PROGETTO FINALE SOPR

A.A. 2024-2025

Versione aggiornata al 25/11/2024

Docenti:

Salvatore Carta

Alessandro Sebastian Podda

Tutor:

Riccardo Balia

Per chiarimenti su quanto riportato nel seguito e, in generale, per ogni tipo di comunicazione correlata al corso, inviare una mail esclusivamente al seguente indirizzo generale (le mail indirizzate a uno solo dei docenti/tutor non riceveranno risposta):

sopr.unica@gmail.com

1. Modalità di consegna e di esame

(estratto dal Regolamento)

FORMAZIONE DEI GRUPPI: lo svolgimento del progetto prevede l'organizzazione in gruppi di **2 persone** (solo in casi *eccezionali*, un gruppo potrà essere formato da un singolo studente). Un solo componente del gruppo dovrà farsi carico di inviare un messaggio di posta elettronica all'indirizzo *sopr.unica@gmail.com*, includendo tra i destinatari anche l'altro studente che compone il gruppo, indicando l'oggetto "**[SOPR] Gruppo progetto**" e specificando nel corpo del messaggio la **matricola**, il **nome** e il **cognome** di tutti i componenti del gruppo. È fortemente consigliato evitare lo svolgimento del progetto da parte dello studente singolo. *NB: l'impegno richiesto per lo svolgimento complessivo del progetto sarà identico sia per i gruppi da due persone che per quelli costituiti da un solo studente.*

CONSEGNA DEL PROGETTO: gli allievi dovranno consegnare il progetto interamente all'interno di un file in formato **ZIP** denominato "**XXXXX**_**YYYYY**_progetto.zip", in cui i placeholder **XXXXX** e **YYYYY** andranno sostituiti con le matricole dei componenti del gruppo (usare "**XXXXX**_progetto.zip" nel caso eccezionale di progetto svolto da uno studente). All'interno del file ZIP dovranno essere presenti due sottodirectory "**versione_processi**" e "**versione_thread**", all'interno di ciascuna delle quali saranno contenuti i rispettivi file di progetto (sorgenti, makefile, ecc.) per la rispettiva versione implementata tramite processi, in un caso, e thread, nell'altro.

Il file di consegna dovrà essere inviato da un solo membro del gruppo, con l'eventuale altro studente in copia carbone (CC), tramite email all'indirizzo *sopr.unica@gmail.com*, specificando l'oggetto "**[SOPR] Consegna progetto**". Sarà opportuno specificare nel corpo del messaggio le eventuali modifiche apportate alla struttura indicata nelle specifiche del progetto, oltre ad eventuali altre comunicazioni di rilievo, **senza dimenticare** di riportare in calce alla email la **matricola**, il **nome** e il **cognome** di tutti componenti del gruppo.

Importante: le email di consegna che non rispettano le prescrizioni sopra indicate (oggetto corretto, corpo del messaggio, dati dei componenti, mancanza di allegato corretto) verranno ignorate e sarà necessario sostenere la discussione del progetto durante il successivo appello disponibile.

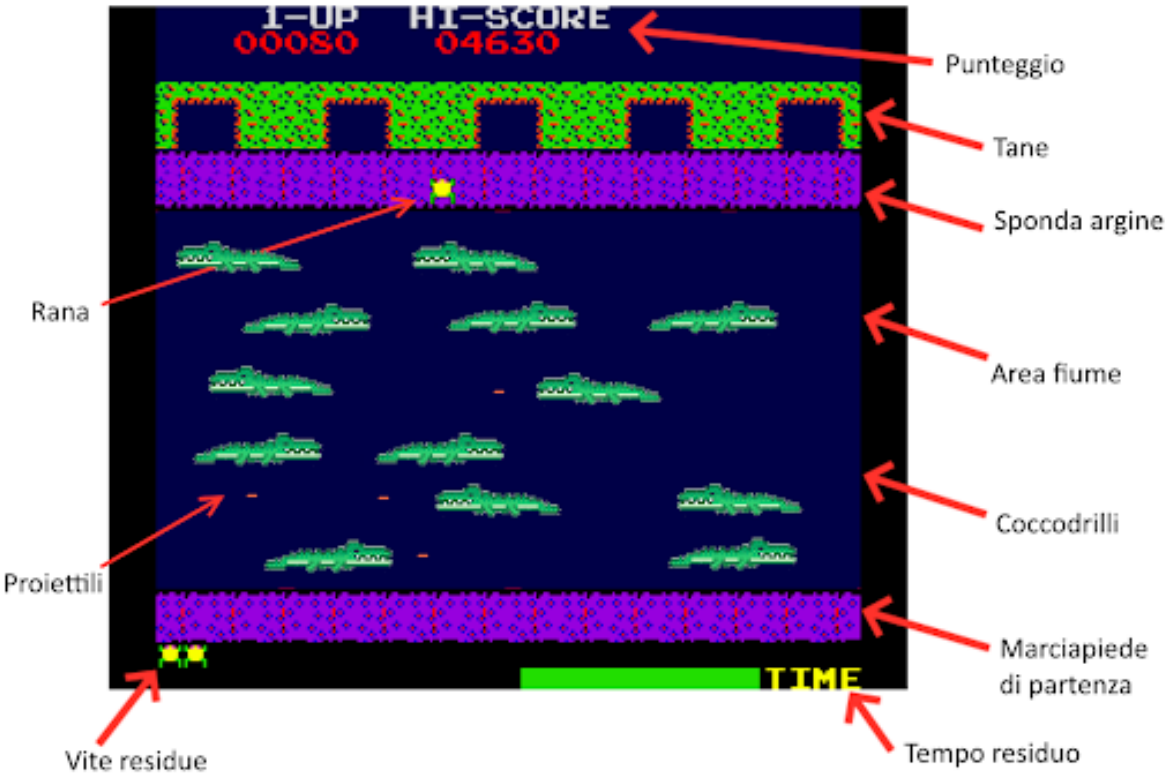
REQUISITI E AUTOVALUTAZIONE: gli studenti dovranno farsi carico di autovalutare il progetto prima della sua consegna e discussione. Non sarà infatti accettata la correzione di progetti malfunzionanti e/o che non completano la fase di compilazione. Nello specifico, il progetto dovrà risultare *COMPILABILE*, *ESEGUIBILE* e correttamente *FUNZIONANTE* sulla macchina virtuale utilizzata durante il corso (**Ubuntu 22.04 versione 64 bit**). Non verranno in alcun modo considerate le consegne di progetti funzionanti unicamente in ambienti differenti da quello suddetto, né prese in considerazione giustificazioni per problemi causati dall'utilizzo di differenti ambienti di sviluppo / librerie.

In caso di valutazione insufficiente (il progetto non funziona e/o lo studente non è in grado di comprendere e commentare opportunamente il proprio codice e/o altre casistiche non facilmente preventivabili) la valutazione verrà rimandata all'appello successivo. In caso di gravi inadempienze è previsto che sia possibile saltare un appello e/o il conseguimento di un voto più basso di quello previsto di default, che NON potrà essere rifiutato. Il superamento del progetto è **condizione strettamente necessaria** per il superamento dell'esame; inoltre è necessario raggiungere la sufficienza in tutti e due i moduli del programma (Teoria e Laboratorio).

2. Tema del progetto: *Frogger Resurrection*

Codificare, compilare ed eseguire su Linux un programma che implementi una funzionalità grafica ispirata ad un videogame vintage: Frogger. Si sottolinea che il progetto è solo **ispirato** al gioco originale, ma si diversifica da esso per numerosi aspetti: per tali ragioni le specifiche di riferimento per la valutazione del progetto sono esclusivamente quelle del presente documento.
L'obiettivo del gioco consiste nel condurre una rana verso la propria tana. Per farlo, il giocatore deve riuscire a far attraversare alla rana un fiume pieno di insidie, evitando di far scadere il tempo a disposizione.

La struttura di massima della mappa di gioco, graficamente solo indicativa, è definita come segue (la posizione di punteggio, tempo e vite può essere variata in base alle esigenze):



3. Specifiche funzionali

Per lo sviluppo del progetto, sarà necessario utilizzare gli strumenti per l'elaborazione parallela, la comunicazione e la sincronizzazione illustrati durante il corso. Si suggerisce di far riferimento all'esercitazione "*Guardie e ladri*" per avere un'idea di una possibile infrastruttura di base del programma.

3.1. Requisiti di carattere generale

- Nell'ambito del gioco da realizzare, la rana dispone di un numero prefissato di vite, definito a tempo di compilazione, che si decrementa di una unità ogni volta che il giocatore perde una *manche*. Il giocatore vince il gioco non appena la rana ha chiuso tutte le tane;
- *Frogger Resurrection* è un gioco a tempo: è pertanto necessario mostrare nella parte inferiore dello schermo (o superiore, a discrezione dell'allievo) un indicatore rappresentante il tempo di gioco (es: contatore, barra, o altro), che si decrementa allo scorrere del tempo della manche.
- L'area di gioco deve visualizzare un punteggio di gioco che si incrementa in base ai risultati raggiunti e al tempo impiegato, secondo una formula definita a piacere dallo studente;
- Come accennato, il gioco è suddiviso in *manche*. All'inizio di ciascuna, la rana parte dal marciapiede e il tempo inizia a scorrere. Ogni manche si conclude con uno dei seguenti eventi: 1) la rana muore (ad esempio è annegata nel fiume); 2) il tempo per la manche corrente è terminato; 3) la rana ha chiuso una tana. Nei primi due casi, il giocatore perde una vita. In ogni caso, ogni volta che la manche si conclude, il tempo viene resettato, il punteggio aggiornato e la rana riparte dalla posizione iniziale;
- È necessario suddividere l'area di gioco nelle seguenti sezioni, orientativamente secondo la struttura descritta in figura precedente, consistenti in: tane, sponda d'erba che costituisce l'argine superiore del fiume, fiume, marciapiede dell'argine di partenza (che costituirà il bordo inferiore di gioco nonché il punto di partenza della rana);
- Nessun oggetto dinamico in gioco (es: rana, coccodrilli, ecc.) dovrà potersi muovere/essere visualizzato al di fuori del perimetro della mappa di gioco;
- È necessario implementare due versioni differenti del gioco, una "versione_processi" utilizzando processi e pipe, e una "versione_thread"

utilizzando thread e shared memory, per l'implementazione degli elementi concorrenti del gioco. In particolare, ogni oggetto dinamico **andrà implementato come singolo thread/processo separato** (la rana, i **singoli** coccodrilli, eventualmente il gestore della matrice grafica). L'unica eccezione consentita, ma non obbligatoria, riguarda i proiettili/granate multipli di ciascun partecipante "attivo" (coccodrilli o rana), che potranno essere gestiti con un unico thread/processo per singolo partecipante (es: non è comunque consentito che il processo che gestisce le granate della rana gestisca anche i proiettili di uno o più coccodrilli).

3.2. Requisiti dell'oggetto "rana"

- È necessario visualizzare sullo schermo un oggetto rana, comandato dal giocatore principale, e in grado di muoversi in tutte e quattro le direzioni principali (↑, ↓, ←, →);
- La rana è rappresentata da una forma scelta dall'utente che occupi un'area minima di n. 2 caratteri, rivolta costantemente verso l'alto (NB: l'area non dev'essere necessariamente quadrata, potete disegnare oggetti che non hanno stessa altezza e stessa larghezza);
- Il movimento della rana è definito dai tasti direzionali della tastiera. In ogni caso la rana deve fare un movimento (salto) massimo pari alla sua dimensione. Ad esempio: una rana rappresentata con un'area di 3x2 caratteri (3 colonne x 2 righe) si muove *al massimo* di 3 caratteri (oppure 1 o 2) quando si sposta orizzontalmente, e al massimo di 2 (oppure 1) quando si sposta verticalmente. La dimensione del salto stabilita per le due direzioni dev'essere comunque costante durante il gioco;
- La rana parte dal limite inferiore dello schermo ("marciapiede dell'argine inferiore", vedi figura) con l'obiettivo di arrivare al perimetro superiore, in cui sono presenti le tane.
- Premendo la barra spaziatrice, la rana genera e rilascia due oggetti granata, rappresentate ognuna da un singolo carattere, una si muoverà in modo indipendente verso destra e una in modo indipendente verso sinistra partendo dalla rana. Utilizzare thread o processi (a seconda della versione) che implementino la gestione dell'oggetto granata. Per ulteriori dettagli consultare le specifiche su *Collisioni*.

3.3. Requisiti del marciapiede dell’argine

- Il marciapiede dell’argine è rappresentato da un'area rettangolare con altezza sufficiente per ospitare la rana e larghezza pari all’area di gioco orizzontale (come definita tramite libreria *ncurses*);
- Il marciapiede dell’argine è un elemento statico. Lo studente potrà scegliere a piacere come rappresentarlo graficamente: esso dovrà essere continuamente stampato a video per l’intera durata della partita.

3.4. Requisiti del fiume

- Il fiume è composto da un numero di flussi (equivalenti a “corsie” orizzontali) non inferiore a 8;
- I flussi hanno una larghezza (coordinata x dello schermo) pari all’area di gioco e una altezza minima (coordinata y dello schermo) sufficiente a contenere la rana;
- Ogni flusso viene percorso da **coccodrilli a pelo d’acqua**, i quali avranno un’altezza (coordinata y) pari all’altezza della rana, una larghezza minima (coordinata x) pari al doppio della larghezza della rana e una larghezza massima pari a 3 volte la larghezza della rana;
- In maniera casuale prestabilita all’avvio di ogni nuova manche, il primo flusso del fiume (quello più in basso, adiacente al marciapiede) sarà percorso orizzontalmente da: a) sinistra verso destra oppure da b) destra verso sinistra Ogni flusso del fiume successivo al primo dovrà avere direzionamento opposto a quello del flusso immediatamente precedente. ;
- Quando l’estremo di un coccodrillo arriva al limite del campo di gioco, il coccodrillo sparisce gradualmente dall’area di gioco, e il suo processo viene terminato (o eventualmente sfruttato per generare un nuovo coccodrillo in un nuovo flusso del fiume);
- I coccodrilli di uno stesso flusso procedono tutti alla stessa velocità e si muovono tutti nella stessa direzione, pertanto NON devono/possono collidere tra loro. La velocità di ogni flusso è definita staticamente a inizio partita: è importante però che flussi diversi abbiano velocità differenti, ma compatibili, per evitare situazioni in cui il gioco diventi impraticabile o impossibile da completare;

- La generazione degli oggetti coccodrillo avviene utilizzando degli intervalli di tempo casuale, in modo che ogni volta che ricomincia la manche (ad esempio perché il giocatore ha perso una vita) la scena di gioco sia sempre originale;
- I coccodrilli si comportano come delle “zattere” in acqua, sulle quali la rana può saltare e muoversi lateralmente, per poter attraversare il fiume verso l’argine opposto e raggiungere la propria tana. In modo casuale, improvviso e pertanto impredicibile, i coccodrilli sparano dei proiettili dalla bocca. I proiettili generati si muoveranno orizzontalmente partendo dal coccodrillo e seguendo la direzione del flusso del fiume. La velocità del movimento dei proiettili è costante.
  Prima di sparare, i coccodrilli possono fornire un feedback grafico che preannunci l’imminente sparo.
- Se la rana è sopra un coccodrillo nel mentre che questo esce dall’area di gioco, la rana cadrà in acqua;
- Se la rana cade (o salta) in acqua, perde una vita e la manche.

3.5. Requisiti sponda d’erba

- La sponda dell’argine di arrivo (quello presente al margine superiore del fiume) è costituita da un prato d’erba, rappresentato da un'area rettangolare spaziosa a sufficienza, di larghezza pari all’area di gioco orizzontale (assegnata tramite libreria *ncurses*) e altezza almeno pari all’altezza della rana;
- Tale sponda d’erba è statica, potrà essere rappresentata graficamente come si preferisce e dovrà essere continuamente stampata a video per l’intera durata del gioco;

3.6. Requisiti delle tane

- Le tane sono mostrate nella parte superiore della schermata di gioco. Le tane dovranno essere n. 5 e dovranno essere rappresentate da sezioni rettangolari, la cui dimensione sarà tale da consentire alla rana di entrare al loro interno. Le tane devono essere distribuite orizzontalmente in proporzione alla larghezza del campo di gioco;

- Ogni volta che la rana raggiunge una tana, quest'ultima verrà chiusa: se ci sono ancora tane aperte, comincerà una nuova manche, altrimenti il giocatore avrà vinto la partita (o il livello, se sono previsti più livelli, vedi specifiche opzionali).

3.7. Gestione delle collisioni

- Ogni qualvolta la rana cade in acqua, essa perde una vita e la manche;
- Ogni qualvolta la rana viene colpita da un proiettile sparato da un coccodrillo, essa perde una vita e la manche;
- Ogni qualvolta una granata *sparata dalla rana* collide con il proiettile sparato da un coccodrillo, la granata neutralizza il proiettile del coccodrillo, entrambi gli oggetti (granata e proiettile) vengono distrutti;
- Ogni qualvolta una granata *sparata dalla rana* collide con un coccodrillo l'oggetto granata attraversa il coccodrillo passandoci sopra e continua il suo percorso;
- Se un qualsiasi proiettile (o granata) raggiunge il bordo dell'area di gioco senza collisioni, il suo oggetto in memoria dev'essere distrutto;
- Se la rana entra all'interno di una tana aperta, vince la manche e viene riportata all'inizio del percorso;
- Se la rana prova ad entrare in una tana già raggiunta (o in una qualsiasi porzione della parte superiore della mappa che non sia una tana disponibile) perde una vita e la manche.

3.8. Fine della partita

- Il gioco termina in caso di vittoria o sconfitta del giocatore;
- Il giocatore vince quando la rana riesce a raggiungere tutte e 5 le tane: in questo caso tutti i thread o processi vengono terminati e viene visualizzato a schermo un messaggio che comunica la vittoria e lo score ottenuto;
- Il giocatore perde quando si esauriscono le vite della rana: in questo caso tutti i thread o processi vengono terminati e viene visualizzato a schermo un messaggio che comunica la sconfitta e lo score del giocatore.
  In questa schermata verrà chiesto al giocatore se intende giocare una nuova partita, in caso di risposta positiva il gioco inizia da capo.

3.9 Specifiche opzionali

Opzionalmente, saranno estremamente apprezzate versioni raffinate delle specifiche, che potranno concorrere all'ottenimento di massimo **2** punti bonus per "progetti eccellenti".

Per l'ottenimento dei punti bonus è necessario svolgere una o entrambe le due seguenti direttive:

- **Implementare versione socket:** l'input della rana, in entrambe le versioni thread e processi, è acquisito da un processo esterno che comunica le coordinate al processo del gioco mediante un socket locale **(max 1 punto)**;
- **Implementare almeno 3 specifiche opzionali** tra quelle di seguito riportate **(max 1 punto)**:
  - Diverse modalità di gioco: facile, normale, difficile;
  - Livelli multipli di gioco, a difficoltà crescente (ad esempio, oggetti più veloci, maggior numero di proiettili, etc.);
  - Nuovi ostacoli all'interno del fiume o dei prati;
  - Suoni e/o musiche di gioco;
  - Gestione di un menu di gioco e/o delle pause;
  - Elevata cura della grafica di gioco (colori, sprite, animazioni), anche eventualmente con uso di librerie differenti da *ncurses*;
  - Ulteriori specifiche non menzionate (*siate creativi*) di complessità non inferiore a quelle precedenti.

NB. Le specifiche del presente capitolo sono opzionali e in fase di valutazione non sostituiscono quelle obbligatorie: lo studente si concentri nello sviluppo delle specifiche obbligatorie e solo dopo, se vuole, può sviluppare alcune specifiche opzionali. Il punteggio base del progetto può essere raggiunto senza sviluppare alcuna specifica opzionale (l'implementazione di feature opzionali è necessario comunque per l'ottenimento dei punti bonus).

4. Architettura generale (IMPORTANTE! ⚠️)

È obbligatorio che l'architettura del programma sia basata su **un singolo task** (thread o processo a seconda della versione) per ciascun oggetto che si muove sullo schermo (rana, coccodrilli; per i proiettili si veda quanto già scritto in 3.1). Ciascuno di questi task si deve occupare di generare e trasmettere le proprie coordinate a un task addizionale che si occupa di: (1) disegnare sullo schermo gli oggetti; (2) verificare le collisioni o l'uscita di un oggetto dallo schermo e comportarsi di conseguenza.

Il cuore del programma è incentrato sulla comunicazione delle coordinate fra *N* produttori (gli oggetti in movimento) e un consumatore (gestore di disegno e movimento). Verificare sempre che siano evitate le scritture su buffer di comunicazione pieno e le letture da buffer di comunicazione vuoto, per ciascuna versione del progetto.

Un'architettura simile che si può usare come base di partenza e dalla quale si può copiare la gestione della grafica è quella dall'esercizio "*Guardie e ladri*". Nell'esercizio ciascun oggetto ha un task associato che genera la sua posizione in maniera random, o secondo una traiettoria, o seguendo i tasti. Vi sono poi task che si occupano di disegnare sullo schermo gli oggetti che si spostano (prima si cancella la vecchia posizione, poi si disegna la nuova). Vengono inoltre monitorate le collisioni. È opportuno scegliere correttamente la durata delle pause negli spostamenti degli oggetti, in modo da avere una visualizzazione comprensibile.

Versione processi: è obbligatorio utilizzare i processi per la gestione della elaborazione concorrente e le pipes per la comunicazione (simile all'esercizio guardie e ladri). È richiesta una singola pipe per le comunicazioni dai processi produttore (multiple entità di gioco) al singolo processo consumatore (gestore di grafica e collisioni). È ammessa l'aggiunta di ulteriori pipes multiple opzionali solo per le comunicazioni in senso inverso (ovvero, dal gestore di grafica e collisioni alle

singole entità di gioco). Saranno invece rifiutate soluzioni che utilizzano pipe multiple per la comunicazione dai produttori al consumatore, così come quelle che implementano la comunicazione con altre soluzioni (e.g., segnali).

Versione thread: è obbligatorio utilizzare: i thread per la gestione della elaborazione concorrente; un **buffer produttore-consumatore** in memoria condivisa per la comunicazione; gli strumenti di sincronizzazione delle libreria pthread (usare come riferimento gli esercizi produttore-consumatore visti a lezione). Nello specifico, il buffer produttore-consumatore condiviso deve essere: limitato (numero massimo di slot allocabili); circolare; correttamente protetto da scritture congiunte sullo stesso slot o che vadano a sovrascrivere i contenuti non ancora consumati. Tale protezione deve essere ottenuta cercando di utilizzare il numero minore possibile di semafori e mutex. Il buffer produttore-consumatore, per definizione, è implementabile come solo array o lista. Saranno pertanto rifiutate soluzioni che implementano una comunicazione tra thread ricorrendo a letture e scritture su strutture condivise diverse dal buffer produttore-consumatore (e.g., sui campi delle strutture passate come thread argument durante l'avvio degli stessi thread).