Specifiche per il Progetto di Laboratorio di Sistemi Operativi

Tutor: Andrea Colledan – andrea.colledan2@unibo.it
Anno Accademico 2020-21

Versione 1.4 – 27 Luglio 2021

In questo documento sono presentate le specifiche del progetto del corso di Sistemi Operativi per la Laurea Triennale in Informatica per il Management dell'Università di Bologna, anno accademico 2020-2021. **Tali specifiche possono essere soggette a cambiamenti e a revisioni. Ogni modifica viene comunicata tempestivamente attraverso la bacheca di annunci ufficiale del corso su Virtuale e sul gruppo ufficiale del corso.** In ogni momento e compatibilmente con la disponibilità dei docenti, è possibile prenotare un ricevimento virtuale sulla piattaforma Teams. A tal fine, contattare il tutor all'indirizzo `andrea.colledan2@unibo.it` specificando la motivazione della richiesta di colloquio. Il progetto verte attorno alla realizzazione di un "generico gioco multigiocatore a blocchi". Le tre fasi del progetto descritte in questo documento sono:

1. La formazione dei gruppi e l'inizializzazione del progetto
2. La realizzazione del progetto
3. La consegna e la discussione del progetto

1 Preliminari

1.1 Formazione dei gruppi

I gruppi sono costituiti da quattro (4) o cinque (5) persone. Coloro che intendono partecipare all'esame comunicano entro e non oltre il **31 Maggio**

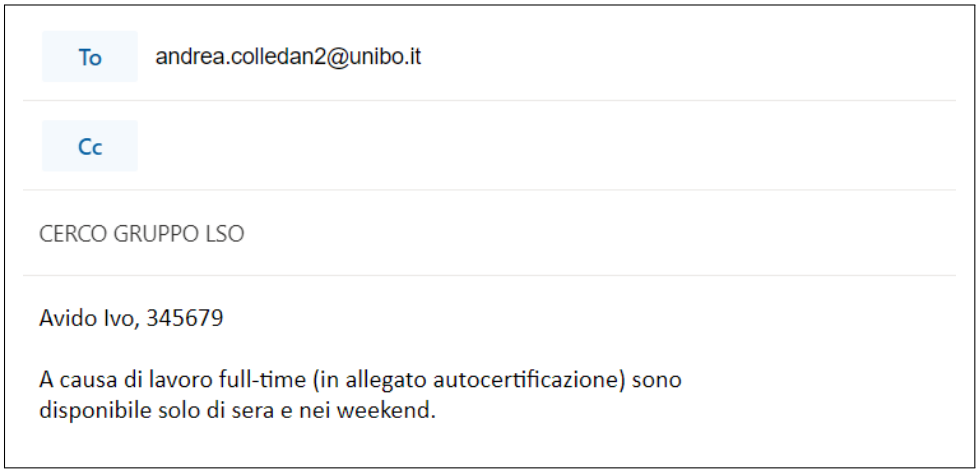
2021 la composizione del gruppo di lavoro al tutor, via posta elettronica. Il messaggio deve essere inviato dalla propria mail istituzionale (dominio `studio.unibo.it`). Deve avere come oggetto **GRUPPO LSO** e contenere:

- Il nome del gruppo.
- Per ciascun componente una riga contenente:
  - Cognome, Nome, Matricola.
- Un indirizzo di posta elettronica di riferimento a cui mandare le notifiche. **N.B. è incarico del referente trasmettere eventuali comunicazioni agli altri membri.**

Esempio di email di formazione gruppo.

Chi non riuscisse a trovare un gruppo può inviare allo stesso indirizzo di posta elettronica un messaggio con oggetto **CERCO GRUPPO LSO**, specificando:

- Cognome, nome, matricola.
- Eventuali preferenze legate a tempi di lavoro. Si cercherà di costituire gruppi di persone con tempi di lavoro compatibili, nel limite delle possibilità.



Esempio di email di ricerca gruppo.

Le persone senza un gruppo vengono assegnate il prima possibile **senza possibilità di ulteriori modifiche**. Per tale motivo è caldamente consigliato rivolgersi al tutor per la ricerca di un gruppo come *ultima* soluzione.

1.2 Creazione della repository

Il codice sorgente del progetto è contenuto in uno spazio cloud del servizio online GitLab. Questo spazio è creato e gestito seguendo la procedura descritta di seguito. **N.B. Le istruzioni che seguono devono essere completate entro e non oltre la scadenza di presentazione dei gruppi.**

1. Ogni membro del gruppo crea un account su GitLab (a meno che non ne abbia già uno).

2. Il referente del gruppo crea un nuovo progetto cliccando sul bottone **"+"** nella barra superiore della schermata principale di GitLab. Inserisce `LabSO_[NOME_GRUPPO]` come nome del progetto, imposta la repository come privata e clicca su **"Create project"**.

3. Il referente aggiunge ogni membro del gruppo alla repository. Per fare ciò, dal menu della repository seleziona "Members". Nella schermata di invito membri, il referente cerca ciascun membro col nome utente con cui quest'ultimo è iscritto a GitLab, seleziona come role permission `Developer` e clicca su **"Invite"**.

4. Allo stesso modo, il referente aggiunge il tutor come reporter della repository. Per fare ciò, invita l'utente `andcol` con role permission `Reporter`.

2 Il progetto

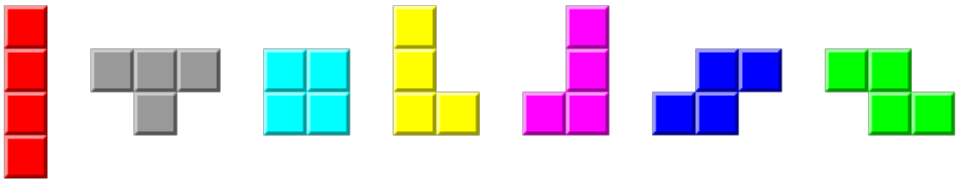
La seguente sezione fornisce le specifiche e le informazioni necessarie alla realizzazione del progetto.

2.1 "Generico gioco multigiocatore a blocchi"

Il progetto verte sull'implementazione di un "generico gioco multigiocatore a blocchi", il cui funzionamento e le cui regole sono descritti nei paragrafi seguenti, e la documentazione del processo implementativo stesso.

2.1.1 Meccaniche di base

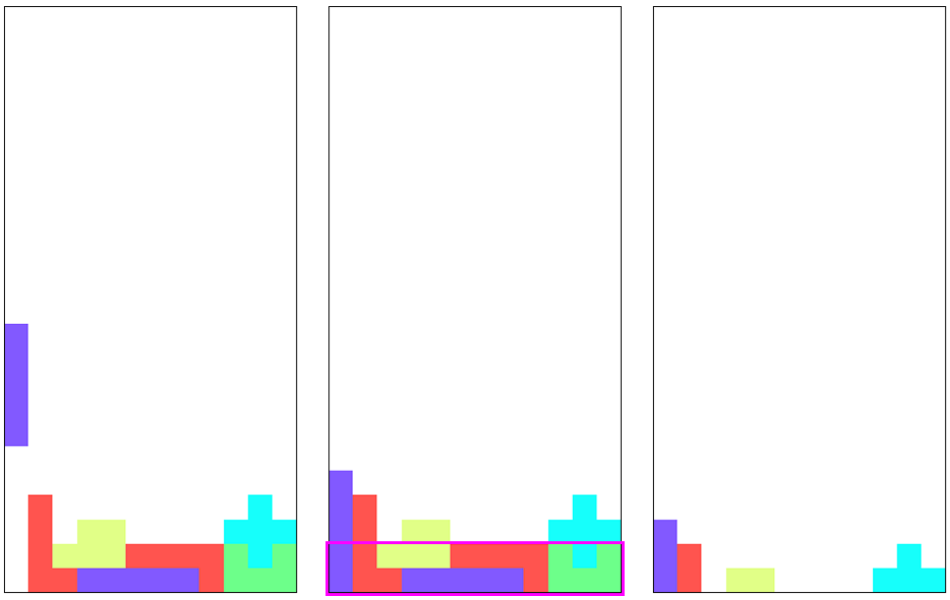
In "generico gioco multigiocatore a blocchi", il giocatore ha a disposizione un campo di gioco largo 12 caselle e alto 24 caselle. All'inizio di una partita, in cima al campo di gioco compare un blocco, la cui forma è scelta casualmente da una pool di *polimini*. Un polimino è una figura geometrica ottenuta componendo lato-a-lato un numero predeterminato di quadrati. Se i quadrati sono due si parla di *domino*, se sono tre si parla di *trimino*, e così via: *tetramino*, *pentamino*, *esamino*, ... Ad esempio, con quattro quadrati è possibile comporre i seguenti sette tetramini:



In seguito alla sua apparizione, il polimino comincia a cadere verso il fondo del campo di gioco, muovendosi verso il basso a un ritmo prefissato, una casella alla volta. Durante la caduta, il giocatore può spostare il polimino a sinistra, a destra e verso il basso (ovvero può accelerarne la caduta) e può ruotarlo in senso orario e antiorario, in incrementi di 90°. Il polimino

continua a cadere fino a quando non collide col fondo del campo di gioco o con un blocco sottostante. Una volta caduto, il polimino non può essere più spostato o ruotato dal giocatore e diventa parte del campo di gioco. In seguito, un nuovo polimino compare in cima al campo di gioco e il ciclo si ripete.

Lo scopo del giocatore è impedire che la pila di polimini caduti raggiunga la cima del campo di gioco. A tal fine, il giocatore cerca di far incastrare i polimini in caduta coi polimini sottostanti in modo tale da riempire una o più righe del campo di gioco. Quando una riga del campo di gioco è piena, essa viene *liberata*: le sue caselle si svuotano ed eventuali blocchi sovrastanti discendono a colmare il vuoto risultante. Se la pila di polimini caduti raggiunge la cima del campo di gioco, il giocatore perde la partita.



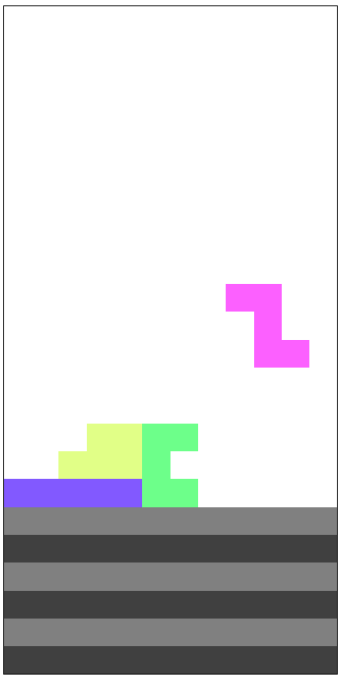
Liberazione di due righe: il pentomino blu in caduta (sinistra) completa le due righe in fondo al campo di gioco (centro), che vengono liberate e scompaiono (destra).

2.1.2 Partita multigiocatore

"Generico gioco multigiocatore a blocchi" è un gioco multigiocatore competitivo. Ciascun giocatore dispone del proprio campo di gioco personale, in cui gioca come descritto nella sezione precedente, e vede i campi di gioco dei propri avversari. Vince la partita il giocatore che sopravvive più a lungo (i.e. quello che perde per ultimo). Per accelerare la sconfitta degli altri giocatori, un giocatore può produrre delle *righe spazzatura* da inviare ai suoi avversari. Per produrre righe spazzatura, il giocatore deve liberare due o più righe con un solo polimino. Più sono le righe liberate dal singolo polimino, maggiore è il numero di righe spazzatura prodotte, secondo la seguente tabella:

2 righe liberate	1 riga di spazzatura
3 righe liberate	2 righe di spazzatura
4 righe liberate	4 righe di spazzatura

Il giocatore può decidere a quale avversario inviare le righe spazzatura che produce. Quando un giocatore riceve delle righe spazzatura, queste si accumulano sul fondo del suo campo di gioco e non possono essere liberate. Le righe spazzatura riducono l'area giocabile del campo di gioco e rendono più facile la sconfitta.



Un campo di gioco che ha accumulato 6 righe spazzatura.

2.2 Requisiti dell'implementazione

Linguaggio e librerie Il gioco è implementato in Java (Java SE 16). La comunicazione di rete è implementata attraverso i socket, come visto a lezione. La gestione della grafica e dell'input utente è implementata attraverso la libreria Java Lanterna (`https://github.com/mabe02/lanterna`). La documentazione di Lanterna, assieme a tutorial ed esempi, è disponibile presso `https://github.com/mabe02/lanterna/blob/master/docs/contents.md`. L'uso di librerie diverse per la gestione della grafica o dell'input utente deve essere preventivamente discusso col tutor e approvato da quest'ultimo. Per una lista delle librerie correntemente permesse/vietate e i criteri per la loro approvazione, consultare le FAQ del progetto su Virtuale (`https://virtuale.unibo.it/mod/page/view.php?id=569216`).

Client Il client viene avviato da linea di comando e accetta come argomenti l'indirizzo IP e la porta del server a cui connettersi. In seguito, permette al

giocatore di giocare al "generico gioco multigiocatore a blocchi" con altri client connessi allo stesso server.

Server Il server viene avviato da linea di comando e accetta come argomento la porta su cui mettersi in ascolto. In seguito, attende che si connettano *almeno* 2 giocatori e *al massimo* 4 giocatori. Durante l'attesa, il server accetta istruzioni da riga di comando e risponde ai seguenti comandi:

- `start`: avvia la partita coi giocatori attualmente connessi (se sono almeno 2).
- `quit`: disconnette i client connessi e chiude il server.

Quando il numero massimo di giocatori viene raggiunto, o quando viene inviato il comando `start`, il server avvia la partita coi giocatori presenti. Durante la partita, il server continua ad accettare istruzioni da riga di comando e risponde ai seguenti comandi:

- `pause`: mette in pausa il gioco per tutti i giocatori. Mentre il gioco è in pausa, i polimini non cadono e i giocatori non possono eseguire alcuna mossa.
- `restart`: termina la partita corrente e disconnette i client connessi. In seguito, il server si rimette in attesa di richieste di connessione.
- `quit`: termina la partita corrente, disconnette i client connessi e chiude il server.

2.3 Altri aspetti del gioco

Per essere valido, il progetto finale deve implementare le meccaniche di gioco descritte nella sezione 2.1 e rispettare i requisiti della sezione 2.2. Ogni altra scelta è a libera discrezione del gruppo, che è incoraggiato a personalizzare l'esperienza di gioco del proprio progetto. In particolare, sono a discrezione del gruppo le scelte riguardanti:

- Il nome del gioco,
- I comandi di gioco,
- Il tipo dei polimini (tetramini, pentamini, esamini, misti, etc.),

- Quando avviene la scelta del destinatario delle righe spazzatura (se prima o dopo la loro produzione), quando queste vengono inviate (se appena prodotte, o appena scelto il destinatario, o in un secondo momento), etc.
- L'estetica del gioco (colori, aspetto dei polimini, etc.),
- Eventuale musica ed effetti audio,
- Eventuali meccaniche di gioco aggiuntive (e.g. accumulo delle righe spazzatura),
- Eventuali easter egg,
- ...

Questi aspetti non influiscono sulla valutazione del progetto, a meno che le scelte fatte in merito non stravolgano le meccaniche di gioco o compromettano fondamentalmente la giocabilità del prodotto finale.

2.4 Elementi di implementazione di videogiochi

In questa sezione sono riportate alcune indicazioni di carattere (molto) generale sull'implementazione di videogiochi. Le informazioni riportate sono adattate in larga parte dal testo *Game Engine Architecture – Second Edition*, di Jason Gregory (CRC Press).

2.4.1 Game loop e subsystem

Indipendentemente dal genere di riferimento, l'implementazione di un videogioco ruota attorno al concetto di *game loop*. Nello scenario più semplificato possibile, questo è un ciclo *while* all'interno del quale:

1. L'input del giocatore viene letto,
2. Lo stato di gioco viene aggiornato,
3. Lo stato di gioco viene mostrato a schermo.

Il game loop è la colonna portante di un videogioco e garantisce il tipo di dinamicità e reattività che distingue i videogiochi da altri pezzi di software. Nonostante l'outline del game loop sia tutto sommato semplice, ciascuna delle operazioni elencate può essere molto complessa e richiedere a sua volta un gran numero di altre funzionalità. Per questo motivo, l'architettura di un videogioco è solitamente suddivisa in *subsystem*, ovvero componenti relativamente indipendenti l'una dall'altra che si occupano ciascuna di un aspetto diverso della logica di gioco. Ad esempio, un videogioco moderno può basarsi sui seguenti subsystem:

- Engine fisico
- Rendering
- Input/Output
- Gestore delle animazioni
- Intelligenza artificiale
- Audio
- Networking (giochi multiplayer)
- ...

Ciascun subsystem implementa le funzionalità necessarie a portare a termine un'attività specifica (e.g. fare rendering dello stato di gioco). I diversi subsystem sono coordinati dal game loop.

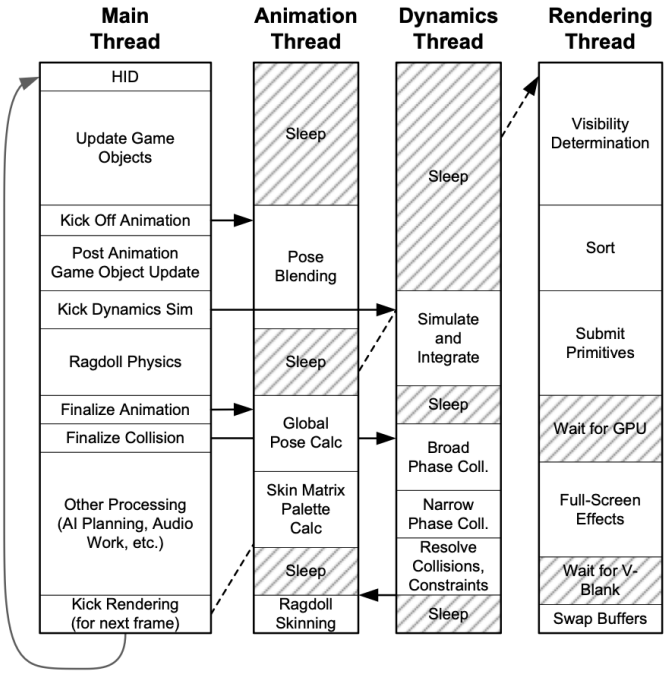
2.4.2 Multithreading

Al giorno d'oggi, l'implementazione di un videogioco non può prescindere dalla programmazione multi-threaded o comunque parallela. I motivi per cui questo è vero sono principalmente due:

- **Efficienza:** la celebre legge di Moore, che prevedeva un raddoppio delle prestazioni delle CPU approssimativamente ogni anno e mezzo, ha smesso di valere più di dieci anni fa. Oggigiorno le macchine su cui lavoriamo non dispongono di processori più potenti, ma di più processori. Pertanto, per sfruttare al massimo le risorse dell'hardware moderno, è essenziale una qualche forma di programmazione parallela.

- **Responsiveness:** i videogiochi sono per definizione pezzi di software altamente interattivi: la stessa applicazione deve renderizzare il mondo di gioco sullo schermo, gestire l'audio, rispondere all'input del giocatore e ad eventuali messaggi dalla rete. Sarebbe inaccettabile se l'audio si interrompesse durante il processo di rendering, o se il gioco si immobilizzasse in attesa di un input del giocatore. Pertanto, emerge la necessità di multithreading per controllare i processi simultanei che costituiscono l'esperienza di gioco.

La suddivisione in subsystem dell'architettura di un videogioco offre una possibile dimensione lungo cui suddividere la logica applicativa in thread concorrenti. Nell'approccio "one thread per subsystem" ciascun subsystem (o comunque ciascuno dei subsystem principali) viene eseguito in un thread dedicato. Il thread principale, che esegue il game loop, continua a gestire la logica ad alto livello del gioco, coordinando gli altri thread, ma il grosso della computazione di ciascun subsystem avviene sul suo thread dedicato.



Esempio di approccio "one thread per subsystem" in un ipotetico videogioco commerciale.

Si prestano ad essere eseguiti in un thread dedicato i subsystem che si occupano di lavori ripetitivi e relativamente indipendenti dal resto della logica applicativa, come ad esempio :

- Engine fisico
- Rendering
- Networking
- Audio
- ...

2.4.3 Multiplayer client-server

In un gioco multigiocatore client-server, il client è la parte che si occupa dell'interazione col giocatore (rendering, I/O, etc.), mentre il server è la parte che si occupa di coordinare l'esperienza di gioco dei diversi giocatori. A seconda di dove risiede lo stato di gioco (e.g. la posizione dei giocatori, il loro punteggio, etc.), distinguiamo due diversi approcci:

- **Server autoritario:** nell'approccio autoritario lo stato di gioco risiede interamente sul server, che ne gestisce l'aggiornamento e che lo inoltra ai client. In questo scenario, i client non fanno altro che inviare al server richieste di compiere determinate azioni (e.g. "muovere il giocatore a destra"). Il server determina se l'azione è permessa e in caso affermativo aggiorna lo stato di gioco, che viene poi inviato a tutti i client. I client si limitano a leggere l'input dell'utente e a renderizzare lo stato ricevuto periodicamente dal server.
- **Server non-autoritario:** nell'approccio non-autoritario lo stato di gioco risiede interamente o in parte presso il client, che ne gestisce l'aggiornamento e informa periodicamente il server dei cambiamenti (e.g. "il giocatore si trova ora alle coordinate $[x : 2, y : 13]$"). Il server si preoccupa di propagare l'aggiornamento agli altri client.

2.5 La documentazione

La documentazione è parte integrante del progetto. Non vi sono vincoli sugli strumenti e i formati utilizzati per redigere la documentazione (e.g. LaTeX, Word, Google Docs, etc.), l'importante è che al termine della stesura venga consegnato un file **PDF** e che tale file rispetti i requisiti descritti in questa sezione. La documentazione deve avere una lunghezza di **almeno 10 pagine** (intese come facciate), compresa l'intestazione, e deve essere scritta con **font di grandezza 12pt**. Il limite è un *lower bound*: non esiste un *upper bound* per la lunghezza della documentazione, che può quindi consistere di un numero arbitrario di pagine.

Struttura della documentazione L'intestazione della documentazione deve avere titolo "Laboratorio di Sistemi Operativi A.A. 2020-21" e deve contenere

- Il nome del gruppo.
- L'indirizzo email del referente del gruppo.
- Per ogni componente del gruppo:
  - Nome, Cognome, Matricola

Il corpo della documentazione deve coprire almeno i seguenti argomenti:

1. Descrizione del progetto consegnato:
   (a) Architettura generale: visione **di alto livello** di quali sono le componenti in gioco, di come interagiscono tra loro e delle informazioni che si scambiano per far funzionare il progetto.
   (b) Descrizione delle singole componenti:
   - Client, server e relativa suddivisione dei compiti.
   - Componenti minori: subsystem, thread, etc.
   - Se necessaria, descrizione di classi e metodi fondamentali.

   (c) Suddivisione del lavoro tra i membri del gruppo.
2. Descrizione e discussione del processo di implementazione:

(a) **Descrizione dei problemi** e degli ostacoli incontrati durante l'implementazione, con discussione e giustificazione delle **soluzioni adottate** e di eventuali **soluzioni alternative**. In particolare:

- Problemi legati alla concorrenza: quali sono le strutture condivise, quando e perché si rende necessaria la mutua esclusione, etc.
- Problemi legati al modello client-server: come vengono instaurate, mantenute e chiuse le connessioni, cosa succede in caso di interruzioni anomale del client o del server, etc.
- Trade-off principali: server autoritario vs. non-autoritario, uso di costrutti bloccanti vs. non bloccanti, etc.

(b) Descrizione degli strumenti utilizzati per l'organizzazione. In particolare, applicazioni, piattaforme, servizi utilizzati per:

- Comunicare tra i membri del gruppo (e.g. Teams, Skype, Discord, etc.)
- Condividere il codice prodotto (e.g. GitLab).
- Tenere traccia del lavoro svolto, del lavoro rimasto da svolgere, delle decisioni ad alto livello prese dal gruppo, etc. (e.g. Google Docs, Trello, etc.)

3. Requisiti e istruzioni per compilare il progetto e giocare.

    - Se presenti, descrizione delle meccaniche di gioco aggiuntive.

L'organizzazione delle sezioni non deve per forza rispecchiare esattamente quella appena riportata. Per esempio, è possibile trattare un argomento in più sezioni, o trattare più argomenti nella stessa sezione. È anche possibile aggiungere informazioni non richieste qualora fossero utili. L'importante è che **almeno** i contenuti elencati siano **facilmente** rintracciabili nel corpo della documentazione.

Scopo della documentazione La documentazione deve puntare a dare al lettore una visione chiara di come funziona il progetto e di come sono stati affrontati gli ostacoli di implementazione, **senza** che il lettore debba conoscere il codice sorgente. I rimandi al codice (e.g. "Vedi `Player.java`, righe 150-155") sono apprezzati, ma il codice sorgente **non** deve sostituire la documentazione. Dall'altro lato, la documentazione non deve essere una

ripetizione delle specifiche. Per la natura del progetto è pressoché scontato che ci sia una classe in cui vengono gestiti i messaggi in arrivo dal server, o una classe in cui viene eseguito il game-loop, etc. Quello che la documentazione deve descrivere è *come* vengono gestiti i messaggi, *cosa* accade nel game loop, etc. nella *vostra specifica implementazione* delle specifiche.

Chi decidesse di redigere la documentazione in LaTeX può trovare un template al seguente indirizzo: `www.overleaf.com/read/hjwrhbzshfxc`.

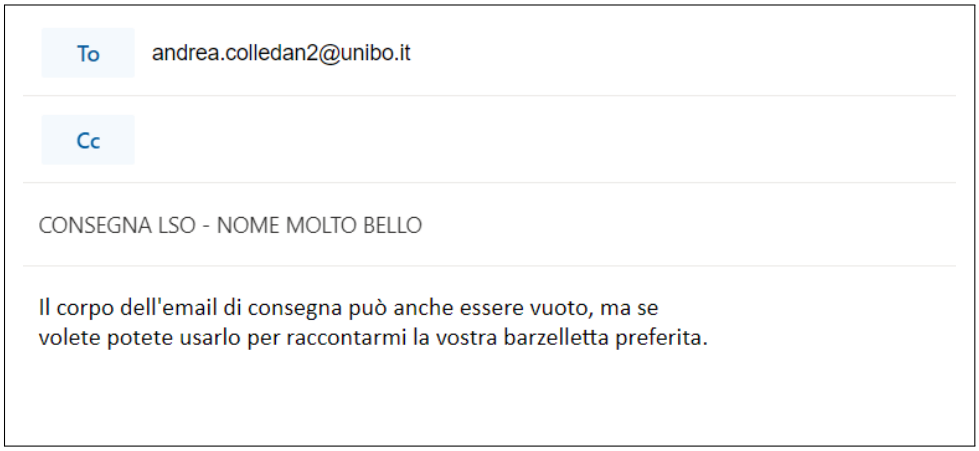
3 Consegna e Discussione

La seguente sezione descrive la procedura di consegna del progetto terminato e le modalità di discussione dello stesso.

3.1 Consegna del Progetto

Al momento della consegna, la repository dovrà contenere i sorgenti del progetto e la documentazione. Quest'ultima deve essere contenuta in un singolo file PDF di nome `DOCUMENTAZIONE_LSO.pdf`. Per effettuare la consegna:

1. Dal menu del progetto, selezionare "Repository" > "Tags".
2. Cliccare su **"New tag"**.
3. Inserire come nome del tag il nome `Consegna` e come messaggio un messaggio qualsiasi *non vuoto*. L'aggiunta di un messaggio è *essenziale* affinché il tag venga registrato con la data e l'ora di creazione.
4. Cliccare su **"Create tag"** per completare la creazione del tag `Consegna`.
5. Una volta creato il tag, notificare la consegna al tutor inviando un'email con oggetto **CONSEGNA LSO - [NOME GRUPPO]** all'indirizzo `andrea.colledan2@unibo.it`.



Esempio di email di notifica di consegna.

L'implementazione e la documentazione del progetto devono essere consegnate assieme. Vi sono due appelli per la consegna del progetto, con le rispettive scadenze:

- **11 Luglio 2021** (11/07/21), ore 23:59.
- **26 Settembre 2021** (26/09/21), ore 23:59.

N.B. La data e l'ora della consegna del progetto sono la data e l'ora di creazione del tag `Consegna`.

Su AlmaEsami è presente un appello per ciascuna di queste scadenze. Il voto finale è individuale, per cui *tutti* i membri del gruppo sono tenuti ad iscriversi su AlmaEsami all'appello in cui il gruppo intende discutere il progetto.

3.2 Discussione del Progetto

In seguito alla consegna, la data e l'ora della discussione verranno fissati e comunicati al referente del gruppo. La discussione del progetto avviene in modalità remota, in un incontro virtuale con tutti i membri. La discussione in sé consiste in:

- Una breve demo del gioco implementato, che può essere effettuata indifferentemente su rete locale (client e server eseguono sulla stessa rete e/o macchina), oppure su internet, coinvolgendo più membri del gruppo.

- Alcune domande ai membri del gruppo sull'implementazione del progetto, l'organizzazione del lavoro e i contributi personali di ciascuno. Durante questa fase verrà richiesto di condividere lo schermo per mostrare e spiegare frammenti di codice sorgente.

Al termine della discussione, ad ogni membro del gruppo viene assegnato un punteggio da 0 a 5 in base all'effettivo contributo alla realizzazione del progetto dimostrato in sede di discussione. Questo punteggio si somma alla valutazione del progetto (vedi Sezione 4) per determinare il voto finale di ogni singolo membro del gruppo.

4 Griglie di Valutazione

La valutazione del progetto tiene conto sia del processo di implementazione del progetto, sia della documentazione. **I due aspetti hanno peso uguale**. La valutazione è basata sui criteri contenuti nelle seguenti griglie di valutazione.

4.1 Implementazione

La valutazione dell'implementazione del progetto si basa sull'analisi del codice Java, sull'uso dei costrutti del linguaggio per la creazione di soluzioni efficienti, sulla tolleranza ai guasti del sistema implementato e sulla gestione delle eccezioni. Da questo punto di vista, l'obiettivo fondamentale del progetto è quello di dimostrare che i membri del gruppo sono in grado di:

- Utilizzare il multithreading per gestire situazioni in cui molti processi (e.g. la lettura dell'input da tastiera, l'aggiornamento del gioco, la ricezione di messaggi dal server, etc.) devono poter eseguire simultaneamente.
- Riconoscere quando una struttura dati è una risorsa condivisa, ovvero quando viene acceduta concorrentemente da più thread, e individuare i problemi di concorrenza associati.
- Adottare di conseguenza i costrutti di mutua esclusione e sincronizzazione adeguati al caso.

Un progetto in cui non viene fatto uso di alcun costrutto di sincronizzazione è pertanto automaticamente insufficiente, così come un progetto che evita i problemi di concorrenza eseguendo tutta la logica applicativa su un solo thread.

La valutazione tiene conto anche di aspetti trasversali come la ripartizione e l'organizzazione del lavoro all'interno del gruppo, la tracciabilità degli artefatti di sviluppo e la partecipazione al gruppo Google del corso. In particolare, una ripartizione precisa dei compiti all'interno del gruppo è importante. Questo non significa che più membri del gruppo non possono collaborare o non devono sapere nulla l'uno del codice degli altri, ma significa che ciascun componente è l'esperto di una (o più) parti circoscritte del progetto (e.g. logica di gioco, comunicazione di rete lato client/server, grafica, etc.) e se ne assume la responsabilità. Per ciascuno degli aspetti riportati nella seguente tabella viene assegnato un punteggio da 0 (insufficiente) a 5 (ottimo).

CRITERIO	**DESCRIZIONE**
Uso dei costrutti di Java	Uso corretto dei costrutti e delle strutture dati offerti da Java per gestire concorrenza e distribuzione. Gestione adeguata di eccezioni e casi limite. Il codice è leggibile e ben commentato.
Distribuzione del lavoro nel gruppo	La distribuzione del lavoro all'interno del gruppo è ben delineata ed omogenea. Ciascun membro del gruppo è capace di indicare i propri contributi.
Grado di partecipazione alla comunità	Presenza di domande e risposte significative sui canali di comunicazione offerti dal corso, richieste di ricevimento e delucidazioni, bug fixing del materiale messo a disposizione dal docente.

4.2 Documentazione

La valutazione della documentazione verte sull'analisi dello scritto e sulla sua capacità di descrivere con chiarezza il prodotto consegnato, i problemi

riscontrati durante l'implementazione e le soluzioni adottate, **soprattutto grazie all'uso di esempi**. Per ciascuno degli aspetti riportati nella seguente tabella viene assegnato un punteggio da 0 (insufficiente) a 5 (ottimo).

CRITERIO	**DESCRIZIONE**
Qualità dell'informazione	La documentazione fornisce una descrizione chiara e completa del progetto implementato.
Uso di esempi	Presenza di esempi (narrativi, grafici, etc.) utili alla comprensione delle scelte implementative del gruppo o di scenari d'uso specifici.
Analisi delle scelte implementative	Individuazione e descrizione dei problemi incontrati durante l'implementazione, con particolare enfasi sui problemi legati alla concorrenza e alla distribuzione. Discussione delle soluzioni adottate e di soluzioni alternative valide.