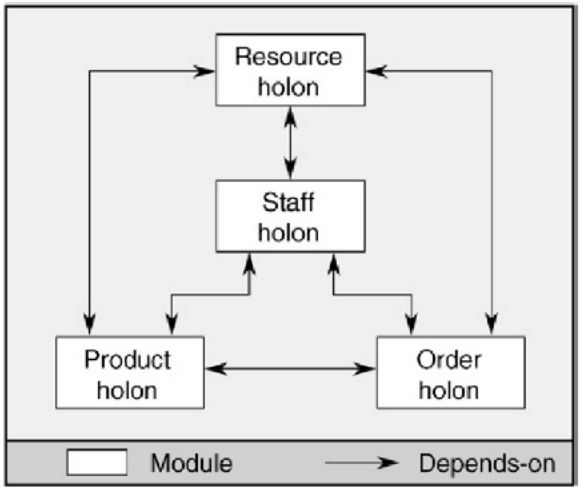

Figura 7 Ogni rettangolo rappresenta un tipo di olole nella architettura PROSA, le frecce rappresentano le interazioni che hanno fra loro

I principali tipi di interazione sono:

1.

Interazione prodotto-ordine:

Gli oloni dell'ordine interagiscono con il loro corrispondente prodotto accordando come eseguire correttamente il loro compito usando determinate risorse.

2.

Interazione prodotto-risorsa:

Gli oloni di prodotto e di risorsa condividono informazioni relative al processo.

Quando si genera una lista di possibili operazioni per un olone di ordine, l'olone di prodotto interagisce con gli oloni di risorsa per sapere quali operazioni possono eseguire le risorse.

Al contrario, l'olone prodotto fornisce all'olone risorsa gli aspetti tecnologici per elaborare correttamente un ordine.

3.

Interazione risorsa-ordine:

Gli oloni delle risorse forniscono agli oloni dell'ordine i risultati delle operazioni virtualmente eseguite.

3.3 PATTERN ARCHITETTONICO DMAS

3.3.1 CARATTERISTICHE DEL PATTERN ARCHITETTONICO DMAS

Essendo l'astrazione olonica un problema di natura distribuita, rende la teoria DMAS appropriata per l'implementazione di HMS.

Il DMAS è un pattern architettonico che permette ad un olone di delegare una responsabilità ad uno sciame di oloni leggeri.

Secondo DMAS:

A) gli oloni possono delegare simultaneamente più responsabilità;

B) l'olone emittente può usare anche più oloni leggeri per una singola responsabilità.

C) gli oloni leggeri eseguono attività per supportare l'olone emittente nello svolgimento delle sue funzioni;

Il pattern DMAS prevede la presenza di tre moduli, come mostrato in figura seguente: l'ambiente, l'agente e la formica.

Gli oloni leggeri sono chiamati anche agenti formica.

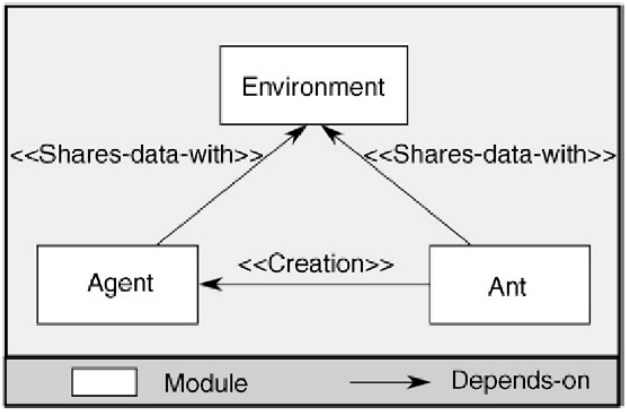

Figura 8 I moduli che compongono DMAS sono tre: Formica,Agente,Ambiente.

Le relazioni di dipendenza tra i moduli sono di due tipi:

i) l'agente (olone emittente) e il modulo formica condividono dati con il modulo ambiente,

ii) la relazione tra l'agente e il modulo formica è una relazione di "creazione".

L'ambiente è una rappresentazione software del mondo di interesse, nel nostro caso della costruzione.

Il modulo formica si muove autonomamente attraverso l'ambiente partendo da un luogo

selezionato dal loro olone emittente.
Un olone che delega una responsabilità a uno sciame di formiche:
- è responsabile del mantenimento della dimensione e della diversità della popolazione;
- sceglie la frequenza di creazione delle formiche.

3.3.2 RUOLO DELL'AGENTE FORMICA ESPLORANTE

Nel sistema di pianificazione conforme al pattern DMAS, l'agente ordine usa agenti formica esploranti per esplorare possibili soluzioni al fine di confrontarne gli esiti.
Per compiere il suo compito, l'agente formica esplorante, in ordine successivo:
1) interroga il suo corrispondente agente prodotto che valuta le informazioni di stato e definisce una lista di operazioni successive fattibili;
2) definisce una risorsa adatta per l'esecuzione dell'operazione, in collaborazione con l'agente prodotto;
3) chiede all'agente risorsa i risultati dell'esecuzione virtuale, cioè il tempo e lo stato finale dopo l'operazione;
4) registra queste informazioni e consulta di nuovo l'agente prodotto per conoscere la prossima operazione richiesta;
5) alla fine dell'attività di esplorazione, quando l'ultima operazione viene eseguita, l'agente formica esploratrice compila i risultati di una possibile soluzione e la riporta all'agente dell'ordine.

Si noti che durante il processo, l'agente d'ordine costruisce possibili soluzioni inviando gli agenti formica esploratori, i quali indagano le prestazioni previste di uno dei molti modi possibili di completare il processo.
L'informazione sulle prestazioni include tipicamente il tempo complessivo di elaborazione e i criteri di qualità dell'ordine, ma può anche includere altri fattori come l'utilizzo delle risorse.
Quando l'agente d'ordine ha raccolto un certo numero di soluzioni, le valuta secondo certi criteri di performance definiti dall'utente.
Poi, seleziona la soluzione più performante che diventa l'intenzione dell'agente.

3.4 SIMULAZIONI ONLINE AD EVENTI DISCRETI IN AMBIENTI OLONICI

3.4.1 VANTAGGI NELL'UTILIZZO DI SIMULAZIONI ONLINE AD EVENTI DISCRETI

In ambienti tradizionali può verificarsi un problema di negoziazione fra più oloni e l'unico parametro decisionale senza strumenti capaci di comprendere più scelte può essere il vincolo di termine ultimo di completamento dell'attività.

L'uso di simulazioni ad eventi discreti considera più eventi possibili e decide in funzione di più parametri decisionali quale sia la scelta migliore.

Per questo la simulazione online costituisce uno strumento di previsione capace di prendere in considerazione i risultati effettivi di più processi decisionali.

Le principali caratteristiche di una simulazione online sono:

- il progetto su cui si basa la simulazione, quindi "l'ambiente" secondo DMAS, viene aggiornato allo stato reale di avanzamento, permettendo di azzerare in ogni simulazione lo scarto fra il pianificato e il realizzato;
- avere un orizzonte di simulazione generalmente breve (dell'ordine di qualche giorno). In questa maniera, non è necessario modellare eventi come guasti, incidenti ecc. poiché il loro verificarsi è troppo raro nell'orizzonte.

Queste caratteristiche migliorano la visibilità a livello di esecuzione permettendo di adattare il pianificato in funzione degli eventi imprevisti.

4. ELABORAZIONE DI UN NUOVO WORKFLOW NEL CASO DI LAVORI DI RIQUALIFICAZIONE

4.1 STRUTTURA DEI SISTEMI DI GESTIONE DELLA COMUNICAZIONE

4.1.1 PRINCIPI DELLO SCAMBIO DI INFORMAZIONI: STANDART O-MI

I "flussi delle risorse" e i " task" sono da considerare parallelamente perché la realizzazione dei secondi dipende fortemente dai primi e i secondi a loro volta dipendono dai primi, la realizzazione dei task.

Secondo [8], ai fini di una fluida comunicazione possono essere adottati gli standard IoT pubblicati dall'Open Group IoT Work Group, cioè l'O-MI (Open- Messaging Interface).

O-MI fornisce interfacce per scambiare informazioni sul processo di costruzione tra una vasta gamma di nodi O-MI, indipendentemente dalle caratteristiche del sistema o dell'applicazione.

Tre tipi di comunicazione sono definiti nelle specifiche O-MI:

i) Write

usato per inviare informazioni di aggiornamento ai nodi O-MI;

ii) Read

usato per recuperare informazioni dai nodi O-MI;

Il meccanismo di sottoscrizione dell'operazione può essere di due tipi:

- *Sottoscrizione con indirizzo di callback:*

i dati sottoscritti sono inviati all'indirizzo di callback dell'individuo richiesto.

Possono essere a loro volta di due tipi: basati su intervalli di tempo fissi o su eventi;

- *Sottoscrizione senza indirizzo di callback:*

i dati sono memorizzati sul nodo O-MI. L'informazione memorizzata può essere recuperata (cioè, polled) emettendo una query di tipo O-MI, ovvero comprensibile al sistema.

iii) Cancel

utilizzato per cancellare le sottoscrizioni prima della loro scadenza.

Lo stesso documento propone una struttura in cui sviluppare lo scambio di informazioni, questa è chiamata VisiLean.

VisiLean è costruito sui seguenti principi:

- *Controllo del flusso di lavoro e pianificazione della produzione:*

i metodi consolidati di lean construction sono presenti in VisiLean.

Come il metodo LPS che supporta cicli di pianificazione a lungo, medio e breve termine, compresa la gestione dei vincoli;

- *Integrazione di processo e del prodotto:*

fornisce la visualizzazione del processo di produzione, il modello BIM e una mappatura dell'attività all'elemento BIM corrispondente.

Tale visualizzazione simultanea migliora l'affidabilità della pianificazione in quanto il metodo LPS ha accesso alle informazioni più complete e aggiornate sulla produzione in un'unica interfaccia durante la pianificazione e l'esecuzione;

- *Controlli visivi e informazioni sulla produzione:*

supporta i sistemi di produzione *"pull",* fornendo direttamente l'interfaccia del flusso di lavoro in modo visivo.

La Figura seguente mostra i tre componenti principali che sono integrati in VisiLean:

(i) le *"App"* di gestione della produzione sul campo;

(ii) il sistema di gestione della produzione principale;

(iii) altri sistemi esterni.

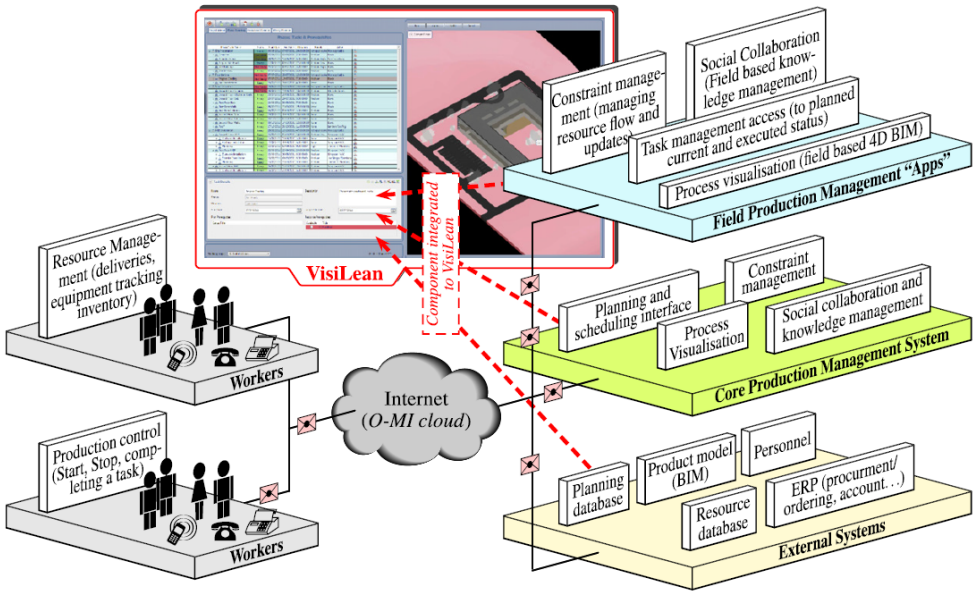

Figura 9 Diagramma schematico della gestione avanzata del Lean Construction Management

Lo standard O-MI è usato per abilitare le comunicazioni tra il sistema VisiLean con i sistemi:

(iv) di gestione delle risorse;

(v) controllo della produzione.

Inoltre Visilean, supporta importanti caratteristiche di comunicazione come i meccanismi basati su *"push"* e *"pull"* che hanno un impatto significativo sul controllo della produzione.

I sistemi *"push"* programmano il rilascio del lavoro, mentre i sistemi *"pull"* autorizzano il rilascio del lavoro sulla base dello stato del sistema.

4.1.2 STRUTTURA DEL SISTEMA KANBIM PER GESTIRE L'INFORMAZIONE

Il pensiero Lean applicato alle costruzioni ha portato allo sviluppo di sistemi di pianificazione e controllo capaci di migliorare le prestazioni del flusso di produzione.

La concettualizzazione di Koskela *"Transformation-Flow-Value" (TFV)* fornisce una base teorica per comprendere gli aspetti di flusso combinati al valore che generano le trasformazioni.

Le simulazioni ad eventi discreti mostrano chiaramente l'impatto negativo delle variazioni dei tassi di produzione e dei benefici legati a flussi di informazione consolidati.

In zone di lavoro, dove si muovono le squadre e non i prodotti, è molto difficile visualizzare il flusso del lavoro in corso e comunicare il suo stato alle squadre e agli individui coinvolti.

La quantità di lavoro in corso accumulato tra squadre differenti non può essere misurato ad occhio nudo nello stesso modo in cui i prodotti possono essere conteggiati tra le stazioni di lavorazione in un impianto di produzione.

A tal proposito, i sistemi di gestione dei processi di costruzione dovrebbero essere basati su piattaforme BIM conformi a flussi di processo *"pull"* con avvisi tramite tabelloni Andon.

Questo sistema, è in fase di sviluppo e viene definito *'KanBIM'*.

Si tratta di un sistema software che integra il metodo LPS fornendo visualizzazioni 3D

dello stato del progetto e permette l'aggiornamento in tempo reale dello stato del processo. In questa maniera è possibile permettere l'accesso al flusso di informazioni a più persone contemporaneamente.
Questo incrementa l'affidabilità del pianificato riducendo la variabilità.

L'obbiettivo del sistema KanBim è quello di inglobare nei concetti di modellazione 4D, oltre al modello della costruzione, un sistema di supporto decisionale volto a definire piani decisionali affidabili ed evitare l'avvio di attività senza vincoli ancora non processati. Il sistema include la rappresentazione grafica dei vincoli, come le consegne di materiale, lavorazioni in corso e vincoli non liberati, attraverso simboli di vario colore nel modello dell'edificio.
Ogni simbolo ha un compito di visualizzazione ed in base al colore assume diversi signficati.
(Esempio: un semaforo, con i tre differenti colori rosso giallo verde, può rappresentare lo stato di attività all'interno di una zona di cantiere)
Utilizzando dei simboli per rappresentare lo stato dei vincoli, rende la comprensione del flusso di produzione allo stato reale semplice ed immediata.
Essendo i simboli dispigeati in un modello tridimensionale della costruzione risulta possibile definire quali zone del cantiere presentano difformità rispetto il piano dei lavori. Il vantaggio è quindi quello di rendere visibile la progressione della produzione visualizzando gli stati attuali e futuri dell'edificio o dell'impianto.
Tuttavia non vengono mostrate esplicitamente le posizioni delle squadre di lavoro o il lavoro in corso.
Di seguito viene mostrato un esempio della visualizzazione tramite simboli e colori dello stato di avanzamento del progetto.
Nel caso di esempio, lo stato delle attività viene regolato da
- un semaforo che può essumere tre differenti colori,
- un segnale che indica lavori in corso,
- un segnale che indica che il lavoro è stato approvato o rifiutato o in pausa.

Tutti i simboli sono integrati nel modello della costruzione, così da avere una visione chiara dello stato di avanzamento associato a determinate zone del cantiere.

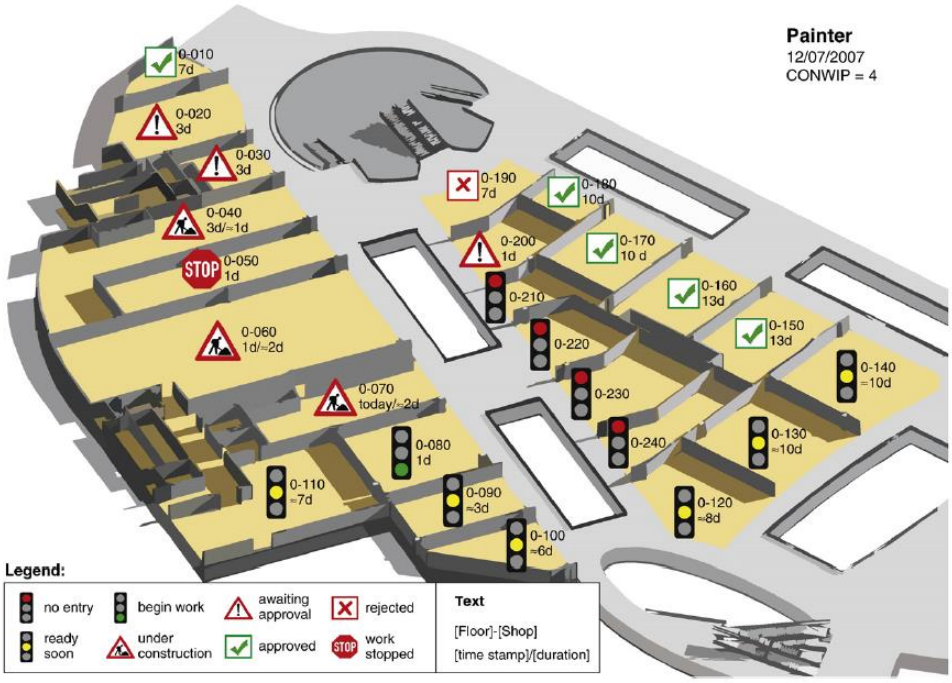

Figura 10 L'obbiettivo è quello di integrare dei simboli che indicano l'avanzamento con un modello della costruzione

Una soluzione possibile per mantere il sistema aggiornato è quella di usare la tracciabilità RFID e i rapporti del personale di cantiere.

Per evitare che il presentarsi di variazioni in singole operazioni possano propagarsi in altre attività ed aumentare la propria entità, in fase di pianificazione settimanale è necessario:

a) essere continuamente informati sullo stato stato delle operazioni.

Quindi il flusso delle operazioni deve essere il piu possibile continuo:

b) avere la capacità di cambiare proattivamente le assegnazioni dei compiti giornalieri in stretta coordinazione con tutte le parti che possono essere interessate dal cambiamento.

C'è quindi la necessità di visualizzare lo stato del processo in tempo reale.

La sfida è quella di rendere l'informazione relativa al prodotto ed il processo onnipresente sul posto di lavoro senza ingombrare i tecnici di attrezzature come dispositivi indossabili o cellulari che possono ostacolare il loro comfort, la sicurezza o la produttività.

Inoltre, al fine di ottimizzare il processo di pianificazione viene introdotto il concetto di *"maturità"* di un'attività, che misura lo stato di preparazione di un pacchetto di lavoro. In funzione dello stato di *"maturità"* è possibile decidere se un pacchetto di lavoro è pronto per poter essere avviato.

L'obbiettivo è quello di evitare di avviare attività che poi devono essere interrotte a causa di vincoli precedenti non processati.

Questo include informazioni riguardo:

- le attività precedenti,
- i materiali,
- le attrezzature,
- la squadra di lavoro,
- lo spazio di lavoro,
- le condizioni esterne.

E'anche possibile aggiungere alla lista i requisiti di sicurezza.

L'*"indice di maturità"* (MI) può essere calcolato se ciascuno dei vincoli elencati sopra, può essere misurato.

L'indice di maturità vuole quindi supportare le decisioni a breve termine durante il processo di pianificazione delle attività settimanali.

Per la pianificazione, siamo interessati al valore futuro previsto della maturità, il che significa che il calcolo deve considerare:

a) lo stato attuale delle precondizioni del compito,

b) lo stato previsto nel momento in cui viene considerato per l'esecuzione del compito,

d*)* l'affidabilità delle predizioni dei valori.

Quando un compito pianificato è definito *"immaturo",* deve essere possibile negoziare un cambiamento di piano con tutti coloro che possono essere interessati.

Per sostenere questo grado di agilità è necessario un robusto sistema informativo che consenta di prendere decisioni.

Per fare questo, il sistema deve:

1. integrare strettamente la pianificazione e il controllo della produzione.

Il livello di dettaglio della pianificazione settimanale deve essere appropriato per il controllo giornaliero della produzione;

2. permettere un feedback dalla postazione di lavoro per assicurare che le informazioni

sullo stato del processo siano aggiornate;

3. fornire un canale di comunicazione per la comunicazione dei cambiamenti di attività pianificate.

Ridurre la finestra di pianificazione a livello giornaliero permettendo di identificare e risolvere i conflitti fra le parti interessate con maggiore agilità.

A maggiore livello di informazione, inteso come mole di dati scambiati e accuratezza, corrisponde agilità di cambiamento.

Allo stesso tempo è necessario che l'indice di maturità di ogni attività sia visualizzabile dal Construction Planner.

Di seguito è mostrato un esempio:

l'indice di maturità è visualizzato ai Construction Planner usando simboli codificati a colori sulle icone delle attività, come mostrato nella figura:

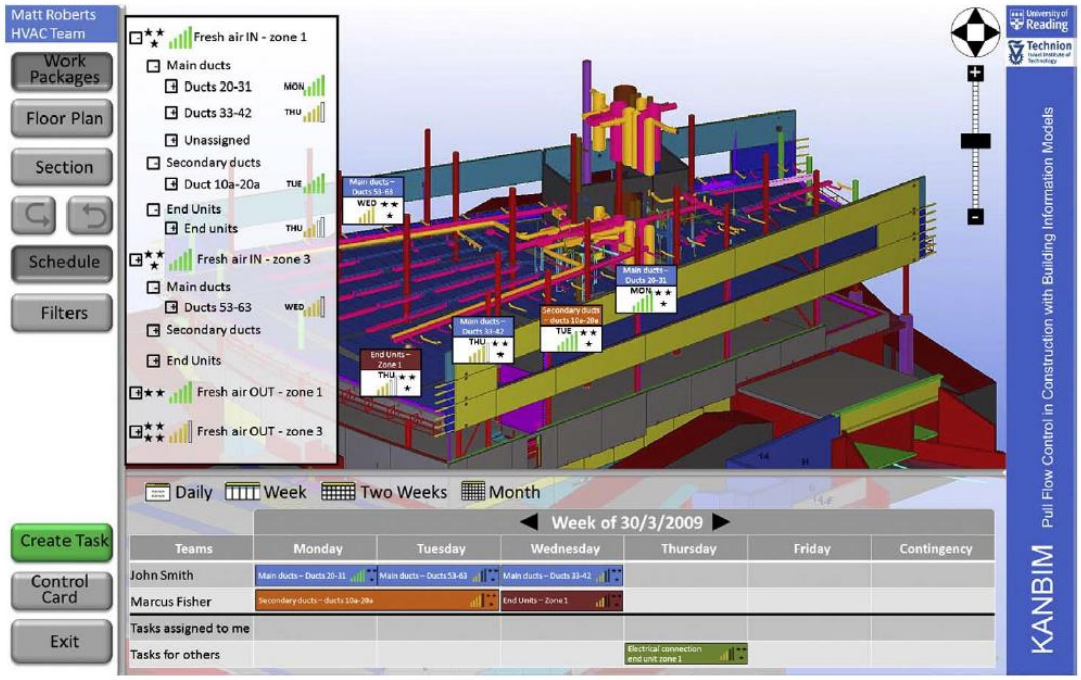

Figura 11 Interfaccia utente che considera gli stati di maturità in un processo di costruzione

Nella stessa schermata vengono riportate:

- le attività,
- il capo squadra che guida l'esecuzione dell'attività,
- l'indice di maturità che indica se un pacchetto di lavori è pronto ad essere avviato.

Perciò, un sistema KanBIM deve implementare due livelli di dettaglio:

a) Pianificazione settimanale del lavoro.

Le richieste iniziali sono comunicate alle squadre sotto forma di pacchetti di lavoro.

b) Esecuzione giornaliera del lavoro.

I capi squadra devono essere in grado di selezionare un compito per l'esecuzione e completarlo nei termini pattuiti.

Nei casi in cui un compito non può essere completato, il sistema richiede una dichiarazione esplicita che il lavoro viene fermato e un dettaglio delle ragioni dell'arresto da parte del capo squadra.

4.1.3 STABILITÀ DEL PIANO DEI LAVORI

Ottenere stabilità in un flusso di lavoro è la caratteristica base per minimizzare lo spreco di tempo nel processo di costruzione.

Nel metodo LPS, la misura della *"percentuale di completamento del piano"* è usata per conoscere l'affidabilità del piano dei lavori e quindi migliorare la stabilità del pianificato, ma come spiegato sopra risulta una misura insufficiente.

La necessità di stabilità del piano implica alcune linee guida per la gestione della produzione, implementate secondo il sistema KanBIM.

Le modifiche improvvise di determinati compiti non devono essere permessi durante la settimana; un arresto di una attività dovuto all'indisponibilità di materiale o a un errore nella sua fabbricazione dovrebbe essere rivelato prima ancora che l'attività abbia inizio.

Anche quando i compiti giornalieri pianificati sono stati completati, l'inizio dei compiti successivi dovrebbe essere evitato, se non prima di aver avvisato tutti gli agenti che interessano le attività.

Non è detto che l'avvio di un attività in anticipo generi necessariamente benefici.

La sua variazione potrebbe, implicitamente, coinvolgere la modifica di una serie di vincoli il cui valore assoluto rallenta il processo totale.

Il sistema KanBIM sfrutta le informazioni sul processo produttivo disponibili nei modelli informativi dell'edificio per supportare il coordinamento operativo settimanale e giornaliero.

A tal proposito risulta necessario un sistema solido e standardizzato di scambio dell'informazione.
A scambi di informazione strutturata, corrisponde semplicità nel definire quali attività hanno raggiunto il grado di maturità necessario per essere avviate, quindi terminate senza variazioni.

4.2 WORKFLOW NEL CASO DI LAVORI DI RIQUALIFICAZIONE

4.2.1 STRUTTURA DEL WORKFLOW NEL CASO DI LAVORO DI RIQUALIFICAZIONE

Il processo di produzione delle costruzioni prevede uno scambio di informazioni ordinato fra più individui, al fine di ottimizzare il flusso.
La struttura di un worflow del processo di produzione varia in funzione della finalità dei lavori o di esigenze specifiche.
Il caso studio proposto nei successivi capitoli, interessa un processo di riqualificazione di una costruzione.
In aggiunta, prevede l'utilizzo di uno strumento di simulazione degli spazi di lavoro.
Lo scopo del successivo worflow è quello di chiarire come lo strumento di simulazione del caso studio può essere integrato all'interno di un flusso di produzione in contesto di riqualificazione.
A tal proposito viene definito il flusso dello scambio di informazioni che supporta la produzione in caso di lavori di riqualificazione.

Il workflow include tre differenti "Lane".
Ogni "Lane" corrisponde ad una divisione orizzontale.
Ogni divisione orizzontale rappresenta un ambiente/responsabile differente.
Lane A: Simulation, (Task color: azzurro);
Lane B: AWOPS, (Task color: viola);
Lane C: Manual task, (Task color: rosso).
I Task inseriti nella *Lane C* sono Task manuali che vengono completati da utenti.
In questo caso, dal Constrcution Planner o dal Zone Managers o da entrambi.
I Task nelle *Lane A e B* sono Task automatizzati che vengono compiuti in automatico dalle rispettive piattaforme, o AWOPS o l'ambiente di simulazione:
I principali simboli da conoscere per comprendere il workflow sono:
1. Alcuni processi richiedono la comunicazione, che può essere rappresentata tramite l'evento messaggio.
Il significato di *"Messaggio"* non è limitato a e-mail o chiamate.

Qualsiasi azione che si riferisce a un destinatario specifico e rappresenta o contiene informazioni per il destinatario è un messaggio.

Figura 12 Un elemento con messaggio rappresenta un elemento contenente informazioni

2. Alcuni Task possono essere processati in parallelo.
I task collegati al simbolo *"Parallelo"* sono processati contemporaneamente.

Figura 13 Segno di parallelismo

3. Il flusso di lavoro può incontrare dei bivi. In funzione del contenuto del *"Messaggio"* possono essere eseguite differenti azioni.
Tutto ciò viene rappresentato con il simbolo *"Exclusive"*.

Figura 14 Il simbolo Exclusive indica quale direzione scegliere nel workflow

Il workflow nel suo complesso si articola in 11 differenti Task.

- TASK 1: Prepare Master Plan
Lane B:AWOPS;
Creazione di un Master Plan.
In questa fase sono determinate più attività di alto livello e pacchetti di lavoro.
Vengono pianificati includendo le assegnazioni di responsabilità e i buffering minimi.

- TASK 2: Prepare Look Ahed Plan
Lane C: Manual task.
Responsabili: Construction Planner, Zone Managers;

Definizione della pianificazione futura dividendo i pacchetti di lavoro in più sotto-attività.
I pacchetti di lavoro, in questa fase, comprendono meno attività e diventano più facili da gestire.
Vengono definiti i vincoli logistici fra le attività sotto forma di connessioni *(ad esempio: fine-inizio,inizio-inizio ecc.)* e l'assegnazione di attrezzature e materiali.
Il Task 2 ha come output un elemento *"messaggio"*.
Questo definisce il Look Ahed Plan.

-TASK 3: Edit workspace
Lane C: Manual task.
Responsabili: Construction Planner, Zone Managers;
In questa fase vengono definite le dimensioni degli spazi necessari alla lavorazione per ogni compito.
Questo volume sarà quello preso in considerazione durante la simulazione.
Lo spazio deve comprendere oltre l'elemento lavorato, quello necessario alla movimentazione, allo stoccaggio dei materiali ed eventuali opere provvisorie.
Il Task 3 ha come output un elemento *"messaggio"*.
Questo definisce le dimensioni degli spazi di lavoro.

-TASK 4: Compile and Detail Tasks
Lane B: AWOPS;
Vengono uniti i piani di lavoro di più squadre in un cronoprogramma unico ed associati gli spazi necessari alla lavorazione richiesti.
Dunque, vengono consolidati definitivamente i compiti da eseguire nella settimana e le rispettive squadre di lavoro responsabili.
Il Task 4 ha come output un elemento *"messaggio"*.
Questo comprende il piano dei lavori che sarà l'input della simulazione.

-TASK 5:Simulation
Lane A: Simulation;
In questa fase avviene la simulazione del pianificato.

Il modello digitale tridimensionale della costruzione è caricato nell'ambiente di simulazione.
Nello specifico, la simulazione è in grado di costruire automaticamente gli spazi di lavoro associati alle attività analizzando il piano dei lavori.
Il Task 5 ha come output un elemento *"messaggio"*.
Questo definisce la lista delle collisioni.

Se la la lista delle collisioni:
- non ricoscontra sovrapposizioni degli spazi di lavoro, quindi è vuota, si passa alla fase successiva *(Task 9: Start Task, Task 10: Manage Logistic);*

- riscontra sovrapposizioni, quindi non è vuota, allora si passa al:

TASK 6: Solve collision
Lane B: AWOPS;
La piattaforma valuta eventuali possibilità automatiche di risoluzione delle collisioni riformulando la successione dei compiti che compongono l'attività.
Se:
- i conflitti sono stati risolti si passa al *Task 8: Update activity plan.*
- i conflitti non possono essere risolti automaticamente allora si procede al:

TASK 7: Negotiate proposed plan
Lane C: Manual task.
Responsabili: Construction Planner, Zone Managers;
In questa fase vengono convocati solo gli individui interessati alla collisione.
Viene deciso come risolvere la collisione, modificando la successione dei compiti o gli spazi richiesti.
Il Task 7 ha come output un elemento *"messaggio"*.
Questo definisce le decisioni prese dalle squadre per risolvere il conflitto. Sarà l'input della piattaforma AWOPS per aggiornare il cronoprogramma.

TASK 8: Update activity plan;

Lane B:AWOPS;

In funzione della risoluzione della collisione adottata, il piano viene aggiornato.

Il Task 8 ha come output un elemento *"messaggio"*.

Questo è il piano aggiornato che permette di iterare il ciclo di simulazione-risoluzione fino a quando non si genera un output vuoto della simulazione.

TASK 9: Start daily Task

Lane C: Manual task.

Responsabili: Construction Planner, Zone Managers;

Il task viene avviato perché nel piano dei lavori non è presente nessuna sovrapposizione degli spazi di lavoro.

TASK 10: Manage Logistic

Lane C: Manual task.

Responsabili: Construction Planner, Zone Managers;

Durante il processo di lavorazione, in parallelo procede il Task *"Manage Logistic"*.

In questa fase, il capo squadra, raccoglie e fornisce alla piattaforma lo stato di avanzamento del task.

In altri termini aggiorna lo stato attuale del processo inserendo informazioni nella piattaforma centrale.

Questo processo assicura un flusso di informazioni continuo e preciso, perché gestito direttamente dal luogo di costruzione .

Allo stesso tempo, il *Construction Planner* ha una visione in tempo reale del flusso di produzione.

Di seguito è rappresentato un esempio della schermata visibile al capo squadra della piattaforma centrale proposta in KanBim.

Ha sia la possibilità di aggiungere informazioni riguardo la sua attività che visionare lo stato delle altre:

Figura 15 Più informazioni definiscono lo stato reale di avanzamento del flusso di produzione

Il Task 10 ha come output un elemento *"messaggio"*.

Questo definisce lo stato di avanzamento dei lavori, necessario per aggiornare il cronoprogramma.

TASK 11: Progress Update

Lane B: AWOPS;

Il piano dei lavori viene aggiornato allo stato di avanzamento dei lavori.

Il Task 11 ha come output un elemento *"messaggio"*.

Questo è il piano dei lavori aggiornato allo stato di avanzamento dei lavori.

Se:

c) i Task sono tutti completati, il processo è concluso.

d) i Task non sono tutti completati, si torna al Task 5 di simulazione.

Di seguito è rappresentato il workflow per intero:

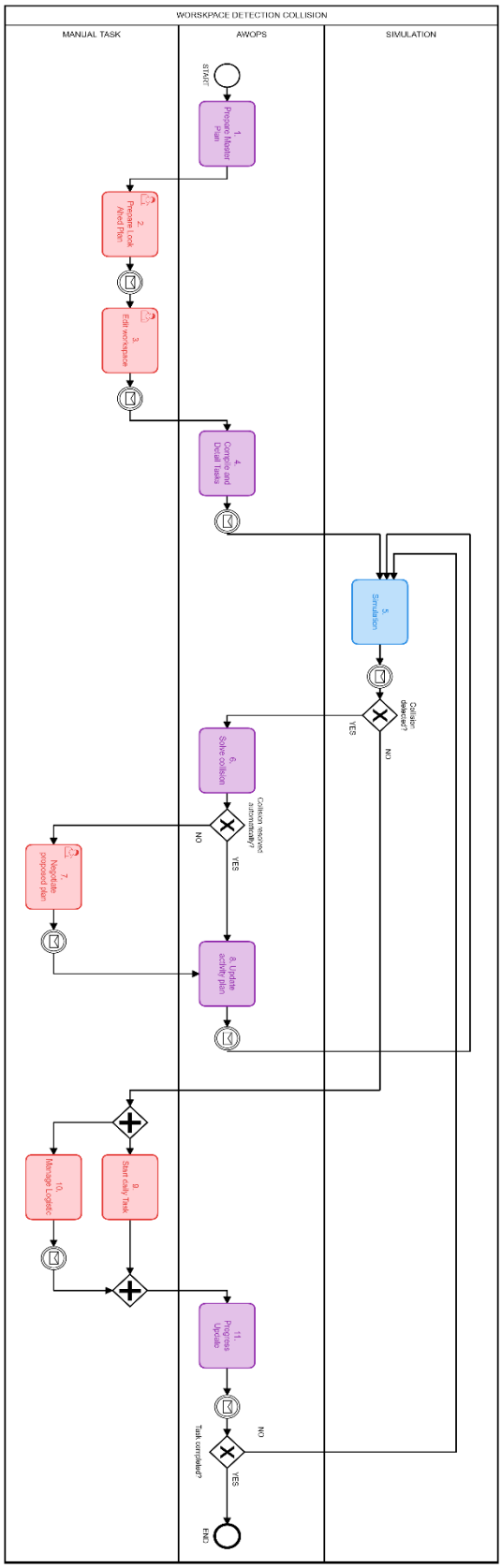

Figura 16 Il worflow si compone di 11 task, 3 line e dimostra come utilizzare lo strumento di simulazione in un processo produttivo

5. IL CASO STUDIO: SVILUPPO DI UN SISTEMA OLONICO PER LA GESTIONE DEI LAVORI IN TEMPO REALE

5.1 STRUTTURA CASO STUDIO

5.1.1 MAPPATURA FRA CASO STUDIO E ARCHITTETTURA PROSA E DMAS

Il sistema di simulazione on-line del caso studio è elaborato secondo l'architettura di riferimento PROSA utilizzando il pattern architettonico DMAS.

Prevede un ambiente nel quale viene eseguita la simulazione ed agenti che interagiscono con lo stesso.

L'ambiente è una riproduzione digitale della realtà, ad esempio il modello as-built del costruito.

Prevede quindi l'esistenza di un modello geometrico entro cui comporre gli spazi di lavoro.

Questo permette al sistema di interagire con i modelli BIM che costituiscono l'output della maggiorparte dei processi di progettazione.

Nell'ambiente vengono composti gli spazi di lavoro associati ad ogni attività.

L'output della simulazione è una lista di collissioni di due o più spazi di lavoro.

Per questo, rappresenta uno strumento di gestione che ha lo scopo di migliorare la qualità delle scelte di pianificazione.

Il caso studio prevede tre macro-fasi:

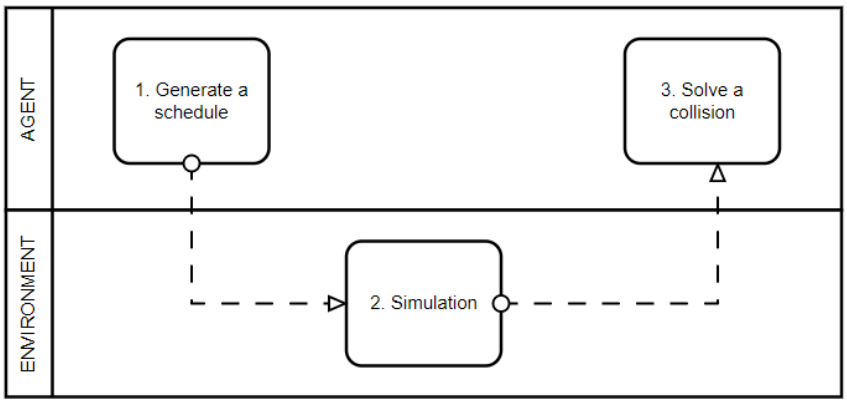

Figura 17 Il caso studio prevede tre task principali: la generazione di un cronoprogramma, la simulazione, la risoluzione di eventuali collisioni

FASE 1:

GENERATE A SCHEDULE

L'agente, ad esempio il Contructor Planner, genera un piano dei lavori oppure unisce quelli di più squadre.

Il piano dei lavori costituisce l'input della simulazione assieme al modello digitale geometrico della costruzione.

Come vedremo successivamente, nel caso studio il modello digitale dell'edificio è in formato glTF ed associato ad un file JSON che ne descrive i componenti.

FASE 2:

SIMULATION

Il sistema importa il piano dei lavori e il modello digitale.

Il piano dei lavori viene deserializzato ed utilizzato per comprendere quali attività, quindi spazio di lavoro, sono processate nell'intervallo di tempo della simulazione definita dall'utente.

A questo punto, all'interno della simulazione è presente il modello della costruzione e gli spazi di lavoro tridimensionali delle attività che vengono eseguite nella finestra temporale della simulazione.

La simulazione viene avviata e si verifica nell'ambiente il cronoprogramma.

Ovvero viene riprodotta tridimensionalmente la risorsa spazio necessaria alla lavorazione di ogni attività segnata nel piano dei lavori.

L'output della simulazione è una lista di collissioni di spazi di lavoro.

Le collisioni consistono in sovrapposizioni di due o più spazi di lavoro di attività differenti che vengono eseguite contemporaneamente.

FASE 3:

SOLVE A COLLISION

L'agente ha il ruolo di considerare le collisioni, output della simulazione, e scegliere il metodo risolutivo.

La simulazione è un processo veloce e non di dispendioso in termini economici, dunque può essere usata come strumento decisionale nello scegliere il metodo risolutivo migliore.

Nella fase 3, quella di risoluzione delle collisioni, secondo il pattern DMAS, l'agente ha due possibilità di azione:

A) risolvere in maniera autonoma le collisioni:
variando le risorse necessarie alla lavorazione o la successione delle attività;

B) demandare la responsabilità di risoluzione agli agenti di prodotto, nonché i responsabili delle squadre delle lavorazioni interessate al conflitto.
In questo caso, sarà suo compito raccogliere la decisione, produrre un nuovo cronoprogramma e verificare attraverso la simulazione eventuali nuove collisioni.
Il passaggio di risoluzione delle collisioni viene spiegato nel capitolo 4.2.1 "Struttura di un workflow nel caso di lavori di riqualificazione"

5.1.2 NECESSITÀ DI UN MOTORE GRAFICO PER VISUALIZZARE CONFLITTI SPAZIALI

I conflitti spaziali consistono nella sovrapposizione di zone di lavoro attive nello stesso momento.
In altri termini, due zone di lavoro sono due componenti tridimensionali *(ad esempio un cubo che rappresenta la zona di lavoro degli infissisti ed un altro dei piastrellisti)* che se si sovrappongono generando un'interferenza.
Di fatto nella realtà, un intereferenza impedisce l'esecuzione contemporanea di entrambe le attività causando inefficienze nel processo di produzione.
Le zone di lavoro comprendono:
- il volume degli elementi che costituiscono l'elemento lavorato (esempio finestra,cavedio ecc);
- lo spazio necessario agli operatori di stazionare e spostarsi;
- lo spazio necessario alle opere provvisorie come ad esempio impalcati;
- lo spazio necessario alle attrezzature.

Una zona di lavoro è dunque rappresentata nel caso studio da due principali variabili:
1) Forma e dimensioni tridimensionale;
2) Collocazione spaziale in una determinata zona del cantiere o della costruzione.

Lo strumento che permette di gestire le due variabili contemporaneamente è un motore grafico.
Un motore grafico è uno strumento che permette di interagire con modelli tridimensionali, come il modello digitale BIM del costruito, lasciando ampia libertà di generare al suo interno nuovi elementi (come gli spazi di lavoro) e generare relazioni fra gli elementi tridimensionali importando file esterni come cronoprogrammi.

In questa maniera, è possibile aggiungere la risorsa spazio ad ogni attività del cronoprogramma ed evidenziare collisioni all'interno dell'ambiente riprodotto nel motore grafico.
Senza strumenti simili, l'agente non ha possibilità di collocare le azioni nello spazio.

Avere uno strumento che riproduce la realtà, permette di cambiare, aggiornare il progetto ed il modello un numero molto alto di volte e simulare quello che accadrebbe per ogni modello decisionale.
In questa maniera è possibile risolvere conflitti prima ancora che nella realtà si verifichino.

5.2 STRUMENTI UTILIZZATI

5.2.1 AMBIENTE SIMULAZIONE: UNITY 3D

Nel caso studio è stato utilizzato il motore grafico Unity 3D come ambiente di simulazione, un motore grafico multipiattaforma sviluppato da *Unity Technologies* che consente di riprodurre processi produttivi.

Unity 3D è stato scelto per le seguenti caratteristiche:
- ha leggi fisiche implementate di default, quindi possibilità di replicare ambienti complessi e realistici all'interno del motore;
- presenza di tool che permettono di integrare la parte di programmazione con esigenze grafiche;
- possibilità di importare ed esportare dati BIM *(Modelli Autodesk Revit, BIM 360, Navisworks, SketchUp e Rhino);*
- possibilità di importare ed esportare file di vario formato, tra cui quelli di tipo JSON;
- presenza di ASSET specifici già sviluppati che facilitano determinate operazioni, come ad esempio l'importazione di file formato GLTF;

Unity si basa su elementi definiti "GameObject".
Un "GameObject" può essere un oggetto vuoto, un oggetto 3D, un effetto, una luce, un audio, un video o un interfaccia utente.
Nel caso studio sono stati utilizzati due i tipi di "GameObject":
- oggetto "empty", a cui non è associata una rappresentazione grafica;
- oggetto 3D a cui è associata una rappresentazione grafica.

In un motore grafico, gli elementi tridimensionali che compongono la scena, quindi nel nostro caso l'ambiente di simulazione, possono essere sia importati attraverso modelli gia esistenti *(ad esempio in formato glTF)* che costruiti al suo interno tramite strumenti di modellazione.
Un motore grafico permette di importare la geometria del modello tridimensionale, i cui elementi vengono automaticamente tradotti in "GameObject".
In aggiunta, una volta che il modello tridimensionale è stato importato, risulta possibile

collocare e modellare geometricamente gli spazi di lavoro.
La schermata principale di Unity3D permette di interagire con il sistema attraverso tre differenti finestre:

(1)
Questi sono gli elementi che costituiscono la simulazione e popolano la schermata *"Gerarchia" (1).*
Anche il modello importato del caso studio è composto da più elementi "GameObject".
Nella schermata *(1)*, gli elementi possono essere raggruppati secondo gerarchie definite.
Il concetto principale è che un oggetto figlio di un altro, eredita le catteristiche dell'oggetto genitore (soprastante).
Modificando ad esempio la posizione di un "GameObject" definito *"parent",* anche la posizione degli elementi *"child"* varia nello spazio.

(2)
La schermata *"Inspector" (2)* permette di aggiungiere componenti ai "GameObject".
I componenti oltre a regolare la posizione, ne determinano il comportamento (posizione, grandezza, generazione di altri GameObject al verificarsi di certe condizioni, ecc…).
Fra i componenti possibili da integrare ci sono gli *"script".*
Questi sono il mezzo principale per mettere in relazione più "GameObject" e strutturare sistemi complessi di relazioni.
Gli *"script"* sono codici sviluppati con librerie prestabilite di Unity, in linguaggio C#.

(3)
Altra finestra di Unity è la finestra *"Project" (3),* tramite cui è possibile caricare elementi e file esterni al progetto.
Nella schermata *"Project"* vengono anche caricati componenti di eventuali ASSET esterni.
La finestra *"Project"* è affiancata dalla *"Console".*
Questa permette, in praticolar modo durante l'avvio della scena, di compiere Debug di eventuali messaggi o variabili ed avvisare della presenza di errori nel codice.
Sarà nella console che avviene la prima fase di restituzione delle collisioni degli spazi di

lavoro.

Di seguito viene allegata una schermata esemplificativa di Unity che indica la posizione delle tre finestre:

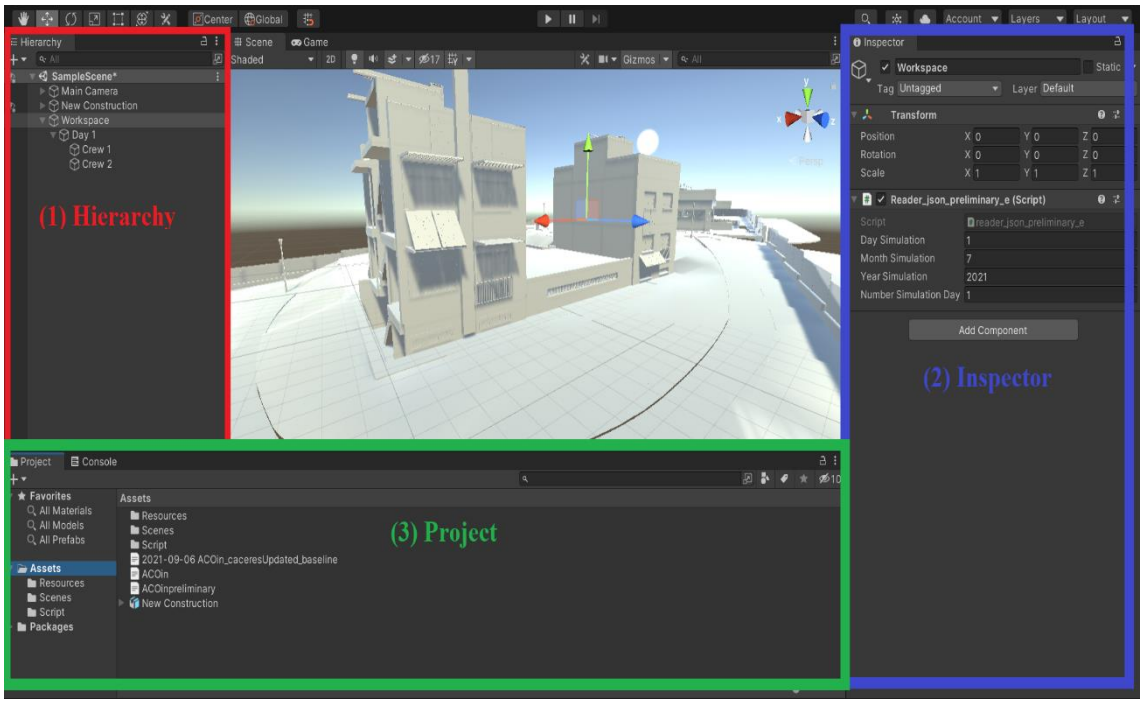

Figura 18 La schermata Unity tradizionale si compone di tre finestre oltre la scena: (1)Hierarchy, (2)Inspector, (3)Project

Inoltre, nella parte centrale della scena il pulsante *"Play"* permette l'avvio della simulazione.

Figura 19 Attraverso il pulsante play è possibile avviare la simulazione manualmente

Come detto, le funzionalità più rilevanti di unity3d sono richiamabili attraverso script.

nel caso studio gli script sono stati sviluppati in linguaggio c#.

questi vengono composti in editor di sviluppo esterni a unity ma comunicanti con il motore grafico stesso.

nel caso studio è stato utilizzato microsoft visual studio.

Il vantaggio degli script è quello di poter utilizzare delle librerie esterne di funzioni che permettono di creare relazioni anche complesse fra più GameObject.

In Unity 3D, gli script presentano una struttura standard che prevede la presenza di due funzioni:

(1)

```
1.  void Start()
2. {
3. }
```

La funzione "Start" permette di avviare dei processi, tutto ciò incluso fra le parentesi grafe, all'avvio della scena, quindi alla pressione del tasto "Play".

(2)

```
1. void Update()
2. {
3. }
```

La funzione "Update" permette di avviare dei processi, tutto ciò incluso fra le parentesi grafe, per ogni frame di simulazione.
Significa che i processi vengono eseguiti con cadenza temporale uguale a quella del frame di simulazione.

5.2.2 PIGLET: GLTF IMPORTER

Nello sviluppo del sistema, viene utilizzato Piglet.
Piglet è un ASSET disponibile nello store di Unity e permette di importare modelli geometri 3D in formato glFT ed glb.
Un file .gltTf può essere autonomo o può fare riferimento a risorse binarie e texture esterne, mentre un file .glb è completamente autonomo.
Piglet permette di:
- importare modelli glTF sia tramite *"drag-and-drop"* nell'editor che in runtime tramite l'esecuzione di script C#.
- l'importazione dei modelli tramite script può avvenire sia impostando il percorso

locale del file che tramite url HTTP.

Il metodo (A) più semplice di importazione è quello del "drag-and-drop".
Il file con estensione .gltf, dopo aver installato Piglet in Unity, viene trascinato nella cartella Asset.
In automatico viene creata una nuova cartella contente il prefab del modello importato ed altri elementi quali texture, materiali ecc.
Sucessivamente il prefab può essere trascinato nella finestra gerarchia.

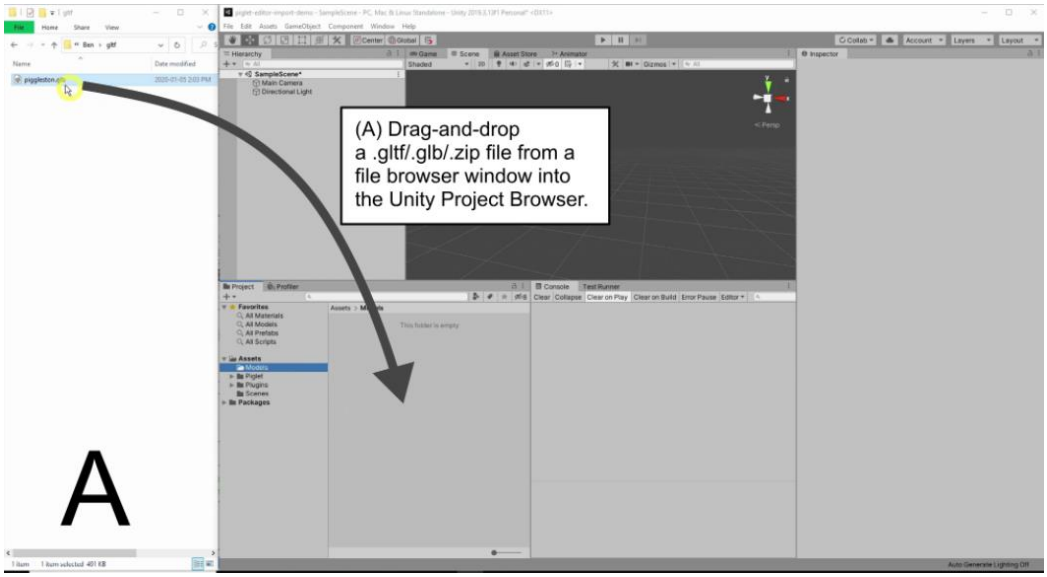

Figura 20 Metodo di importazione A drag-and-drop

E' possibile evitare la generazione automatica del prefab corrispondende, tenendo premuto ctrl durante l'importazione.

Secondo metodo (B) di importazione è quello tramite path.
Si pressupone che il file .gltf sia salvato in memoria locale.
Viene utilizzata la funzione *"RuntimeGltfImporter.*

Il percorso del file.gltf da caricare deve essere specificato per intero.
Ad esempio:
"D:/Univpm/2.MAGISTRALE/TESI/20.Importazione/caceres.gltf"

Il task "RunTimeGltfImporter" viene specificato nella funzione "Start" di Unity, quindi quella riferita all'avvio della simulazione.

```
1.    _task = RuntimeGltfImporter.GetImportTask(
2.     "D:/Univpm/2.MAGISTRALE/TESI/20.Importazione/caceres.gltf");

3.    _task.OnCompleted = OnComplete;
```

Poi viene eseguito con la funzione *"MoveNext()"* al primo frame della simulazione, nella funzione *Update* di Unity.

```
1. void Update()
2. {
3.     _task.MoveNext();
4. }
```

Può essere necessario avviare altri script al completamento del caricamento del modello .gltf.
Quindi Piglet considera due utili funzioni che rappresetano lo stato di caricamento del modello:
1) (OnProgress): definisce il numero di texture, materiali e nodi caricati rispetto al totale tramite messaggi in console;
2) (OnComplete): definisce quando il modello è stato caricato completamente.
Questo permette di associare al modello generato altri script che si basano sul modello stesso.

```
1. private void OnComplete(GameObject importedModel)
2. {
3.     //Import the model .gltf
4.     _model = importedModel;
5.     Debug.Log("Success!");
6.     //when the model is loaded do something
7. }
```

Perciò ad importazione avvenuta in console sarà visualizzato il seguente messaggio:

Figura 21 Debug.Log("Success!");

La documentazione di piglet è consultabile al seguente link:

https://awesomesaucelabs.github.io/piglet-manual/

5.3 OBBIETTIVO DEL SISTEMA SVILUPPATO

I processi produttivi nell'industria delle costruzioni possono essere migliorati introducendo strumenti decisionali di supporto nella programmazione dei lavori.
In particolar modo sono i flussi delle risorse a dover essere misurati affinchè questi non determinino attese ed inefficenze.
Tra questi, quello della risorsa spazio rappresenta un elemento che se processato senza interruzioni genera in termini totali ottimizzazione del sistema produttivo.
Il caso studio, si pone come obbiettivo quello di definire uno strumento in grado di rappresentare in un motore grafico lo spazio di lavoro per ogni attività.
Lo strumento proposto, viene inserito in un sistema capace di sviluppare simulazioni in ambiente digitale il cui scopo è quello di restituire la lista delle sovrapposizioni degli spazi di lavori, che di fatto limitano l'efficienza produttiva.
In funzione dei risultati della simulazione, il piano dei lavori può essere modificato variando l'ordine delle azioni o la dimensione degli spazi di lavoro necessari all'esecuzione.
La simulazione, rappresenta uno strumento decisionale che permette di comprendere quale fra più scelte progettuali risulta la migliore.
Il livello di dettaglio della simulazione prevede una suddivisione temporale pari al giorno, definendo per ogni intervallo giornaliero quali crew sono in azione e in quali deliverable.
Le attività vengono lette automaticamente dal programma dei lavori importato nell'ambiente di simulazione.
Per ogni deliverable viene rappresentata una zona di lavoro, a cui è associato un *"gameObject"* a forma di cubo le cui dimensioni comprendono in automatico l'elemento lavorato e lo spazio necessario alla movimentazione dei tecnici.
Le dimensioni del cubo possono, al momento, essere variate manualmente dall'utente.
Ogni cubo rappresenta un deliverable.
L'obbiettivo della simulazione è restituire una lista di sovrappozioni di due o più cubi, definendo il deliverbale, la crew e la data associata.

5.4 STRUTTURA DEL SISTEMA SVILUPPATO PER LA GESTIONE DEI LAVORI IN TEMPO REALE

5.4.1 WORKFLOW REALIZZATIVO DEL SISTEMA

Lo sviluppo del sistema è stato articolato in quattro differenti fasi:

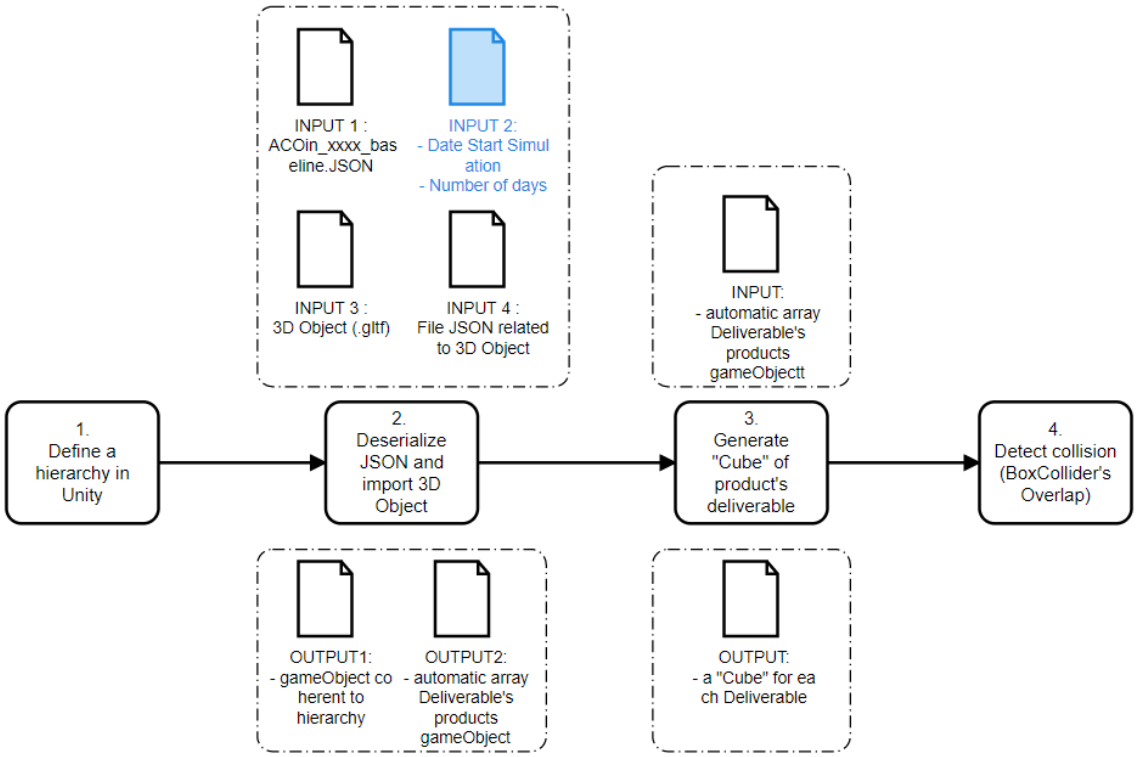

Figura 22 Lo sviluppo del sistema si è articolato in quattro task principali

(1)

Generare la struttura della simulazione attraverso la definizione di una gerarchia di elementi nel motore grafico;

(2)

Importare:

- il piano dei lavori in formato JSON e deserializzarlo per renderlo fruibile al motore grafico;
- il modello geometrico in formato .glTF ed il relativo file JSON associato che ne descrive gli elementi.
- elaborare gli input manuali dell'utente;

(3)

Generare per ogni deliverable attivo nell'intervallo di simulazione, un cubo che rappresenti lo spazio di lavoro.

(4)

Restituire una lista delle collisioni indicanti il deliverbale in questione, la crew associata e la data in cui avviene.

Le quattro fasi si sviluppano all'interno dell'ambiente di simulazione, Unity 3D.

Le fasi 2 e 3 prevedono degli Input e degli Output.

Questi sono digitalizzati, quindi processati in automatico dalla piattaforma, ad eccezione dell'*Input2* della Fase 2, colorato di azzurro.

Questo input prevede l'inserimento manuale dell'utente

- della data di inizio simulazione;
- il numero di giorni di estensione della simulazione stessa.

5.4.2 FASE 1: DEFINIZIONE DI UNA GERACHIA DI ELEMENTI NELL'AMBIENTE DI SIMULAZIONE

Il processo logico di sviluppo del sistema prevede che:

- ogni deliverable definisce un'area di lavoro;
- ogni crew lavora uno o più deliverbale nella stessa giornata;
- le collisioni degli spazi di lavoro sono da rilevare solo se verificatesi in deliverables lavorati da squadre differenti.

Esempio:

nella stessa giornata la crew 100 lavora il deliverables 3 e 4.

I deliverables 3 e 4 si sovrappongono. La sovrapposizione non viene rilevata perché riconducibile alla stessa crew.

E' quindi necessario suddivere i deliverables in funzione della squadra di lavoro che li esegue.

Ricordiamo che il motore grafico permette di concatenare le proprietà di più elementi *"GameObject"* tramite la generazione di parentele.

Oggetti *"Parent"* possono avere uno o più elementi *"Child"*.

Le proprietà dei *"Child"* vengono ereditate dagli elementi *"Parent",* come ad esempio

posizione ecc.

La simulazione online prevede una suddivisione temporale pari al giorno.

La gerarchia proposta presuppone:

Level 1: Workspace

Level 2: Days

Level 3: Crews

Level 4: Deliverables

In questa maniera è possibile leggere le sovrapposizioni a livello 3.

L'elemento workspace sarà solo uno, gli elementi al livello due saranno proporzionali all'intervallo di simulazione scelto dall'utente, mentre al livello 3 e 4 il numero di elementi dipende dal programma dei lavori.

Di seguito è allegata un'immagine dimostrativa della gerarchia proposta in Unity3D:

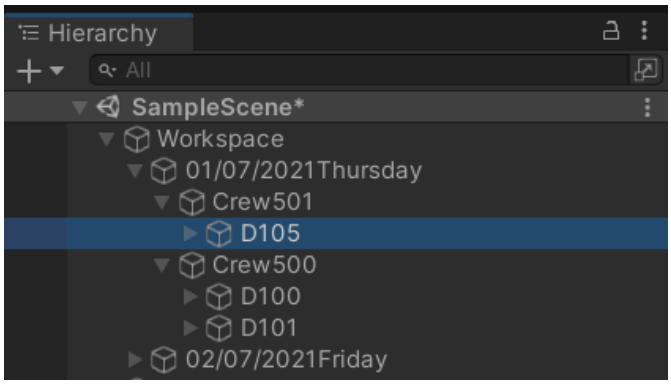

Figura 23 Estratto della finestra Hierarchy in Unity della struttura proposta: Day,Crew,Deliverable.

In questo caso la simulazione ha un orizzonte temporale pari a due giorni (01/07/2021 e 02/07/2021).

Nello specifico nel giorno 01/07/2021:

Level 1: Workspace

Level 2: 01/07/2021;

Level 3: Crew501 e Crew500

Level 4: D105 per la Crew501 e D100+D101 per la Crew500

5.4.3 FASE 2: IMPORTAZIONE DEL PIANO DEI LAVORI E DEL MODELLO 3D

La Fase 2 ha come obbiettivo di :

(A) Deserializzazione il cronoprogramma in formato JSON;

(B) Importazione del modello 3D formato .glTf e deserializzazione del file JSON associato;

(C) Organizzazione degli input manuali.

La deserializzazione in *(A)* e *(B)* ha come obbiettivo:

1) generare automaticamente gli elementi dei livelli prima elencati: day, crew, deliverable coerentemente alla gerarchia definita in TASK 1.

2) associare alle stringe "*Guid*" presenti negli oggetti "deliverable" del file "*ACOin_xxxx_baseline*"(esportazione JSON del cronoprogramma), i "GameObject" presenti nel modello geometrico .glTF utilizzando il file JSON associato.

Il Task 2 si compone di sette differenti Task, i primi due riguardano il processo di deserializzazione *(colore blu),* il successivo l'organizzazione degli input manuali *(colore giallo),* i successivi tre la generazione degli elementi *"GameObject" (colore rosso)* e l'ultimo l'associazione degli elementi a partire dalle stringe *"Guid" (colore verde):*

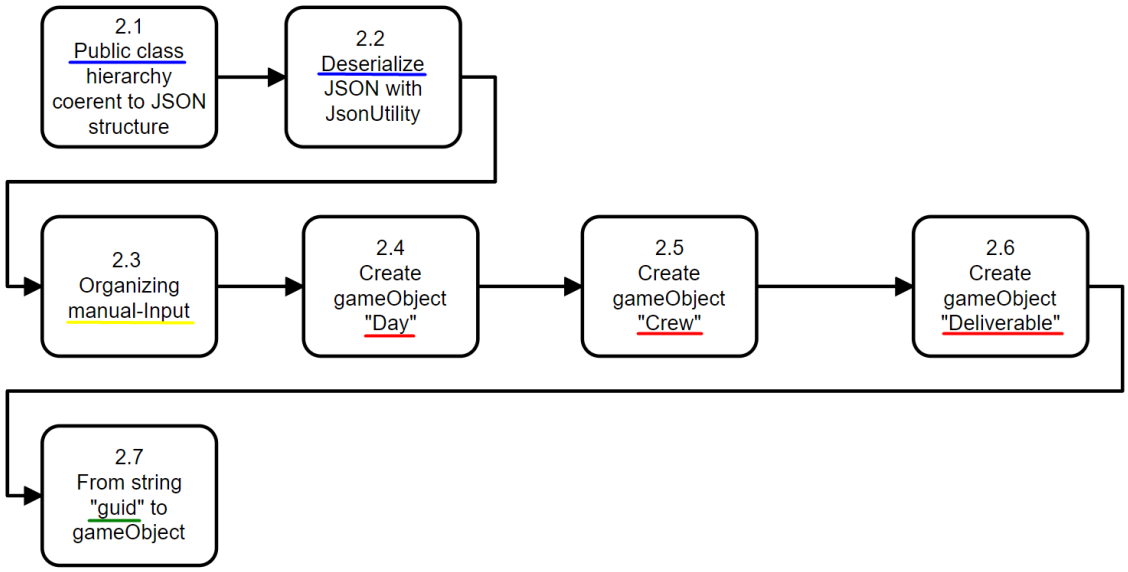

Figura 24 Il processo di deserializzazione e generazione dei gameObject si articola in sette passaggi

2.1 Public class hierarchy coerent to JSON structure

Viene generata una gerarchia di classi pubbliche coerente al file JSON da deserializzare. A tale scopo è stato utilizzato un convertitore automatico online, di seguito il link: *https://json2csharp.com/*

Nello specifico a titolo di esempio, vengono riportate le classi pubbliche generate per deserializzare il file JSON "*ACOin_xxxx_baseline.JSON"*:

```
1.      [System.Serializable]
2.     public class Deliverable
3.     {
4.         public string x_comment;
5.         public int id;
6.         public string name;
7.         public Size size;
8.         public Location location;
9.         public List<object> startDateTime;
10.        public List<object> dueDateTime;
11.        public List<string> guids;
12.    }
13.
14.    public class Crew
15.    {
16.        public string x_comment;
17.        public int crewID;
18.        public List<double> unitLabour;
19.        public int setupMatrixID;
20.        public int id;
21.        public string name;
22.        public List<Worker> workers;
23.        public object equipment;
24.        public double speed;
25.    }
26.
27.    public class Activity
28.    {
29.        public string x_comment;
30.        public int id;
31.        public string priceListCode;
32.        public string name;
33.        public string start;
34.        public string end;
35.        public int progress;
36.        public Crew crew;
37.        public object preconditions;
38.        public int deliverables;
39.        public DateTime startDateTime;
40.        public DateTime dueDateTime;
41.        public double size;
42.        public string priceListCode;
43.        public string name;
44.        public double unitCost;
```

```
45.         public int quantity;
46.     }
47.
48.     public class Datum
49.     {
50.         public string id;
51.         public string type;
52.         public object text;
53.         public bool open;
54.         public string start_date;
55.         public object end_date;
56.         public string parent;
57.         public string planned_start;
58.         public string planned_end;
59.         public object duration;
60.         public object progress;
61.     }
```

2.2 Deserialize JSON with JsonUtility (INPUT1: "ACOin_xxxx_baseline.JSON")

Viene deserializzato il file JSON attraverso le seguenti operazioni:

2.2.1 Il file JSON viene letto e tradotto in una stringa;

2.2.2 Viene generato un oggetto C# "root" pronto a contenere tutti i valori e le liste di valori del file json;

2.2.3 Con la funzione "*JsonUtility.FromJson<>()*" viene deserializzato il file json.

```
1. //1. DESERIALIZE BASELINE FILE JSON
2.      string json = File.ReadAllText(Application.dataPath + "/2021-09-
   06 ACOin_caceresUpdated_baseline.json");
3.      // Root2 contain all the data from JSON file Baseline
4.      Root2 rootObject;
5.      rootObject = new Root2();
6.      // FromJson is the function that Deserialize file.json
7.      rootObject = JsonUtility.FromJson<Root2>(json);
```

Il file JSON da deserializzare è inserito manualmente nella cartella di importazione del motore grafico.

Precisazione:

l'oggetto *"rootObject2"* è un elemento riconosciuto come un oggetto nel contesto del codice C#, ma non ha una rappresentazione in Unity3D.

Il suo compito è quello di essere una chiave di accesso al file deserializzato.

Lo stesso procedimento di deserializzazione verrà utililizzato nell'importazione del file JSON associato al modello 3D Object.

2.3 Organizing manual-Input (INPUT2: Date Start Simulation, Number of days)

L'organizzazione degli input manuali si compone di due fasi:

2.3.1 Inserimento manuale dell'input:

A) Data di inizio simulazione

Viene generata inserendo il numero del giorno, del mese e dell'anno.

B) Il numero di giorni della simulazione

Il numero di giorni è compresivo della giornata di inizio e dei giorni non lavorativi.

(Esempio se startDate = 17/09/2021,Friday e NumberSimulationDay= 5: la simulazione considera 1.Friday,4.Monday,5. Tuesday.)

Di seguito viene mostrata una schermata di esempio:

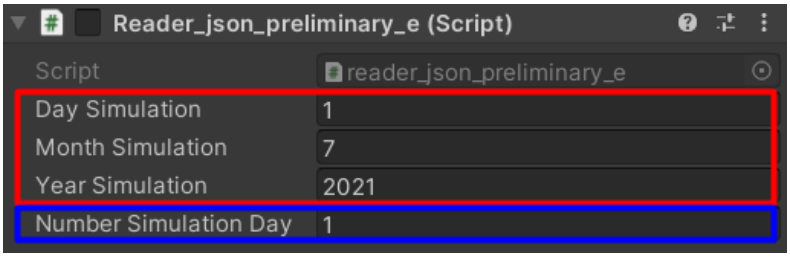

Figura 25 Schermata di esempio degli imput manuali: data e numero di giorni

2.3.2 Generazione della data a partire dagli input manuali:

I quattro input manuali sono quattro numeri interi.

- La data di inizio viene generata combinando i primi 3 valori, con funzione *"DateTime"* .

- La data di fine viene generata aggiungendo "Number Simulation Day" alla data di inizio, con funzione *"DateTime.AddDays"*.

```
1. //CALENDAR:StartDateSimulation + EndDateSimulation
2.         DateTime StartDateSimulation = new DateTime( yearSimulation, m
   onthSimulation, daySimulation);
3.         DateTime EndDateSimulation = StartDateSimulation.AddDays(numbe
   rSimulationDay);
```

La data di fine simulazione è individuata come la somma fra quella di inizio e i giorni della simulazione.

2.4 Create gameObject "Day" (OUTPUT1)

Nel passaggio seguente viene generato il primo livello della gerarchia: la suddivisione temporale della simulazione, che nel seguente caso è il giorno, attraverso le seguenti operazioni:

2.4.1 Con ciclo "for" incrementale di 1 Giorno si entra nella finestra temporale di giorni compresi fra StartDateSimulation ed EndDate Simulation.

2.4.2 Viene generato un GameObject per ogni giorno e denominato con *"date + DayOfWeek"*.

Dalla generazione di oggetti vengono esclusi quelli con *DayOfWeek* uguale a "Sunday"Saturday".

2.4.3 Ogni giorno viene aggiunto alla lista *"ListDaySimulation"* contenente tutti i giorni della simulazione.

Questa sarà necessaria nel passaggio successivo, durante il confronto fra le date della simulazione e quelle delle attività, ai fini di trovare le attività in esecuzione nella finestra temporale della simulazione.

```
  //GENERATE GAMEOBJECT DAYS
var ListDaySimulation = new List<DateTime>();
for (DateTime date = StartDateSimulation; date < EndDateSimulation; date =
 date.AddDays(1))
{
    DayOfWeek day = date.DayOfWeek;
    if ((day != DayOfWeek.Saturday) && (day != DayOfWeek.Sunday))
    {
        objToSpawn_Day = new GameObject();
        objToSpawn_Day.name = date.ToString("d") + date.DayOfWeek;
        objToSpawn_Day.transform.SetParent(this.transform);
        //List of day simulation
        var DaySimulation = date;
        ListDaySimulation.Add(DaySimulation);
    }
}
```

Per ogni giorno viene generato un GameObject e denominato in funzione della data di simulazione, vengono esclusi i giorni non lavorativi.
Nell'esempio seguente viene mostrato come l'orizzonte temporale della simulazione pari a 3 giorni, esclude dalla generazione degli elementi i giorni non lavorativi mostrandone solamente due.
In questo caso il sabato.

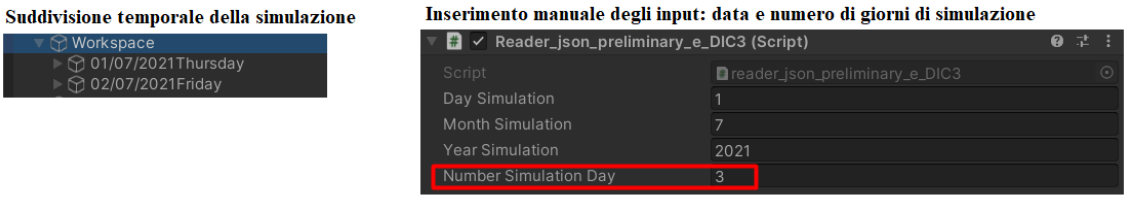

Figura 26 Il sistema esclude dall'orizzonte temporale i giorni non lavorativi.

2.5 Create gameObject "Crew" (OUTPUT1)

Nel passaggio seguente vengono generati i GameObject "crew" del secondo livello con data di inizio e fine coerente all'intervallo temporale scelto della simulazione:
2.5.1 Si accede alla baseline, filtrando gli oggetti con type=task.
Le date di inizio e fine delle attività vengono tradotte da stringe a date con funzione "*DateTime.TryParse()*".
2.5.2 Per coerenza temporale ad entrambe le date viene azzerato l'orario alla mezzanotte.
In questa maniera:
A) Nel confronto fra le date dei task e le date della simulazione è possibile riconoscere la congruenza, risultano uguali essendo entrambe relative a h 00.00.
B) Nella fase successiva, in cui è presento un ciclo for con incremento 24h, non si generino ambiguità.
Se entrassi nel ciclo con una data con orario di inizio alle ore 9.00 AM ed incrementassi +24h, avrei Data+1 ore 9.00 AM.
Se la stessa attività finisse alle ore 17 non rientrebber nell'intervallo di simulazione.

2.5.3 Ciclo for con inizio= "data di inizio dell'attività", fine= "data di fine dell'attività", con incremento di 1 giorno.

- Per ogni data in cui il task viene svolto, la data viene paragonata a tutte le date della simulazione.
- Se una data in cui è attivo il task è uguale ad una data della simulazione, significa che il task rientra nella simulazione.
- Dunque, si genera un GameObject "Crew" se non già presente nella data in questione.

```
//BASELINE
        foreach (Datum data in datumList)
        {
            if (data.type == "task")
            {
                // from string to a DateTime with DateTime.TryParse
                string dateStringStart = data.planned_start;
                string dateStringEnd = data.planned_end;
                DateTime dtStart;
                var BooleanStartDate = DateTime.TryParse(dateStringStart, out dtStart
                DateTime dtStartMidnight = new DateTime(dtStart.Year, dtStart.Month,
                dtStart.Day, 0, 0, 0);
                DateTime dtEnd;
                var BooleanEndDate = DateTime.TryParse(dateStringEnd, out dtEnd);
                DateTime dtEndMidnight = new DateTime(dtEnd.Year, dtEnd.Month, dtEnd.
                0, 0, 0);
                //GENERATE GAMEOBJECT CREW
                for (DateTime dateCrew = dtStartMidnight; dateCrew <= dtEndMidnight;
                dateCrew = dateCrew.AddDays(1))
                {
                    foreach (DateTime dateTime in ListDaySimulation)
                    {
                        //Compare Simulation DateTime and Activities DateTime
                        int resultComparison = DateTime.Compare(dateTime, dateCrew);
                        if (resultComparison == 0)
                        {
                            //1.Spawn gameObject CREW
                            var dayi = GameObject.Find(dateTime.ToString("d") +
                            dateTime.DayOfWeek);
                            {
                                var crewFind = dayi.transform.Find(data.parent);
                                if (!crewFind)
                                {
                                    //spawn object
                                    objToSpawn_Crew = new GameObject();
                                    objToSpawn_Crew.name = data.parent;
                                    objToSpawn_Crew.AddComponent<set_color_crew>();
                                    var inComponent = objToSpawn_Crew.GetComponent
                                    <set_color_crew>();
                                    inComponent.color_crew = dictionaryC[data.parent]

                                    //crew as a child of date;
                                    objToSpawn_Crew.transform.SetParent(dayi.transfor
                                }
```

Le date delle attività vengono paragonate alle date della simulazione, se sono uguali allora si genera il GameObject "Crew".

Il risultato che viene ottenuto nell'ambiente di simulazione è il seguente:

la generazione delle *"Crew501"* e *"Crew 500"* divise in funzione della data in cui eseguono azioni.

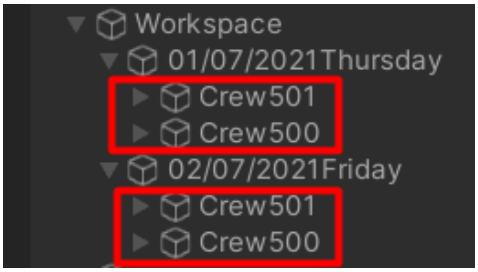

Figura 27 Ogni elemento "Crew" viene diviso in funzione della data in cui esegue attività

2.6 Create gameObject "Deliverable" (OUTPUT2)

L'obbiettivo della seguente fase è:

1) Generare i gameObject Deliverable;

2) Raccogliere le stringe Guids in una lista ed associarli a gameObject;

Soluzione:

PS: Ricordiamo di essere all'interno del ciclo for each datetime=datecrew, quindi ad ogni elemento di baseline.data

A) In Baseline.Data viene indicato il nome dell'Attività;

B) In Attività viene indicato il Deliverbale associato:

C) Accedendo al Deliverable è possibile accedere alle stringe "guids"

Figura 28 Al fine di generare l'oggetto deliverable è necessario passare attraverso l'oggetto attività

Per questo nel componente *"reader_json_simulation"* si seguono i seguenti passaggi:

2.6.1 Vengono ordinati gli oggetti: Data(Baseline), Attività e Deliverables in tre differenti liste:

A) I Data della baseline vengono raccolti nella lista *"dictionarylist1"*.

```
01
02 //A. list of data
03 var dictionarylist1 = new List<Datum>();
04 foreach (Datum data in datumList)
05 {
06     if (data.type == "task")
07     {
08         var pas1 = data;
09         dictionarylist1.Add(pas1);
10     }
}
```

La lista di Data(Baseline) verrà utilazzata come *"chiave"* nel *"dictionaryA"* generato nelle fasi seguenti.

Gli elementi vengono ordinati in ordine crescente dall' *"Activity 1"* all'*"Activity42"*.

A tal fine, non può essere utilizzato un metodo di ordinamento alfabetico il quale avrebbe fatto susseguire *"Activity 11"* ad "*Activity1"*.

Viene così eseguito il metodo *"PadNumers"*, il quale ordina le attività in progessione considerando il numero che accompagna la parola *"Activity"*.

Dopo aver ordinato gli elementi all'interno della lista, la lista viene trasformata in Array.

Il metodo è stato seguito dal seguente link:

https://stackoverflow.com/questions/5093842/alphanumeric-sorting-using-linq/5093939#5093939

```
List<Datum> queryOrder = dictionarylist1.OrderBy(data => PadNumbers(data.id)).ToList();
1       static string PadNumbers(string input)
2       {
3           return Regex.Replace(input, "[0-9]+", match =>
4           match.Value.PadLeft(10, '0'));
5       }
```

B) Le attività vengono raccolte nella lista *"dictionarylist2"*.

La lista di attività verrà utilizzata come *"valori"* nel *"dictionaryA"* generato nelle fasi seguenti.

```
var dictionaryarray1 = queryOrder.ToArray();
//B. list of activities
var dictionarylist2 = new List<Activity>();
foreach (Activity Activity in ActivityList)
{
    var pas2 = Activity;
    dictionarylist2.Add(pas2);
}
var dictionaryarray2 = dictionarylist2.ToArray();
```

Anche in questo caso la lista viene poi trasformata in array.

Non è necessario un ordinamento come nel caso dei Data(Baseline) poiché le attività sono già ordinate in ordine crescente in funzione del loro *id.*

C) I Deliverables vengono raccolti nella lista *"dictionarylist3"*.

```
//C. list of deliverables
        var dictionarylist3 = new List<Deliverable>();
        foreach (Deliverable deliverable in deliverableList)
        {
            var pas3 = deliverable;
            dictionarylist3.Add(pas3);
        }
        var dictionaryarray3 = dictionarylist3.ToArray();
```

Anche in questo caso la lista viene poi trasformata in array.

Non è necessario un ordinamento come nel caso dei Data(Baseline) poiché il metodo con cui vengono associate alle attività non è dipendente ad un dizionario.

2.6.2 Viene generato il "*dictionaryA"*.

Questo ha:

- come chiave la lista di Data(Baseline);

- come valore la lista delle Attività.

```
1. //Generate Dicitionary A:Data-Activity
2.
3. Dictionary<Datum, Activity> dictionaryA = new Dictionary<Datum, Activi
   ty>();
4. for (int i = 0; i < dictionaryarray1.Length; i++)
5. {
6.     dictionaryA.Add(dictionaryarray1[i], dictionaryarray2[i]);
7. }
```

Avendo ordinato le due liste in ordine crescente in funzione dell'id dell'attività, queste coppie sono uniche e coerenti.

Esempio:

alla chiave Data "Activity3" corrisponde il valore: "Attività con activity.id=3".

2.6.3 A partire dall'id dell'attività viene generato il Deliverable e localizzato.

In questa maniera è possibile accedere alla stringa di Guids, necessaria nella fase 2.7.

Nella porzione di script seguente:

- viene indagato il *dictionaryA* con la chiave data, interna al ciclo "foreach (*Datum Data in DatumList)"* con date coerenti a quelle dell'intervallo di simulazione;

- all'attività trovata viene associata il deliverable attraverso il confronto di *activity.deliverables* e *deliverable.id*;

- infine viene generato il *gameObjec*t Deliverable se questo all'interno della crew di riferimento non è gia stato creato.

Questo potrebbe avvenire se ad esempio la stessa crew lavora lo stesso giorno nello stesso deliverable.

```
Activity temp = null;
                            if (dictionaryA.TryGetValue(data, out temp))
01                          {
02                          //connect activities and deliverable
03                          var deliverableQuery = dictionaryarray3.Where(d =>
04                          d.id == temp.deliverables).FirstOrDefault();
05                          var DeliverableFind = crewFind2.transform.Find("D"
06                         + deliverableQuery.id);
07                              if (!DeliverableFind)
08                              {
09                              //spawn object
10                              objToSpawn_Deliverable = new GameObject();
11                              //Change name GameObect
12                              objToSpawn_Deliverable.name = "D" + deliverableQuery.i
13                              //Add deliverable as child activity
14                              objToSpawn_Deliverable.transform.SetParent(crewFind2.
                                transform);
```

Il risultato ottenuto è l'associazione fra i deliverable e le squadre che eseguono azioni all'interno di essi:

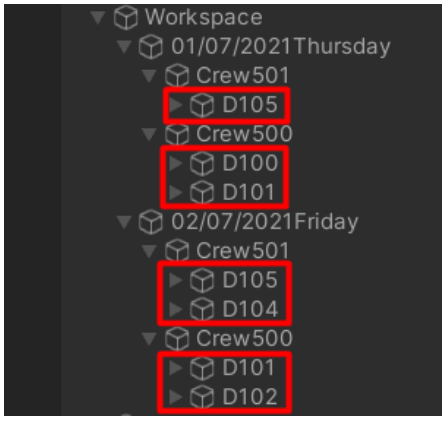

Figura 29 I deliverable sono divisi in funzione della crew che esegue attività al loro interno.

Nell'esempio:

- la Crew501 il giorno 01/07/2021 lavora all'interno del D105;
- la Crew500 il giorno 01/07/2021 lavora all'interno del D100 e D101;
- la Crew501 il giorno 02/07/2021 lavora all'interno del D105 e D104;
- la Crew500 il giorno 02/07/2021 lavora all'interno del D101 e D102;

2.7 From string "guid" to gameObject

L'obbietivo è quello di associare alle stringe "Guid" presenti negli oggetti "deliverable" del file "ACOin_xxxx_baseline", i *gameObject* presenti nel modello geometrico glTF utilizzando il file json associato.

Le stringe "Guid" sono codici alfanumerici che identificano l'elemento geometrico del modello.

La fase 2.7 prevede tre passaggi:

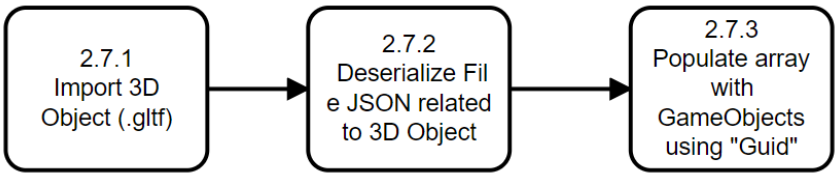

Figura 30 La fase 2.7 si compone di tre task: un'importazione, una deserializzazione ed un associazione fra guids e GameObjects

2.7.1 Import 3D Object (.gltf)

Nella fase seguente viene importato il modello geometrico della costruzione utilizzando l'asset Unity Piglet.

Lo script di importazione del modello è cosi composto:

A) Start

Allo "Start" viene richiamata la funzione accessibile attraverso la libreria di Piglet *"RuntimeGltfImporter.GetImportTask()"*.L'importazione viene svolta presupponendo che il modello sia salvato in locale.

Per questo, è utilizzato il percorso del file.gltf per indicare a Piglet quale file considerare.

Successivamente viene richiamata la funzione *"OnComplete"*.

```
1.  void Start()
2.  {
3.      _task = RuntimeGltfImporter.GetImportTask(
4.       "D:/Univpm/2.MAGISTRALE/TESI/20.Importazione/caceres.gltf");
5.      _task.OnCompleted = OnComplete;
```

```
6.      toDelete = GameObject.Find("Variables Saver");
7.      Destroy(toDelete);
8.  }
```

B) Update

In "Update" il task di importazione viene avviato e mantenuto attivo fino a quando il modello non sarà stato importato completamente.

```
1.  void Update()
2.  {
3.      _task.MoveNext();
4.  }
```

C) OnComplete

Piglet permette di avviare script o funzioni quando il modello è completamente importato.

Questo è comodo per casi, dove l'esistenza di successivi script prevede l'importazione di tutti gli elementi del modello.

Ad esempio, nel caso studio un obbiettivo è quello di cercare un *gameObject* a partire dal suo nome Guid.

Per farlo perciò è necessario che nella scena sia presente il *gameObject*.

Nota A:

Il modello importato con PIGLET rinomina i *gameObject* con nome **suddivisibile in due parti:**

- porzione anteriore a #: è il nome dell'elemento,

- porzione successiva a #: è il codice IfcEntityLabel presente in "caceres.json".

Di seguito viene mostrato un esempio:

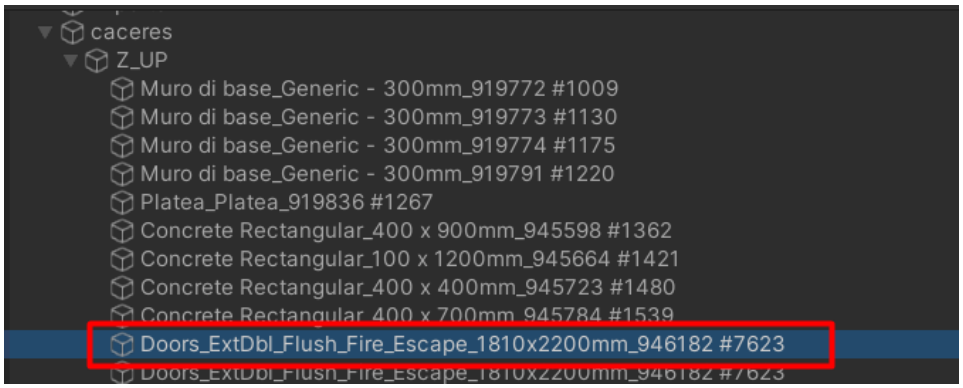

Figura 31 La rinominazione degli elementi importati con Piglet è scomponibile in due porzioni: il nome e il Guid

NotaB:

Elementi come porte, finestre o arredi sono formati da più *gameObject* con lo stesso nome.

Ogni *gameObject* è una porzione dell'elemento completo: la maniglia, il telaio fisso ecc.

Di seguito è rappresentato l'esempio di una porta:

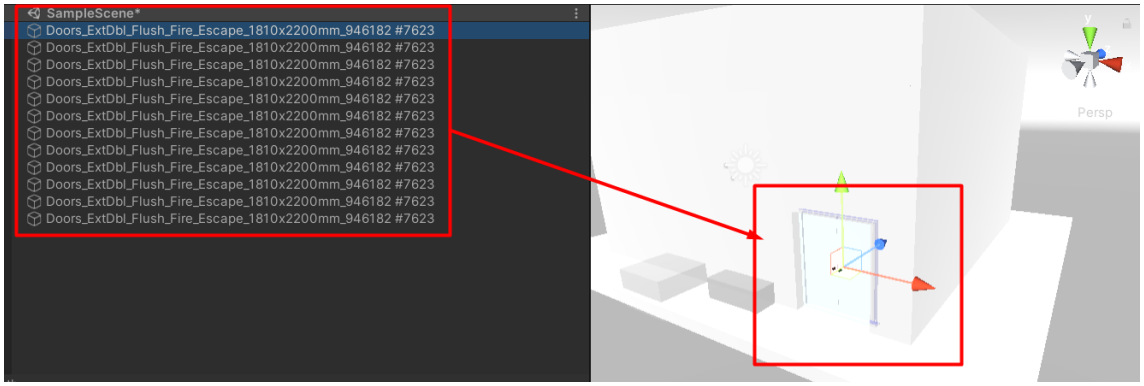

Figura 32 Elementi come gli infissi sono comporti da più gameObject aventi lo stesso nome

A tal proposito, quando il modello è completamente importato:

- vengono cambiati i nomi degli elementi conservando solo il codice *IfcEntityLabel.*

Per farlo viene utilizzato il # come divisore e viene indicato di conservare solo la parte successiva allo stesso.

Di seguito viene rappresentato il codice responsabile al cambio nome:

```
01
02public void OnComplete(GameObject importedModel)
03    {
04        //Import the model .gltf
05        _model = importedModel;
06        //Debug.Log("Success!");
07        //when the model is loaded:
08        GameObject childLevel1 = _model.transform.GetChild(0).gameObject;
09        var transformChildLevel1 = childLevel1.transform;
10        foreach (Transform child in transformChildLevel1)
11        {
12            string initialName = child.name;
13            output = initialName.Split('#').Last();
14            child.name = output;
        }
```

Il risultato ottenuto è il seguente:

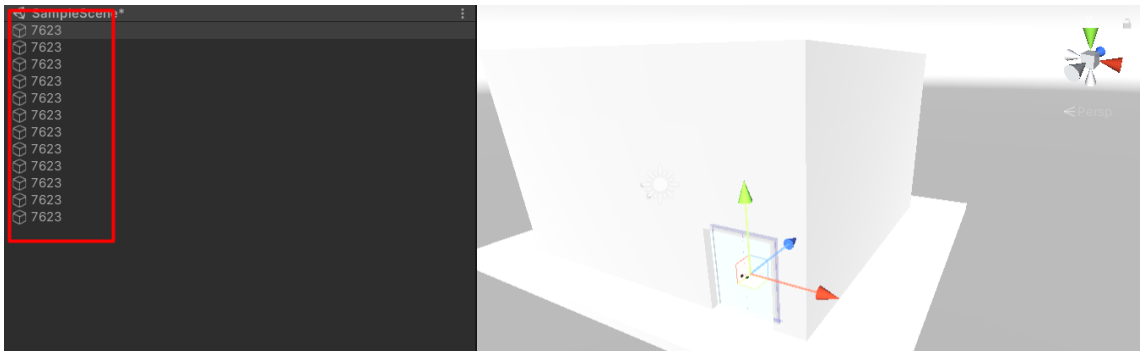

Figura 33 Risultato della rinominazione degli elementi importati

In questa manierà ogni gameObject è rinominato in funzione del proprio *IfcEntityLabel.* Così nei prossimi passaggi viene fornito un criterio attraverso cui è possibile cercare oggetti all'interno della scena.

2.7.2 Deserialize File JSON related to 3D Object

Nel passaggio seguente viene deserializzato il file JSON attraverso le seguenti operazioni:

- Il file json viene letto e tradotto in una stringa;
- Viene generato un oggetto C# "root" pronto a contenere tutti i valori e le liste di valori del file json;
- Con la funzione "*JsonUtility.FromJson<>()*" viene deserializzato il file json.

```
1. //2. DESERIALIZE CACERES FILE JSON
2. string json2 = File.ReadAllText(Application.dataPath + "/caceres.json"
   );
3. // Root contain all the data from JSON file Baseline
4. Root rootObject1;
5. rootObject1 = new Root();
6. // FromJson is the function which Deserialize file.json
7. rootObject1 = JsonUtility.FromJson<Root>(json2);
```

Il risultato ottenuto dell'importazione del modello tridimensionale glTF tramite Piglet e le due serializzazioni si presenta nella seguente disposizione nell'ambiente di simulazione Unity3D:

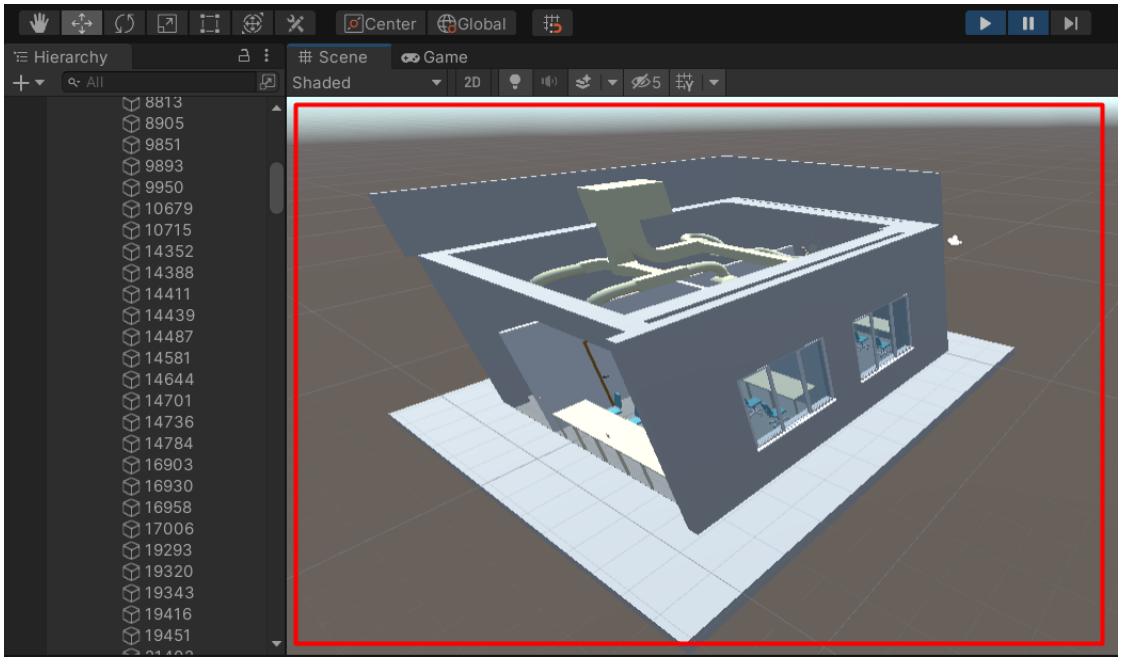

Figura 34 Il modello importato tramite Piglet è composto da piu elementi "GameObject"

Nell'immagine soprastante sono stati resi invisibili alcuni elementi per mostrare l'interno della struttura.

2.7.3 Populate array with GameObjects using "Guid"

L'obbiettivo è quello di associare ad ogni stringa *"Guid"* il relativo *gameObject*. Selezionare il *gameObject* o i *gameObject* ed inserirli nella lista che definisce l'input dello script *"merge_workspace"*.

Tutto ciò è possibile seguendo il seguente percorso logico:

A) In "ACOin_xxx_baseline" vengono indicati i *"guids"*.

B) In caceres.json (file.json associato al modello importato) è indicato il *"IfcGuid"* e *"IfcEntityLabel"*.

C) In Unity ogni gameObject importato dal modello.gltf è nominato con *"IfcEntityLabel"*.

Figura 35 Processo logico secondo cui è possibile cercare l'elemento in Unity3D attraverso il codice Guid

Perciò vengono seguiti i determinati pasaggi:

A:

E' definita la lista di stringe"guid" come proprietà dell'oggetto *"deliverableQuery"*, che è il deliverable trovato nei passaggi precedenti.

Viene anche definita una lista *"passList"* di *gameObject*, gli stessi che saranno poi l'input di *"merge_workspace"*.

B:

Per ogni stringa Guid, viene associato l'elemento di caceres.json che ha lo stesso valore IfcGuid.

C:

Trovato l'elemento corrispondente, allo stesso viene definito l' ifcEntityLabel.
Vengono raccolti nella lista "sameName"list", tutti i gameObject che hanno nome in Unity uguale all'ifcEntityLabel trovato.
Per la ricerca dei gameObject viene utilizzato il metodo "Where()" appartenente alla libreria LinQ.
Questo permette di includere nella lista più elementi con lo stesso nome, quindi anche elementi come infissi o porte definiti da più gameObject con lo stesso nome.

D:

Ogni gameObject presente in *"sameNameList"* viene aggiunto alla lista *"passList"*.
La lista *"passList"* viene definita come la variabile pubblica *"Products"* in *"merge_workspace"*.

```
A  //Define what is "GuidList"
   GuidList = deliverableQuery.guids;
   //Define a list which it can contain GameObject
   var passlist = new List<GameObject>();
     foreach (string guids in GuidList)
B    {
         //elements with element.ifcGuid == guids in GuidList
         var elementQuery = elementList.Where(d => d.ifcGuid == guids).FirstOrDefault();

C        string ifcEntityLabelQuery = elementQuery.ifcEntityLabel.ToString();

         var sameNameList = Resources.FindObjectsOfTypeAll<GameObject>().Where(obj => obj.name == ifcEntityLabelQuery).ToList();
D        foreach (GameObject gameObject1 in sameNameList)
         {
             passlist.Add(gameObject1);
         }
     }
   //Add Components merge_workspace and gameobject in products
   objToSpawn_Deliverable.AddComponent<merge_workspace>();
   var component = objToSpawn_Deliverable.GetComponent<merge_workspace>();
   component.Products = passlist.ToArray();
   }
```

Figura 36 I passaggi in cui si articola lo script relativo alla fase 2.7 sono quattro: si definisce la lista di guid, si cerca l'elemento con guid associato, allo stesso viene definito il ifclabelentity e successivamente vengono cercati i gameObject

Di seguito vengono mostrati degli esempi di associazione fra la stringa guid ed il gameObject:

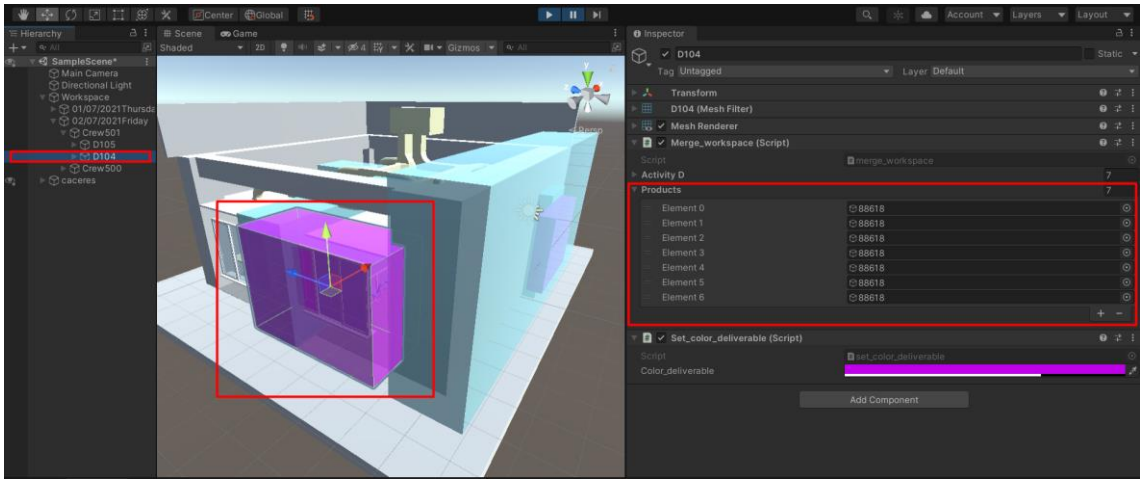

Figura 37 Più elementi con lo stesso nome rappresentano il deliverable, in questo caso una finestra

E' necessario includere nella raccolta non un solo elemento, ma tutti gli elementi con lo stesso nome.

In questa maniera, anche deliverable come gli infissi, composti da più elementi con lo stesso nome vengono rappresentati nella loro completezza.

Di seguito è mostrato l'esempio di una parete muraria interna, a cui è associato un solo elemento:

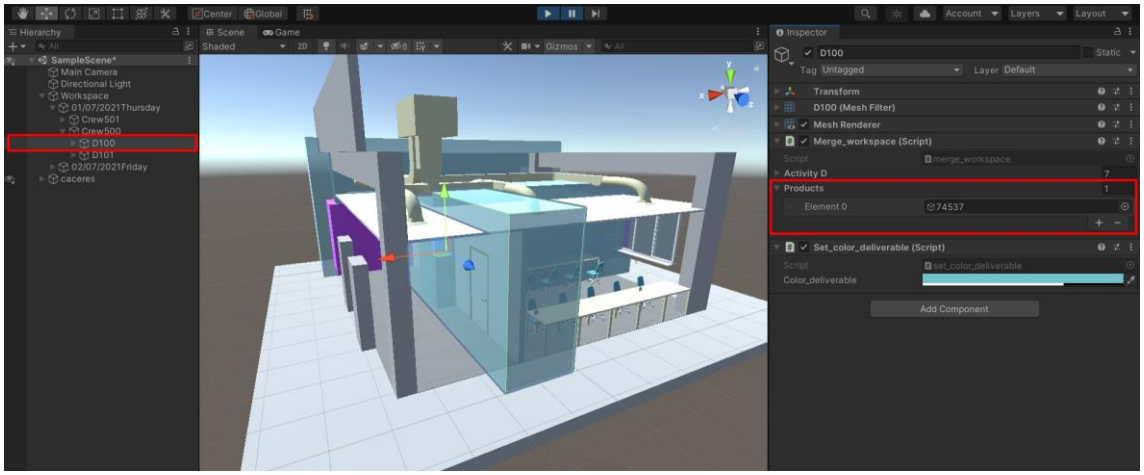

Figura 38 In questo caso ad una intera parete muraria corrisponde un solo elemento

5.4.4 FASE 3: GENERAZIONE DEGLI SPAZI DI LAVORO

In questa fase, il modello geometrico 3D del costruito, il relativo file JSON descrittivo degli elementi che lo compongono e il piano dei lavori in formato JSON sono già stati importati e letti nell'ambiente di simulazione.

La simulazione, quindi, è in grado di riconoscere quali attività, crew e deliverable vengono eseguiti nell'orizzonte temporale della simulazione.

Il passaggio seguente, eseguito nel Task 3, è quello di associare ad ogni deliverbale un elemento *gameObject* a forma di cubo che rappresenti la zona di lavoro associata a quell'azione.

Le dimensioni dello spazio di lavoro vengono aumentate di 1.5 unità Unity per ogni lato oltre l'elemento lavorato per simulare lo spazio necessario alla movimentazione dei tecnici e dello stoccaggio del materiale.

La forma e la posizione dello spazio di lavoro è regolata in funzione di più elementi.

Ogni deliverable è l'insieme di più elementi *gameObject.*

La sua forma dunque è strettamente legata ai prodotti che lo compongono.

Ad esempio un deliverable finestra ha come prodotto: la maniglia, il telaio fisso, il vetro ecc.

Per questo gli elementi prodotto vengono suddivisi in funzione del deliverable e raccolti in una lista.

La forma del cubo è regolata in funzione degli elementi presenti nella lista.

Dunque, ad ogni elemento del livello 4, quindi ad ogni deliverable viene aggiunto lo script "merge_workspace", responsabile della generazione dei cubi, che si compone di tre passaggi:

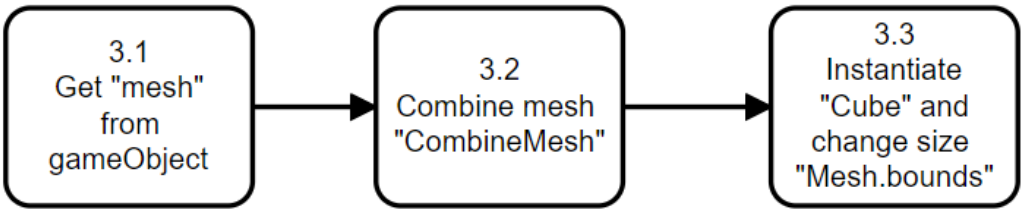

Figura 39 Lo script "merge_workspace" si compone di tre passaggi: estrarre le mesh, combinarle ed instanziare un cubo come inviluppo

Lo script si compone di tre passaggi, quindi:

3.1 Get "mesh" from gameObject (INPUT)

Ad ogni gameObject presente nell'array viene presa la mesh ed inserita nella array MeshFilters[].

Una mesh è composta da triangoli disposti nello spazio 3D per creare l'impressione di un oggetto solido.

E' quindi quel componente che definisce la forma di ogni GameObject nello spazio 3D.

```
1. meshFilters = new MeshFilter[Products.Length];
2.  for (int a = 0; a < Products.Length; a++)
3.  {
4.     meshFilters[a] = Products[a].GetComponent<MeshFilter>();
5.  }
6. CombineInstance[] combine = newCombineInstance[meshFilters.Length];
```

3.2 Combine mesh with "CombineMesh"

Attraverso un ciclo for vengono estratte la Mesh per ogni gameObject dall'array.

Poi unite con la funzione "CombineMeshes".

Il risultato che si ottiene è una unica mesh che ha come forma la forma dell'unione di tutte le mesh presenti nell'array.

```
1. while (i < meshFilters.Length)
2. {
3.     combine[i].mesh = meshFilters[i].sharedMesh;
4.     combine[i].transform = meshFilters[i].transform.localToWorldMatrix
   ;
5.     meshFilters[i].gameObject.SetActive(true);
6.     i++;
7. }
8. transform.GetComponent<MeshFilter>().mesh = new Mesh();
9.        transform.GetComponent<MeshFilter>().mesh.CombineMeshes(combine
   );
10.     transform.gameObject.SetActive(true);
```

3.3 Instantiate "Cube" and change size with "Mesh.bounds" (OUTPUT)

Viene instanzializzato un prefab "Cube" le cui dimensioni, con la funzione "mesh.bounds" si modellano fino a definire l'inviluppo convesso dei prodotti.
Per simulare lo spazio necessario agli spostamenti, alle attrezzature e ai materiali viene definito un offset di default pari ad 1.5 metri negli assi orizzontali e 0.5 metri nell'asse verticale. L'aumento delle dimensioni viene aggiunto dal centro del inviluppo delle mesh.

```
01
02 //instanciate workspace
03 GameObject workspaceObj = Instantiate(Resources.Load("workspace")) as GameObject;
04 workspaceObj.transform.SetParent(this.transform);
05 //bound size of workspace
06 this.GetComponentInChildren<Transform>().Find("workspace(Clone)").position
   = transform.GetComponent<MeshFilter>().mesh.bounds.center;
07 this.GetComponentInChildren<Transform>().Find("workspace(Clone)").localScale
08 = transform.GetComponent<MeshFilter>().mesh.bounds.size;
09 //set transform.x=+1 ecc
10 scaleChange = new Vector3(workspace_sizeX, workspace_sizeY, workspace_sizeZ);
   workspaceObj.transform.localScale += scaleChange;
11 workspaceObj.AddComponent<BoxCollider>();
12 // change name according their parent's name
13 workspaceObj.name = "Workspace" + this.name;
14 // change layer in Workspace
15 workspaceObj.layer = LayerMask.NameToLayer("Workspace");
16 workspaceObj.AddComponent<OverlapBox_JSON>();
17
   var layerQuery = workspaceObj.GetComponent<OverlapBox_JSON>();
18 layerQuery.m_LayerMask = LayerMask.GetMask("Workspace");
19 layerQuery.listActivity = ActivityD;
```

Esempi dei risultati così ottenuti sono mostrati seguentemente:

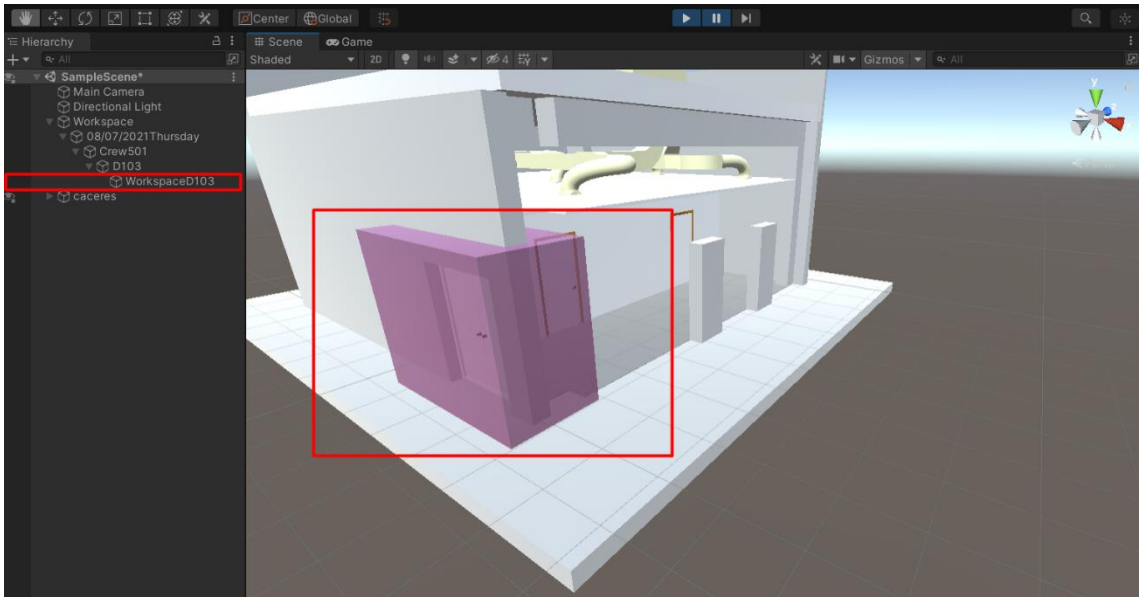

Figura 40 Lo spazio di lavoro relativo all'infisso comprende sia l'elemento lavorato che un offset di default

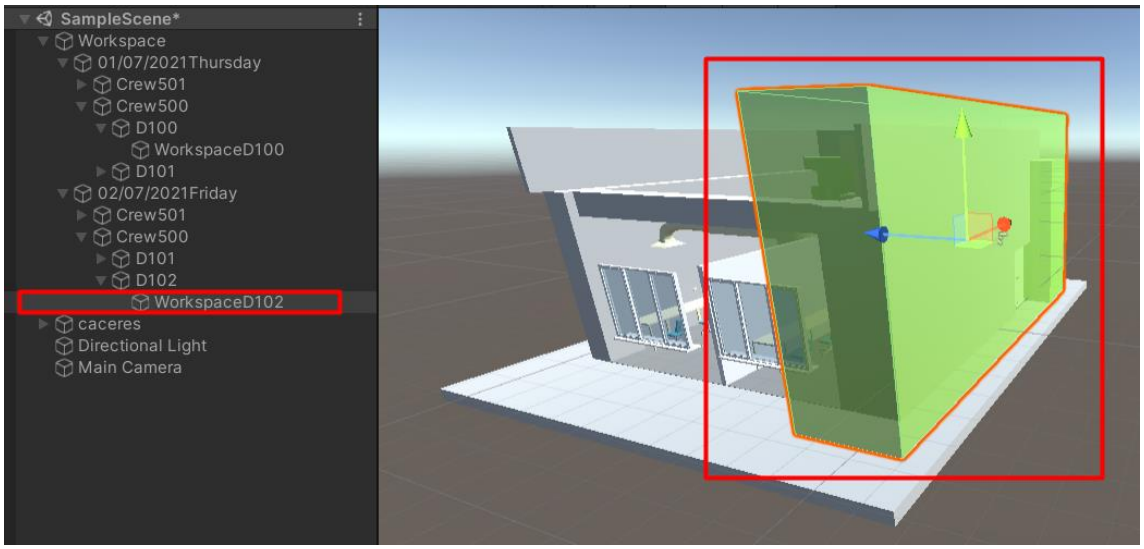

Figura 41 Lo spazio di lavoro relativo alla parete muraria comprende sia l'elemento lavorato che un offset di default

In questo caso i deliverables mostrati consistono in un infisso porta ed una parete esterna.

Lo spazio di lavoro generato oltre a comprendere l'elemento comprendere una porzione di spazio aggiuntiva così da simulare la zona necessario alla movimentazione dei tecnici dello stoccaggio del materiale e delle attrezzature.

5.4.5 GENERAZIONE DEI COLORI ASSOCIATI A SPAZI DI LAVORO

Al fine di rendere maggiormente visibili le zone di lavoro in simulazione queste sono colorate di differenti colori.

In altri termini, viene cambiato il colore al materiale del cubo, instanzializzato con l'obbiettivo di rappresentare la zona di lavoro.

La rappresentazione colorata delle zone di lavoro non influisce sull'esportazione della collisione che avviene nella fase successiva 6.2.6.

Il colore dei cubi è scelto con funzione randomica ma con un criterio prestabilito:

- deliverable lavorati dalla stessa crew hanno stesso colore.

Questo perché la sovrapposizione di zone di lavoro processate dalla stessa crew, ha una rilevanza molto inferiore rispetto alla sovrapposizione di zone di lavoro processate da crew differenti, con esigenze e attrezzature non uguali.

Si può anticipare che oltre a colorare le zone di lavoro elaborate dalle stesse crew dello stesso colore, le sovrapposizioni fra le stesse non vengono esportate nella fase successiva, perché appunto considerate irrilevanti.

La generazione dei colori essendo legata al numero delle crew, uno per crew, prevede i seguenti passaggi:

1. Raccogliere in una lista tutti gli elementi "crew" presenti nel file JSON baseline, rappresentante il piano dei lavori.

In questa maniera avremo le crew organizzate in una lista pronte per associargli un colore.

```
1. foreach (Datum data in datumList)
2. {
3.     if (data.parent == "Project")
4.     {
5.         var stringPass = data.id;
6.         stringNameCrewList.Add(stringPass);
7.     }
8. }
```

2. Generare un dizionario che abbia come chiave la *"crew"* e come valore un colore randomico.

In questa maniera nella fase successiva è possibile richiamare la *"crew"* all'interno del dizionario ed ottenere il colore associato.

```
1. Dictionary<String, Color> dictionaryC = new Dictionary<String, Color>(
   );
2. for (int i = 0; i < stringNameCrewList.Count; i++)
3. {
4.     Color colorCrewPass = new Color(Random.value, Random.value, Random
   .value, 0.70f);
5.
6.     dictionaryC.Add( stringNameCrewList[i], colorCrewPass);
7. }
```

3. Alla generazione dell'elemento *gameObject "crew"* viene aggiunto uno script responsabile del cambio di colore del materiale del cubo che rappresenta lo spazio di lavoro.

Lo script ha una variabile di tipo *"Color"* pubblica.

Il colore della variabile sarà il colore dello spazio di lavoro.

Perciò, quando viene generata la crew, le viene aggiunto lo script *"set_color-crew"*.

Poi viene settato il parametro pubblico colore corrispondente a quello della squadra.

Per farlo viene interrogato il dizionario con l'elemento squadra.

L'output dell'interrogazione del dizionario è appunto il colore associato alla squadra.

Nel riquadro rosso è rappresentata l'interrogazione del dizionario C e la successiva assegnazione della variabile pubblica:

```
if (!crewFind)
{
    //spawn object
    objToSpawn_Crew = new GameObject();
    objToSpawn_Crew.name = data.parent;
    objToSpawn_Crew.AddComponent<set_color_crew>();
    var inComponent = objToSpawn_Crew.GetComponent<set_color_crew>();
    inComponent.color_crew = dictionaryC[data.parent];


    //crew as a child of date;
    objToSpawn_Crew.transform.SetParent(dayi.transform);
}
```

Figura 42 Il dizionario C associa ad ogni Crew un Colore

4. Ogni crew ha uno script con una variabile colore pubblica. Questo colore è uguale per tutte le crew con stesso nome.

L'ultimo passaggio è quello di trasferire il colore al workspace, così che questo cambi il colore in funzione della crew associata.

La colorazione avviene variando il colore del materiale che compone il cubo "*workspace*".

Questo è possibile farlo attraverso il componente *"Renderer"* del *"GameObject"*.

```
1. {
2.     rend = GetComponent<Renderer>();
3.     rend.material.color = color_workspace;
4. }
```

I risultati ottenuti sono mostrati nelle immagini seguenti:

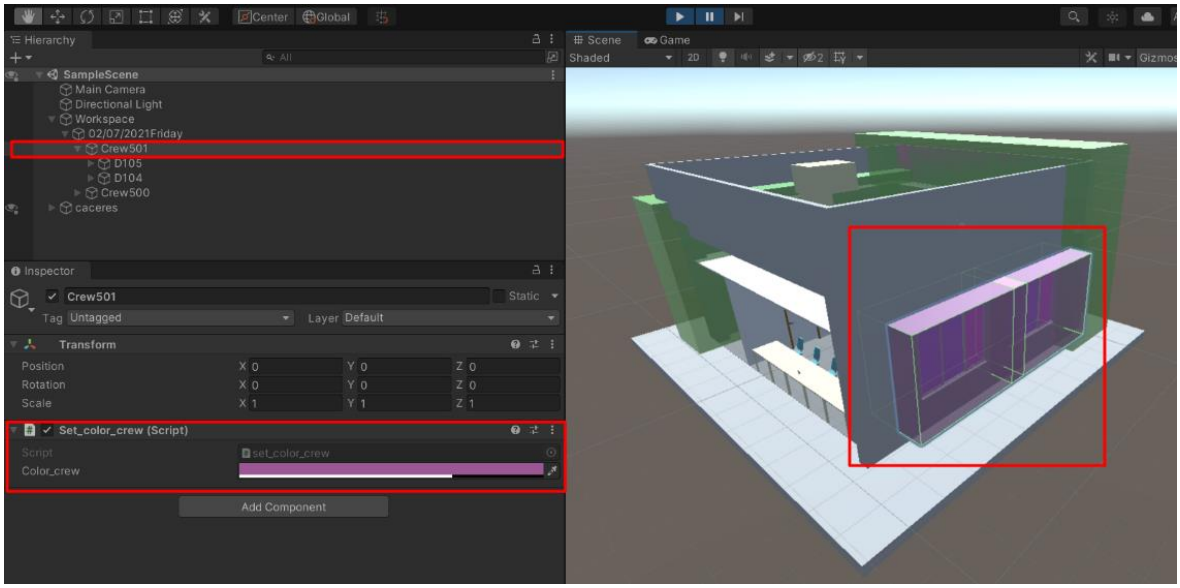

Figura 43 I colori degli spazi di lavoro vengono cambiati in funziona della crew che li esegue

La *crew501* il giorno 02/07/2021 esegue attività in due differenti deliverable, due finestre della facciata EST.

Entrambi sono colorati di viola, il colore che rappresenta i deliverable lavorati dalla *crew501*.

Quelli verdi sono lavorati dalla *crew500.*

Ricordo che la generazione dei colori è randomica e processata ad ogni inizio di simulazione quindi varia in funzione della stessa.

Non in tutte le simulazioni le crew mantengono lo stesso colore di deliverable.

Sotto riporto un esempio analogo al precedente ma con colori differenti dei deliverable lavorati da Crew501:

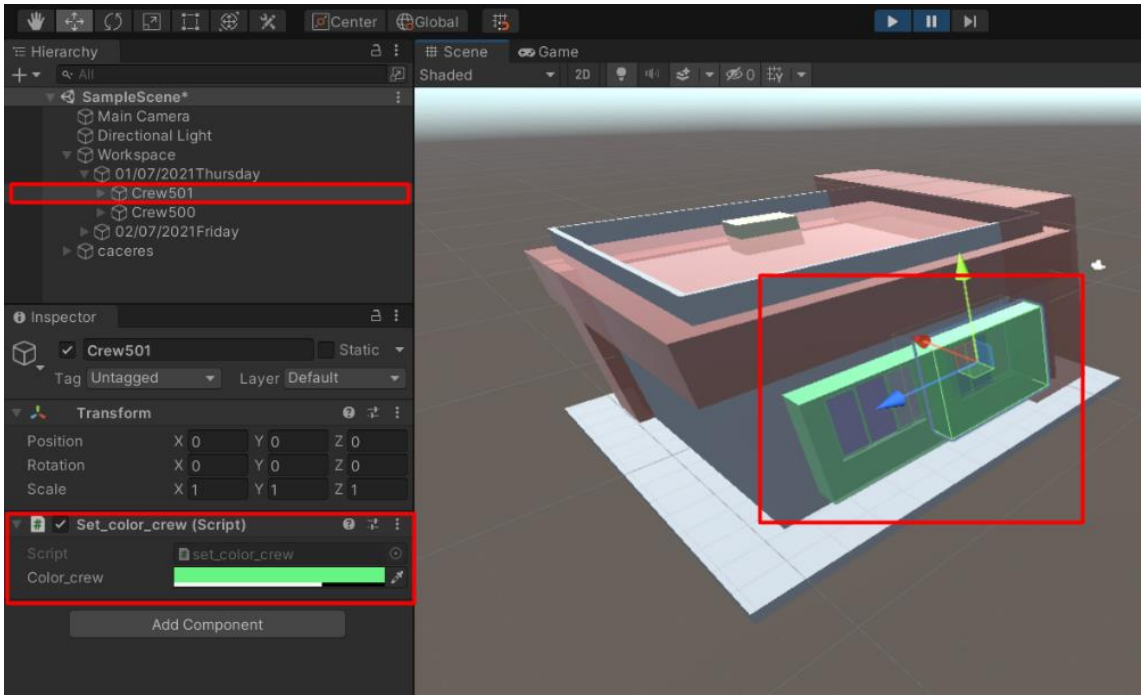

Figura 44 Ulteriore esempio riguardo la colorazione degli spazi di lavoro

5.4.6 FASE 4: OUTPUT SIMULAZIONE: LISTA DELLE SOVRAPPOSIZIONI DI SPAZI DI LAVORO

L'obbiettivo della simulazione è permette di riconoscere le sovrapposizioni di più spazi di lavoro al fine di modificare il piano dei lavori.

Il riconoscimento delle sovrapposizioni viene poi esportato in un file JSON.

Dunque il passaggio di esportazione si articola in due task differenti:

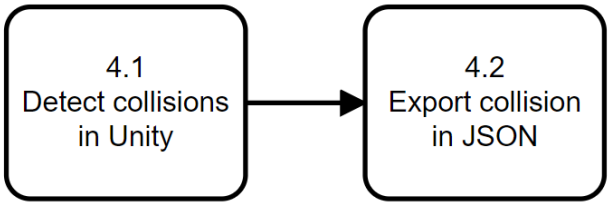

Figura 45 Il riconoscimento delle intersezioni è composto da un task di rilevazione delle collisioni ed uno di esportazione

(4.1) DETECT COLLISION IN UNITY

Le collissioni sono sovrapposizioni di cubi che rappresentano workspace.

Ogni cubo rappresenta un workspace.

Ogni workspace è processato da una squadra in una determinata data.

L'obbiettivo della rilevazione delle collisioni è quello di riconoscere collisioni suddivise per data e per crew.

Significa che se due deliverable eseguiti dalla stessa squadra si sovrappongono nella stessa giornata, questi non verranno notificati.

A questo scopo, viene utilizzata la funzione Unity *"Physics.OverlapBox"*.

Lo scopo è quello di generare uno script con questa funzione ed aggiungerlo alle stazioni di lavoro.

In questa maniera, lo script genera un box immaginario con le stesse dimensioni e rotazione del gameObject a cui è stato aggiunto lo script.

Ogni altro elemento che intercetta il box immaginario viene riconosciuto ed in questa prima fase stampato in console.

Al fine di riuscire ad intercettare le collisioni gli elementi devono avere un componente *"BoxCollider"* attivo.

L'individuzione delle intersezioni dei workspace viene avviata all'attivazione di Void *"MyCollision"*.

Questa combinata alla funzione *"Physics.OverlapBox"* restituisce la lista di elementi che intersecano il box.

La funzione può essere sensibile solo ad alcuni layer. In questa maniera è stato importato come layer di riferimento nel nostro caso *"workspace"*.

```
1. public void MyCollisions()
2.     {
3.
4.         //variables this transform
5.         var m_Collider = this.GetComponent<BoxCollider>();
6.         var parentDel = this.transform.parent;
7.         var parentCrew = parentDel.transform.parent;
8.         var date = parentCrew.transform.parent;
9.         m_Collider.enabled = false;
10.        Collider[] hitColliders = Physics.OverlapBox(gameObject.transf
   orm.position, transform.localScale / 2, Quaternion.identity, m_LayerMa
   sk);
11.        int i = 0;
12.
13. //Check when there is a new collider coming into contact with the box
14.        while (i < hitColliders.Length)
15.        {
16. //variables hitColliders[i]
17.            var parentDelDetected = hitColliders[i].transform.parent;

18.            var parentCrewDetected = parentDelDetected.transform.paren
   t;
19.            var dateDetected = parentCrewDetected.transform.parent;
20.
21. //Debug Collision
22.            if (date.name == dateDetected.name && parentCrew.name != p
   arentCrewDetected.name)
23.            {
24.              string readerActivity = string.Join(",", listActivity);

25.                Debug.Log(date.name + "Collision:" + transform.name +
   "by" + parentCrew.name + "#" + hitColliders[i].name + "by" + parentCre
   wDetected.name + "Activity" + readerActivity);
26.            }
27.            i++;
28.        }
29.        m_Collider.enabled = true;
30.    }
```

Si noti che:

i) prima di indagare quali *BoxCollider* intersecano il cubo immaginario è stato disattivato quello dell'oggetto stesso.

In questa maniera viene esclusa l'intersecazione fallacea che determinerebbe lo stesso deliverable intersecando se stesso.

ii) la stampa delle sovrapposizioni avviene solo se due zone di lavoro:

a) hanno stesso elemento data: sono eseguiti lo stesso giorno,

b) hanno elemento crew differente: sono eseguiti da cew differenti.

Il risultato ottenuto è una stampa in console Unity delle sovrapposizioni:

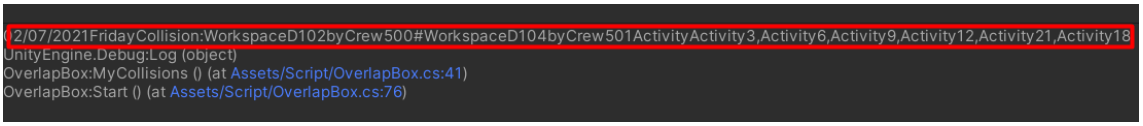

Figura 46 il primo passaggio della rilevazione delle collisioni in Unity termina con la stampa delle stesse

Di seguito viene mostrata la collisione interecettata dalla simulazione fra il D102 lavorato dalla Crew 500 e il D104 lavorato dalla crew 501:

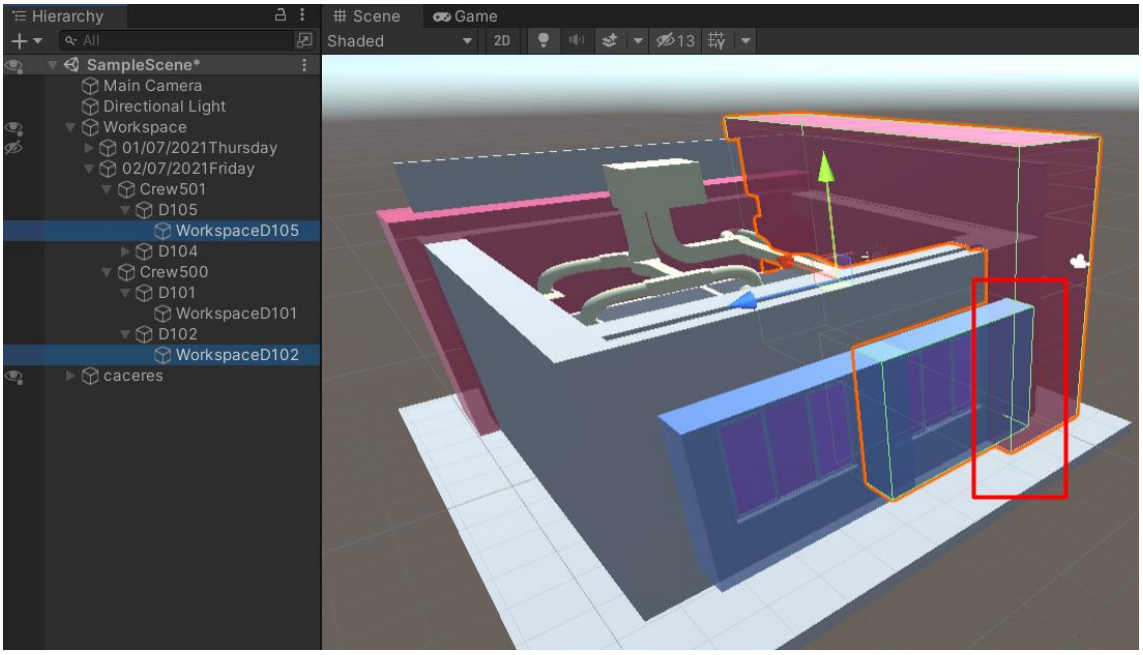

Figura 47 Dimostrazione di sovrapposizione di due zone di lavoro

Di seguito, inoltre, viene mostrato come vengono esclusi dall'esportazione dei conflitti: *A)* il conflitto che avviene fra i deliverable D100 e D101 perché lavorati dalla stessa crew, infatti colorati entrambi di color rosso.

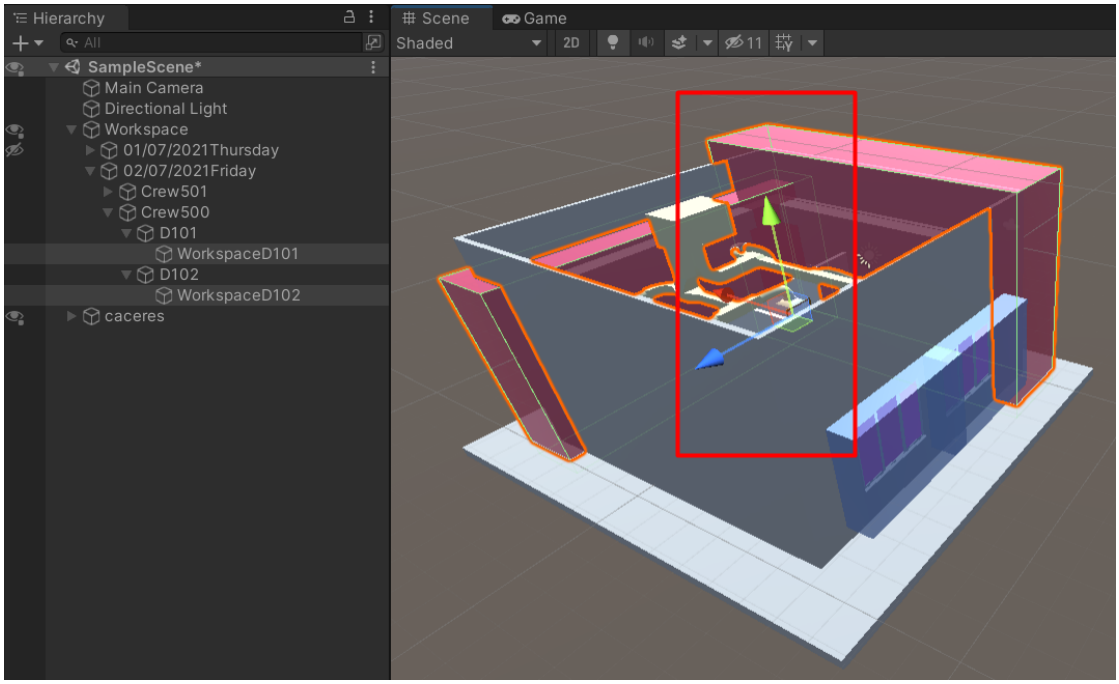

Figura 48 Deliverables dello stesso colore sono processari dalla stessa crew, quindi non definiscono una collisione 1

B) il conflitto che avviene fra i deliverable D105 e D104 perché lavorati dalla stessa crew, infatti colorati entrambi di color blu.

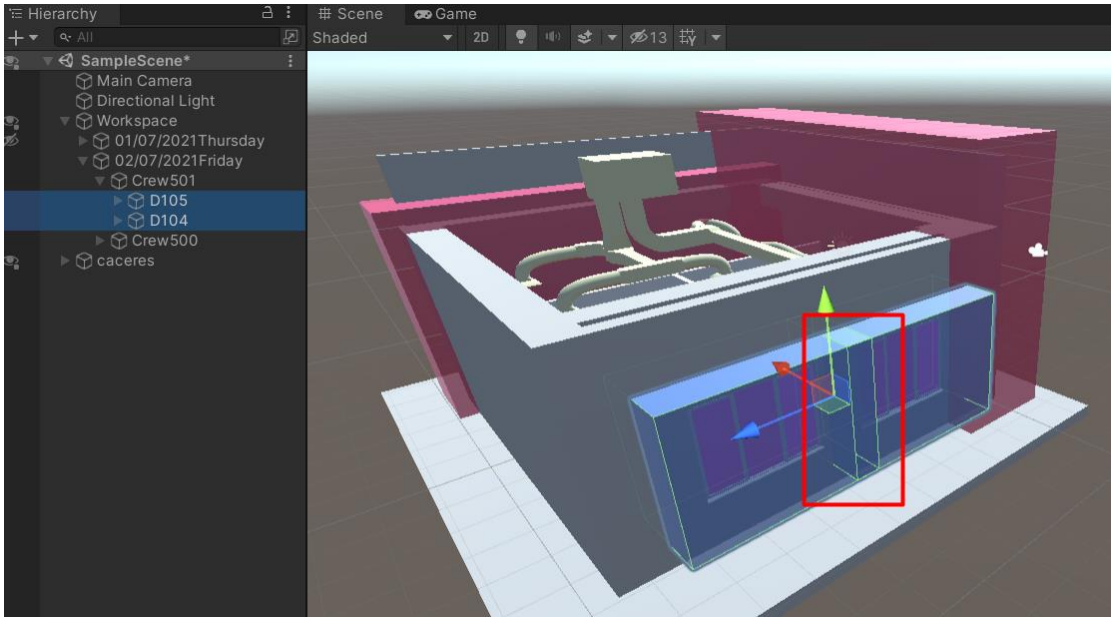

Figura 49 Deliverables dello stesso colore sono processati dalla stessa crew, quindi non definiscono una collisione

(4.2) EXPORT COLLISION IN JSON

Unity permette di compilare file.JSON e quindi compiere l'esportazione dei conflitti in formato JSON.

A tale scopo è necessario definire la struttura di esportazione.

Si è deciso di suddividere i conflitti per data e per crew.

Ogni conflitto è definito da una coppia di sovrapposizioni.

Quella del primo deliverable sul secondo e viceversa quella del secondo sul primo.

Perciò la struttura di esportazione JSON proposta è la seguente:

```
{"date": "",
"conflicts": [
              {
                     "crew": "",
                     "activity": null,
                     "deliverable": ""
            },{
                     "crew": "",
                     "activity": null,
                     "deliverable": ""
                },]
"interference": "",
}
```

In questa fase si è deciso di non esportare la lista delle attività perché intese come un pacchetto di lavoro sul dato deliverable.

Qualora volesse essere esportata, la lista delle attività è già strutturata come variabile pubblica dello script *"merge_workspace"*.

Per esportare oggetti in formato Json è necessario:

1.

Definire una struttura di classi pubbliche coerente alla struttura sopra citata.

Ogni classe deve essere preceduta da `[System.Serializable]`.

In questa maniera è possibile utilizzare le funzioni di JsonUtility.

Per questo è stata definita una classe **`Collision`** ed una lista di **`Conflicts.`**

`Collision` rappresenta l'oggetto della sovrapposizione ed è caratterizato da una data, un'interferenza percentuale ed una lista di conflitti.

`Conflicts` invece rappresenta l'oggetto che determina il conflitto.

Quindi comprende una crew, un campo attività ed uno deliverable.

In ogni collisione saranno presenti due oggetti conflitto.

```
1. [System.Serializable]
2.  public class Collision
3.  {
4.      public string day;
5.      public List<Conflicts> conflicts;
6.      public string interference;
7.  }
8.  [System.Serializable]
9.  public class Conflicts
10.       {
11.           public string crew;
12.           public string activity;
13.           public string deliverable;
14.       }
```

2.

Dopo aver rilevato le sovrapposizioni è necessario definire la data della collissione e i due oggetti conflitto.

```
1. Collision objCollision = new Collision();
2.           objCollision.day = date.name;
3.
4.           List<Conflicts> conflictsList = new List<Conflicts>();

5.           objCollision.conflicts = conflictsList;
6.
7.           Conflicts myObject1 = new Conflicts();
8.           myObject1.deliverable = transform.name;
9.           myObject1.activity = null;
10.          myObject1.crew = parentCrew.name;
11.          conflictsList.Add(myObject1);
12.
13.           Conflicts myObject2 = new Conflicts();
14.           myObject2.deliverable = hitColliders[i].name;
15.           myObject2.activity = null;
16.           myObject2.crew = parentCrewDetected.name;
17.           conflictsList.Add(myObject2)
```

3.

Con la funzione `JsonUtility.ToJson` l'oggetto `objCollision` è formattato nella stringa `SaveToString` con formato JSON.

Successivamente con la funzione `File.AppendText`, la stringa viene aggiunta ad ogni ciclo al file di testo `Save.txt`.

```
1. string SaveToString = JsonUtility.ToJson(objCollision, true);
2.
3. using (StreamWriter sw = File.AppendText(@"D:\Programmi\Save.txt
   "))
4.                 {
5.                     sw.WriteLine(SaveToString);
6.                 }
```

Il risultato dell'esportazione è mostrato di seguito:


```
{
    "day": "02/07/2021Friday",
    "conflicts": [
        {
            "crew": "Crew501",
            "activity": "",
            "deliverable": "WorkspaceD105"
        },
        {
            "crew": "Crew500",
            "activity": "",
            "deliverable": "WorkspaceD102"
        }
    ],
    "interference": ""
}
{
    "day": "02/07/2021Friday",
    "conflicts": [
        {
            "crew": "Crew500",
            "activity": "",
            "deliverable": "WorkspaceD102"
        },
        {
            "crew": "Crew501",
            "activity": "",
            "deliverable": "WorkspaceD105"
        }
    ],
    "interference": ""
}
```

5.4.7 SVILUPPI FUTURI: MISURAZIONE DELLA SOVRAPPOSIZIONE VOLUMETRICA DELLE COLLISIONI

Obbiettivo di questo passaggio è quello di misurare il volume di sovrapposizione delle collisioni.
Ovvero misurare il volume dell'interferenza per valutarne l'entità.
In questa maniera, si aggiunge un altro parametro decisionale così da escludere collisioni che interessano volumetrie inferiori ad un determinato valore percentuale.
La gravità della sovrapposizione è definita come:

GRAVITA' = (VOLUME DELL'INTERSEZIONE) / (VOLUME DEL DELIVERABLE MINORE) LIMITATO AL 100%.

In questa maniera:
- gravità pari al 100% corrisponde a massima interferenza.
Il deliverable più grande ingloba completamente il più piccolo.
- gravità pari al 0% corrisponde a intereferenza nulla. I deliverable non si sovrappongono.

Tuttavia al momento Unity3D non presenta una funzione che definisce in automatico il valore di un volume di interesezione fra due boxCollider.
Perciò di seguito vengono rappresentati i metodi seguiti e i limiti riscontrati:

A) L'unica funzione che valuta l'interferenza fra due box collider è la seguente *Bounds.Intersects.*
Questa valuta se un box collider collide con altri box collider, tuttavia il risultato che restituisce è una booleana.

B) Unity permette di lavorare con metodi legati alla funzione Bounds.
Questi sono di natura geometrica e il loro utilizzo nel contesto corrente porterebbe ad un processo articolato.

Questi sono:

- Bounds.center = restituisce il centro del collide.r

- Bounds.min o max = definisce l'estensione rispetto al punto (0,0,0)

Di seguito ne è rappresentato un esempio:

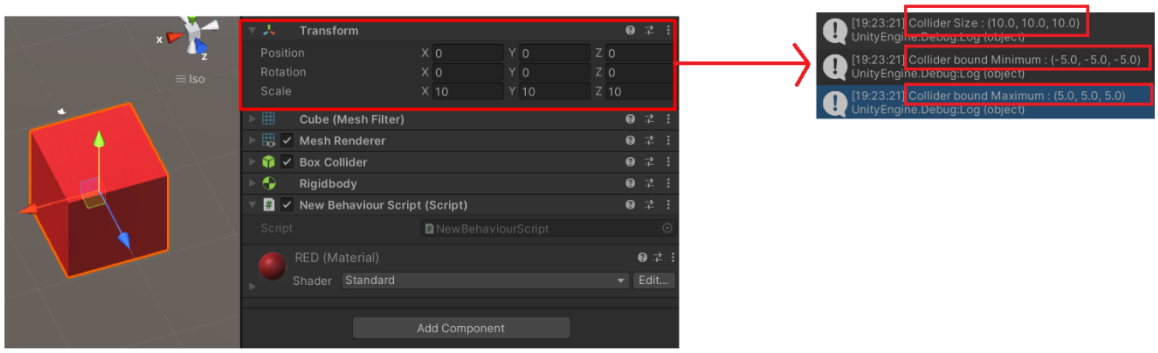

Figura 50 Esempio di calcolo di bounds.center e bound.min e max

Altre funzioni simili permettono di compiere valutazioni riguardo alla posizione nello spazio del box. L'output è sempre una coordinata o estensione nello spazio.

Presupponendo che, sia possibile definire più punti di due box (ad esempio quelli tondi in figura), rimangono i seguenti problemi riguardo questo metodo geometrico:

i) come definire se un punto è interno alla superficie del secondo.

ii) come definire in Unity a quale spigolo facciamo riferimento durante il calcolo.

Dovrei avere una logica per la quale definisca che il punto trovato sia in una determinata posizione rispetto ad un altro punto. Se ci basassimo sulle coordinate di due punti, in che modo queste devono essere confrontate?

Non è detto che quella con coordinate minore sia per forza quella inclusa nel secondo Box e viceversa.

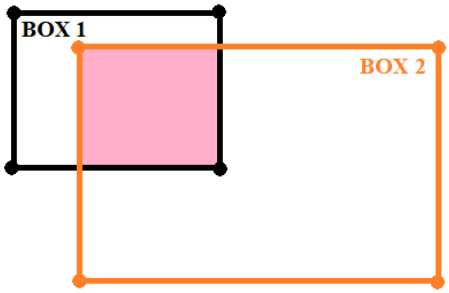

Figura 51 Ipotetica sezione trasversale di due box

Ipoteticamente sarebbe necessario avere una mappatura di tutti i punti nello spazio della sovrapposizione e confrontare il punto di riferimento con quelli della mappatura.
Al momento il metodo non è stato approfondito.

C) Unity permette di definire un oggetto Collision come l'interesezione di due Box.
Una funzione legata al Collision è GetContacs. Questa restituisce un array di punti che sono l'interesezione di due Box. Il metodo è stato testato.
Questo è lo script:

```
void OnCollisionStay(Collision collisionInfo)
{
    //variables this transform
    var m_Collider1 = this.GetComponent<BoxCollider>();
    var parentDel1 = this.transform.parent;
    var parentCrew1 = parentDel1.transform.parent;
    var date1 = parentCrew1.transform.parent;

    //variables hitColliders[i]
    var parentDelDetected1 = collisionInfo.transform.parent;
    var parentCrewDetected1 = parentDelDetected1.transform.parent;
    var dateDetected1 = parentCrewDetected1.transform.parent;

    if (date1.name == dateDetected1.name && parentCrew1.name != parentCrewDetected1.name)
    {
        Debug.Log(date1.name + "This" + this.gameObject.name + "#" + collisionInfo.gameObject.name);
        List<ContactPoint> contacts = new List<ContactPoint>();
        collisionInfo.GetContacts(contacts);
        foreach (var contact in contacts)
        {
            GameObject intersection = Instantiate(Resources.Load("Sphere")) as GameObject;
            intersection.transform.position = contact.point;
        }
    }
}
```

Figura 52 Vengono instanziati nei punti di contatto delle sfere per compiere valutazioni

Il risultato ottenuto:

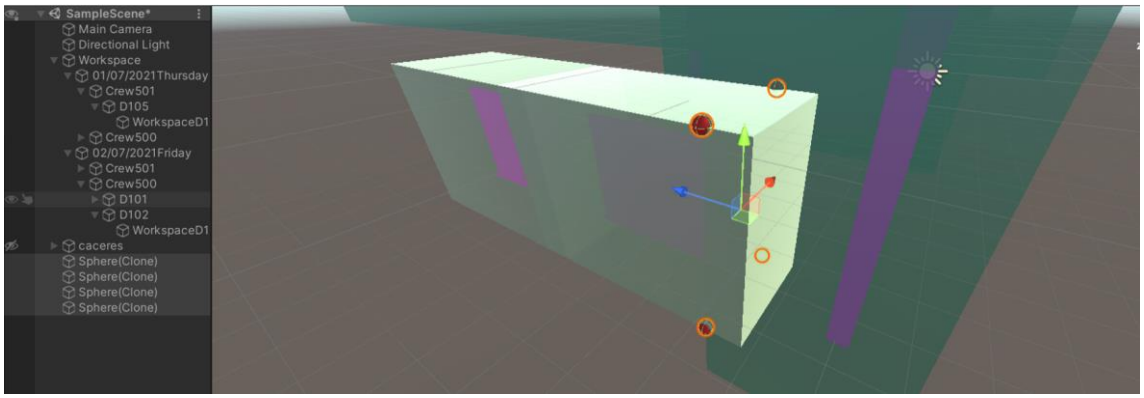

Figura 53 In corrispondenza dei punti rilevati viene instanziata una sfera

I punti di contatto che vengono restituiti sono quelli di entrata di un box nell'altro.
In questa maniera la percezione della profondità di interesezione non è restituita.

Nel caso in cui avesse restituito anche i punti "di profondità" sarebbe stato possibile costruire su quei punti un ulteriore cubo di cui calcolare il volume.

Altre proprietà della classe Collision sono documentate nel seguente link:
https://docs.unity3d.com/ScriptReference/Collision.html

6. DISCUSSIONE DEI RISULTATI E CONCLUSIONI

Il sistema proposto, non ha l'intendo di sostituire metodi di pianificazione già presenti nell'industria delle costruzioni, ma quello di affiancarli definendo un metodo decisionale che vada oltre la semplice data ultima di consegna dell'attività.
Valorizza il flusso delle risorse e mira a darne una misura nel tempo.
I metodi che non considerano i flussi delle risorse, risultano miopi al reale flusso produttivo delle costruzioni.
Le attività per poter essere concluse necessitano di risorse.
Qualora non sia disponibile una risorsa, l'attività tarderà ad essere realizzata, le prestazioni dei tecnici diminuiranno e di conseguenza anche la qualità del costruito, intesa come la capacità di rispettare la pianificazione.

Il caso studio, dopo aver analizzato le tecniche di pianificazione dell'industria delle costruzioni si è concentrato sulla tracciabilità della risorsa spazio di lavoro.
Ad ogni attività nel processo di costruzione è associata una zona di lavoro, che se risulta occupata o non totalmente fruibile ne impedisce l'esecuzione del compito al suo interno.
Il metodo di simulazione integra il piano dei lavori, importandolo nell'ambiente di simulazione e deserializzando le attività in esecuzione.
Al momento, le dimensioni dello spazio di lavoro sono definite di default con misure uguali per ogni deliverable, a prescindere dalla lavorazione.
Successivamente, la simulazione è in grado di associare al modello geometrico tridimensionale della costruzione più spazi di lavoro.
Poi, il sistema simula il piano dei lavori ed esporta una lista di collisioni.
Una collisione è definita come la sovrapposizione di due o più zone di lavoro.
Per ogni collisione è definita la data, le crew interessate e i deliverable le cui aree di lavoro si sovrappongono.
Le collisioni sono sensibili sia alla data che alla crew.
Significa che collisioni di deliverable lavorati dalla stessa crew non vengono esportate.
Questo perché una crew non ostacola se stessa.
Lo stesso ragionamento è stato applicato all'orizzonte temporale.
Affinchè avvenga una collisione, i deliverable devono essere processati lo stesso giorno.
Nello specifico è di seguito illustrato un esempio di applicazione.

Viene deserializzato in Unity un file .json il cui estratto è allegato in appendice 7.4. Questo rappresenta l'esportazione del cronoprogramma dalla piattaforma AWOPS. All'interno del cronoprogramma in formato json, sono tre gli oggetti di particolare rilevanza:

i) `"Baseline.Data":` Definiscono le date in cui le Attività vengono processate:

ii) `"activities":` Definiscono il deliverable in cui avviene l'Attività.

iii) `"deliverables":` Definiscono la lista di elementi che compongolo il deliverable attraverso la lista di stringe Guid.

Figura 54 Gli oggetti Baseline.Data, activities e deliverables sono quelli usati dalla simulazione

Attraverso l'interfaccia utente viene definita la data di simulazione del cronoprogramma *(rettangolo rosso)* e l'orizzonte temporale *(rettangolo arancione)*.

In questo caso:

- la data di simulazione è il: 1/07/202;
- il numero di giorni di simulazione è: 2;

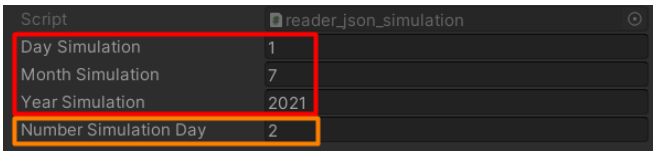

Figura 55 Gli input della data di simulazione e del numero di giorni sono numeri interi

Avviata la simulazione vengono confrontate la date della simulazione, con le date delle attività negli oggetti Baseline.data.

Perciò, prese in considerazione le attività con data coerente a quella della simulazione e rappresentate nella simulazione.

Nell'esempio:

- viene importato il modello geometrico della costruzione;
- generati gli oggetti :"Day 01/01/2021" e "Day 02/01/2021";
- generati gli oggetti crew e deliverable delle azioni con date di esecuzione coerente a quelle della simulazione.

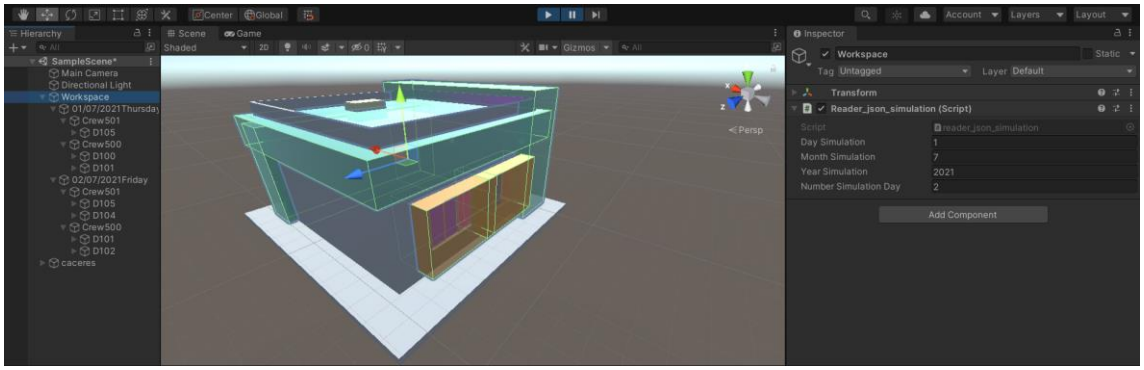

Figura 56 Il modello è importato ed il cronoprogramma deserializzato

In automatico, viene compilato il file.txt posizionato nella cartella Asset con le sovrapposizioni rilevate dalla simulazione.

La struttura di esportazione è coerente alla struttura json.

Di seguito viene allegato:


```
{
  "day": "02/07/2021Friday",
  "conflicts": [
    {
      "crew": "Crew501",
      "activity": "",
      "deliverable": "WorkspaceD105"
    },
    {
      "crew": "Crew500",
      "activity": "",
      "deliverable": "WorkspaceD102"
    }
  ],
  "interference": ""
}
```

In questo caso, viene rilevata la collisione con data 02/07/2021 fra la Crew501 che lavora nel WorkspaceD105 e la Crew500 che lavora nel WorkspaceD102.
Infatti, come mostra la figura seguente, le due zone di lavoro collidono:

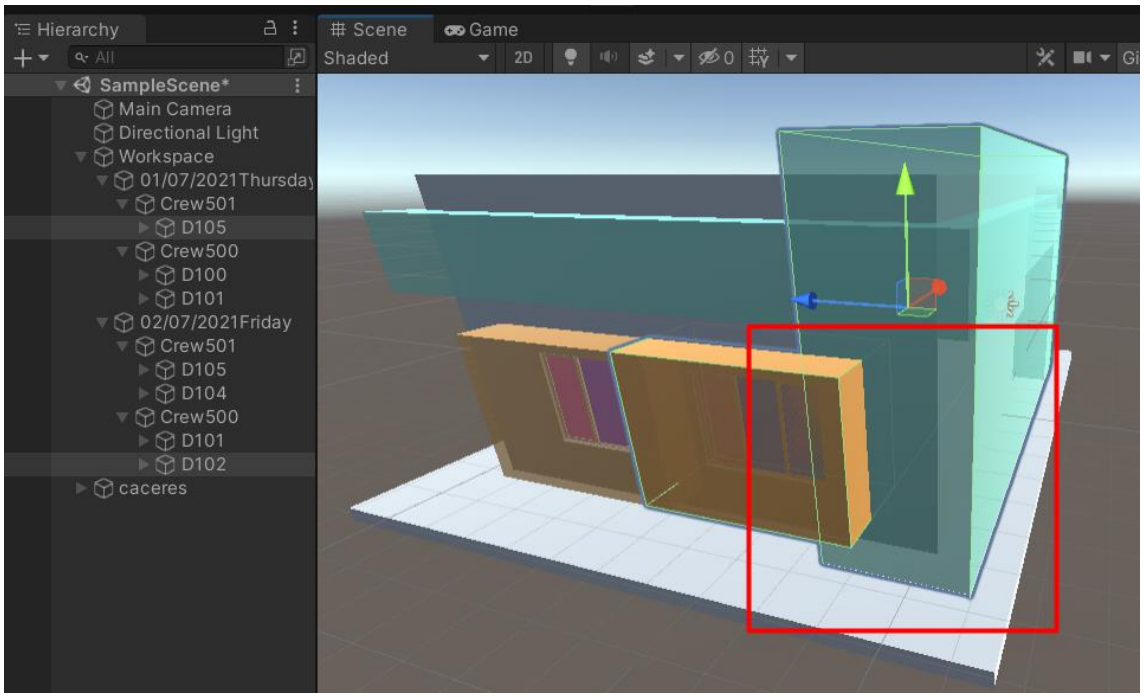

Figura 57 La sovrapposizione è nettamente visibile grazie alla colorazione differente delle zone di lavoro

Senza uno strumento capace di riconoscere sovrapposizioni di più zone di lavoro, alcune attività potrebbero essere avviate senza poter essere terminate con processi di produzione fluidi privi di interruzioni.

In questa maniera, è possibile usufruire della simulazione ogni volta che viene rilevata una collisione.

Quindi, modificare la sequenza delle attività o lo spazio necessario alla lavorazione e iterare il processo fino a quando la simulazione rileva zero collisioni.

Per questo, la simulazione del caso studio si inserisce nell'industria delle costruzioni come uno strumento capace di vagliare più ipotesi progettuali al fine di scegliere quella che determina la massima ottimizzazione.

7. APPENDICE

7.1 "READER_JSON_SIMULATION" SCRIPT

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System;
using System.Globalization;
using System.Linq;
using System.Text.RegularExpressions;
using Piglet;
using Random = UnityEngine.Random;

public class reader_json_simulation : MonoBehaviour
{
    //PIGLET GLTF TASK AND GAMEOBJECT
    private GltfImportTask _task;
    private GameObject _model;
    private GameObject toDelete;

    //Public class for baseline.json deserialize caceres
    [System.Serializable]
    public class Property
    {
        public string Name;
        public string Value;
    }
    [System.Serializable]
    public class PropertySet
    {
        public string propertySetName;
        public List<Property> properties;
    }
    [System.Serializable]
    public class IfcInfo
    {
        public string Name;
        public string Value;
    }
    [System.Serializable]
    public class Element
    {
        public int ifcSystemIndex;
        public int ifcStoreyIndex;
        public int ifcSpaceIndex;
        public List<PropertySet> propertySets;
        public string ifcExpressTypeName;
        public string ifcGuid;
        public int ifcEntityLabel;
        public List<IfcInfo> ifcInfo;
    }
```

```
    [System.Serializable]
    public class Storey
    {
        public double Elevation;
        public string ifcExpressTypeName;
        public string ifcGuid;
        public int ifcEntityLabel;
        public List<IfcInfo> ifcInfo;
    }
    [System.Serializable]
    public class System2
    {
        public string ifcExpressTypeName;
        public string ifcGuid;
        public int ifcEntityLabel;
        public List<IfcInfo> ifcInfo;
    }
    [System.Serializable]
    public class Root
    {
        public string geometryExporterVersion;
        public string uniZiteMetadataExporterVersion;
        public string ifcSchemaVersion;
        public List<Element> elements;
        public List<System2> systems;
        public List<Storey> storeys;
    }
    //Public class for baseline.json deserialize
ACOin_xxxx_baseline
    [System.Serializable]
    public class Site
    {
        public string x_comment;
        public string name;
        public int unitCost;
        public DateTime startDateTime;
        public DateTime dueDateTime;
        public string currency;
        public string timeUnit;
        public string distanceUnit;
    }
    [System.Serializable]
    public class WorkingTime
    {
        public string from;
        public string to;
    }
    [System.Serializable]
    public class WeekDay
    {
        public string x_comment;
        public int dayOfWeek;
        public int workingHours;
        public string dayName;
        public string isWorking;
        public List<WorkingTime> workingTimes;
```

```
    }
    [System.Serializable]
    public class Calendar
    {
        public List<WeekDay> weekDays;
    }
    [System.Serializable]
    public class Size
    {
        public string x_comment;
        public double value;
        public string unit;
    }
    [System.Serializable]
    public class Location
    {
        public string x_comment;
        public string facade;
        public string level;
    }
    [System.Serializable]
    public class Deliverable
    {
        public string x_comment;
        public int id;
        public string name;
        public Size size;
        public Location location;
        public List<object> startDateTime;
        public List<object> dueDateTime;
        public List<string> guids;
    }
    [System.Serializable]
    public class Crew
    {
        public string x_comment;
        public int crewID;
        public List<double> unitLabour;
        public int setupMatrixID;
        public int id;
        public string name;
        public List<Worker> workers;
        public object equipment;
        public double speed;
    }
    [System.Serializable]
    public class Activity
    {
        public string x_comment;
        public int id;
        public string priceListCode;
        public string name;
        public string start;
        public string end;
        public int progress;
        public Crew crew;
```

```
        public object preconditions;
        public int deliverables;
        public DateTime startDateTime;
        public DateTime dueDateTime;
        public double size;
    }
    [System.Serializable]
    public class Worker
    {
        public string x_comment;
        public int id;
        public string priceListCode;
        public string name;
        public double unitCost;
        public int quantity;
    }
    [System.Serializable]
    public class Equipment
    {
        public int id;
        public string priceListCode;
        public string name;
        public double unitCost;
    }
    [System.Serializable]
    public class OperationalResources
    {
        public List<Worker> workers;
        public List<Equipment> equipment;
        public List<Crew> crew;
    }
    [System.Serializable]
    public class SetupMatrix
    {
        public int id;
        public string x_comment1;
        public string x_comment2;
        public List<int> deliverables;
        public List<List<double>> matrix;
    }
    [System.Serializable]
    public class BestSol
    {
        public double cost;
        public List<int> task;
        public List<int> crew;
        public List<double> departureTime;
        public List<double> startTime;
        public List<double> endTime;
        public double overhead;
        public double BestCost;
        public List<List<double>> tau;
    }
    [System.Serializable]
    public class Datum
    {
```

```
        public string id;
        public string type;
        public object text;
        public bool open;
        public string start_date;
        public object end_date;
        public string parent;
        public string planned_start;
        public string planned_end;
        public object duration;
        public object progress;
    }
    [System.Serializable]
    public class Links
    {
        public int id;
        public int source;
        public int target;
        public int type;
    }
    [System.Serializable]
    public class Baseline
    {
        public List<Datum> data;
        public Links links;
    }
    [System.Serializable]
    public class Root2
    {
        public string x_comment;
        public Site site;
        public Calendar calendar;
        public List<Deliverable> deliverables;
        public List<Activity> activities;
        public OperationalResources operationalResources;
        public List<SetupMatrix> setupMatrix;
        public BestSol BestSol;
        public Baseline baseline;
    }

    //0.Simulation time horizon variables
    public int daySimulation;
    public int monthSimulation;
    public int yearSimulation;
    public int numberSimulationDay;
    //1.BASELINE
    Baseline baselineOb = new Baseline();
    List<Datum> datumList = new List<Datum>();
    //1.1DeliverableList + GuidList
    List<Deliverable> deliverableList = new List<Deliverable>();
    List<string> GuidList = new List<string>();
    //1.2ActivitiesList + Crew
    List<Activity> ActivityList = new List<Activity>();
    Crew crewOb = new Crew();
    //2.CACERES
    List<Element> elementList = new List<Element>();
```

```
    List<System2> systemList = new List<System2>();
    private string output;
    //3.store gameObject to spawn reference
    GameObject objToSpawn_Day;
    GameObject objToSpawn_Deliverable;
    GameObject objToSpawn_Activity;
    GameObject objToSpawn_Crew;

    void Start()
    {
        _task = RuntimeGltfImporter.GetImportTask(

"D:/Univpm/2.MAGISTRALE/TESI/20.Importazione/caceres.gltf");
        _task.OnCompleted = OnComplete;
        toDelete = GameObject.Find("Variables Saver");
        Destroy(toDelete);
        File.WriteAllText(@"D:\Programmi\13. UNITY\Progetti
salvati\0.Simulation_Collision_Workspace\Assets\Save.txt",
string.Empty);
    }
    public void OnComplete(GameObject importedModel)
    {
        //Import the model .gltf
        _model = importedModel;
        //Debug.Log("Success!");
        //when the model is loaded:
        GameObject childLevel1 =
_model.transform.GetChild(0).gameObject;
        var transformChildLevel1 = childLevel1.transform;
        foreach (Transform child in transformChildLevel1)
        {
            string initialName = child.name;
            output = initialName.Split('#').Last();
            child.name = output;
        }

        //CALENDAR:StartDateSimulation + EndDateSimulation
        DateTime StartDateSimulation = new DateTime(
yearSimulation, monthSimulation, daySimulation);
        DateTime EndDateSimulation =
StartDateSimulation.AddDays(numberSimulationDay);
        //GENERATE GAMEOBJECT DAYS
        var ListDaySimulation = new List<DateTime>();
        for (DateTime date = StartDateSimulation; date <
EndDateSimulation; date = date.AddDays(1))
        {
            DayOfWeek day = date.DayOfWeek;
            if ((day != DayOfWeek.Saturday) && (day !=
DayOfWeek.Sunday))
            {
                objToSpawn_Day = new GameObject();
                objToSpawn_Day.name = date.ToString("d") +
date.DayOfWeek;
                objToSpawn_Day.transform.SetParent(this.transform);
                //List of day simulation
                var DaySimulation = date;
```

```
                ListDaySimulation.Add(DaySimulation);
            }
        }
        //1. DESERIALIZE BASELINE FILE JSON
        string json = File.ReadAllText(Application.dataPath +
"/2021-09-06 ACOin_caceresUpdated_baseline.json");
        // Root2 contain all the data from JSON file Baseline
        Root2 rootObject;
        rootObject = new Root2();
        // FromJson is the function which Deserialize file.json
        rootObject = JsonUtility.FromJson<Root2>(json);

        //2. DESERIALIZE CACERES FILE JSON
        string json2 = File.ReadAllText(Application.dataPath +
"/caceres.json");
        // Root contain all the data from JSON file Baseline
        Root rootObject1;
        rootObject1 = new Root();
        // FromJson is the function which Deserialize file.json
        rootObject1 = JsonUtility.FromJson<Root>(json2);
        //GENERATE LISTS CACERES JSON OBJECTS
        elementList = rootObject1.elements;

        //GENERATE DICTIONARIES LISTS IN BASELINE JSON
        datumList = rootObject.baseline.data;
        ActivityList = rootObject.activities;
        deliverableList = rootObject.deliverables;
        //A. list of data
        var dictionarylist1 = new List<Datum>();
        foreach (Datum data in datumList)
        {
            if (data.type == "task")
            {
                var pas1 = data;
                dictionarylist1.Add(pas1);
            }
        }
        List<Datum> queryOrder = dictionarylist1.OrderBy(data =>
PadNumbers(data.id)).ToList();
        static string PadNumbers(string input)
        {
            return Regex.Replace(input, "[0-9]+", match =>
match.Value.PadLeft(10, '0'));
        }

        var dictionaryarray1 = queryOrder.ToArray();
        //B. list of activities
        var dictionarylist2 = new List<Activity>();
        foreach (Activity Activity in ActivityList)
        {
            var pas2 = Activity;
            dictionarylist2.Add(pas2);
        }
        var dictionaryarray2 = dictionarylist2.ToArray();
        //C. list of deliverables
        var dictionarylist3 = new List<Deliverable>();
```

```
        foreach (Deliverable deliverable in deliverableList)
        {
            var pas3 = deliverable;
            dictionarylist3.Add(pas3);
        }
        var dictionaryarray3 = dictionarylist3.ToArray();

        //GENERATE DICTIONARY A: DATA-ACTIVITY
        Dictionary<Datum, Activity> dictionaryA = new
Dictionary<Datum, Activity>();
        for (int i = 0; i < dictionaryarray1.Length; i++)
        {
            dictionaryA.Add(dictionaryarray1[i],
dictionaryarray2[i]);
        }
        //Generate dictionary B
        //Dictionary<Activity,Deliverable> dictionaryB = new
Dictionary<Activity, Deliverable>();

        //GENERATE DICTIONARY C: CREW-COLOR
        var stringNameCrewList = new List<string>();

        foreach (Datum data in datumList)
        {
            if (data.parent == "Project")
            {
                var stringPass = data.id;
                stringNameCrewList.Add(stringPass);
            }
        }

        Dictionary<String, Color> dictionaryC = new
Dictionary<String, Color>();

        for (int i = 0; i < stringNameCrewList.Count; i++)
        {
            Color colorCrewPass = new Color(Random.value,
Random.value, Random.value, 0.70f);
            dictionaryC.Add( stringNameCrewList[i], colorCrewPass);
        }



        //BASELINE
        foreach (Datum data in datumList)
        {
            if (data.type == "task")
            {
                // from string to a DateTime with DateTime.TryParse
                string dateStringStart = data.planned_start;
                string dateStringEnd = data.planned_end;
                DateTime dtStart;
                var BooleanStartDate =
DateTime.TryParse(dateStringStart, out dtStart);
                DateTime dtStartMidnight = new
DateTime(dtStart.Year, dtStart.Month, dtStart.Day, 0, 0, 0);
```

```
                DateTime dtEnd;
                var BooleanEndDate =
DateTime.TryParse(dateStringEnd, out dtEnd);
                DateTime dtEndMidnight = new DateTime(dtEnd.Year,
dtEnd.Month, dtEnd.Day, 0, 0, 0);
                //GENERATE GAMEOBJECT CREW
                for (DateTime dateCrew = dtStartMidnight; dateCrew
<= dtEndMidnight; dateCrew = dateCrew.AddDays(1))
                {
                    foreach (DateTime dateTime in
ListDaySimulation)
                    {
                        //Compare Simulation DateTime and
Activities DateTime
                        int resultComparison =
DateTime.Compare(dateTime, dateCrew);
                        if (resultComparison == 0)
                        {
                            //1.Spawn gameObject CREW
                            var dayi =
GameObject.Find(dateTime.ToString("d") + dateTime.DayOfWeek);
                            {
                                var crewFind =
dayi.transform.Find(data.parent);
                                if (!crewFind)
                                {
                                    //spawn object
                                    objToSpawn_Crew = new
GameObject();
                                    objToSpawn_Crew.name =
data.parent;

objToSpawn_Crew.AddComponent<set_color_crew>();
                                    var inComponent =
objToSpawn_Crew.GetComponent<set_color_crew>();
                                    inComponent.color_crew =
dictionaryC[data.parent];

                                    //crew as a child of date;

objToSpawn_Crew.transform.SetParent(dayi.transform);
                                }
                            }
                            //2.Spawn gameObject DELIVERABLE
                            var crewFind2 =
dayi.transform.Find(data.parent);
                            //dictionaryA: connect data with
activities
                            Activity temp = null;
                            if (dictionaryA.TryGetValue(data, out
temp))
                            {
                            //connect activities and deliverable
                            var deliverableQuery =
dictionaryarray3.Where(d => d.id ==
temp.deliverables).FirstOrDefault();
```

```
                            var DeliverableFind =
crewFind2.transform.Find("D" + deliverableQuery.id);
                                if (!DeliverableFind)
                                {
                                //spawn object
                                objToSpawn_Deliverable = new
GameObject();
                                //Change name GameObect
                                objToSpawn_Deliverable.name = "D" +
deliverableQuery.id;
                                //Add deliverable as child activity

objToSpawn_Deliverable.transform.SetParent(crewFind2.transform);
                                //Define what is "GuidList"
                                GuidList = deliverableQuery.guids;
                                //Define a list which it can
contain GameObject
                                var passlist = new
List<GameObject>();
                                   foreach (string guids in
GuidList)
                                   {
                                        //elements with
element.ifcGuid == guids in GuidList
                                        var elementQuery =
elementList.Where(d => d.ifcGuid == guids).FirstOrDefault();

                                        string ifcEntityLabelQuery
= elementQuery.ifcEntityLabel.ToString();

                                        var sameNameList =
Resources.FindObjectsOfTypeAll<GameObject>().Where(obj => obj.name
== ifcEntityLabelQuery).ToList();
                                        foreach (GameObject
gameObject1 in sameNameList)
                                        {

passlist.Add(gameObject1);
                                        }
                                   }
                                //Add Components merge_workspace
and gameobject in products

objToSpawn_Deliverable.AddComponent<merge_workspace>();
                                var component =
objToSpawn_Deliverable.GetComponent<merge_workspace>();
                                component.Products =
passlist.ToArray();
                                }
                                //Add data.id=activity in
activityListDeliverable;
                                var DeliverableFind2 =
crewFind2.transform.Find("D" + deliverableQuery.id);
                                var component2 =
DeliverableFind2.GetComponent<merge_workspace>();
```

```
                                    var nameActivity = data.id;
                                    var str = new List<string>();
                                    str.Add(nameActivity);

                                    component2.ActivityD.AddRange(str);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    void Update()
    {
        _task.MoveNext();
    }
```

7.2" MERGE_WORKSPACE" SCRIPT

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

// Copy meshes from array into the empty Gameobject's Mesh.
// CombineInstance stores the list of meshes.  These are combined
and assigned to the attached Mesh.

[RequireComponent(typeof(MeshFilter))]
[RequireComponent(typeof(MeshRenderer))]
public class merge_workspace : MonoBehaviour
{
    //mesh_variables
    public List<string> ActivityD = new List<string>();

    public GameObject[] Products;
    private MeshFilter[] meshFilters;

    //size_variables
    private Vector3 scaleChange;
    private float workspace_sizeX = 1.5f;
    private float workspace_sizeY = 1;
    private float workspace_sizeZ = 1.5f;

    //change color
    Renderer rend;

    void Start()
    {

        meshFilters = new MeshFilter[Products.Length];
        for (int a = 0; a < Products.Length; a++)
        {
            meshFilters[a] =
Products[a].GetComponent<MeshFilter>();
        }
        CombineInstance[] combine = new
CombineInstance[meshFilters.Length];
        int i = 0;
        while (i < meshFilters.Length)
        {
            combine[i].mesh = meshFilters[i].sharedMesh;
            combine[i].transform =
meshFilters[i].transform.localToWorldMatrix;
            meshFilters[i].gameObject.SetActive(true);
            i++;
        }
        transform.GetComponent<MeshFilter>().mesh = new Mesh();

transform.GetComponent<MeshFilter>().mesh.CombineMeshes(combine);
        transform.gameObject.SetActive(true);

        //instanciate workspace
```

```
        GameObject workspaceObj =
Instantiate(Resources.Load("workspace")) as GameObject;
        workspaceObj.transform.SetParent(this.transform);
        //bound size of workspace

this.GetComponentInChildren<Transform>().Find("workspace(Clone)").p
osition = transform.GetComponent<MeshFilter>().mesh.bounds.center;

this.GetComponentInChildren<Transform>().Find("workspace(Clone)").l
ocalScale = transform.GetComponent<MeshFilter>().mesh.bounds.size;
        //set transform.x=+1 ecc
        scaleChange = new Vector3(workspace_sizeX, workspace_sizeY,
workspace_sizeZ);
        workspaceObj.transform.localScale += scaleChange;
        workspaceObj.AddComponent<BoxCollider>();
        // change name according their parent's name
        workspaceObj.name = "Workspace" + this.name;
        // change layer in Workspace
        workspaceObj.layer = LayerMask.NameToLayer("Workspace");
        workspaceObj.AddComponent<OverlapBox_JSON>();

        var layerQuery =
workspaceObj.GetComponent<OverlapBox_JSON>();
        layerQuery.m_LayerMask = LayerMask.GetMask("Workspace");
        layerQuery.listActivity = ActivityD;

    }
}
```

7.3 "OVERLAPBOX" SCRIPT

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System;
using System.Globalization;
using System.Linq;
using System.Text.RegularExpressions;
using Piglet;
using Random = UnityEngine.Random;
using System.Threading.Tasks;

public class OverlapBox_JSON : MonoBehaviour
{
    bool m_Started;
    public LayerMask m_LayerMask;
    public List<string> listActivity = new List<string>();

    //public class for JsonExport

    [System.Serializable]
    public class Collision
    {
        public string day;
        public List<Conflicts> conflicts;
        public string interference;
    }
    [System.Serializable]
    public class Conflicts
    {
        public string crew;
        public string activity;
        public string deliverable;
    }
    public void MyCollisions()
    {
        //variables this transform
        var m_Collider = this.GetComponent<BoxCollider>();
        var parentDel = this.transform.parent;
        var parentCrew = parentDel.transform.parent;
        var date = parentCrew.transform.parent;

        //m_Collider.enabled = false;
        Collider[] hitColliders =
Physics.OverlapBox(gameObject.transform.position,
transform.localScale / 2, Quaternion.identity, m_LayerMask);
        int i = 0;
        //Check when there is a new collider coming into contact
with the box
        while (i < hitColliders.Length)
        {
            //variables hitColliders[i]
```

```
            var parentDelDetected =
hitColliders[i].transform.parent;
            var parentCrewDetected =
parentDelDetected.transform.parent;
            var dateDetected = parentCrewDetected.transform.parent;
            //Debug Collision
            if (date.name == dateDetected.name && parentCrew.name
!= parentCrewDetected.name)
            {

                //string readerActivity = string.Join(",",
listActivity);
                //Debug.Log(date.name + "Collision:" +
transform.name + "by" + parentCrew.name + "#" +
hitColliders[i].name + "by" + parentCrewDetected.name + "Activity"
+ readerActivity);

                Collision objCollision = new Collision();
                objCollision.day = date.name;

                List<Conflicts> conflictsList = new
List<Conflicts>();
                objCollision.conflicts = conflictsList;

                Conflicts myObject1 = new Conflicts();
                myObject1.deliverable = transform.name;
                myObject1.activity = null;
                myObject1.crew = parentCrew.name;
                conflictsList.Add(myObject1);

                Conflicts myObject2 = new Conflicts();
                myObject2.deliverable = hitColliders[i].name;
                myObject2.activity = null;
                myObject2.crew = parentCrewDetected.name;
                conflictsList.Add(myObject2);

                string SaveToString =
JsonUtility.ToJson(objCollision, true);
                Debug.Log(SaveToString);

                using (StreamWriter sw =
File.AppendText(@"D:\Programmi\13. UNITY\Progetti
salvati\0.Simulation_Collision_Workspace\Assets\Save.txt"))
                {
                    sw.WriteLine(SaveToString);
                }
            }
            i++;
        }
        m_Collider.enabled = true;
    }
    void Start()
    {
        MyCollisions();
    }
```

```
    void OnDrawGizmos()
    {
        Gizmos.color = Color.blue;
        if (m_Started)
            Gizmos.DrawWireCube(transform.position,
transform.localScale);
    }
}
```

7.4 PORZIONE BASELINE DI "ACOIN_XXXX_BASELINE.JSON"


```
{
  "deliverables": [{
            "x_comment": "Data about each deliverable of
production.",
            "id": 100,
            "name": "M_Window-Casement-Double:850 x 900mm:1322710",
            "size": {
                "x_comment": "Main size of the object.",
                "value": 0.77,
                "unit": "m^2"
            },
            "location": {
                "x_comment": "Location of the deliverable inside
the jobsite. All fields are optional and conventional. They are
used for defining building topology.",
                "facade": "South",
                "level": "1"
            },
            "startDateTime": [],
            "dueDateTime": [],
            "guids": [ "0d_pDVIf91FRQz7X6BnzDB" ]

        }, {
            "id": 101,
            "name": "M_Window-Casement-Double:1250 x 1890
mm:1324839",
            "size": {
                "value": 2.36,
                "unit": "m^2"
            },
            "location": {
                "facade": "North",
                "level": "1"
            },
            "startDateTime": [],
            "dueDateTime": [],
            "guids": [ "1evipg2MH7OQvwN7M5aGCc",
"2OBPgDlOz0DB$u1TN$GlRJ", "2OBPgDlOz0DB$u1TN$GlRG" ]
        }, {
            "id": 102,
            "name": "M_Window-Casement-Double:1250 x 1890
mm:1324519",
            "size": {
                "value": 2.36,
                "unit": "m^2"
            },
            "location": {
                "facade": "North",
                "level": "1"
            },
            "startDateTime": [],
            "dueDateTime": [],
```

```
            "guids": [ "1evipg2MH7OQvwN7M5aGCb",
"0d_pDVIf91FRQz7XEBnzrG", "1evipg2MH7OQvwN7M5aMM3" ]
        }, {
            "id": 103,
            "name": "Basic Wall:1.1_Plaster_New Construction
0.148:1338676",
            "size": {
                "value": 18.26,
                "unit": "m^2"
            },
            "location": {
                "facade": "North",
                "level": "1"
            },
            "startDateTime": [],
            "dueDateTime": [],
            "guids": [ "1evipg2MH7OQvwN7M5aNlz" ]
        }, {
            "id": 104,
            "name": "Basic Wall:5_Drywall_New
Construction:1317261",
            "size": {
                "value": 6.93,
                "unit": "m^2"
            },
            "location": {
                "facade": "South",
                "level": "2"
            },
            "startDateTime": [],
            "dueDateTime": [],
            "guids": [ "3NkI86N8r6oAltlsh6ExqE" ]
        }, {
            "id": 105,
            "name": "Basic Wall:5_Drywall_New
Construction:1315249",
            "size": {
                "value": 6.93,
                "unit": "m^2"
            },
            "location": {
                "facade": "North",
                "level": "2"
            },
            "startDateTime": [],
            "dueDateTime": [],
            "guids": [ "3NkI86N8r6oAltlsh6ExQZ" ]
        }
    ],
    "activities": [{
            "x_comment": "",
            "id": 1,
            "priceListCode": "TOS20_02.A03.033.001",
            "name": "Remove existing panels, fixed frame and rough
frame",
            "start": "",
```

```
            "end": "",
            "progress": 0,
            "crew": {
                "x_comment": "",
                "crewID": 500,
                "unitLabour": [0.57, 0],
                "setupMatrixID": 0
            },
            "preconditions": [],
            "deliverables": 100,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 0.77
        }, {
            "x_comment": "",
            "id": 2,
            "priceListCode": "TOS20_02.A03.033.001",
            "name": "Remove existing panels, fixed frame and rough
frame",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "x_comment": "",
                "crewID": 500,
                "unitLabour": [0.57, 0],
                "setupMatrixID": 0
            },
            "preconditions": [],
            "deliverables": 101,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "x_comment": "",
            "id": 3,
            "priceListCode": "TOS20_02.A03.033.001",
            "name": "Remove existing panels, fixed frame and rough
frame",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "x_comment": "",
                "crewID": 500,
                "unitLabour": [0.57, 0],
                "setupMatrixID": 0
            },
            "preconditions": [],
            "deliverables": 102,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 4,
            "priceListCode": "TOS20_02.A03.039.001",
```

```
            "name": "Remove existing sill",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.45, 0],
                "setupMatrixID": 0
            },
            "preconditions": 1,
            "deliverables": 100,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 0.77
        }, {
            "id": 5,
            "priceListCode": "TOS20_02.A03.039.001",
            "name": "Remove existing sill",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.45, 0],
                "setupMatrixID": 0
            },
            "preconditions": 2,
            "deliverables": 101,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 6,
            "priceListCode": "TOS20_02.A03.039.001",
            "name": "Remove existing sill",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.45, 0],
                "setupMatrixID": 0
            },
            "preconditions": 3,
            "deliverables": 102,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 7,
            "priceListCode": "TOS20_02.C01.032.002",
            "name": "Resize wall compartment",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
```

```
                "crewID": 500,
                "unitLabour": [0.4, 0.41],
                "setupMatrixID": 0
            },
            "preconditions": 4,
            "deliverables": 100,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 0.77
        }, {
            "id": 8,
            "priceListCode": "TOS20_02.C01.032.002",
            "name": "Resize wall compartment",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.4, 0.41],
                "setupMatrixID": 0
            },
            "preconditions": 5,
            "deliverables": 101,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 9,
            "priceListCode": "TOS20_02.C01.032.002",
            "name": "Resize wall compartment",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.4, 0.41],
                "setupMatrixID": 0
            },
            "preconditions": 6,
            "deliverables": 102,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "x_comment": "Here size is 1 since the unitLabour is
per frame and not per square meter (main size of deliverable)",
            "id": 10,
            "priceListCode": "TOS20_01.E04.001.002",
            "name": "Install new rough frame",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.9, 0.9],
                "setupMatrixID": 0
```

```
            },
            "preconditions": 7,
            "size": 1,
            "deliverables": 100,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z"
        }, {
            "x_comment": "Here size is 1 since the unitLabour is
per frame and not per square meter (main size of deliverable)",
            "id": 11,
            "priceListCode": "TOS20_01.E04.001.002",
            "name": "Install new rough frame",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.9, 0.9],
                "setupMatrixID": 0
            },
            "preconditions": 8,
            "size": 1,
            "deliverables": 101,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z"
        }, {
            "x_comment": "Here size is 1 since the unitLabour is
per frame and not per square meter (main size of deliverable)",
            "id": 12,
            "priceListCode": "TOS20_01.E04.001.002",
            "name": "Install new rough frame",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.9, 0.9],
                "setupMatrixID": 0
            },
            "preconditions": 9,
            "size": 1,
            "deliverables": 102,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z"
        }, {
            "id": 13,
            "priceListCode": "TOS20_01.E04.003.004",
            "name": "Install new window shutters",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.63, 0.63],
                "setupMatrixID": 0
            },
```

```
            "preconditions": 10,
            "deliverables": 100,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 0.77
        }, {
            "id": 14,
            "priceListCode": "TOS20_01.E04.003.004",
            "name": "Install new window shutters",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.63, 0.63],
                "setupMatrixID": 0
            },
            "preconditions": 11,
            "deliverables": 101,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 15,
            "priceListCode": "TOS20_01.E04.003.004",
            "name": "Install new window shutters",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.63, 0.63],
                "setupMatrixID": 0
            },
            "preconditions": 12,
            "deliverables": 102,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 16,
            "priceListCode": "TOS20_02.E06.006.003",
            "name": "Restore/execute internal plaster",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.53, 0.35],
                "setupMatrixID": 0
            },
            "preconditions": 10,
            "deliverables": 100,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 0.77
```

```
        }, {
            "id": 17,
            "priceListCode": "TOS20_02.E06.006.003",
            "name": "Restore/execute internal plaster",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.53, 0.35],
                "setupMatrixID": 0
            },
            "preconditions": 11,
            "deliverables": 101,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 18,
            "priceListCode": "TOS20_02.E06.006.003",
            "name": "Restore/execute internal plaster",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.53, 0.35],
                "setupMatrixID": 0
            },
            "preconditions": 12,
            "deliverables": 102,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 19,
            "priceListCode": "TOS20_01.E04.003.002",
            "name": "Install new fixed frame and panels
(accessories included)",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.675, 0.675],
                "setupMatrixID": 0
            },
            "preconditions": 10,
            "deliverables": 100,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 0.77
        }, {
            "id": 20,
            "priceListCode": "TOS20_01.E04.003.002",
```

```
            "name": "Install new fixed frame and panels
(accessories included)",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.675, 0.675],
                "setupMatrixID": 0
            },
            "preconditions": 11,
            "deliverables": 101,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 21,
            "priceListCode": "TOS20_01.E04.003.002",
            "name": "Install new fixed frame and panels
(accessories included)",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 500,
                "unitLabour": [0.675, 0.675],
                "setupMatrixID": 0
            },
            "preconditions": 12,
            "deliverables": 102,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 2.36
        }, {
            "id": 22,
            "priceListCode": "TOS20_03.A03.004.003",
            "name": "Remove existing wall layers",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 501,
                "unitLabour": [0.2, 0, 0.24],
                "setupMatrixID": 0
            },
            "preconditions": [],
            "deliverables": 103,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 18.26
        }, {
            "id": 23,
            "priceListCode": "TOS20_03.A03.004.003",
            "name": "Remove existing wall layers",
            "start": "",
            "end": "",
```

```
            "progress": 0,
            "crew": {
                "crewID": 501,
                "unitLabour": [0.2, 0, 0.24],
                "setupMatrixID": 0
            },
            "preconditions": [],
            "deliverables": 104,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 6.93
        }, {
            "id": 24,
            "priceListCode": "TOS20_03.A03.004.003",
            "name": "Remove existing wall layers",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 501,
                "unitLabour": [0.2, 0, 0.24],
                "setupMatrixID": 0
            },
            "preconditions": [],
            "deliverables": 105,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 6.93
        }, {
            "id": 25,
            "priceListCode": "TOS20_01.B03.004.001",
            "name": "Install metallic profiles' system ('L' and 'T'
profiles)",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 501,
                "unitLabour": [0.0225, 0.0225, 0.0235],
                "setupMatrixID": 0
            },
            "preconditions": 22,
            "deliverables": 103,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 18.26
        }, {
            "id": 26,
            "priceListCode": "TOS20_01.B03.004.001",
            "name": "Install metallic profiles' system ('L' and 'T'
profiles)",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 501,
```

```
                "unitLabour": [0.0225, 0.0225, 0.0235],
                "setupMatrixID": 0
            },
            "preconditions": 23,
            "deliverables": 104,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 6.93
        }, {
            "id": 27,
            "priceListCode": "TOS20_01.B03.004.001",
            "name": "Install metallic profiles' system ('L' and 'T'
profiles)",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 501,
                "unitLabour": [0.0225, 0.0225, 0.0235],
                "setupMatrixID": 0
            },
            "preconditions": 24,
            "deliverables": 105,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 6.93
        }, {
            "id": 28,
            "priceListCode": "TOS20_01.D01.039.002",
            "name": "Install waterproofing membrane and existing
sill",
            "start": "",
            "end": "",
            "progress": 0,
            "crew": {
                "crewID": 501,
                "unitLabour": [0.027, 0, 0.05],
                "setupMatrixID": 0
            },
            "preconditions": [25, 20, 21],
            "deliverables": 103,
            "startDateTime": "2021-07-01T08:00:00Z",
            "dueDateTime": "2021-07-31T24:00:00Z",
            "size": 18.26
        }, {
// ULTERIORI ATTIVITA' NON ESPORTATE IN APPENDICE
            }

    "baseline": {
        "data": [{
                "id": "Project",
                "type": "project",
                "text": "<Name of the construction site>",
                "open": true,
                "start_date": "01-07-2021 08:00",
                "end_date": "01-08-2021 00:00",
```

```
                "parent": "root",
                "planned_start": "01-07-2021 09:00",
                "planned_end": "08-07-2021 09:00",
                "duration": [],
                "progress": []
            }, {
                "id": "Crew500",
                "type": "project",
                "text": "Crew for Windows Replacement",
                "open": true,
                "start_date": "",
                "end_date": [],
                "parent": "Project",
                "planned_start": "",
                "planned_end": "",
                "duration": "",
                "progress": []
            }, {
                "id": "Crew501",
                "type": "project",
                "text": "Crew for Envelope Renovation",
                "open": true,
                "start_date": "",
                "end_date": [],
                "parent": "Project",
                "planned_start": "",
                "planned_end": "",
                "duration": "",
                "progress": []
            }, {
                "id": "Activity24",
                "type": "task",
                "text": ["Remove existing wall layers"],
                "open": true,
                "start_date": "01-07-2021 09:00",
                "end_date": "01-07-2021 10:39",
                "parent": "Crew501",
                "planned_start": "01-07-2021 09:00",
                "planned_end": "01-07-2021 10:39",
                "duration": 1.6631999999999998,
                "progress": 0
            }, {
                "id": "Activity27",
                "type": "task",
                "text": ["Install metallic profiles' system ('L'
and 'T' profiles)"],
                "open": true,
                "start_date": "01-07-2021 10:39",
                "end_date": "01-07-2021 10:49",
                "parent": "Crew501",
                "planned_start": "01-07-2021 10:39",
                "planned_end": "01-07-2021 10:49",
                "duration": 0.16285499999999997,
                "progress": 0
            }, {
                "id": "Activity30",
```

```
                "type": "task",
                "text": ["Install waterproofing membrane and
existing sill"],
                "open": true,
                "start_date": "01-07-2021 10:49",
                "end_date": "01-07-2021 11:10",
                "parent": "Crew501",
                "planned_start": "01-07-2021 10:49",
                "planned_end": "01-07-2021 11:10",
                "duration": 0.34649999999999981,
                "progress": 0
            }, {
                "id": "Activity1",
                "type": "task",
                "text": ["Remove existing panels, fixed frame and
rough frame"],
                "open": true,
                "start_date": "01-07-2021 09:00",
                "end_date": "01-07-2021 09:26",
                "parent": "Crew500",
                "planned_start": "01-07-2021 09:00",
                "planned_end": "01-07-2021 09:26",
                "duration": 0.43889999999999996,
                "progress": 0
            }, {
                "id": "Activity4",
                "type": "task",
                "text": ["Remove existing sill"],
                "open": true,
                "start_date": "01-07-2021 09:26",
                "end_date": "01-07-2021 09:47",
                "parent": "Crew500",
                "planned_start": "01-07-2021 09:26",
                "planned_end": "01-07-2021 09:47",
                "duration": 0.34649999999999992,
                "progress": 0
            }, {
                "id": "Activity7",
                "type": "task",
                "text": ["Resize wall compartment"],
                "open": true,
                "start_date": "01-07-2021 09:47",
                "end_date": "01-07-2021 10:06",
                "parent": "Crew500",
                "planned_start": "01-07-2021 09:47",
                "planned_end": "01-07-2021 10:06",
                "duration": 0.31570000000000009,
                "progress": 0
            }, {
                "id": "Activity33",
                "type": "task",
                "text": ["Install insulation panels"],
                "open": true,
                "start_date": "01-07-2021 11:10",
                "end_date": "01-07-2021 12:33",
                "parent": "Crew501",
```

```
                "planned_start": "01-07-2021 11:10",
                "planned_end": "01-07-2021 12:33",
                "duration": 1.3859999999999997,
                "progress": 0
            }, {
                "id": "Activity10",
                "type": "task",
                "text": ["Install new rough frame"],
                "open": true,
                "start_date": "01-07-2021 10:06",
                "end_date": "01-07-2021 11:00",
                "parent": "Crew500",
                "planned_start": "01-07-2021 10:06",
                "planned_end": "01-07-2021 11:00",
                "duration": 0.90000000000000013,
                "progress": 0
            }, {
                "id": "Activity16",
                "type": "task",
                "text": ["Restore/execute internal plaster"],
                "open": true,
                "start_date": "01-07-2021 11:00",
                "end_date": "01-07-2021 11:24",
                "parent": "Crew500",
                "planned_start": "01-07-2021 11:00",
                "planned_end": "01-07-2021 11:24",
                "duration": 0.40810000000000013,
                "progress": 0
            }, {
                "id": "Activity13",
                "type": "task",
                "text": ["Install new window shutters"],
                "open": true,
                "start_date": "01-07-2021 11:24",
                "end_date": "01-07-2021 11:53",
                "parent": "Crew500",
                "planned_start": "01-07-2021 11:24",
                "planned_end": "01-07-2021 11:53",
                "duration": 0.48510000000000009,
                "progress": 0
            }, {
                "id": "Activity19",
                "type": "task",
                "text": ["Install new fixed frame and panels
(accessories included)"],
                "open": true,
                "start_date": "01-07-2021 11:53",
                "end_date": "01-07-2021 12:24",
                "parent": "Crew500",
                "planned_start": "01-07-2021 11:53",
                "planned_end": "01-07-2021 12:24",
                "duration": 0.51975000000000016,
                "progress": 0
            }, {
                "id": "Activity2",
                "type": "task",
```

```
                "text": ["Remove existing panels, fixed frame and
rough frame"],
                "open": true,
                "start_date": "01-07-2021 12:24",
                "end_date": "01-07-2021 14:57",
                "parent": "Crew500",
                "planned_start": "01-07-2021 12:24",
                "planned_end": "01-07-2021 14:57",
                "duration": 1.5452000000000004,
                "progress": 0
            }, {
                "id": "Activity5",
                "type": "task",
                "text": ["Remove existing sill"],
                "open": true,
                "start_date": "01-07-2021 14:57",
                "end_date": "01-07-2021 16:01",
                "parent": "Crew500",
                "planned_start": "01-07-2021 14:57",
                "planned_end": "01-07-2021 16:01",
                "duration": 1.0619999999999994,
                "progress": 0
            }, {
                "id": "Activity36",
                "type": "task",
                "text": ["Install drywall panels"],
                "open": true,
                "start_date": "01-07-2021 12:33",
                "end_date": "01-07-2021 15:34",
                "parent": "Crew501",
                "planned_start": "01-07-2021 12:33",
                "planned_end": "01-07-2021 15:34",
                "duration": 2.0096999999999996,
                "progress": 0
            }, {
                "id": "Activity8",
                "type": "task",
                "text": ["Resize wall compartment"],
                "open": true,
                "start_date": "01-07-2021 16:01",
                "end_date": "01-07-2021 16:59",
                "parent": "Crew500",
                "planned_start": "01-07-2021 16:01",
                "planned_end": "01-07-2021 16:59",
                "duration": 0.9676,
                "progress": 0

    }
}
```

8. BIBLIOGRAFIA

[1] "Using online simulation in Holonic Manufacturing Systems": *Olivier Cardin, Pierre Castagna.*

[2] "Design for the unexpected from holonic manufacturing systems towards a humane mechatronics society": *Paul Valckenaers, Hendrik Van Brussel.*

[3] "PROSA and Delegate MAS for Open-Air Engineering Processes": *P. Valckenaers, J. Van Belle; O. Ali.*

[4] "Development and applications of holonic manufacturing systems: a survey": *Radu Babiceanu, Frank Chen.*

[5] "Construction beyond lean: a new understanding of construction management": Sven Bertelsen and Lauri Koskela.

[6] "A Construction Workflow Model for Analyzing the Impact of In-Project Variability": Nelly P. Garcia-Lopez and Martin Fischer.

[7] "Construction flow index: a metric of production flow quality in construction": Rafael Sacks, Olli Seppänen, Vitaliy Priven & Jonathan Savosnick.

[8] "Opportunities for enhanced lean construction management using Internet of Things standards": Bhargav Dave, Sylvain Kubler,Kary Främling,Lauri Koskela.

[9] " CIFE 2016 - Flow based field management": Martin Fischer, Research Assistant: Nelly Garcia-Lopez.

[10] " Mosheli Workspace planning non deterministic": Abdelhady Hosny, Mazdak Nik-Bakht, Osama Moselhi.

[11] " Construction workspace management within an Industry Foundation Class-Compliant 4D tool": M. Kassem, N. Dawood, R. Chavada.

[12] " Spatio-temporal planning for tower cranes in construction projects with simulated annealing": Keyi Wu, Borja García de Soto, Feilian Zhang.

[13] "Requirements for building information modeling based lean production management
systems for construction": Rafael Sacks, Milan Radosavljevic, Ronen Barak.

[14] "Examination of the effects of a KanBIM production control system on subcontractors' task selections in interior works":Ury Gurevich, Rafael Sacks.

9. INDICE DELLE FIGURE

Figura 1 Il rilascio dello spazio di lavoro è difficile da cogliere attraverso la rappresentazione Gantt..... 11

Figura 2 La struttura di un workflow variabile prevede due tipologie di variabilità..... 14

Figura 3 Formula del Construction Flow Indexj..... 16

Figura 4 Flowline charts di tre differenti progetti che mostrano il reale progresso di sette lavorazioni in sette spazi..... 17

Figura 5 Indicazione delle grandezze che influiscono nel calcolo di CFI..... 18

Figura 6 Il modello deterministico di gestione dello spazio è composto da tre fasi differenti: Pre-Collision, Collision, Post-Collision 20

Figura 7 Ogni rettangolo rappresenta un tipo di olole nella architettura PROSA, le frecce rappresentano le interazioni che hanno fra loro..... 29

Figura 8 I moduli che compongono DMAS sono tre: Formica,Agente,Ambiente. 31

Figura 9 Diagramma schematico della gestione avanzata del Lean Construction Management 35

Figura 10 L'obbiettivo è quello di integrare dei simboli che indicano l'avanzamento con un modello della costruzione..... 38

Figura 11 Interfaccia utente che considera gli stati di maturità in un processo di costruzione. 40

Figura 12 Un elemento con messaggio rappresenta un elemento contenente informazioni..... 44

Figura 13 Segno di parallelismo..... 44

Figura 14 Il simbolo Exclusive indica quale direzione scegliere nel workflow..... 44

Figura 15 Più informazioni definiscono lo stato reale di avanzamento del flusso di produzione 48

Figura 16 Il worflow si compone di 11 task, 3 line e dimostra come utilizzare lo strumento di simulazione in un processo produttivo..... 49

Figura 17 Il caso studio prevede tre task principali: la generazione di un cronoprogramma, la simulazione, la risoluzione di eventuali collisioni 50

Figura 18 La schermata Unity tradizionale si compone di tre finestre oltre la scena: (1)Hierarchy, (2)Inspector, (3)Project..... 56

Figura 19 Attraverso il pulsante play è possibile avviare la simulazione manualmente..... 56

Figura 20 Metodo di importazione A drag-and-drop 58

Figura 21 Debug.Log("Success!"); *60*

Figura 22 Lo sviluppo del sistema si è articolato in quattro task principali *62*

Figura 23 Estratto della finestra Hierarchy in Unity della struttura proposta: Day,Crew,Deliverable. *64*

Figura 24 Il processo di deserializzazione e generazione dei gameObject si articola in sette passaggi *65*

Figura 25 Schermata di esempio degli imput manuali: data e numero di giorni *68*

Figura 26 Il sistema esclude dall'orizzonte temporale i giorni non lavorativi. *70*

Figura 27 Ogni elemento "Crew" viene diviso in funzione della data in cui esegue attività *72*

Figura 28 Al fine di generare l'oggetto deliverable è necessario passare attraverso l'oggetto attività *72*

Figura 29 I deliverable sono divisi in funzione della crew che esegue attività al loro interno. . *76*

Figura 30 La fase 2.7 si compone di tre task: un'importazione, una deserializzazione ed un associazione fra guids e GameObjects *77*

Figura 31 La rinominazione degli elementi importati con Piglet è scomponibile in due porzioni: il nome e il Guid *79*

Figura 32 Elementi come gli infissi sono comporti da più gameObject aventi lo stesso nome .. *79*

Figura 33 Risultato della rinominazione degli elementi importati *80*

Figura 34 Il modello importato tramite Piglet è composto da piu elementi "GameObject" *81*

Figura 35 Processo logico secondo cui è possibile cercare l'elemento in Unity3D attraverso il codice Guid *82*

Figura 36 I passaggi in cui si articola lo script relativo alla fase 2.7 sono quattro: si definisce la lista di guid, si cerca l'elemento con guid associato, allo stesso viene definito il ifclabelentity e successivamente vengono certi i gameObject *83*

Figura 37 Più elementi con lo stesso nome rappresentano il deliverable, in questo caso una finestra *84*

Figura 38 In questo caso ad una intera parete muraria corrisponde un solo elemento *84*

Figura 39 Lo script "merge_workspace" si compone di tre passaggi: estrarre le mesh, combinarle ed instanziare un cubo come inviluppo *85*

Figura 40 Lo spazio di lavoro relativo all'infisso comprende sia l'elemento lavorato che un offset di default *88*

Figura 41 Lo spazio di lavoro relativo alla parete muraria comprende sia l'elemento lavorato che un offset di default 88

Figura 42 Il dizionario C associa ad ogni Crew un Colore 90

Figura 43 I colori degli spazi di lavoro vengono cambiati in funziona della crew che li esegue 91

Figura 44 Ulteriore esempio riguardo la colorazione degli spazi di lavoro 92

Figura 45 Il riconoscimento delle intersezioni è composto da un task di rilevazione delle collisioni ed uno di esportazione..... 93

Figura 46 il primo passaggio della rilevazione delle collisioni in Unity termina con la stampa delle stesse Di seguito viene mostrata la collisione interecettata dalla simulazione fra il D102 lavorato dalla Crew 500 e il D104 lavorato dalla crew 501:..... 95

Figura 47 Dimostrazione di sovrapposizione di due zone di lavoro..... 95

Figura 48 Deliverables dello stesso colore sono processari dalla stessa crew, quindi non definiscono una collisione 1..... 96

Figura 49 Deliverables dello stesso colore sono processati dalla stessa crew, quindi non definiscono una collisione..... 96

Figura 50 Esempio di calcolo di bounds.center e bound.min e max 101

Figura 51 Ipotetica sezione trasversale di due box 101

Figura 52 Vengono instanziati nei punti di contatto delle sfere per compiere valutazioni 102

Figura 53 In corrispondenza dei punti rilevati viene instanziata una sfera 102

Figura 54 Gli oggetti Baseline.Data, activities e deliverables sono quelli usati dalla simulazione..... 105

Figura 55 Gli input della data di simulazione e del numero di giorni sono numeri interi 105

Figura 56 Il modello è importato ed il cronoprogramma deserializzato..... 106

Figura 57 La sovrapposizione è nettamente visibile grazie alla colorazione differente delle zone di lavoro 107