LABORATORIO DI SISTEMI OPERATIVI A.A. 2020-21

AlmenoNonèJolie	
alessandro.pasi6@studio.unibo.it	
Nome	Matricola
Pasi Alessandro	**893268**
Carboni Carlotta	893185
Palmieri Luca	839445
Alsina Gabriel Riccardo	890823
Halychanska Anhelina	906420

- **DESCRIZIONE DEL PROGETTO**

  Per avviare il gioco la prima azione da svolgere è "runnare" la classe MainSchermata contenente il main e selezionare "*Host game*", che inserendo un nome e la porta (preventivamente aperta nel router) permetterà di creare un Server. Per connettersi ad un server bisogna selezionare dalla home "*Find game*" e compilare i campi con Nickname, indirizzo IP del server e relativa Porta.
  Se i Client proveranno a collegarsi prima che venga istanziato un Server la connessione fallirà. Una volta raggiunto il numero minimo di client il server potrà decidere se far partire la partita o in alternativa aspettare che si raggiunga il numero massimo di client, ovvero 4.
  Il gioco risiede interamente sui client (sulla classe schermo) e sono loro che invieranno continui aggiornamenti al server che a sua volta si occuperà di aggiornare tutti i client sullo stato degli altri client connessi (Server non-autoritario); questo meccanismo di condivisione delle informazioni vale anche per quanto riguarda la vittoria o la sconfitta: sono gli stessi client che informano il server se hanno vinto o perso. Possiamo quindi dire che il gioco è composto da Client, un Server e la classe schermo su cui risiede la meccanica di gioco.
  I polimini durante la loro caduta possono essere ruotati a destra (tasto D) e a sinistra (tasto S), possono anche essere spostati in entrambe le direzioni e verso il basso con le freccine o nel caso si decida di voler farlo atterrare direttamente si può usare la barra spaziatrice.
  Le righe spazzatura vengono generate quando si riesce a eliminare più di una riga contemporaneamente e verranno mandate al giocatore selezionato in precedenza. Per capire chi sia il giocatore a cui verranno mandate le righe basta guardare quale sia il campo evidenziato e nel caso lo si voglia cambiare basterà schiacciare il numero sulla tastiera che fa riferimento al giocatore prescelto; attenzione però che quando un giocatore perde (verrà segnalato con una scritta "Ha perso" sul suo campo) sarà ancora possibile selezionarlo per le righe spazzatura, ma in questo modo si starà perdendo l'opportunità di far perdere qualche altro giocatore!

Di seguito elencheremo i tasti (mentre i comandi sono descritti nelle rispettive classi) da conoscere per poter giocare al “Generico gioco multigiocatore a blocchi”:

- **Frecce direzionali**: movimento del pezzo
- **s**: ruota il pezzo in senso antiorario
- **d**: ruota il pezzo in senso orario
- **Barra spaziatrice**: fai cadere il pezzo istantaneamente
- **Numeri 1 2 3**: seleziona l’avversario a cui inviare le righe spazzatura

- **DIVISIONE DEI COMPITI**

  - **Alsina**: meccanica di gioco (Griglia, RiceviStato, InviaStato, package Pezzi, package Blocchi), rotazione dei polimini, controllo collisioni e formattazione visiva della classe Schermo;
  - **Carboni**: grafica e inserimento dei dati in MainSchermata, meccanica delle righe spazzatura, /restart, /startagain, /quit, segnalazione di quando un client perde;
  - **Halychanska**: Classe Youwin e Gameover e i System.out.println per controllare quello che succede durante il ciclo di vita del programma;
  - **Pasi**: struttura Client/Server (Client, Sender, ClientHandler, ConncetionListener, ServerSender,Server) e PlayAgain;
  - **Palmieri**: adeguamento grafico dello schermo, regolamento del gioco, controlli laterali;

- **STRUMENTI UTILIZZATI**

  Durante la realizzazione del progetto sono state eseguite riunioni di gruppo settimanali tramite la piattaforma Teams di Microsoft, essendo l’unica che avevamo tutti. In ogni riunione ci aggiornavamo sui progressi fatti durante la settimana; nel primo periodo le riunioni servivano per assegnare a ciascuno delle “sfide” che servissero poi per la realizzazione del progetto, per esempio, creare una chat per comprendere a pieno la comunicazione client/server oppure mostrare un quadrato sullo schermo e far si che si muovesse alla pressione di un tasto. Una volta raggiunti i livelli di conoscenza dei vari costrutti che avremmo dovuto utilizzare abbiamo iniziato con la vera stesura del progetto. In queste riunioni ci informavamo sui progressi raggiunti, sulle difficoltà riscontrate, sui compiti finiti e sui nuovi punti da svolgere. I compiti venivano assegnati ad un soggetto con relative tempistiche di completamento; nel caso avesse avuto difficoltà o non fosse riuscito a risolverlo nel tempo richiesto gli si affiancava un componente del gruppo. Durante il resto della settimana continuavamo a rimanere in contatto tramite whatsapp per dubbi, confronti e problematiche relative al codice. Verso la fine del progetto le riunioni non si sono fatte solo al lunedì, ma più volte nell’arco della settimana per avere sotto controllo i possibili problemi da risolvere e rifinire al meglio il codice. Per quanto riguarda invece la condivisione del codice abbiamo sempre usato il repository su gitlab.

- **DESCRIZIONE DELLE CLASSI E DEI METODI DEL PROGETTO**

a. Architettura Client/Server

- ***MainSchermata***: Classe dove risiede il main, è qui che si può decidere se iniziare il gioco come client (sarà comunque necessario che il Server a cui ci si vuole collegare sia già attivo) o come Server per istanziare una nuova partita. Inoltre, qui vengono effettuati i controlli sui nickname e sulla correttezza dell'inserimento dei parametri per creare sia client che Server. Si potranno anche leggere le regole del gioco, con la spiegazione dei comandi, e i crediti.

- ***Client***: Il client viene avviato selezionando "*Find game*" e accetta come argomenti l'indirizzo IP del server a cui connettersi e un nickname (univoco all'interno del sistema) come elemento di distinzione tra i vari Client. Una volta che la connessione al Server sarà avvenuta con successo si visualizzerà la chat pre-partita in cui è possibile inviare e ricevere messaggi da Client e Server. Alla ricezione del messaggio *"/start"* proveniente dal Server verrà inizializzato il "generico gioco multigiocatore a blocchi", tramite la classe Schermo di cui parleremo più avanti.

- ***Sender***: Thread del Client (ogni client ha il proprio Sender) che si occupa dell'invio di messaggi verso il server che a sua volta si occuperà di inoltrarli agli altri client connessi tramite il metodo "broadcastMessage". Se il messaggio che si vuole inviare è "*/quit*" il client tornerà alla schermata iniziale dove potrà collegarsi a un nuovo Server o creare lui stesso un nuovo Server; nel caso in cui si volesse uscire definitivamente dal gioco non si dovrà scrivere "*/quit*", ma si dovrà cliccare sul bottone "*Close*".

- ***Server***: Il server viene avviato selezionando, come detto in precedenza, "*Host game*" e accetta come argomento la porta su cui mettersi in ascolto. In seguito, attende che si connettano almeno 2 giocatori e al massimo 4. Nel caso in cui i client connessi siano 4 non si avrà bisogno di nessun comando da parte del Server per iniziare la partita, ma inizierà in automatico. Esso si occupa di inizializzare tutti i Thread, i componenti e le variabili pubbliche più importanti necessarie al funzionamento del programma.

- ***ConnectionListener***: Il Thread listenerThread si occupa di restare in ascolto per nuove connessioni provenienti da client che vogliono collegarsi al server. Vengono accettate e gestite un massimo di 4 connessioni contemporaneamente. Qualora si connettesse il numero massimo di Client la partita inizierebbe istantaneamente attraverso l'invio automatico del comando *"/start"* a tutti i giocatori. Per ogni client che si connette al Server viene creato un handlerThread (ClientHandler), così che possano venire gestiti singolarmente dal server.

- ***ClientHandler*****:** Per ogni client che si connette al server viene creato un Thread handlerThread che si occupa di gestirlo. Nello specifico questa classe è dove il Server riceve i messaggi di ogni client. I client non possono comunicare tra di loro in modo diretto: tutti i messaggi vengono prima inviati al Server che poi si occuperà a sua volta di inviare il messaggio a tutti gli altri tramite il metodo ""*broadcastMessage*" che inoltra il messaggio a tutti i giocatori tranne che al mittente. Tramite questa classe i client possono scollegarsi dal Server, senza spegnersi: se un client invia il comando "*/quit*" il Server lo scollegherà e comunicherà a tutti gli altri client il

cambiamento. All'interno di questa classe viene eseguito inoltre il controllo sul Nickname per verificare che non sia presente un altro Client con lo stesso nome. In caso di rilevamento del duplicato il Server rigetta la connessione e invita l'utente a modificarlo prima di ritentare il collegamento.
ClientHandler righe 51-60

- ***ServerSender***: Thread di comunicazione del server che permette al server di mandare messaggi a tutti i client durante la chat prepartita, ma non solo, tramite il metodo "broadcastServerMessage". All'interno troviamo il controllo del numero dei giocatori (se i client sono meno di 2 non può iniziare il gioco). Durante l'attesa dell'inizio del match è il Thread ServerSender che permette al Server, oltre a poter partecipare alla chat prepartita, di eseguire i seguenti comandi:

    - */start*: avvia la partita coi giocatori attualmente connessi (se sono almeno 2).
      ServerSender righe 48-89, Client righe 170-224
    - */quit*: disconnette i client connessi e chiude il server.
      ServerSender righe 89-112, Client righe 361-380
    - */clear*: permette di ripulire la chat prepartita

  Per iniziare la partita o si aspettano 4 giocatori oppure, con un minimo di 2 client connessi, si aspetta un comando dal Server come visto in precedenza. Durante la partita il server, sempre tramite questo Thread, può lanciare una serie di comandi di Utility:

    - */pause*: mette in pausa il gioco per tutti i giocatori, il server manda il messaggio, ma sono i client quando ricevono "*/pause*" che si mettono in pausa. Mentre il gioco è in pausa, i polimini non cadono e i giocatori non possono eseguire alcuna mossa.
      Client righe 321-326
    - */resume*: permette, dopo aver messo in pausa il gioco, di riprendere da dove si era lasciato; esattamente come per "*/pause*", sono i client che ricevendo il messaggio riprendono il gioco.
      Client righe 326-331
    - /*clear*: permette di ripulire la chat prepartita
    - */restart*: termina la partita corrente scollegando i client dal Server.
      ServerSender righe 112-124,Client righe 344-361
    - */startagain*: termina la partita corrente iniziandone immediatamente una nuova.
      ServerSender righe 124-138, Client righe 331-344
    - */quit:* termina la partita corrente, disconnette i client connessi e chiude il server.

- ***Schermo***: Thread schermo viene lanciato dai client (uno per ciascuno) nel momento in cui ricevono dal Server il comando "*/start*". Le meccaniche di gioco avvengono tutte in questa classe. Il campo di gioco più grande è il proprio campo, mentre gli altri più piccoli sono i campi degli avversari, riconoscibili dal nome sotto al campo; un solo minicampo sarà evidenziato di giallo: è il giocatore a cui si invieranno le righe

spazzatura generate. Nel caso si voglia cambiare persona a cui inviarle basterà schiacciare sulla tastiera il numero che fa riferimento al giocare desiderato.

b. Il gioco

- ***GameLoop***:

Il game loop è il ciclo while che va avanti fino alla fine del gioco (`while (!Schermo.gameOver)`), durante il gameLoop facciamo dei controlli necessari per la continuazione del gioco e per verificare le azioni compiute dai giocatori.
Schermo righe 172-244

I controlli che facciamo sono:

```
if(Client.winner){
```

Controlliamo se si è rimasti gli unici giocatori rimasti, nel caso si stoppa la partita e si fa partire il thread di YouWin.

```
for(KeyStroke key : keyStrokes) {
```

Per ogni elemento nella lista KeyStroke richiamiamo il metodo della classe schermo processKeyInput che ci permette di tradurre in eventi tutti i tasti che abbiamo premuto.

```
if((pezzoScelto.collisioneSotto() && brickDropTimer.getDropBrick()) ||
barra)
```

Con questo if stiamo controllando se sia avvenuta una collisione e se è successa renderemo il nuovo pezzo parte della struttura. Controllorighe ritorna se e quante righe abbiamo liberato dalla struttura con la nuova collisione: nel caso combo, ovvero le righe liberate, sia maggiore di uno invieremo al Server questo messaggio

```
datas = "spazzatura-" + usernameDestinatario + "-" + combo;
```

“spazzatura” serve per far capire si tratti di righe spazzatura, usernameDestinatario per indicare il destinatario delle righe e combo per indicare il numero di righe da aggiungere nel campo di usenameDestinatario.
Schermo righe 192-197

Il Server invierà il messaggio a tutti i giocatori, saranno loro che se nel messaggio troveranno il loro nome dovranno aggiungersi le righe spazzatura.
I client quando riceveranno messaggi dal Server controllano se contiene la parola spazzatura, in tal caso divideranno il messaggio in un array di dimensione 3, se [1] è uguale al nome del client allora controllerà anche l’ultima parte dell’array per sapere quante righe spazzatura aggiungersi.
Client righe 290-315

Successivamente disegnerà le nuove righe spazzatura con aggiungiSpazzatura.
Griglia righe 102-151

```
for(int i=4; i<8; i++) {
    if(campo.griglia[i][0].stato>1){
```

Qui controlliamo che la prima riga al centro sia libera, nel caso sia occupata il gioco richiamerà il metodo “haPerso”
Schermo righe 476-486

che avviserà gli altri Client della nostra sconfitta e farà partire il thread con la schermata GameOver.

```
pezzoScelto = prossimoPezzo(schermo);
```

Avendo avuto una collisione dovrà scendere un nuovo pezzo, questo sarà scelto casualmente dal metodo ProssimoPezzo in Schermo
Schermo righe 274-287

```
if(aggiungiSpazzatura != 0){
```

Controlla che aggiungiSpazzatura sia uguale a zero, se non è così vorrà dire che qualcuno ci ha inviato delle righe spazzatura e dovremo aggiungerle al nostro campo, per farlo utilizziamo il metodo aggiungiSpazzatura di Griglia
Griglia righe 102-151

```
if(brickDropTimer.getDropBrick()) {
```

Il brickDropTimer scandisce il tempo per la caduta dei pezzi e questo if controlla se è finito il tempo, nel caso richiamerà il metodo di pezzo Scendi, inoltre manda l'aggiornamento a tutti gli altri giocatori del proprio stato della griglia con InviaStato.
Schermo righe 223-234

```
screen.refresh();
```

Dopo tutti i controlli "refresha" lo schermo e riparte il ciclo while.

- ***Meccanica di gioco***:

Il gioco si basa su una matrice [12][24] chiamata **Griglia** che costituisce il campo da gioco, questa griglia è composta da oggetti **Blocco**, un blocco è un quadrato disegnato con il metodo di Lanterna **fillRectangle**, la classe Blocco è estesa da quattro classi figlie, ogni classe ha un ID, che le identifica, chiamato Stato:

- **BloccoVuoto**: indica un quadrato nella griglia vuoto, stato = 0.
- **BloccoPieno**: ovvero un blocco dove sta passando un blocco del pezzo che cade, stato = 1.
- **BloccoStruttura**: per struttura intendiamo tutti i quei pezzi che avendo avuto una collisione non possono più muoversi, verrà tolta solo quando verrà completata la riga, stato = 2.
- **BloccoSpazzatura**: un blocco della riga spazzatura inviata da un avversario, questo blocco non potrà più essere tolto, stato = 3.

Il costruttore di Blocco è importante perché è il momento che smettiamo di pensare in modo astratto nella griglia e lavoriamo in modo concreto sullo schermo.

Un **Pezzo,** che rappresenta il polimino**,** è un insieme di quattro **BloccoPieno**, e possono essere di diversi tipi che estendono la classe Pezzo e si trovano tutti nel Package pezzi. All'interno della classe Pezzo ci sono tutti i controlli necessari per far funzionare il gioco in modo corretto: collisione sotto, che non possa ruotare nel caso un blocco del polimino esca lateralmente (se non controllato genererebbe un java.lang.ArrayIndexOutOfBoundsException). I metodi di pezzo non fanno il vero e proprio controllo, ma con un for fanno i controlli per ogni blocco di pezzo, i metodi di

controllo si trovano dentro Blocco.
Blocco righe 81-87,Pezzo righe 90-95 righe 103-108 righe 119-131 righe 137-142

La classe Griglia si occupa di controllare il proprio stato e di aggiornarsi ad un determinato evento, per esempio, il metodo *controlloRighe* controlla tutta la griglia, se trova una riga composta da tutti **BloccoStruttura** allora svuoterà quella riga (con il metodo *eliminaRiga*) e farà cadere la struttura sopra (con il metodo cadutaStruttura).
Griglia righe 43-92

- ***La spazzatura***:

Quando un giocatore riesce a liberare 2 o più righe contemporaneamente genera le righe spazzatura che verranno mandate a un singolo giocatore che vedrà il proprio campo rimpicciolirsi. Più righe verranno liberate più righe spazzatura verranno generate:

Righe	Righe spazzatura
2	1
3	2
4	4

usernameDestinatario, come abbiamo detto prima, è la variabile che contiene il nome del client a cui vogliamo inviare le righe spazzatura e di seguito spiegheremo come gli viene assegnato il nome.

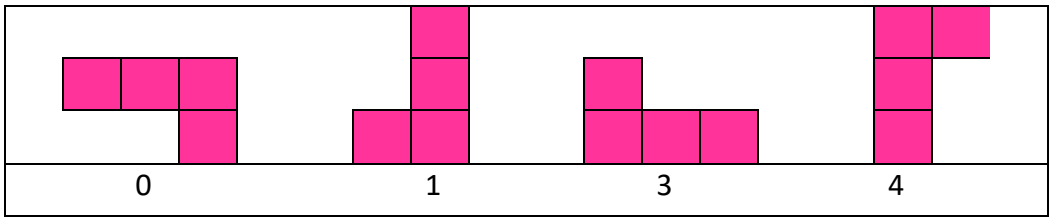
Quando viene avviata una partita Schermo disegna tanti mini campi tanti quanti sono gli avversari e quando disegna il primo campo lo evidenzia in automatico di giallo e assegna alla variabile usernameDestinatario il nome del client assegnato a quel campo.
Schermo righe 130-152

Durante la partita, se si è in più di due giocatori, si può cambiare persona a cui inviare le righe spazzatura schiacciando il tasto corrispondente al numero sotto al campo del giocatore a cui vogliamo inviarle.
Schermo righe 354-391

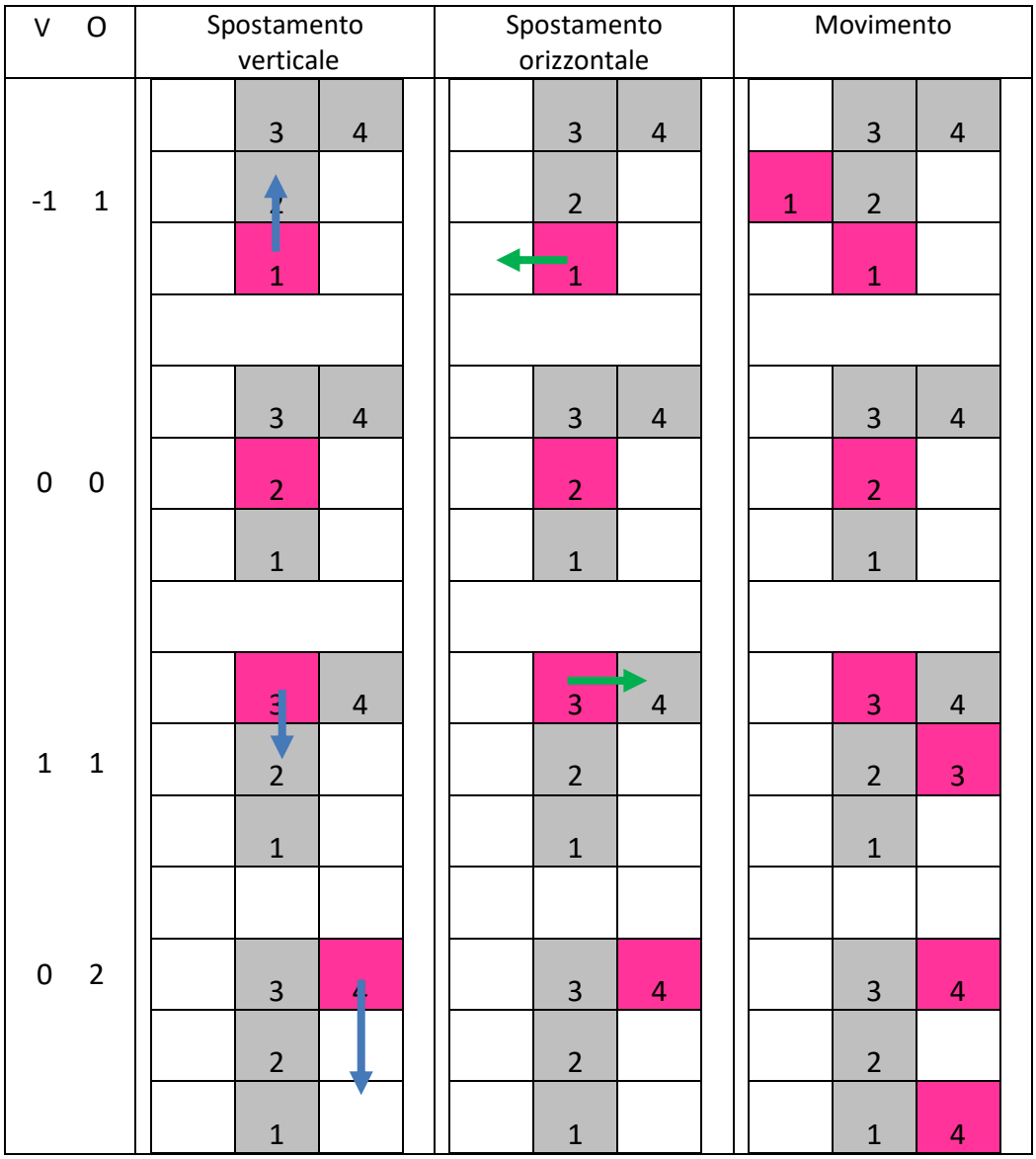
- ***Le rotazioni***:

Nel generico gioco a blocchi un pezzo deve poter ruotare per incastrarsi meglio nella griglia, le rotazioni si basano su due matrici chiamate spostamentoVerticale e spostamentoOrizzontale. Le rotazioni si basano sul muovere ogni singolo blocco di pezzo nella griglia: per capire bene prendiamo come esempio pezzoJ. PezzoJ ha le due matrici[4][4], questo perché il pezzoJ può avere 4 tipi di rotazioni diverse:

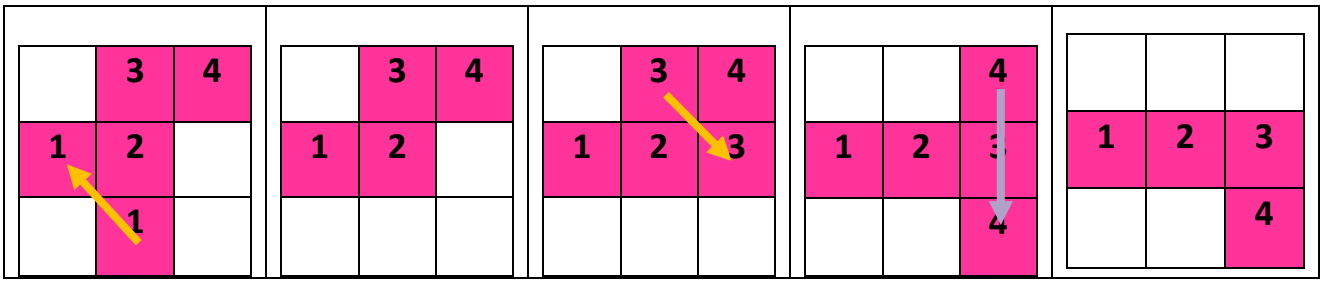
```
spostamentoVerticale= new int[][]{{-1, 0 , 1, 0},  {1, 0 , -1, -2}, {1, 0, -1,
0}, {-1, 0 , 1, 2}};
spostamentoOrizzontale = new int[][]{{1 , 0 , -1, -2}, {1, 0, -1, 0}, {-1, 0, 1,
2}, {-1, 0, 1, 0}};
```



Diverso è il caso di un pezzo con solo due rotazioni (come il pezzoLungo), ma la logica rimane la stessa.
Nella matrice spostamentoVerticale il primo numero, ovvero quello che indica la riga [n][] quale rotazione dobbiamo svolgere, il secondo numero [][m], invece, indica come dobbiamo spostare quel pezzo <u>verticalmente</u> (per questo spostamentoVerticale). Stesso discorso per spostamentoOrizzontale e il movimento in orizzontale.
SpostamentoVerticale + spostamentoOrizzontale indica dove dovrà andare il blocco.



Spostamento insieme



La rotazione nell'altro senso (antiorario) si annulla la rotazione prima, quando richiamiamo ruota gli passiamo la rotazione-1 (nel caso di 0 gli passiamo la rotazioneMax) e moltiplichiamo ogni spostamento sia orizzontale che verticale per -1, annullando così lo spostamento. Questo metodo ci aiuta anche con le collisioni con l'esterno e con altri blocchi della struttura o spazzatura visto che così sappiamo dove si muoverà il blocco.

Pezzo righe 34-36, BloccoPieno righe 18-24 righe 26-29

- **Multithreading**:

Durante l'esecuzione del gioco vanno in esecuzione più thread in contemporanea:

- **Schermo**: la classe principale del gioco è un thread, viene fatto partire dal Client quando riceve il comando "*/start*"
- **CadutaBlocco**: la classe CadutaBlocco è un thread che durante il gameLoop avvisa il sistema per far cadere un nuovo pezzo.
- **InputKey**: questo thread è sempre in ascolto sui tasti che vengono premuti dal giocatore e l'inserisce in una lista, poi ogni elemento della lista viene analizzato dal metodo processKeyInput in Schermo.
- **InviaStato**: invia lo stato della griglia in quel momento, la griglia viene tradotta da Blocchi a numeri creando una matrice [12][24] e traducendo ogni blocco per il suo Stato, successivamente viene creata una Stringa che inizia con il proprio *Username* + ":" e successivamente tutti i numeri degli stati, chiude con ":" + colore. Il colore è il colore del pezzo che sta scendendo, questa stringa viene mandata al Server che poi la manderà a tutti.
- **RiceviStato**: se il Client riceve una stringa che contiene ":0" capisce che deve tradurre lo stato di una griglia di un altro giocatore. Come prima cosa splitta la stringa ricevuta ogni ":", avendo così 3 strighe:
  [0]: sarà l'username del giocatore a cui va riaggiornato lo stato della griglia.
  [1]: lo stato della griglia.
  [2]: il colore del pezzo che sta cadendo.

  Lo stato della griglia viene nuovamente splittato, su ogni carattere viene eseguita un'operazione di casting e messo in una matrice [12][24], successivamente ogni numero viene tradotto nel suo corrispondente Mini_Blocco. I mini-blocchi sono strutturati come i blocchi, ma più piccoli e senza metodi per guardare la collisione con muri o struttura questo perché il

controllo avviene già in blocco i mini-blocchi sono solo la stampa di quello che sta succedendo in un altro campo, è quindi impossibile che escano dalla griglia o diano origine a qualsiasi tipo di problema riguardante i blocchi.

- **PROBLEMI AFFRONTATI**

a. Problematiche di concorrenza

Durante la creazione del progetto abbiamo riscontrato tre grandi problemi con la concorrenza. Per risolverli abbiamo usato semafori e monitor: i primi per controllare le parti di codice critiche, ovvero processi o sub-processi che potrebbero accedere simultaneamente alla stessa risorsa condivisa, invece i monitor per controllare i Thread.

- ***Semafori***

  - **semaforoColore:** questo semaforo ha risolto due grossi problemi. Quando si selezionava il giocatore a cui inviare la spazzatura l'evidenziatura gialla attorno a volte non si colorava o a volte prendeva colori diversi dal giallo e durante la partita si coloravano dei blocchi casuali sia nel campo che nel mini_campo. Questo problema si verificava perché più processi (blocchi, miniblocchi, evidenziatura) accedevano alla risorsa schermo. setForegroundColor creado appunto problemi di concorrenza. Per risolvere questo problema abbiamo messo un semaforo intorno alle parti di codice che accedevano a questa variabile, in questo modo i processi ci accedevano uno alla volta senza causare problemi.
    Schermo righe 436-445 righe 460-469 righe 502-508 righe 513-520 righe 525-532, Blocco righe 33-39, Mini_Blocco righe 32-37 righe 45-50

  - **Traduzione:** questo semaforo serve per un altro grande problema, ovvero quello che durante il gioco i polimini del campo degli avversi cadevano un blocco alla volta (ricordiamo che i polimini, ovvero i pezzi, sono formati da più blocchi) sembrando che ci fosse un "lag" nella connessine. Il problema era che il gioco inviava il proprio stato proprio nel momento che stava facendo cadere il pezzo e l'altro giocatore (giustamente) stampava la griglia con le informazioni ricevute: quindi separando i blocchi. Per aggiustare questo problema abbiamo messo un semaforo intorno ai metodi di **riceviStato** e intorno al refresh dello schermo.
    Schermo righe 237-242; RiceviStato righe 36-53

  - **semaforoSpazzatura:** questo semaforo è servito per gestire il caso in cui mentre un polimino sta per entrare in contatto con la struttura arriva una riga spazzatura. Prima dell'inserimento di questo semaforo si aveva un problema di concorrenza in cui il polimino non percepiva la collisione, ma riusciva ad andare oltre alla struttura; invece con l'uso del semaforo questi due processi (campo.aggiungi spazzatura e pezzo Scelto.scendi) non possono essere più eseguiti contemporaneamente.
    Schermo righe 212-218 righe 225-232 righe 297-302 righe 311-318.

- **SemaforoConnectedClient:** questo semaforo serve a gestire l'accesso e la modifica dell'HashMap "connectedClients" utilizzato per memorizzare tutti i client connessi in tempo reale. Essendo un componente critico la presenza del semaforo è stata necessaria per ovviare ai problemi di concorrenza sia durante il gioco che durante la prepartita
  ClientHandler righe 145-169

- ***Deadlock***

  Nonostante i semafori abbiano risolto il problema della concorrenza in più casi avevano generato problemi di deadlock. Il primo deadlock generato è stato difficile da riconoscere, non capivamo fosse un blocco generale del programma; invece, nei casi successivi l'abbiamo riconosciuto subito e abbiamo solo dovuto capire quale fosse il punto corretto dove rilasciare i semafori per evitare il deadlock. In quasi tutti i casi il problema era che quando si ricominciava una partita c'era il rischio che la partita precedente fosse finita con i semafori acquisiti e che quindi nella partita successiva nel momento di acquisizione del semaforo il processo venisse bloccato, dovevamo quindi rilasciare i semafori prima che la partita riiniziasse.

- ***Monitor***

  Dichiarando un metodo **synchronized** si vincola l'esecuzione del metodo ad un solo thread per volta. I Thread che ne richiedono l'esecuzione mentre già uno sta eseguendo vengono automaticamente sospesi, in attesa che il thread in esecuzione esca dal metodo. All'interno del programma tutti i Thread sono synchronized così che l'intero corpo diventi sincronizzato e si migliori la concorrenza.

b. Problematiche Client/Server

Durante la stesura del codice riguardante le connessioni il primo obiettivo raggiunto è stato la creazione di una chat tra un unico Client e un Server. In quel momento è sorto il primo grande problema, ovvero, come poter stabilire e gestire le connessioni di più Client contemporaneamente. Per risolverlo ci sono venuti in aiuto i Thread ovvero parti di codice che vengono eseguite contemporaneamente in modo autonomo. Sono stati quindi creati Thread per gestire i Task più importanti come:

- Invio e Ricezione di dati a lato Server: Server, ServerSender e ClientHandler;
- Invio e Ricezione di dati a lato Client: Sender e Client;
- Gestione delle connessioni: ConnectionListener.

c. Problematiche vittoria e sconfitta

Potendo i Client comunicare solamente col Server e possedendo solamente dati in locale è nato il problema di come far sapere a tutti, durante una partita, lo stato di gioco e, nel caso, gestire la Vittoria o la Sconfitta.
Il metodo più semplice e veloce che è stato implementato varia a seconda dei due casi.
Come primo analizziamo il caso di sconfitta; quando un giocatore perde (Il polimino una volta creato impatta subito con la struttura) invia un messaggio al

Server con la seguente sintassi *"<Nickname>-lost"*, a quel punto il clientHandler relativo al Client che ha inviato il messaggio controlla la sintassi e, verificando che è conforme ad un messaggio di sconfitta lo trasmette agli altri giocatori che lo riceveranno nel Client e, attraverso un altro controllo di sintassi rimuoveranno il Player dalla lista di giocatori ancora in gara e aggiungeranno sul campo di quel client la scritta "Ha perso".
Client righe 314-321
Solo adesso l'utente sconfitto chiuderà la connessione col Server in automatico e verrà aperta la schermata GameOver.
Schermo righe 476-486

Per la vittoria è diverso, il Client ciclicamente controlla se nella lista in cui inizialmente sono presenti tutti i giocatori si trovi solo lui. Se ciò accade significa che tutti gli altri Players hanno perso, in quel momento l'utente che ha vinto invia al Server un messaggio con la sintassi *"<Nickname>-winner"* cosicché il clientHandler associato riconosca il messaggio di vittoria, resetti la variabile di gioco "gameStarted = false" e chiuda la connessione con l'utente che farà partire la schermata YouWin.

In entrambi i casi, la schermata presenta 3 button:

- *Playagain*: permette di ricollegarsi al Server senza dover passare per la schermata iniziale, quindi evitando l'inserimento dei dati.
  YouWin righe 143-155, GameOver 117-129

- *Close*: ferma l'esecuzione del programma.
  YouWin righe 157-167, GameOver 131-141

- *Esci dal server*: in realtà quando si perde o si vince ci si scollega già dal Server, ma con questo bottone si può tornare alla schermata iniziale dove si potrà collegare a un nuovo Server o diventare lui stesso un nuovo Server.
  YouWin righe 169-181, GameOver 143-155

- **Sconfitta**

  Nel gioco ci sono tre modi per perdere:

  - Interrompendo l'esecuzione del programma chiudendolo, in questo modo non si perde veramente, perché non si aprirà la schermata di GameOver, ma i giocatori visualizzeranno lo stesso sul quel campo la scritta di sconfitta.

  - Se prima dello spawn del nuovo polimino almeno uno dei quattro blocchi centrali è occupato, può succedere, che quando il pezzo "spawni" lo faccia dentro la struttura, questo non riuscirà a scendere prolungando la vita di un turno e si perderà.

  - L'ultimo modo è se la struttura con l'arrivo delle righe spazzatura va oltre la riga 0 della griglia.

Quando si perde parte (oltre che la vergogna) il Thread chiamato GameOver.

- **Vittoria**

L'ultimo che rimane in gioco vince, tecnicamente siamo una battle royale, ma rimaniamo umili. Quando si vince fa partire il thread YouWin.