Gioco chiamato "Trova il Tesoro", dove i partecipanti devono trovare un "tesoro" nascosto in una griglia rappresentata da una matrice.

### **Descrizione del progetto**

1. **Scopo:** Sviluppare un gioco in cui i giocatori devono navigare una griglia per trovare il tesoro nascosto.
2. **Tecnologie:** C#, WPF.
3. **Struttura:**
   * **MainWindow.xaml e MainWindow.xaml.cs:** Gestione della grafica e delle interazioni utente.
   * **GameLogic.cs:** Implementazione della logica di gioco (posizionamento del tesoro, aggiornamento stato).
   * **GridManager.cs:** Gestione della matrice e operazioni correlate.

### **Struttura delle classi**

1. **Classe** GameLogic**:** Gestisce la logica del gioco, come il posizionamento del tesoro e la verifica delle mosse.
2. **Classe** GridManager**:** Gestisce la matrice di gioco e le operazioni su righe e colonne.
3. **Classe** Player**:** Tiene traccia del giocatore (es. numero di mosse, coordinate attuali).
4. **Classe** GameUI **(connessa al MainWindow.xaml):** Si occupa dell’interazione grafica.

### **Codice suddiviso**

Ecco un esempio di come potrebbe essere strutturata la soluzione.

**1. Classe** GameLogic Questa classe implementa la logica del gioco.

using System; // Importa il namespace System per utilizzare funzionalità come Random.

namespace TreasureHuntGame // Definisce il namespace del progetto.

{

public class GameLogic // Definizione della classe principale che gestisce la logica del gioco.

{

private int treasureX; // Variabile per memorizzare la coordinata X del tesoro.

private int treasureY; // Variabile per memorizzare la coordinata Y del tesoro.

private Random random; // Instanza per generare numeri casuali.

public GameLogic() // Costruttore della classe GameLogic.

{

random = new Random(); // Inizializza l'oggetto Random per generare numeri casuali.

}

// Metodo per posizionare il tesoro casualmente nella griglia.

public (int, int) PlaceTreasure(int rows, int cols) // Ritorna una tupla con le coordinate del tesoro.

{

treasureX = random.Next(rows); // Genera una posizione casuale per la coordinata X basata sulle righe.

treasureY = random.Next(cols); // Genera una posizione casuale per la coordinata Y basata sulle colonne.

return (treasureX, treasureY); // Ritorna la posizione del tesoro.

}

// Metodo per verificare se il tesoro è stato trovato.

public bool IsTreasureFound(int x, int y) // Controlla se le coordinate del giocatore corrispondono al tesoro.

{

return x == treasureX && y == treasureY; // Ritorna true se le coordinate coincidono.

}

}

**2. Classe** GridManager Questa classe gestisce la matrice e le operazioni su di essa.

namespace TreasureHuntGame // Definisce il namespace del progetto.

{

public class GridManager // Classe responsabile della gestione della griglia.

{

private int[,] grid; // Dichiarazione della matrice che rappresenta la griglia.

public GridManager(int rows, int cols) // Costruttore della classe GridManager.

{

grid = new int[rows, cols]; // Inizializza la matrice con le dimensioni specificate.

}

// Metodo per resettare la griglia, impostando tutti i valori a 0.

public void ResetGrid()

{

for (int i = 0; i < grid.GetLength(0); i++) // Ciclo per ogni riga della griglia.

for (int j = 0; j < grid.GetLength(1); j++) // Ciclo per ogni colonna della griglia.

grid[i, j] = 0; // Imposta il valore della cella corrente a 0.

}

// Metodo per aggiornare una cella specifica nella griglia.

public void UpdateCell(int x, int y, int value)

{

grid[x, y] = value; // Aggiorna la cella con il valore specificato.

}

// Metodo per ottenere la matrice completa.

public int[,] GetGrid()

{

return grid; // Ritorna la matrice.

}

}

}

**3. Classe** Player Tiene traccia del giocatore e delle sue interazioni.

namespace TreasureHuntGame // Definisce il namespace del progetto.

{

public class Player // Classe che rappresenta il giocatore.

{

public int Moves { get; private set; } // Proprietà che tiene traccia del numero di mosse effettuate.

public Player() // Costruttore della classe Player.

{

Moves = 0; // Inizializza il numero di mosse a 0.

}

// Metodo per incrementare il conteggio delle mosse.

public void IncrementMoves()

{

Moves++; // Aumenta il numero di mosse di 1.

}

}

}

**4. Interfaccia Utente (**GameUI **e** MainWindow.xaml**)** Questa parte si occupa dell'interazione grafica. Nella finestra principale (XAML), creiamo una griglia dinamica.

using System.Windows; // Importa il namespace per gestire la grafica in WPF.

using System.Windows.Controls; // Importa il namespace per i controlli GUI come i pulsanti.

namespace TreasureHuntGame // Definisce il namespace del progetto.

{

public partial class MainWindow : Window // Classe principale della finestra.

{

private GameLogic gameLogic; // Istanza di GameLogic per la logica del gioco.

private GridManager gridManager; // Istanza di GridManager per gestire la griglia.

private Player player; // Istanza di Player per tenere traccia delle mosse del giocatore.

public MainWindow() // Costruttore della finestra principale.

{

InitializeComponent(); // Inizializza i componenti grafici.

gameLogic = new GameLogic(); // Inizializza l'oggetto GameLogic.

gridManager = new GridManager(5, 5); // Crea una griglia di dimensione 5x5.

player = new Player(); // Inizializza l'oggetto Player.

SetupGrid(); // Metodo per configurare la griglia grafica.

}

private void SetupGrid() // Metodo per configurare la griglia grafica.

{

for (int i = 0; i < 5; i++) // Itera per ogni riga della griglia.

{

for (int j = 0; j < 5; j++) // Itera per ogni colonna della griglia.

{

Button btn = new Button // Crea un nuovo pulsante per rappresentare una cella.

{

Content = "", // Imposta il contenuto iniziale del pulsante.

Tag = (i, j) // Memorizza la posizione della cella nelle proprietà del pulsante.

};

btn.Click += OnCellClick; // Associa l'evento di click al pulsante.

GameGrid.Children.Add(btn); // Aggiunge il pulsante alla griglia grafica.

}

}

}

private void OnCellClick(object sender, RoutedEventArgs e) // Metodo per gestire il click su una cella.

{

Button clickedButton = (Button)sender; // Identifica il pulsante cliccato.

var (x, y) = ((int, int))clickedButton.Tag; // Recupera le coordinate della cella cliccata.

player.IncrementMoves(); // Incrementa il numero di mosse del giocatore.

if (gameLogic.IsTreasureFound(x, y)) // Controlla se il tesoro è stato trovato.

{

clickedButton.Content = "🎉"; // Aggiorna la cella con un'icona di successo.

MessageBox.Show("Tesoro trovato in " + player.Moves + " mosse!"); // Mostra un messaggio di vittoria.

}

else

{

clickedButton.Content = "❌"; // Aggiorna la cella con un'icona di tentativo fallito.

}

}

}

}

**Schema XAML** Aggiungi una griglia nel tuo MainWindow.xaml:

<Grid Name="GameGrid" Margin="10" />

### **Schema logico**

Il flusso delle responsabilità tra le classi:

1. MainWindow (UI) interagisce con il giocatore, mostra la griglia e riceve input.
2. GameLogic si occupa della logica (posizionamento del tesoro, verifica delle coordinate).
3. GridManager gestisce lo stato della matrice.
4. Player tiene traccia delle mosse del giocatore.

**"Scomposizione del problema"** strutturata per passi numerati, che mostra il processo logico seguito per progettare e implementare la soluzione:

### **Scomposizione del problema**

1. **Definizione degli obiettivi del progetto:**
   * Creare un gioco interattivo basato su una griglia (matrice).
   * Implementare la logica per trovare un "tesoro" nascosto.
   * Garantire una chiara separazione delle responsabilità attraverso classi ben definite.
   * Utilizzare WPF per fornire un'interfaccia grafica all'utente.
   * Assicurarsi che il tempo stimato per completare l'attività sia adeguato (80 minuti).
2. **Identificazione degli elementi chiave del problema:**
   * **Griglia (matrice):** Una struttura dati per rappresentare il campo di gioco.
   * **Tesoro:** Un elemento nascosto in una posizione casuale della griglia.
   * **Giocatore:** Deve interagire con la griglia per trovare il tesoro.
   * **Logica del gioco:** Gestione della casualità e verifica delle azioni del giocatore.
   * **Interfaccia utente (UI):** Un sistema per visualizzare la griglia e ricevere input.
3. **Definizione dei requisiti principali:**
   * Deve esserci una matrice gestibile sia dal punto di vista della logica che della visualizzazione.
   * La posizione del tesoro deve essere generata casualmente.
   * Deve essere possibile registrare e conteggiare il numero di mosse del giocatore.
   * L'interfaccia utente deve essere reattiva e mostrare i tentativi (es. falliti o riusciti).
4. **Suddivisione del progetto in componenti modulari:**
   * **Classe** GameLogic**:** Per gestire la logica del tesoro (posizionamento, verifica).
   * **Classe** GridManager**:** Per gestire lo stato e l'aggiornamento della matrice.
   * **Classe** Player**:** Per tenere traccia delle azioni del giocatore.
   * **Interfaccia grafica (UI):** Configurazione della griglia e interazione con l'utente.
5. **Progettazione della struttura delle classi:**
   * Identificare responsabilità univoche per ogni classe per garantire la separazione.
   * Progettare un'interazione fluida tra le classi (es. MainWindow comunica con GameLogic e GridManager).
6. **Scelta della tecnologia e strumenti:**
   * **WPF (Windows Presentation Foundation):** Per costruire l'interfaccia grafica.
   * **C#:** Per implementare la logica del gioco e la gestione della matrice.
   * **Commenti nel codice:** Inserire spiegazioni dettagliate per rendere il progetto comprensibile.
7. **Implementazione delle funzionalità principali:**
   * **Logica del tesoro:** Generare una posizione casuale e controllare se è stata trovata.
   * **Gestione della griglia:** Creare e aggiornare dinamicamente lo stato della matrice.
   * **Conteggio mosse:** Registrare il numero di tentativi effettuati dal giocatore.
   * **UI interattiva:** Configurare pulsanti che rappresentano le celle e rispondono agli input.
8. **Integrazione delle componenti:**
   * Collegare MainWindow.xaml.cs con le classi logiche per sincronizzare grafica e dati.
   * Testare l'interazione tra pulsanti e metodi della logica di gioco (es. aggiornamenti delle celle).
9. **Test del progetto:**
   * Verificare che il tesoro sia generato correttamente.
   * Assicurarsi che i tentativi falliti vengano gestiti correttamente.
   * Controllare che il messaggio "Tesoro trovato!" appaia quando necessario.
   * Valutare la reattività della UI in risposta agli input del giocatore.
10. **Ottimizzazione e rifinitura finale:**
    * Aggiungere ulteriori dettagli grafici (es. simboli, colori) per migliorare l'esperienza utente.
    * Garantire che il codice sia commentato e leggibile.
    * Documentare la logica e il collegamento tra le classi per facilitare la comprensione.

### **Griglia di valutazione per la verifica**

| **Criterio** | **Descrizione** | **Punteggio massimo** |
| --- | --- | --- |
| **Correttezza funzionale** | Il programma funziona correttamente: il tesoro viene generato casualmente e la ricerca produce risultati coerenti. | 25 punti |
| **Struttura del codice** | Il codice è suddiviso in classi con responsabilità ben definite (es. GameLogic, GridManager, Player). | 20 punti |
| **Commenti nel codice** | Ogni riga/parte del codice è accompagnata da commenti chiari che spiegano le funzionalità e le scelte progettuali. | 15 punti |
| **Implementazione della UI (WPF)** | La griglia è visualizzata correttamente e risponde agli input dell'utente (es. clic sui pulsanti). | 15 punti |
| **Gestione degli errori** | Il programma gestisce correttamente i possibili errori (es. input fuori dai limiti) o casi eccezionali. | 10 punti |
| **Ottimizzazione e leggibilità** | Il codice è ben strutturato e facilmente leggibile grazie all'indentazione, ai nomi significativi delle variabili e ai metodi. | 10 punti |
| **Creatività ed esperienza utente** | Eventuali elementi extra (es. simboli, colori, notifiche) che migliorano l'interattività e la presentazione visiva. | 5 punti |

**Totale punteggio massimo:** **100 punti**

### **Dettaglio dei criteri**

1. **Correttezza funzionale (25 punti):**
   * Il tesoro è posizionato correttamente nella matrice.
   * Il programma identifica il tesoro quando trovato.
   * Non ci sono crash o bug evidenti.
2. **Struttura del codice (20 punti):**
   * Ogni classe è progettata per una responsabilità specifica.
   * Nessuna funzionalità risulta duplicata o confusa.
3. **Commenti nel codice (15 punti):**
   * Commenti utili che spiegano chiaramente il funzionamento del codice.
   * Assenza di commenti inutili o ridondanti.
4. **Implementazione della UI (15 punti):**
   * La griglia WPF viene generata correttamente.
   * Le celle reagiscono ai clic e mostrano feedback all'utente (es. ❌ o 🎉).
5. **Gestione degli errori (10 punti):**
   * Input o azioni anomale (es. clic ripetuti sulla stessa cella) sono gestiti correttamente.
   * Assenza di comportamenti imprevedibili.
6. **Ottimizzazione e leggibilità (10 punti):**
   * Buona organizzazione del codice, con metodi e variabili ben nominati.
   * Struttura modulare che facilita la lettura e la modifica.
7. **Creatività ed esperienza utente (5 punti):**
   * Aggiunta di dettagli grafici o messaggi che migliorano l’esperienza complessiva del giocatore.

### **Note per la valutazione**

* **Sufficienza:** 60/100 (il programma funziona ma presenta carenze in alcuni aspetti come commenti o UI).
* **Ottimo:** 90+/100 (soluzione completa, ben strutturata e arricchita con dettagli creativi).
* **Eccellenza:** 100/100 (soluzione impeccabile, con funzionalità avanzate o miglioramenti extra).

TRACCIA

## **Verifica di laboratorio: sviluppo di un gioco in C# con WPF**

### **Obiettivo**

Realizzare un’applicazione Windows basata su WPF che implementi un gioco chiamato *Trova il Tesoro*. Il programma dovrà utilizzare vettori e matrici per gestire la griglia del gioco e seguire i principi di programmazione ad oggetti per una corretta suddivisione delle responsabilità.

### **Consegne**

1. **Funzionalità richieste:**
   * Creare una griglia 5x5 (matrice) che rappresenti il campo di gioco.
   * Posizionare casualmente un "tesoro" in una cella della griglia.
   * Consentire al giocatore di cliccare sulle celle della griglia per cercare il tesoro.
   * Mostrare feedback visivo (es. simboli o colori) per i tentativi falliti e riusciti.
   * Conteggiare e mostrare il numero di mosse effettuate dal giocatore.
2. **Struttura del progetto:**
   * Il programma deve essere suddiviso in classi distinte:
     + GameLogic: Gestisce la logica del posizionamento del tesoro e verifica i tentativi.
     + GridManager: Gestisce la matrice e le operazioni correlate.
     + Player: Tiene traccia del numero di mosse effettuate.
     + GameUI: Gestisce l'interfaccia utente.
   * Commentare ogni riga di codice per spiegare chiaramente ogni operazione.
3. **Interfaccia utente:**
   * Utilizzare WPF per creare una griglia dinamica di pulsanti che rappresentano le celle.
   * Mostrare messaggi grafici e/o pop-up per il feedback del giocatore (es. *“Tesoro trovato in X mosse!”*).
4. **Gestione degli errori:**
   * Prevedere la gestione di situazioni anomale (es. input non validi).

### **Durata e ambiente**

* **Tempo di svolgimento:** 80 minuti.
* **Ambiente di sviluppo:** Visual Studio con il linguaggio C#.

### 

### 

### **Criteri di valutazione**

| **Criterio** | **Descrizione** | **Punteggio massimo** |
| --- | --- | --- |
| **Correttezza funzionale** | Il programma funziona correttamente: il tesoro viene generato casualmente e la ricerca produce risultati coerenti. | 25 punti |
| **Struttura del codice** | Il codice è suddiviso in classi con responsabilità ben definite (es. GameLogic, GridManager, Player). | 20 punti |
| **Commenti nel codice** | Ogni riga/parte del codice è accompagnata da commenti chiari che spiegano le funzionalità e le scelte progettuali. | 15 punti |
| **Implementazione della UI (WPF)** | La griglia è visualizzata correttamente e risponde agli input dell'utente (es. clic sui pulsanti). | 15 punti |
| **Gestione degli errori** | Il programma gestisce correttamente i possibili errori (es. input fuori dai limiti) o casi eccezionali. | 10 punti |
| **Ottimizzazione e leggibilità** | Il codice è ben strutturato e facilmente leggibile grazie all'indentazione, ai nomi significativi delle variabili e ai metodi. | 10 punti |
| **Creatività ed esperienza utente** | Eventuali elementi extra (es. simboli, colori, notifiche) che migliorano l'interattività e la presentazione visiva. | 5 punti |

### **Totale punteggio massimo:** **100 punti**

### 

### **Indicazioni per lo sviluppo**

1. Progettare le classi separatamente, seguendo i principi SRP (Single Responsibility Principle).
2. Implementare la matrice nel codice, garantendo che venga aggiornata correttamente.
3. Configurare l’interfaccia grafica (griglia di pulsanti) con eventi di click collegati alla logica del gioco.
4. Testare il programma per verificare il corretto funzionamento delle funzionalità richieste.
5. Documentare ogni fase dello sviluppo con commenti esplicativi e organizzare il codice per facilitare la comprensione.

### **Obiettivi extra (facoltativi)**

* Aggiungere animazioni o effetti visivi per rendere l’esperienza più coinvolgente.
* Implementare una funzionalità per resettare il gioco senza chiudere l’applicazione.

**Ecco come dovrebbe funzionare il gioco Trova il Tesoro:**

1. **Avvio del gioco:**
   * L'applicazione apre una finestra che mostra una griglia di dimensione 5x5 (o altra dimensione specificata), composta da pulsanti che rappresentano le celle.
   * Il tesoro viene posizionato in modo casuale in una cella della griglia, senza che l'utente possa conoscere la posizione.
2. **Interazione del giocatore:**
   * Il giocatore clicca sui pulsanti della griglia per cercare il tesoro.
   * Ogni clic corrisponde a un tentativo e viene registrato come una "mossa".
3. **Feedback visivo:**
   * Quando il giocatore clicca su una cella:
     + Se **non** contiene il tesoro, il pulsante cambia (es. mostrando un simbolo "❌" o un colore diverso) per indicare il tentativo fallito.
     + Se la cella contiene il tesoro, il pulsante mostra un'icona di successo (es. "🎉") e appare un messaggio di congratulazioni.
4. **Messaggio finale:**
   * Quando il giocatore trova il tesoro, l'applicazione mostra un pop-up con un messaggio del tipo: *"Tesoro trovato in X mosse!"*.
   * Il gioco termina e il giocatore può scegliere di ricominciare (opzionale).
5. **Conteggio delle mosse:**
   * Il numero totale di mosse viene registrato e mostrato alla fine per indicare la performance del giocatore.
6. **Gestione degli errori:**
   * Se il giocatore clicca più volte sulla stessa cella, il gioco non conta ulteriori mosse (opzionale).
   * Eventuali input non validi vengono gestiti in modo da non causare crash dell'applicazione.

SOLUZIONE SEMPLIFICATA, TUTTO IN UNA CLASSE

using System;

namespace TreasureHuntGame

{

public class GameLogic

{

private int[,] grid; // Matrice che rappresenta la griglia di gioco.

private int treasureX; // Coordinata X del tesoro.

private int treasureY; // Coordinata Y del tesoro.

// Costruttore della classe: inizializza la griglia e posiziona il tesoro.

public GameLogic(int rows, int cols)

{

grid = new int[rows, cols];

PlaceTreasure();

}

// Metodo per posizionare casualmente il tesoro nella griglia.

private void PlaceTreasure()

{

Random random = new Random();

treasureX = random.Next(grid.GetLength(0)); // Genera una posizione casuale per X.

treasureY = random.Next(grid.GetLength(1)); // Genera una posizione casuale per Y.

grid[treasureX, treasureY] = 1; // 1 indica che il tesoro è presente.

}

// Metodo per verificare se il giocatore ha trovato il tesoro.

public bool CheckTreasure(int x, int y)

{

return x == treasureX && y == treasureY;

}

}

}