Corso di Laboratorio di Programmazione
Progetto finale – Monopoly

Vi viene richiesto di sviluppare un programma in C++ che implementi una versione semplificata del gioco Monopoly. Chi non ha familiarità con questo gioco può trovare la descrizione al link seguente:

https://it.wikipedia.org/wiki/Monopoly
https://www.wikihow.it/Giocare-a-Monopoly (regole ufficiali)

1.Regole per la versione semplificata

Per semplificare il design e l'implementazione vi forniamo di seguito delle regole semplificate a cui il vostro programma **deve** attenersi.

1.1 Tabellone

- Il tabellone è composto da 8 x 8 caselle
- Le caselle sono di due tipi: 4 angolari e 24 laterali
- Una delle caselle angolari corrisponde alla casella di partenza, le altre tre sono vuote
- Le caselle laterali possono essere di tre categorie:
  - economica
  - standard
  - lusso
- Il tabellone sarà quindi formato da:
  - 4 caselle angolari (di cui una è una casella di partenza)
  - 8 caselle laterali di categoria economica
  - 10 caselle laterali di categoria standard
  - 6 caselle laterali di categoria lusso
- Ogni terreno può essere migliorato costruendo una casa, oppure un albergo (per semplicità, si suppone che su ogni proprietà esista al massimo una casa, e il passaggio successivo sia l'albergo).
- A parte le caselle angolari, le restanti sono posizionate in modo casuale all'inizio di ogni partita

1.2 Giocatori

- Esistono due tipi di giocatori:
  - giocatore umano
  - computer
- La partita inizia **sempre** con 4 giocatori, di cui solo 1 **può** essere umano

1.3 Tiro di dadi

- Un tiro di dadi corrisponde alla somma di 2 numeri casuali compresi fra 1 e 6

1.4 Meccaniche di gioco

- Ogni giocatore possiede un budget iniziale di 100 fiorini
- All'inizio di ogni partita, ogni giocatore tira i dadi. Il giocatore con il numero più alto inizierà la partita e poi si seguirà un ordine decrescente (in caso di parità di punteggio fra due o più giocatori, i giocatori interessati dovranni ritirare il dado)
- Una volta determinato l'ordine di gioco, il primo giocatore tirerà di nuovo i dadi e muoverà la sua pedina di un numero di caselle pari al numero uscito
- A questo punto possono verificarsi le seguenti condizioni:

	Giocatore Umano	**Computer**
Casella angolare	Non succede nulla, il turno del giocatore finisce.	Non succede nulla, il turno del giocatore finisce.
Casella laterale non appartenente a nessuno dei giocatori	Può decidere se comprare il terreno (la casella), se possiede abbastanza soldi. Altrimenti, finisce il turno.	Può comprare il terreno (la casella) con una probabilità del 25%, se possiede abbastanza soldi. Altrimenti, finisce il turno
Casella laterale di proprietà del giocatore, senza casa/albergo	Può decidere se comprare una casa (il passaggio diretto da terreno semplice ad albergo non è possibile), se possiede abbastanza soldi. Altrimenti, finisce il turno.	Può comprare una casa (il passaggio diretto da terreno semplice ad albergo non è possibile) con una probabilità del 25%, se possiede abbastanza soldi.
Casella laterale con casa di proprietà	Può decidere di migliorare la propria casa in albergo, se possiede abbastanza soldi. Altrimenti, finisce il turno.	Può migliorare la propria casa in albergo con una probabilità del 25%, se possiede abbastanza soldi.
Casella laterale con casa/albergo di proprietà di un altro giocatore	Deve pagare il pernottamento nella struttura.	Deve pagare il pernottamento nella struttura.

- Se con il tiro di dado, un giocatore passa per la casella di partenza, ritira 20 fiorini (cioè il suo budget viene incrementato di 20)
- I prezzi (in fiorini) per l'acquisto di una casa, il miglioramento in albergo di una casa esistente e il pernottamento nelle due diverse strutture e nelle diverse categorie di caselle, è riportato di seguito:

	economica	**standard**	**lusso**
Acquisto terreno	6	10	20
Acquisto casa	3	5	10
Miglioramento ad albergo	3	5	10
Pernottamento in casa	2	4	7
Pernottamento in albergo	4	8	14

- Un giocatore viene eliminato se non ha abbastanza fiorini al momento del pagamento del pernottamento. Gli altri giocatori continuano a giocare. Le caselle di proprietà del giocatore eliminato vengono svuotate di possibili case o alberghi e i terreni tornano ad essere disponibili per la vendita.
- Vince l'ultimo giocatore che rimane in gioco.

1.5 Visualizzazione

Nel caso di partita con un giocatore umano, è possibile richiedere al programma la visualizzazione del tabellone corrente. Verranno visualizzate le caselle del tabellone, la presenza di eventuali case o alberghi e la posizione dei giocatori attualmente in gioco.

La visualizzazione avviene stampando un carattere con un eventuale suffisso per ciascuna posizione:

Casella	**Carattere**
Angolare	spazio
Partenza	P
Economica	E
Standard	S
Lusso	L

Suffissi	**Carattere**
Casa	*
Albergo	^
Giocatore	1-4

Esempio tabellone vuoto:

```
    1     2     3     4     5     6     7     8
A | | | L | | L | | E | | S | | S | | S | | |
B | L |                                       | S |
C | S |                                       | E |
D | L |                                       | S |
E | S |                                       | E |
F | E |                                       | L |
G | E |                                       | E |
H | | | S | | S | | L | | S | | E | | E | | P |
```

Esempio tabellone con giocatori, case e alberghi:

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	| |	| L*1 |	| L |	| E^4 |	| S |	| S |	| S |	| 3 |
B	| L |							| S |
C	| S |							| E |
D	| L^ |							| S |
E	| S |							| E |
F	| E |							| L |
G	| E |							| E |
H	| |	| S*2 |	| S |	| L |	| S |	| E* |	| E |	| P |

Legenda:
Giocatore 1 nella casella A2 con una casa
Giocatore 2 nella casella H2 con una casa
Giocatore 3 nella casella A8
Giocatore 4 nella casella A4 con albergo
Casella D1 con albergo
Casella H6 con casa

Nota: Sul tabellone non vengono visualizzate l'appartenenza di una casa/albergo a un determinato giocatore. Un ulteriore comando stamperà sul terminale la lista di case/alberghi che ogni giocatore possiede. Esempio:

Giocatore 1: A2, H6
Giocatore 2: A4
Giocatore 3: H2, D1
Giocatore 4:

1.6 Svolgimento del gioco

Ogni evento del gioco (ad esempio, tiro di dado, acquisto terreni/case/alberghi, pagamenti e così via) **deve** essere stampato a schermo e **deve** essere salvato su un file di log testuale usando i seguenti messaggi:

```
- Giocatore N è passato dal via e ha ritirato 20 fiorini
- Giocatore N ha tirato i dadi ottenendo un valore di X
- Giocatore N è arrivato alla casella Y
- Giocatore N ha acquistato il terreno Y
- Giocatore N ha costruito una casa sul terreno Y
- Giocatore N ha migliorato una casa in albergo sul terreno Y
- Giocatore N ha pagato Z fiorini a giocatore M per pernottamento nella casella Y
- Giocatore N ha finito il turno
- Giocatore N è stato eliminato
- Giocatore N ha vinto la partita
```

Inoltre in qualsiasi momento del proprio turno il giocatore può utilizzare il comando `show` da terminale al fine di:

- visualizzare il tabellone
- visualizzare lista terreni/case/alberghi posseduti da ogni giocatore
- visualizzare l'ammontare di fiorini posseduto da tutti i giocatori

1.7 Partite

Le partite sono di due tipi:

- partite con 4 giocatori computer;
- partite 3 giocatori computer e un giocatore umano.

Il programma deve gestirle entrambe, in base a un argomento fornito da riga di comando (letto tramite `argv` e `argc`): `computer` per i quattro giocatori gestiti dal computer, e `human` per un giocatore umano contro tre giocatori gestiti dal computer. Nel caso di partite tra computer, dovete fissare un numero massimo di turni, oltre il quale vince il giocatore con più fiorini (in caso di quantità uguale di fiorini, è permessa la vittoria ex-equo). Durante ogni partita deve essere effettuato un log su file, che elenca tutti gli eventi, in ordine. Nel caso di partite con un giocatore umano, il programma interagisce con l'utente in caso di:

- arrivo su una casella non ancora venduta (chiede all'utente se desidera comprarla);
- arrivo su una casella di proprietà senza una casa (chiede all'utente se desidera costruire una casa);
- arrivo su una casella di proprietà con una casa (chiede all'utente se desidera migliorare la casa in albergo).

In questi tre casi, la risposta dell'utente deve essere `S` (per sì) o `N` (per no); in tutti gli altri casi il programma regola automaticamente la meccanica di gioco (per esempio, procede automaticamente ai pagamenti, alle riscossioni delle somme dovute, ecc.).

2. Note allo sviluppo

Il progetto sviluppato **deve** essere gestito tramite CMake e compilabile sulla macchina virtuale Taliercio.2020. Non è necessario che sviluppiate tutto sulla macchina virtuale, ma dovete fare verifiche periodiche per essere sicuri di non aver utilizzato codice fuori standard.

2.1 Indicazioni per lo svolgimento

Il progetto deve essere suddiviso in più file sorgente. **Ogni file deve essere scritto da un solo studente**. È tuttavia possibile controllare che il codice dei propri compagni di gruppo funzioni correttamente. Un errore in un file che inficia il funzionamento del progetto potrebbe causare una penalizzazione della valutazione dell'intero gruppo. **Il nome dell'autore deve essere indicato in un commento a inizio file**. Ovviamente, è necessario e positivo che si discuta all'interno di ciascun gruppo su come realizzare il codice, ma ogni studente è responsabile della gestione del codice che deve scrivere.

Saranno valutati:

- Chiarezza e correttezza del codice;
- Corretta gestione della memoria e delle strutture dati;
- Efficienza del codice e della soluzione trovata;
- Utilizzo di strumenti appropriati.

Il codice deve essere adeguatamente commentato.

Il software deve essere basato unicamente sulla libreria standard del C++.

2.2 Plagi

Il codice verrà controllato per verificare eventuali plagi tra gruppi appartenenti allo stesso canale e in canali diversi. Verrà effettuato anche un controllo rispetto a codice disponibile online.

2.3 Consegna

Il compito deve essere consegnato su Moodle (consegna di gruppo come per la prova intermedia). Ricordatevi che tutti i membri del gruppo devono approvare il progetto perché la consegna risulti effettuata. Il progetto dovrà essere compresso in un archivio che include una sola directory contenente:

- Il codice sorgente (eventualmente organizzato in sottodirectory);
- Il file CMakeLists.txt necessario alla compilazione (uno solo e posto nella directory principale del progetto);
- Un file di log di una partita con soli giocatori computer e un file di log di una partita fra giocatori computer e un giocatore umano;
- Un file readme.txt in cui riportate eventuali note che volete comunicare in fase di correzione. Questo file non è la documentazione, che deve essere inserita sotto forma di commenti nel codice, ma un elenco di note aggiuntive per chi corregge, per esempio problemi riscontrati e non risolti.

L'archivio non deve contenere l'eseguibile, perché il sorgente sarà compilato in fase di correzione. Il sistema CMake deve compilare con le opzioni di ottimizzazione attivate (-O2).
Dopo la consegna su moodle, **verificate ciò che avete consegnato** con i passi seguenti:

- Scaricate il vostro progetto da moodle in una directory diversa da quella usata per sviluppare;
- Lanciate cmake;
- Compilate il codice e verificate la corretta esecuzione.

Si suggerisce di consegnare anche compiti non completi. È tuttavia necessario che il software consegnato sia **compilabile** ed **eseguibile**.

La consegna in ritardo degli elaborati sarà **fortemente penalizzata** e considerata solo in un brevissimo lasso temporale dopo la scadenza (pochi minuti). Consegne effettuate successivamente saranno ignorate.

Alcuni esempi (non esaustivi) delle voci nella griglia di valutazione del progetto:

- Il progetto non compila
- Il progetto non esegue (dà sempre segmentation fault)
- Il progetto produce segmentation fault saltuari
- Il progetto ha memory leak

- CMakeLists sbagliato o inesistente
- Erroneo (assente o eccessivo) uso dell'ereditarietà
- Erroneo uso della std library per gestire elementi multipli
- Mancato controllo bound array / accesso a memoria non consentito
- Mancato controllo argomenti da linea di comando
- Utilizzo allocazione dinamica in maniera non opportuna
- Il programma non funziona correttamente
- Bad code style lieve/grave
- Mancanza log file richiesto dal progetto