“BlockWar” implementa un generico gioco multigiocatore a blocchi ispirato a Tetris. Lo scopo del giocatore è impedire che la pila di polimini caduti raggiunga la cima del campo di gioco.
Il progetto è strutturato in quattro pacchetti: Tetramini, Logic, Net e GUI.

Il pacchetto Tetramini rappresenta le figure di gioco. Nel pacchetto Net, invece, vi sono classi che implementano la connessione client-server e, nel pacchetto GUI, viene implementata la renderizzazione grafica.

La struttura dati che rappresenta il campo da gioco è una matrice di interi definita nel seguente modo:

- Fuori campo: numero 9
- Bordo del campo: numero 8
- Tetramini standard: da 1 a 7
- Pentamino speciale: 11
- Campo di gioco vuoto: 0

9	9	8	0	0	0	0	0	0	0	0	8	9	9
9	9	8	0	0	0	6	6	0	0	0	8	9	9
9	9	8	0	0	0	6	6	0	0	0	8	9	9
9	9	8	0	0	0	0	0	0	0	0	8	9	9
9	9	8	0	0	0	0	0	0	0	0	8	9	9
9	9	8	0	0	0	0	0	0	0	0	8	9	9
9	9	8	0	0	0	0	0	0	5	0	8	9	9
9	9	8	0	1	1	0	0	0	5	0	8	9	9
9	9	8	0	7	1	1	0	0	5	0	8	9	9
9	9	8	7	7	7	0	0	0	5	0	8	9	9

Pacchetto Tetramini

Il pacchetto contiene le caratteristiche delle figure

Classe Tetramino

È una classe astratta e viene estesa dalle classi che definiscono le forme. Essa contiene un riferimento all’ enumerazione direction che ne rappresenta l’orientamento e ha due metodi: rotateRight() e rotateLeft().
In base alla variabile direction della classe che estende Tetramino.
Richiamando il metodo rotateRight() nel caso in cui la figura è nel caso UP, viene ruotata verso destra di 90° e quindi la direzione diventa RIGHT. Con questo metodo viene cambiata solo la variabile direction, la rotazione effettiva della figura si effettua con il cambio di coordinate che avviene conseguentemente in base al contenuto di direction.

Le Figure
Le classi delle figure: I, L, O, J, T, Z, C sono caratterizzate da numero d'identità, ID, e da un insieme di coordinate. L'ID serve per identificare i tetramini in modo univoco sul campo di gioco, permettendoci di colorarli diversamente in fase di disegno. Nel momento in cui scende la figura, ogni valore 0 viene cambiato nel valore di ID.

Classe Coordinates
I tetramini sono definiti da un insieme di coordinate, Coordinates, definite nel pacchetto Logic. Queste coordinate sono considerate assolute rispetto al piano cartesiano e rappresentano i punti di cui sono composte le figure, rappresentate da due interi x, y. Una lista di coppie di punti (x, y) definisce la forma del tetramino.
Ad esempio, quando il tetramino J è ruotato a destra avrà questo insieme di coordinate.

```
case RIGHT:
    coordinates.add(new Coordinates(0, 0));
    coordinates.add(new Coordinates(0, 1));
    coordinates.add(new Coordinates(1, 0));
    coordinates.add(new Coordinates(2, 0));
    break;
```

Per muovere i tetramini sul campo di gioco vengono utilizzati gli offset di X e Y. Questi vengono incrementati per poi applicarsi su ogni punto che compone la figura. il funzionamento è spiegato più avanti.

Pacchetto Logic

Inizio e svolgimento game loop
Le classi necessarie per la logica di gioco sono le seguenti: GameLoop, ControlGameLoop e BrickFaller.
Il thread della classe GameLoop esegue l'elaborazione dei comandi e la relativa modifica da apportare alla matrice. Il thread della classe ControlGameLoop serve per memorizzare i comandi dati in input dall' utente all' interno di una lista (simile ad un buffer) in modo tale da poter processarli uno ad uno senza che si perdano. Infine, il thread della classe BrickFaller gestisce il timing di caduta del pezzo.

Classe TetraminoGenerator
Le figure vengono scelte in modo casuale, perciò, è definito il metodo che restituisce in modo randomico il tetramino.

Classe GameLoop
Il Loop di gioco si sviluppa in una serie di fasi in successione:

- Fase di acquisizione degli input
- Controllo sulla posizione in cui il tetramino intende spostarsi.
- Cancellazione della forma alla posizione precedente
- Processo di elaborazione dell'input
- Disegno nella nuova posizione

Ad ogni ciclo di game loop, a prescindere dall'input inserito, viene verificata una condizione che controlla mediante la classe BrickFaller se è il momento di far o meno scendere verso il basso il tetramino. Questo consente di ricreare la caduta del pezzo.
Durante la caduta, una volta che il tetramino non è più in grado di compiere uno spostamento verso il fondo, viene settata la variabile nextShape nuovo tetramino generato da TetraminoGenerator.
Viene poi controllata l'eventuale presenza di righe complete e spazzatura (nel caso vi siano). Il ciclo termina con l'update della board, e con il metodo di disegno che si occuperà di colorare in base al valore numerico letto sulla matrice totale (con i vari giocatori). La condizione di Game Over viene controllata ogni volta che un tetramino si è posato; ovvero non riesce a completare la caduta.

Classe Controller
Questa classe contiene tutti i metodi che verificano la possibilità di spostare, o meno, una figura in una data posizione per ogni possibile movimento del pezzo.
Tutti i metodi di movimento richiamano il metodo di controllo principale canDrawAtPosition: metodo booleano che ritorna se è possibile fare una determinata mossa.
Nel metodo, si controlla, che la posizione da raggiungere sia effettivamente libera (presenza di altre figure o limiti del campo), e nel caso lo sia, di eseguire la modifica della posizione, cancellando e poi ridisegnandolo (nella nuova posizione). Un ulteriore controllo eseguito dal metodo consiste nel determinare i requisiti di game over.

Gli altri metodi (canGoRight(), canGoLeft(), canRotateRight(), canRotateLeft()) cambiano gli argomenti di canDrawAtPosition() per seguire i propri controlli. In base a dove si trova la figura, si controlla se può essere spostata: a destra, a sinistra, oppure ruotata.

I metodi undrawShape() e drawShape() descrivono i due passaggi che avvengono quando un tetramino viene mutato di posizione, ovvero: cancellazione nella posizione attuale e disegno in quella nuova.

Il movimento del tetramino:
Le coordinate di offset sono utilizzate per determinare la posizione effettiva del tetramino sulla matrice di gioco. Su queste ultime, saranno fatti controlli sulla direzione, descritti nella classe Controller.
Le coordinate che compongono le figure vengono considerate assolute, mentre gli offset sono coordinate relative alla posizione 0(riga), 0(colonna) dello Screen di Lanterna.
Quando viene spostata una figura si agisce sull'Offset, senza modificare le coordinate che la definiscono, così da evitare che venga deformata.

0	4	0	0	0
0	4	0	0	0
0	4	4	0	0
0	0	0	0	0
0	0	0	0	0

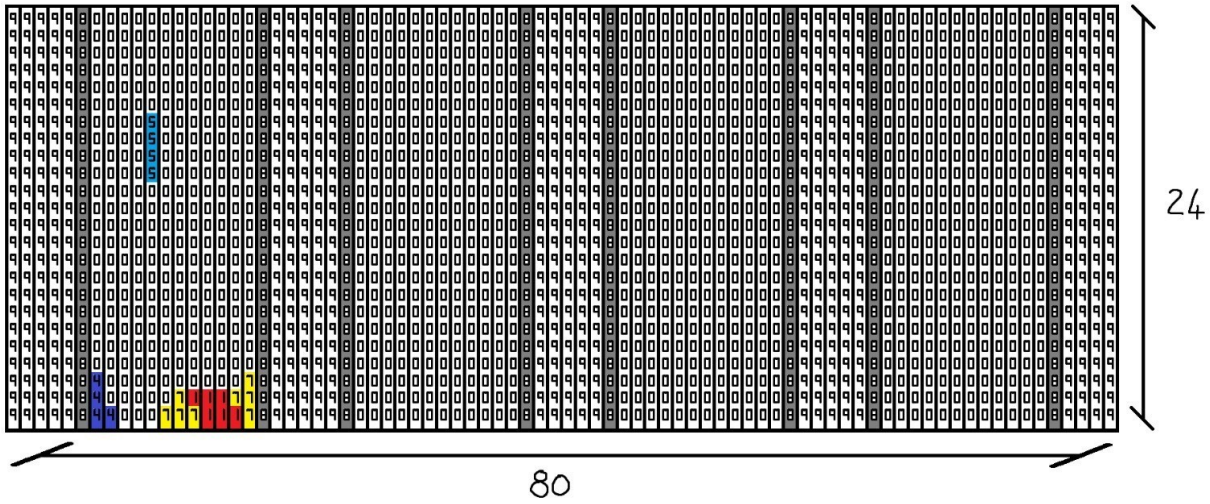
0	0	0	0	0
0	4	0	0	0
0	4	0	0	0
0	4	4	0	0
0	0	0	0	0

In rosso evidenziamo il cambiamento dei valori dei numeri d'identità quando viene spostato un tetramino in una nuova posizione.

Classe BoardManager
Questa classe contiene tutto ciò che riguarda l'aggiornamento e la creazione del campo di gioco totale (completeBoard).

Contiene i seguenti metodi:
UpDateMyBoard(): metodo che consente di aggiornare la parte di campo che riguarda il proprio giocatore.
upDateBoard(): metodo che aggiorna la board dei campi di tutti gli altri giocatori (non il proprio)
MergeMatrix(): metodo che consente di unire tutte le matrici per formarne una unica

Nella figura sottostante è raffigurata la matrice completa assemblata da mergeMatrix() nel caso in cui ci sono quattro giocatori connessi.



Classe Players
Questa classe rappresenta il singolo giocatore, contenendo tutte le informazioni necessarie per descriverlo, ovvero: id, righe spazzatura totali, bersaglio delle righe spazzatura e lo stato del game over (se ha perso o no). Da essa si accede alla matrice nella quale vengono eseguite le operazioni sul game loop. Quando il giocatore vince, verrà settata la matrice defaultWinnerBoard nella sua board, mentre quando perde, verrà settata la defaultEndGameBoard.

Classe BrickFaller

Questa classe implementa un thread che allo scadere di un tempo determinato, cambia una booleana che attiva il controllo di discesa della figura verso il basso. Questo viene eseguito per evitare che la figura rimanga in stallo quando il giocatore la vuole spostare, aumentando la reattività di gioco.

Classe ScreenUtil

Questa classe è usata per contenere metodi di utilità come il metodo checkCompleteLines() che controlla se sono presenti delle righe piene. All'interno di checkCompleteLines() c'è uno switch-case che in base alla quantità di righe completate definisce quante righe spazzatura, conseguentemente, vanno inviate.
Una volta eliminate le righe complete, le righe soprastanti, scendono verso il basso in modo da riempire lo spazio che si era svuotato. Viene quindi definito il metodo shiftDown() per spostarle verso il basso. Al contrario, quando l'avversario riceve le trashlines le righe già presenti sul campo vengono spostate verso l'alto con il metodo shiftUp().

Funzionamento Trashline

Il bersaglio delle trashline viene settato di default ad inizio partita in base al l'id del giocatore.
Il bersaglio di default è sempre il giocatore con l'id successivo (della lista di Player ordinata); l'ultimo giocatore della lista invece avrà come bersaglio di default il primo.
In alternativa, durante la partita, si può decidere a chi inviare le righe spazzatura premendo i tasti '1', '2', '3', '4' che corrispondono al ID giocatore, visualizzato anche sulla GUI. Un bersaglio valido deve essere sempre impostato, quindi, sono eseguiti dei controlli che impediscono di scegliere sé stessi, inoltre ad ogni ciclo di gioco viene anche controllato che il bersaglio impostato sia ancora valido. Ad esempio, se un giocatore avversario è in game over non può più essere bersagliato, quindi il programma cambia automaticamente il target scegliendone uno valido.
Le linee spazzatura sono inviate automaticamente appena disponibili, al target programmato. Esse verranno aggiunte al fondo del suo campo di gioco quando il suo tetramino in movimento avrà raggiunto il fondo del campo.

Problemi riscontrati nella scrittura delle classi del pacchetto LOGIC

Problema della Struttura di dati:
Un problema iniziale è stato trovare una struttura di dati adeguata al gioco: inizialmente si pensava ad un campo di gioco unico condiviso da tutti i giocatori. Successivamente si è optato per una struttura separata per ogni giocatore. In questo modo ogni player ha un campo di gioco indipendente, separando in questo modo i processi di gioco in locale e l'aggiornamento dello stato dei campi dei giocatori online.

Problema della pausa:
Una volta arrivato il comando dal server il client modifica la variabile booleana "pause", la quale fa entrare il GameLoop nel ciclo while dove è richiamato un wait su di un lock presente in ScreenUtil. Questo lock è utilizzato anche dal wait nel thread della classe ControlGameLoop, il quale verra` attivato dall' ingresso nel wait del GameLoop. Questo riesce a far interrompere e riprendere il gioco, ma la lettura dei keystroke non sembra fermarsi, permettendo ad un giocatore in pausa di eseguire delle mosse che verranno renderizzate una volta ripreso il gioco.

Problema del game over
Originariamente si pensava di controllare l'altezza del campo di gioco occupato da tetramini, oppure effettuare controlli di posizione sulle single figure. Tuttavia, sono sorti vari problemi, come il fatto che una figura durante il suo ciclo di vita si troverà sempre nella parte superiore dello schermo perché è lì che viene generata, si è quindi pensato di utilizzare degli stati per i tetramini, ovvero differenziare un tetramino in movimento da uno caduto quindi fermo, per poi eseguire i controlli solo su questi ultimi. Tutto ciò risultava molto macchinoso, si è giunti ad una idea molto più semplice: controllare la presenza di blocchi nella zona di spawn, non appena un blocco è caduto e quindi fermo, Il controllo viene effettuato all' interno di canDrawAtPosition().

Ottimizzazione del gioco
in GameLoop è creata una variabile fps che setta il timing di gioco, viene usata all' interno di un wait, che rallenta il ripetersi dei cicli del Gameloop, con il valore attuale (1000/30) si ottiene un aggiornamento di gioco a circa 30 volte al secondo. Originariamente era impostato a 1000/60, non necessario per il nostro tipo di gioco.

Pacchetto NET

Nel pacchetto NET si trovano le classi che riguardano la parte di connessione client-server.
In questo gioco multiplayer è stato utilizzato un approccio di tipo server non-autoritario: il server si occupa solamente di inizializzare la partita (inviando l'id dei vari player ai rispettivi client), raccogliere le informazioni inviate dai client, elaborarle (come per esempio per le trash lines) e ridistribuirle a questi ultimi.

LATO SERVER

Classe Server:
Contiene il main() e quello che fa è semplicemente creare un thread ServerRoutine e farlo partire.

Classe ServerRoutine:
Il thread della classe ServerRoutine una volta fatto partire gestisce l'inizializzazione del gioco, il suo svolgimento e la sua fine.

Inizializzazione:
1- Inizia un loop.
2- Viene creato e fatto partire un thread SocketConnection (prima che parta il conto alla rovescia).
3- Si aggiorna la lista di ClientHandler con la lista aggiornata che si trova all'interno dell'istanza della classe SocketConnection appena creata.
4- Si controlla che la dimensione di questa lista sia > 1 e conseguentemente si fa partire il conto alla rovescia.
5- Una volta che il conto alla rovescia è arrivato a 0 vengono creati e fatti partire tutti i threads degli oggetti ClientHandler all'interno della lista, e si esce dal loop.
6- Dopo che ad ogni ClientHandler è arrivato il giocatore dal corrispettivo client, viene creata una lista di Player inserendoci i giocatori presenti in ogni ClientHandler.
7- Viene creato e fatto partire il thread CmdServer.

Svolgimento del gioco:
Finché qualche giocatore non ha vinto la partita viene costantemente scorsa (in un loop) la lista dei ClientHandler da cui si ricavano i Player aggiornati che hanno ricevuto i singoli ClientHandler e li si inserisce nella lista dei Player (della classe ServerRoutine).
Viene fatto un controllo su un possibile comando impartito in input dalla console del server.
Viene controllato se qualche giocatore ha inviato delle righe spazzatura. In caso siano state inviate, viene mandato un messaggio al giocatore bersaglio contente il numero di righe spazzatura che quest'ultimo dovrà elaborare.
Infine, si manda la lista dei Player a tutti i client.

Fine del gioco:
Si continua ad inviare ai Client le liste dei Player e si continuano a ricevere i Player dai ClientHandler finché non viene digitato (da console) il comando "close": il quale farà terminare l'esecuzione di tutti i thread attivi (sia lato client che lato server).

Classe SocketConnection:
Il thread della classe **SocketConnection** stabilisce le connessioni con i client: nel caso in cui vadano a buon fine vengono creati i ClientHandler e aggiunti alla lista.

Classe ClientHandler:
Il thread di questa classe ha come unico scopo di inviare (in fase di inizializzazione) gli id dei giocatori ai Client e quello di ricevere messaggi dai Client (tramite il MessageBuffer) per tutta la durata della partita.

Classe CmdServer:
Il thread di questa classe resta sempre attivo ad ascoltare eventuali comandi impartiti da console.

Classe ServerUtil:
È una classe di utilità dove sono contenuti semafori e lock.

Classe CountDown:
Il thread di questa classe contiene il conto alla rovescia.

LATO CLIENT

Classe Client:
Questa classe si occupa di ricevere ed inviare le informazioni al server (tramite il MessageBuffer) per l'inizializzazione lo svolgimento e la fine della partita.

Inizializzazione:
1- Riceve l'id del player dal server.
2- Crea l'oggetto Player (con l'id ricevuto) e lo invia al server.
3- Riceve la lista di Player dal server.
4- Setta il target (delle righe spazzatura) di default.
5- Crea l'oggetto BoardManager.
6- Crea il thread GameLoop e lo fa partire.

Svolgimento del gioco:
Ascolta in loop i messaggi in arrivo dal server, controlla che il bersaglio delle righe spazzatura del Player sia ancora valido (che non si riferisca ad un giocatore già eliminato) e nel caso non lo sia setta un nuovo bersaglio.
Controlla se sono stati ricevuti comandi dal server ("pausa", "resume", "close" ecc.) e li gestisce.

Fine del gioco:
Controlla se il Player (di quel client) è il vincitore della partita e, nel caso, setta la board di quel Player con la board di default del vincitore.

CLASSI COMUNI SIA AL LATO SERVER CHE AL LATO CLIENT

Classe GameWinner:
È anche questa una classe di utilità (utilizzata sia dal lato server che dal lato client) che contiene un metodo che verifica le condizioni di vittoria della partita: totale giocatori - numero giocatori che hanno perso = 1.

Classe BufferMessage:
Il thread di questa classe rimane in ascolto di messaggi dal BufferedReader.
Ogni volta che viene ricevuto un messaggio, viene inserito all'interno di una coda FIFO.
Tramite la classe ReceiveData verrà eseguita la poll dalla coda.

Classe ReceiveData:
Questa classe permette di estrarre messaggi dalla coda del BufferMessage, identificarli e memorizzarli all'interno della variabile (di classe) corretta.

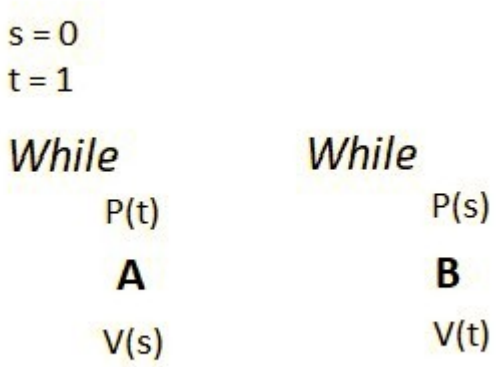
Classe SendData:
Questa classe permette tramite i suoi metodi di trasformare un qualunque dato da inviare (es. Un oggetto player, un intero, una lista di Player ecc.) in una stringa con un identificatore e inviarla.

Id	Contenuto del messaggio
1	Comando server
2	Player id
3	Target + Sender + numero di righe spazzatura da inviare
4	Oggetto Player
5	ArrayList di Player
6	Numero di righe spazzatura da elaborare

Problemi riscontrati nella scrittura delle classi del pacchetto NET

Uno dei primi problemi che abbiamo incontrato è stato quello di passare da un'architettura single-client ad una multi-client. Il problema è stato risolto creando più istanze di connessioni tramite socket e memorizzandole in una lista per potersi poi riferire ad una specifica istanza.

Un altro problema è stata la mancata sincronizzazione tra i thread di ServerRoutine e SocketConnection: ciò permetteva al thread ServerRoutine di eseguire il suo loop costantemente senza permettere al thread SocketConnection di eseguire e quindi di accettare le connessioni con i client. Abbiamo risolto ciò con l'implementazione di due semafori che hanno permesso un'alternanza perfetta dell'esecuzione dei due thread.



Abbiamo riscontrato un problema con il metodo accept() di ServerSocket, che risultava bloccante, perché dopo aver accettato due connessioni con due client mandava il programma in starvation poiché la ServerSocket rimaneva in ascolto di una terza connessione (da parte di un terzo client) all'infinito, non permettendo il rilascio del lock del semaforo e non permettendo conseguentemente al thread ServerRoutine di continuare l'esecuzione.

Il problema è stato risolto impostando un timer all'accettazione della serverSocket in modo tale da non rendere bloccante tale metodo.

In fase di test delle righe spazzatura, il risultato ottenuto riportava diversi problemi nella corretta elaborazione delle righe. Poteva accadere che le righe venissero aggiunte ad ogni caduta del pezzo, senza resettarsi; oppure risultati più casuali come di mancato aggiornamento della riga. Il problema era attribuito un macchinoso sistema di "conservazione" del dato delle righe: esso era conservato all' interno della lista di Player e doveva aggiornare l'oggetto player utilizzato nel game loop. Il problema è stato risolto generando un messaggio dedicato, direttamente all' elaborazione delle righe spazzatura.

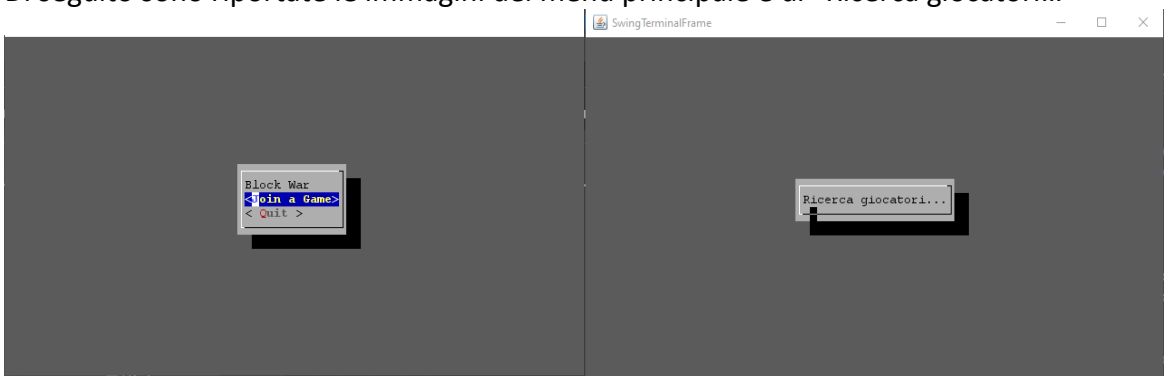
Pacchetto GUI

Qui abbiamo le classi che implementano la rappresentazione grafica.

Screen_Start:

È la classe che avvia il gioco. All'inizio abbiamo un menu principale che permette di connettersi al gioco cliccando sul tasto "Join a Game" oppure di uscire da gioco premendo il tasto "Quit". Quando ci si connette, dopo aver premuto il tasto "Join a Game" compare la schermata "Ricerca giocatori…" in quanto, i giocatori devono essere minimo due. Una volta che tutti i giocatori che volevano partecipare si sono connessi, viene lanciato il gioco. Compare quindi la schermata disegnata da ScreenDrawer.

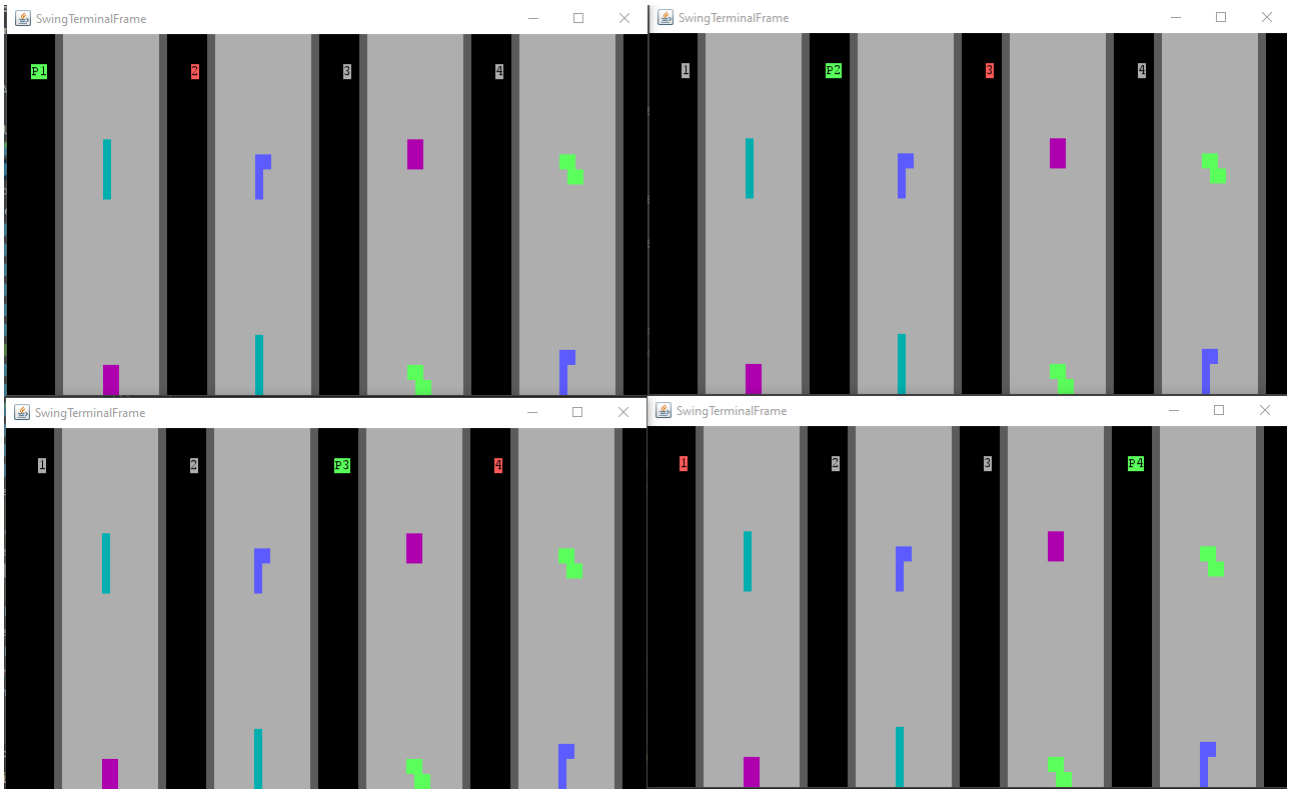
Di seguito sono riportate le immagini del menu principale e di "Ricerca giocatori…"



ScreenDrawer

La classe disegna lo schermo utilizzando la libreria grafica Lanterna, inserendo informazioni come ad esempio indicazione del proprio campo di gioco e target degli avversari.

Riportiamo l'immagine nel caso in cui ci sono quattro giocatori connessi.



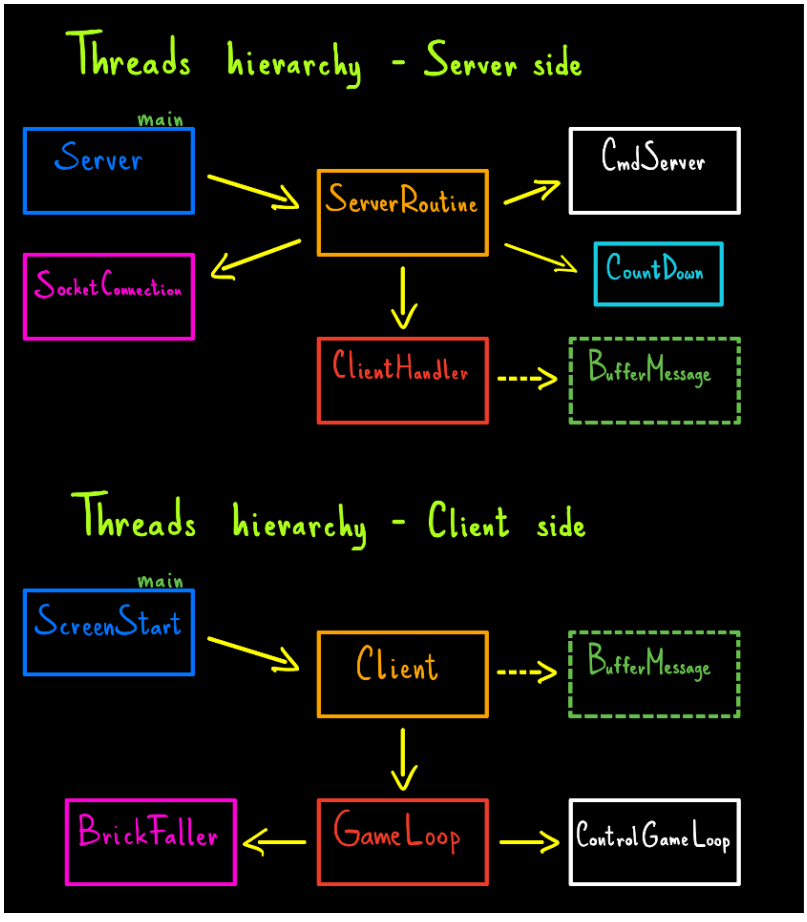
Sullo schermo di gioco sono visibili tanti campi quanti giocatori connessi.

Il campo del giocatore locale corrente è indicato dalla lettera P affiancata dal numero del giocatore, in un riquadro verde. Tutti gli altri giocatori hanno il proprio ID, visibile in un riquadro grigio a fianco del rispettivo campo di gioco.
Il target per le trashlines impostato dal giocatore corrente è indicato da un riquadro rosso che sostituisce quello grigio dell'avversario.
In questa classe vengono definiti anche i colori dei polimini.

Lista dei comandi di gioco definiti in GameLoop:

- Tasti freccia: per spostare lateralmente (a destra oppure a sinistra), oppure in basso il tetramino.
- X: rotazione oraria del tetramino.
- Z: rotazione antioraria del tetramino.
- Spacebar: caduta rapida del tetramino.
- Tasti numerici da 1 a 4: per impostare il target per le righe spazzatura.
- Escape: chiusura del file per tutti i giocatori.

Gerarchia dei thread



Questo grafico mostra tutti i thread presenti nel progetto (esclusi Server e ScreenStart che fanno solo partire thread ma non lo sono). Le frecce mostrano quale classe fa lo start del/dei thread di quale/i altre classi (es. In ServerRoutine sono presenti gli start dei thread delle classi ClientHandler, SocketConnetion, CmdServer e CountDown).

STRUMENTI UTILIZZATI PER L'ORGANIZZAZIONE:

L'IDE utilizzato per il progetto è Intellij IDEA. Si è sviluppato prevalentemente in ambiente Windows.

Pacchetti usati: Lanterna e Gson per i files Json.

La maggior parte delle riunioni virtuali sono state svolte su Discord poiché consente lo streaming simultaneo di schermi multipli, in alternativa sono stati utilizzati Microsoft Teams o Google Meet. Per la condivisone dei files di lavoro è stata usata la piattaforma teams. Per la condivisione del codice di progetto sono stati usati Teams, Codeshare e GitLab. È stato inoltre usato un gruppo Whatsapp per tenerci quotidianamente aggiornati.

COMPILAZIONE

Per eseguire il programma all' interno dell'IDE è possibile eseguire il run() del main per il Server e per il Client. È necessario modificare la configurazione di run, per abilitare istanze multiple del client e per inserire gli argomenti di avvio del client e server.

In alternativa è possibile produrre un Jar per il client ed uno per il server utilizzando il plugin Maven, eseguendo il comando package, in Lifecycle. In questo modo si conservano le dipendenze.

Per avviare il file jar da linea di comando del Server si dà il comando:

java -jar Server.jar [PORTA]

Per avviare il file jar da linea di comando del Client si dà il comando:

javaw -jar Client.jar [INDIRIZZO_IP] [PORTA]

Alla comparsa della finestra con <join game> premere sul bottone usando [enter]

DIVISIONE DEL LAVORO

Inizialmente pensavamo di dividerci il lavoro in sottogruppi, tuttavia per affrontare meglio le difficoltà emerse dal progetto, si è proceduti prevalentemente assieme, tranne per l'implementazione di piccole task svolte in autonomia.