

Progetto Programmazione a Oggetti e Ingegneria del Software

Anno Accademico 2016-2017 – Sessione Estiva

Unipoly

Giorgia Remedi - Matricola 272743

Paola Persico - Matricola 271220

Indice

1. Specifica del problema
2. Specifica dei requisiti
 - 2.1 Diagramma dei casi d'uso
 - 2.2 Relazioni e descrizione dei casi d'uso
3. Analisi e progettazione
 - 3.1 Modello di processo
 - 3.2 Linguaggio di programmazione
 - 3.3 Ambiente di sviluppo
 - 3.4 GUI
 - 3.5 Design pattern
 - 3.6 Organizzazione progetto
 - 3.7 Classi
 - 3.8 Diagramma delle classi
 - 3.9 Diagramma di sequenza
 - 3.10 Diagramma di attività
4. Implementazione
5. Test
 - 5.1 WhiteBox Test
 - 5.2 BlackBox Test
6. Compilazione ed esecuzione

1. SPECIFICA DEL PROBLEMA

Si vuole realizzare un'applicazione desktop che proponga un'alternativa al gioco Monopoly realizzata appositamente per studenti universitari di Informatica, ma adattabile anche ad altri corsi di laurea.

Avvio del gioco

Il numero di giocatori può variare da 2 a 4. Ogni giocatore può scegliere il proprio ruolo tra i seguenti: studente fuori sede, in sede, pendolare o straniero, e si inizia con una quota di 2000 euro e 0 CFU. Proprio come nel gioco tradizionale, si ci muove sul tabellone tirando a turni un dado a 6 facce.

Descrizione del tabellone

Il tabellone è composto da 40 caselle, così suddivise:

- 5 caselle di **Formazione matematico-fisica**:
 - Analisi Matematica (12 CFU)
 - Matematica Discreta (6 CFU)
 - Fisica I (6 CFU)
 - Fisica II (6 CFU)
 - Probabilità e Statistica (6 CFU)
- 2 caselle di **Formazione informatica di base**
 - Algoritmi e Strutture Dati (12 CFU)
 - Architettura degli Elaboratori (12 CFU)
- 3 caselle di **Discipline Informatiche**
 - Sistemi Operativi (12 CFU)
 - Basi di Dati (12 CFU)
 - Reti di Calcolatori (12 CFU)
- 3 caselle di **Programmazione**
 - Programmazione Procedurale e Logica (12 CFU)
 - Programmazione a Oggetti e Ingegneria del Software (12 CFU)
 - Linguaggi di Programmazione e Verifica del Software (12 CFU)
- 2 caselle di **Esami di curriculum**
 - Elaborazione di Segnali e Immagini (12 CFU)
 - Simulazione Numerica (6 CFU)
- 2 caselle di **Esami a Scelta**
 - Economia e Gestione delle Imprese (6 CFU)
 - Piattaforme Digitali per la Gestione del Territorio (6 CFU)
- 1 casella di **Tesi** (6 CFU)
- 1 casella di **Lingua Inglese** (6 CFU)
- 2 caselle di **Tirocini formativi**
 - tirocinio in azienda (3 CFU)
 - tirocinio in università (3 CFU)
- 1 casella “VIA” -> chi arriva su questa casella deve pagare una tassa di iscrizione di 500 euro
- 1 casella “vai in stallo per aver pagato in ritardo le tasse” -> chi passa qui va sulla casella di stallo
- 1 casella di “stallo” -> chi arriva qui resta fermo per 2 turni
- 1 casella “giovedì universitario” -> chi passa qui perde 24 CFU
- 3 caselle probabilità -> chi passa qui pesca una delle 16 carte probabilità
- 3 caselle imprevisti -> chi passa qui pesca una delle 16 carte imprevisti
- 2 caselle “tasse”
 - seconda rata -> chi passa qui paga 400 euro

- terza rata -> chi passa qui paga 100 euro + il proprio capitale/3
- 1 casella “borsa di studio” -> chi passa qui ottiene 500 euro
- 1 casella “Erasmus” -> chi passa qui ottiene 24 CFU
- 3 caselle “vacanze” (Natale, Pasqua, Estate) -> chi passa qui, resta fermo un turno
- 2 caselle “fuoricorso” -> chi passa qui, si ritrova il capitale decrementato del 10%

Svolgimento del gioco

Il primo giocatore che arriva su una casella di tipo esame può comprare il libro per l’esame secondo la seguente tabella prezzi:

- Libri per esami da 3 CFU -> 30 euro
- Libri per esami da 6 CFU -> 60 euro
- Libri per esami da 12 CFU -> 120 euro

ottenendo così i CFU corrispondenti. Ottenuti tutti gli esami di una stessa categoria, il giocatore può associare a pagamento degli appunti alle caselle (al prezzo di 20 euro codauno) proprio come nella versione tradizionale in cui si possono associare case e alberghi. I giocatori che passeranno successivamente sopra quelle caselle, dovranno pagare il libro e gli appunti allo stesso prezzo. I giocatori che vanno in rosso si ritrovano i CFU dimezzati.

Terminazione del gioco

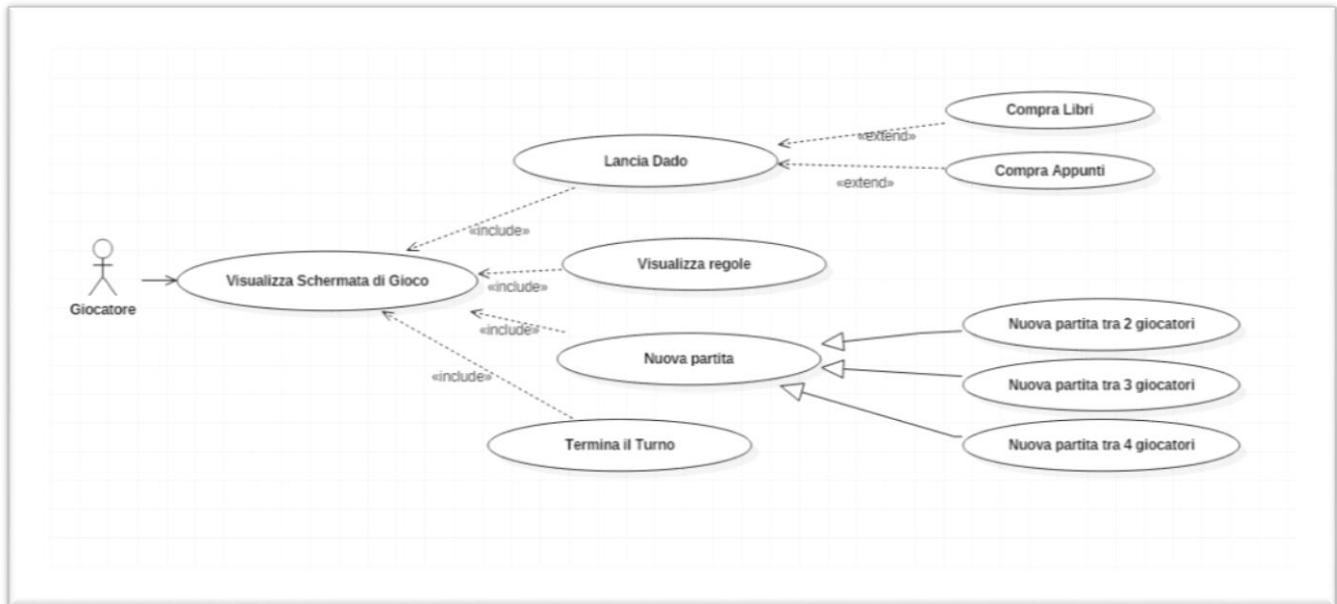
Il gioco termina se sussiste almeno una delle seguenti condizioni:

- uno studente si laurea (ottiene 180 CFU o più)
- finisce il tempo (2 ore) e vince lo studente con più CFU. A parità di CFU, vince quello con più soldi

2. SPECIFICA DEI REQUISITI

Di seguito vengono riportati i casi d'uso dell'applicativo che ci aiuteranno a comprendere al meglio le funzionalità del gioco.

Il diagramma UML dei casi d'uso ha evidenziato la partecipazione di un unico attore: il giocatore, il quale potrà scegliere inizialmente il proprio ruolo (fuori sede, pendolare, in sede, straniero).



Relazioni e descrizione dei casi d'uso

Si mostreranno di seguito le descrizioni dei casi d'uso individuati.

Il giocatore avvia l'applicazione GUI e visualizza il tabellone con un pulsante per avviare una nuova partita.

Caso d'uso	Inizia una nuova partita
ID	#1
Attore	Giocatore
Precondizioni	Il form visualizzato è quello della schermata di gioco e l'utente clicca sul pulsante "Avvia Partita"
Corso base	Compare un nuovo form e il giocatore sceglie i nomi e i ruoli dei giocatori, e conferma cliccando sul pulsante "Chiudi".
Postcondizioni	Viene chiuso il form e vengono visualizzate le pedine sulla casella "VIA" del tabellone, insieme ai dati personali dei giocatori inseriti, i pulsanti di azione, e viene evidenziato il segnaposto del giocatore corrente
Percorsi alternativi	L'utente non inserisce abbastanza giocatori e viene visualizzato un messaggio d'errore

Caso d'uso	Visualizza regole di gioco
ID	#2
Attore	Giocatore
Precondizioni	Il form visualizzato è quello della schermata di gioco e il giocatore clicca sul pulsante "Visualizza Regole di Gioco"
Corso base	Compare un nuovo form e il giocatore visualizza le regole di gioco.
Postcondizioni	Viene chiuso il form e viene visualizzata la schermata di gioco

Caso d'uso	Termina il turno
ID	#3
Attore	Giocatore
Precondizioni	Il form visualizzato è quello della schermata di gioco e il giocatore clicca sul pulsante "Termina Turno"
Corso base	Viene passato il turno
Postcondizioni	Vengono aggiornate le informazioni relative ai giocatori e viene evidenziato il segnaposto del giocatore successivo

Caso d'uso	Lancia il dado
ID	#4
Attore	Giocatore
Precondizioni	Il form visualizzato è quello della schermata di gioco e il giocatore clicca sul pulsante "Tira il dado"
Corso base	Il giocatore transita su una nuova casella in base all'esito del lancio
Postcondizioni	Viene visualizzato sulla sezione di flusso di gioco il risultato del lancio del dado (un numero da 1 a 6) e la pedina si sposta sul tabellone dello stesso numero di caselle, viene visualizzata la casella (ed eventuali carte pescate) e, se la casella è di tipo esame, vengono visualizzati i pulsanti di compravendita. Possono essere visualizzati dei messaggi sulla sezione di flusso di gioco.

Caso d'uso	Compra Libro
ID	#5
Attore	Giocatore
Precondizioni	Il giocatore si trova su una casella di tipo Esame
Corso base	Il giocatore clicca sul pulsante "Compra Libro"
Postcondizioni	Il giocatore si ritrova il conteggio di CFU aumentato del numero di CFU dell'esame corrispondente e il proprio capitale decrementato in base al costo della casella, e viene visualizzato sulla casella un segnaposto che indica il proprietario
Percorso alternativo	<ul style="list-style-type: none"> - Se il giocatore non ha abbastanza soldi, viene visualizzato sulla sezione di flusso di gioco un messaggio di errore. - Se la casella appartiene già ad un altro giocatore, viene visualizzato sulla sezione di flusso di gioco un messaggio di errore.

Caso d'uso	Compra Appunti
ID	#6
Attore	Giocatore
Precondizioni	Il giocatore si trova su una casella di tipo Esame
Corso base	Il giocatore clicca sul pulsante "Compra Appunti"
Postcondizioni	Al giocatore viene decrementato il conteggio dei soldi e viene messo in evidenza il segnaposto relativo al proprietario della casella
Percorso alternativo	<ul style="list-style-type: none"> - Se il giocatore non ha abbastanza soldi, viene visualizzato sulla sezione di flusso di gioco un messaggio di errore - Se la casella appartiene già ad un altro giocatore, viene visualizzato sulla sezione di flusso di gioco un messaggio di errore - Se il giocatore non possiede tutti gli esami dell'area formativa, viene visualizzato sulla sezione di flusso di gioco un messaggio di errore

3. ANALISI E PROGETTAZIONE

Si descrivono in questa sezione le principali scelte di progettazione e di implementazione fatte per realizzare il gioco.

Scelte di progetto

- Modello di processo
- Linguaggio di programmazione
- Ambiente di sviluppo
- GUI
- Design pattern
- Organizzazione progetto
- Classi
- Diagramma delle classi
- Diagramma di sequenza
- Diagramma di attività

Modello di processo

La metodologia di sviluppo software adottata durante la progettazione e la realizzazione del progetto è **Extreme programming** (abbreviato XP) basata su semplicità e comunicazione, e su un insieme di pratiche:

- **Design semplice** -> si segue il principio KISS (keep it simple, stupid!)
- **Refactoring** -> il refactoring consiste nel modificare il codice per renderlo più chiaro e coeso, preservando la logica funzionale.
- **Pair programming** -> il software è sviluppato da due programmatore che lavorano alla stessa postazione. Il codice risulta migliore, poiché c'è sempre qualcuno che lo revisiona, ed inoltre c'è una diffusione della conoscenza.
- **Sviluppo guidato dai test** -> i test vengono eseguiti ossessivamente per assicurarsi software funzionante.
- **Integrazione continua** -> si eseguono test di integrazione una decina di volte al giorno, poiché l'integrazione può portare a problemi seri nello sviluppo.
- **Proprietà collettiva del codice** -> tutti i programmatore possono lavorare a qualsiasi pezzo di codice in ogni momento.
- **Standard di coding** -> il codice deve sembrare scritto da un'unica persona
- **Metafore** -> si usa un sistema comune di nomi per avere una visione chiara del software

Linguaggio di programmazione : C#

Il linguaggio di programmazione C# è un linguaggio orientato ad oggetti. Uno dei vantaggi di questo linguaggio deriva dalla possibilità di utilizzo delle API del framework .NET che mette a disposizione numerose classi base per lo sviluppo di applicazioni di vario genere, sia Console che Windows Form.

C# sfrutta a pieno i principi della programmazione orientata ad oggetti, quali: l'incapsulamento dei vari componenti per riuscire a gestire il grado di accessibilità dei vari elementi; l'ereditarietà tra le varie classi e il polimorfismo che ci consentono di avere codice riusabile e più facilmente mantenibile per successive modifiche.

Ambiente di sviluppo

Si è utilizzato per lo sviluppo del progetto l'ambiente **Visual Studio C# 2015** che utilizza il framework .NET. La versione utilizzata è quella che Microsoft mette gratuitamente a disposizione cioè la **Community**. Questo ambiente di sviluppo permette di creare facilmente applicazioni grafiche e di gestire in modo intuitivo gli eventi.

GUI

Per ottenere un livello di gioco che si avvicinasse al Monopoly giocato realmente, si è deciso di realizzare un'applicazione di tipo grafico. Infatti lo stessa tipologia di gioco, via console, non avrebbe coinvolto in egual modo il giocatore.

L'intero progetto è stato quindi concepito attraverso l'utilizzo di Windows Form e dei suoi elementi sfruttando classi già presenti nel framework .NET.

Tra le classi grafiche utilizzate abbiamo:

- *PictureBox* che rappresenta un controllo casella dell'immagine
- *Button* che rappresenta un pulsante che risponde all'evento Click
- *MessageBox* visualizza una finestra di messaggio che può contenere testo, pulsanti e simboli
- *TextBox* che rappresenta un controllo casella di testo
- *ComboBox* che rappresenta dei possibili valori da selezionare
- *Label* che rappresenta un'etichetta testuale
- *Timer* che rappresenta un timer
- *Panel* che rappresenta un gruppo di controlli

Abbiamo introdotto principalmente tre form:

1. Form tavolo da gioco

All'avvio del programma viene visualizzata la schermata di gioco priva di informazioni, con i pulsanti “Lancia Dado”, “Termina Turno”, “Compra Libro”, “Compra Appunti”, “Stop” disabilitati. Gli unici pulsanti abilitati sono “Visualizza Regole” e “Inizia Partita”.

2. Form avvia partita

Attraverso l'evento MouseClick sul pulsante “Inizia Partita”, si apre un nuovo form per iniziare la partita, scegliendo attraverso delle TextBox i nomi dei giocatori e attraverso delle ComboBox i ruoli. Una volta cliccato sul pulsante “Conferma”, nel form tavolo di gioco vengono attivati i pulsanti “Lancia Dado” e “Stop”, vengono visualizzate le informazioni sui giocatori (attraverso TextBox, Label e PictureBox) in alto, le informazioni di gioco nella TextBox scrollabile sulla destra, e le pedine come delle PictureBox sulla casella “VIA” del tabellone (rappresentato anch'esso attraverso un Panel). Inoltre viene fatto partire il timer (gestito da un Timer e visualizzato attraverso una Label).

3. Form regole di gioco

Cliccando sul pulsante “Visualizza Regole”, compare un Form con le regole del gioco.

Le dimensioni del form tavolo da gioco non possono essere modificate dall'utente durante il gioco.

Design Pattern

- **Observer:** Pattern comportamentale, usato per definire una dipendenza tra oggetti in modo tale che, quando un oggetto cambia stato, tutti i suoi dipendenti vengono notificati e si aggiornano automaticamente. Si rende necessario poiché un effetto collaterale comune del partizionare un sistema in una collezione di classi cooperanti è la necessità di mantenere consistenza tra gli oggetti correlati. Gli oggetti chiave in questo pattern sono il soggetto e l'osservatore. L'osservatore viene notificato quando il soggetto subisce un cambiamento di stato. Per tutta risposta, l'osservatore chiederà al soggetto di sincronizzare il suo stato con lo stato del soggetto. I partecipanti sono:
 - **Subject** -> conosce il suo osservatore
 - **Observer** -> definisce un'interfaccia di aggiornamento per l'oggetto che dovrebbe essere notificato dei cambiamenti in un soggetto
 - **ConcreteSubject** -> conserva lo stato di interesse dell'oggetto ConcreteObserver. Manda una notifica al suo osservatore quando cambia il suo stato
 - **ConcreteObserver** -> mantiene un riferimento ad un oggetto ConcreteSubject. Conserva lo stato che dovrebbe restare consistente col soggetto. Implementa l'interfaccia di aggiornamento Observer per mantenere il suo stato consistente con quello del soggetto.
- **Singleton:** Pattern creazionale usato per assicurare che una classe abbia una sola istanza, e per fornire un punto di accesso globale. Si ottiene ciò definendo l'operazione Instance() che permette ai client di accedere alla sua unica istanza. Instance() è un'operazione di classe, cioè un metodo statico. Potrebbe essere responsabile di creare la sua propria istanza unica. I client accedono a singleton esclusivamente attraverso il metodo statico Instance(). La variabile instance è inizializzata a null, e il metodo Instance() ritorna il suo valore, inizializzandolo con l'unica istanza se è null. Instance() usa la lazy invocation: il valore che ritorna non è creato e conservato fino a quando non viene acceduto per la prima volta. Bisogna notare che il costruttore è protetto. Un client che prova a istanziare Singleton direttamente avrà un errore a tempo di compilazione. Questo assicura che solo un'istanza potrà essere creata.

Per quanto riguarda Observer, esso è già integrato nel sistema eventi-delegati di C#: la notifica all'osservatore avviene tramite la generazione di un evento, a cui si può rispondere attraverso un metodo gestore di eventi (event handler) dedicato. Un esempio di evento nel nostro programma è il click sul pulsante di lancio del dado, che triggerà il gestore dell'evento *dado_Click()*, che risponde chiamando il metodo *LanciaDadoSpostaGiocatore()*.

Per quanto riguarda invece il pattern Singleton, l'abbiamo utilizzato per assicurarci che ci fosse una sola istanza della classe Gioco. Per fare ciò, abbiamo definito un metodo statico *OttieniGioco()* per invocare il costruttore protected e istanziare l'attributo statico *istanza*.

Organizzazione progetto

L'applicazione sviluppata separa logica funzionale e parte grafica. Infatti si è creata una libreria che funge da modello del gioco, che viene poi importata nell'applicazione Windows Form.

- Per sviluppare il modello si è ricorso ai principali concetti offerti dalla programmazione ad oggetti: l'**ereditarietà** tra caselle e tra carte, per poter adottare un approccio tendente al **polimorfismo** tramite l'upcasting (l'aggiunta delle caselle alla lista *tabellone*, e l'aggiunta delle carte alle liste *cartel* e *carteP*) e l'uso dell'overriding dei metodi ereditati (*Transita()* e *Usa()*).
- Nell'applicazione grafica abbiamo inserito una classe che eredita da Windows.Form per ogni finestra, insieme ad una classe Pedina. Nel lancio iniziale dell'applicazione verrà creato il tavolo da gioco, attraverso la pressione di un pulsante verrà aperta una finestra di avvio della partita, e attraverso la pressione di un altro pulsante verrà visualizzata una finestra contenente le regole del gioco.

Abbiamo quindi questi elementi fondamentali:

LibreriaUnipoly	Gioco Giocatore Dado Configurazione Casella CasellaEsame CasellaCarta CasellaSpeciale CasellaFuoriCorso CasellaTerzaRata Carta CartaMovimento CartaStallo CartaCfu CartaSoldi AreaFormativa
UnipolyGUI	AvviaPartita Tavolo Regole Pedina

Si descrivono di seguito le classi istanziate e utilizzate per lo sviluppo del progetto attraverso l'utilizzo di diagrammi UML.

Diagramma delle classi

Nota: le proprietà verranno indicate in questo modo: *visibilità nome {get; set;}(): tipo*

Modello

1) Gioco

Gioco	
<pre>-istante: Gioco +MIN_GIOCATORI: int = 2 {readonly} +MAX_GIOCATORI: int = 4 {readonly} +tempoPartita: TimeSpan {readonly} -tabellone: List<Casella> -cartel: List<Carta> -carteP: List<Carta> -giocatori: List<Giocatore> -dato: Dado -turno: int -giocatoreCorrente: Giocatore -infoGioco: string -inizioPartita: DateTime</pre> <pre>#Gioco() +OttieniGioco(): Gioco +InfoGioco {get; set;}(): string +GiocatoreCorrente {get;}(): Giocatore +NumCarteP {get;}(): int +NumCartel {get;}(): int +Giocatori {get;}(): List<Giocatore> +IniziaPartita(): bool +CambiaTurno(): bool -ControllaPartita(): bool +AggiungiGiocatore(nome: string, ruolo: string): Giocatore +OttieniCasella(indice: int): Casella +OttieniIndice(casella: Casella): int +LanciaDadoSpostaGiocatore(): int +AggiungiCartaP(carta: Carta) +AggiungiCartal(carta: Carta) +AggiungiCasella(casella: Casella) +LeggiCartaP(indice: int): Carta +LeggiCartal(indice: int): Carta +DecretaClassifica(): List<Giocatore></pre>	

La classe Gioco è incaricata di controllare la partita, rispondendo alle azioni dei giocatori sulla base delle regole. Poiché è stato utilizzato il pattern Singleton, è stato necessario inserire un attributo statico per l'istanza, un costruttore protetto ed un metodo statico che richiama il costruttore. Si hanno 2 attributi di classe che indicano il numero minimo e quello massimo di giocatori, un attributo a sola lettura che indica la durata massima della partita ed un attributo che indica la data di inizio partita per poter calcolare il tempo residuo e fermare il gioco. L'attributo tabellone è una lista di caselle di vario tipo, mentre cartel e carteP rappresentano rispettivamente la lista delle carte imprevisto e quella di carte probabilità. Un'altra lista presente è quella di giocatori, e ad ogni cambio turno viene aggiornato l'attributo giocatoreCorrente e l'attributo turno, che è il suo indice nella lista. Infine abbiamo gli attributi dado e la stringa infoGioco.

Per quanto riguarda i metodi, **IniziaPartita()** controlla se il numero di giocatori aggiunti rientra nel range, e nel caso affermativo inizializza i parametri di gioco. **CambiaTurno()** invece invoca **ControllaPartita()** per assicurarsi che la partita possa continuare (cioè se non ci sono giocatori con più di 180 CFU e se c'è ancora tempo residuo), poi se il giocatore corrente

ha lanciato il dado, effettua il passaggio ignorando i giocatori in stallo. Il metodo **LanciaDadoSpostaGiocatore()** è il fulcro della classe: invoca il dado per ottenere l'esito del lancio e sposta il giocatore sulla casella target. **DecretaClassifica()** permette di determinare il vincitore al termine della partita sulla base della chiave primaria Cfu e della chiave secondaria Soldi. Alcuni di questi metodi aggiornano la stringa infoGioco che potrà essere visualizzata per avere un log degli eventi di gioco.

2) Giocatore

Giocatore	
-nome: string {readonly}	
-ruolo: string {readonly}	
-soldi: double	
-cfu: int	
-turniDaSaltare: int	
-posizione: Casella	
-gioco: Gioco	
-dadoLanciato: bool	
+CAPITALE_INIZIALE: double = 2000 {readonly}	
+Giocatore(nome: string, ruolo: string, gioco: Gioco)	
+Nome {get;}: string	
+Ruolo {get;}: string	
+DadoLanciato {get; set;}: bool	
+Soldi {get; set;}: double	
+Cfu {get; set;}: int	
+TurniDaSaltare {get; set;}: int	
+Posizione {get; set;}: Casella	
+Spostati(numero: int)	
+Spostati(casella: Casella)	

La classe Giocatore presenta un attributo identificativo di tipo stringa, che viene inizializzato nel costruttore all'avvio della partita insieme al ruolo (fuori sede, pendolare, in sede e straniero), vari attributi che indicano CFU, capitale iniziale e soldi, un attributo di tipo intero che indica i turni da saltare se si è in stallo, un attributo che indica la posizione corrente sul tabellone.

Si hanno due metodi in overload chiamati **Spostati()**: uno prende in input l'esito del lancio del dado e ricava attraverso il metodo **OttieniCasella()** della classe Gioco la nuova posizione, l'altro prende in input direttamente la casella target. Entrambi invocano il metodo **Transita()** della casella target. Una particolarità della proprietà Soldi è che, coerentemente con le regole, se il giocatore va in rosso, allora i suoi CFU vengono dimezzati di volta in volta.

3) Dado

Dado	
-max: int {readonly}	
-min: int {readonly}	
-md: Random	
+Dado(min: int, max: int)	
+LanciaDado(): int	
+Max {get;}: int	
+Min {get;}: int	

La classe Dado è molto semplice: dati un valore massimo ed uno minimo, restituisce un numero casuale compreso tra i due (estremi inclusi). Nel nostro caso, il valore sarà un numero tra 1 e 6 per quanto riguarda il lancio del dado, e tra 0 e 15 per quanto riguarda l'estrazione della carta dal mazzo.

4) Configurazione

Configurazione	
-gioco: Gioco	
+Configurazione(gioco: Gioco)	
+CaricaTabellone()	

La classe Configurazione consente, attraverso il metodo **CaricaTabellone()**, di definire le specifiche del tabellone, cioè le caselle, le carte e le aree formative attraverso l'invocazione dei metodi presenti nella classe Gioco. Separare la configurazione dal gioco consente una maggior possibilità di riuso: infatti basta cambiare il file

di configurazione per adattare il gioco a diverse specifiche (ad esempio inserendo caselle esame di altri corsi di laurea o aggiungendo nuove carte imprevisti e probabilità).

5) Casella

Casella
#nome: string #istruzioni: string #gioco: Gioco #tipoCarta: string #areaFormativa: AreaFormativa +Nome {get;}: string +Istruzioni {get;}: string +Carta {get;}: Carta +TipoCarta {get;}: string +AreaFormativa {get;}: AreaFormativa +Transita() +Compra(): bool +Potenzia(): bool

La classe astratta Casella è la generalizzazione dei vari tipi di casella, che hanno come attributi comuni solo una stringa che rappresenta il nome e una stringa che rappresenta una descrizione delle istruzioni da eseguire nel caso si transiti su di essa. Alcune caselle presentano una carta associata (le caselle di tipo CasellaCarta e di tipo CasellaSpeciale) ed una stringa che identifica il tipo di carta (nel caso di CasellaCarta si hanno sia carte imprevisto che carte probabilità). Inoltre le caselle di tipo CasellaEsame devono specificare l'area formativa di appartenenza. Alla classe appartengono tre metodi: **Transita()**, che è un metodo astratto ridefinito in modo diverso dalle sottoclassi; **Compra()** e **Potenzia()**, che sono due metodi virtuali che di default lanciano un'eccezione, ma che possono essere ridefiniti dalle sottoclassi comprabili e/o potenziabili.

6) CasellaEsame

CasellaEsame
-pesoCfu: int {readonly} -prezzoLibro: double {readonly} <u>-PREZZO APPUNTI: double = 20 {readonly}</u> -appunti: int -proprietario: Giocatore +CasellaEsame(nome: string, istruzioni: string, pesoCfu: int, gioco: Gioco, areaFormativa: AreaFormativa) +Proprietario {get; set;}: Giocatore +PrezzoLibro {get;}: double +PesoCfu {get;}: int +Appunti {get;}: int +Compra(): bool +Transita() +Potenzia(): bool +ToString(): string

La classe CasellaEsame specializza la classe Casella, e si differenzia da essa ridefinendo **Compra()** e **Potenzia()**, oltre al metodo **ToString()** ereditato dalla classe Object. Il proprietario inizialmente è un valore null, mentre il numero di appunti assume inizialmente il valore zero, ed entrambi vengono aggiornati di volta in volta quando viene comprato il libro tramite **Compra()** o aggiunti degli appunti tramite il metodo **Potenzia()**. Prima che avvenga ciò, però, vengono effettuati dei controlli: per entrambi si richiede che il giocatore abbia abbastanza soldi e che la casella non appartenga già ad un altro giocatore, ed in **Potenzia()** si richiede anche che il giocatore possegga tutte le caselle dell'area formativa (il controllo viene effettuato invocando il metodo **AcquisitaDa()**). Se gli acquisti hanno successo, viene decrementato anche il capitale del giocatore.

7) CasellaSpeciale

CasellaSpeciale
+CasellaSpeciale(nome: string, istruzioni: string, gioco: Gioco, carta: Carta)
+Transita()
+ToString(): string

La classe CasellaSpeciale è un'altra specializzazione di Casella. Per garantire il massimo riuso, ad ogni casella speciale viene associata una carta che ne identifichi il comportamento in caso di transito su di essa: infatti molte caselle speciali hanno un comportamento simile (ad esempio le tre caselle “di vacanza”, che comportano tutte che il giocatore salti un turno).

8) CasellaCarta

CasellaCarta
+CasellaCarta(nome: string, istruzioni: string, gioco: Gioco)
+Transita()
+ToString(): string

Anche la classe CasellaCarta specializza Casella. La stringa nome in questo caso identifica il tipo di carta che verrà pescata nel metodo **Transita()**: se il tipo è “Imprevisto”, la carta verrà pescata dal mazzo di carte imprevisto secondo l’indice generato casualmente, altrimenti verrà pescata dal mazzo probabilità. Pescata la carta, vengono eseguite le istruzioni relative. Anche in questo caso viene ridefinito il metodo **ToString()**.

9) CasellaFuoriCorso

10) CasellaTerzaRata

CasellaFuoriCorso
+CasellaFuoriCorso(nome: string, istruzioni: string, gioco: Gioco)
+Transita()
+ToString(): string

CasellaTerzaRata
+CasellaTerzaRata(nome: string, istruzioni: string, gioco: Gioco)
+Transita()
+ToString(): string

Si è reso necessario creare due caselle speciali a parte perché queste accedono al capitale del giocatore corrente e non possono essere legate ad una carta istanziata a priori. Per quanto riguarda la casella Fuori Corso, essa ridefinisce il metodo **Transita()** decrementando il capitale del giocatore corrente del 10%, mentre la casella Terza Rata decrementa il capitale del giocatore corrente di una quota correlata al suo capitale. Entrambe ridefiniscono anche **ToString()**.

11) Carta

Carta
#ruolo: string #valore: double #casella: Casella #descrizione: string #gioco: Gioco +Ruolo {get;}(): string +Valore {get;}(): double +Casella {get;}(): Casella +Descrizione {get;}(): string +Usa()

La classe astratta Carta modelizza le carte che verranno pescate nel caso in cui si transiti sulla casella Casella Carta, e le carte associate alle Caselle Speciali. Consta di 5 attributi protetti, che verranno inizializzati dalle carte concrete: una stringa ruolo necessaria poiché alcune carte hanno effetto solo su giocatori con un determinato ruolo, un valore di tipo double che può indicare il numero di cfu, soldi o turni da saltare da incrementare o decrementare, una casella nel caso in cui la carta sia di tipo Movimento, una descrizione delle caratteristiche della carta, il gioco a cui ci si riferisce. Il metodo astratto **Usa()** verrà implementato dalle carte concrete in maniera coerente col tipo.

12) Carta Movimento

13) CartaSoldi

14) CartaCfu

15) CartaStallo

CartaMovimento
+CartaMovimento(ruolo: string, casella: Casella, gioco: Gioco, descrizione: string) +Usa()

CartaSoldi
+CartaSoldi(ruolo: string, valore: double, gioco: Gioco, descrizione: string) +Usa()

CartaCfu
+CartaCfu(ruolo: string, valore: double, gioco: Gioco, descrizione: string) +Usa()

CartaStallo
+CartaStallo(ruolo: string, valore: double, gioco: Gioco, descrizione: string) +Usa()

Le carte concrete sottoclassi di Carta si differenziano per il modo in cui implementano **Usa()**: la carta Movimento determina uno spostamento del giocatore sulla casella indicata; la carta Soldi determina un incremento o decremento del capitale; la carta Cfu determina un incremento o decremento dei CFU; la carta Stallo determina un incremento dei turni da saltare.

16) Area Formativa

Area formativa
-nome: string {readonly} -esami: List<CasellaEsame> +AreaFormativa(nome: string) +Nome {get;}(): string +AggiungiEsame(esame: CasellaEsame) +AcquisitaDa(giocatore: Giocatore): bool

Le aree formative create dalla classe Configurazione rappresentano degli insiemi di Caselle Esame. La classe Area Formativa è quindi composta da due soli metodi: **AggiungiEsame()**, e **AcquisitaDa()**. Quest'ultimo prende in input un giocatore e restituisce true se il giocatore possiede tutti gli esami di quell'area formativa.

17) TavoloGioco

TavoloGioco
<pre> -gioco: Gioco -config: Configurazione -corrente: Casella -tempo: TimeSpan -immaginiPedine: PictureBox * -segnaposti: PictureBox * -coloriPerGiocatore: Color * -colorePerArea: Dictionary<string, Color> -facceDado: Bitmap * -nomi: Label * -dati: TextBox * -pulsanti: Button * +TavoloGioco() +AvviaPartita{get;}: Button +Dado{get;}: Button +Stop{get;}: Button +Turno{get;}: Button +Libro{get;}: Button +Appunti{get;}: Button +Regole{get;}: Button +TimerGioco{get;}: Label +Nomi{get;}: Label * +SegnaPosti{get;}: PictureBox * +ImmaginiPedine{get;}: PictureBox * +CartaPescata{get;}: PictureBox +Dati{get;}: TextBox * +NomeCarta{get;}: TextBox +DescrizioneCarta{get;}: TextBox +AvviaTimer() +dado_Click(sender: object, e: EventArgs) +turno_Click(sender: object, e: EventArgs) +libro_Click(sender: object, e: EventArgs) +appunti_Click(sender: object, e: EventArgs) +avviaPartita_Click(sender: object, e: EventArgs) +regole_Click(sender: object, e: EventArgs) +stop_Click(sender: object, e: EventArgs) +veroTimer_Tick(sender: object, e: EventArgs) +timerMovimento_Tick(sender: object, e: EventArgs) +Muoviti(prima: Casella, seconda: Casella) +MostraProprietario() +TrovalImmagineProprietario(): PictureBox +MostraCarta(mostra: bool) +ColoraCarta(colore: Color) +DatiGiocatore(indice: int): string +StampaInfoGioco() +TerminaPartita() +load() </pre>

La classe TavoloGioco eredita da Form ed è la finestra principale. Essa importa la libreria LibreriaUnipoly, ed ha come attributi importati Gioco, Configurazione e Casella (quest'ultimo rappresenterà la casella corrente). Inoltre presenta un attributo di tipo TimeSpan che rappresenta il tempo residuo, utilizzato dall'evento **veroTimer_Tick()**.

Oltre agli attributi di tipo Control generati automaticamente da Visual Studio, si è ritenuto opportuno aggiungere delle Collection di controlli che ne raggruppassero alcuni per poter accedere più facilmente ad essi: immaginiPedine raggruppa le PictureBox delle pedine sul tabellone; segnaposti raggruppa le PictureBox che compaiono accanto ai nomi dei giocatori; coloriPerGiocatore è un array di colori ordinati con lo stesso indice dei giocatori; colorePerArea è un tabella hash che associa al nome di un'area formativa un determinato colore (per colorare le caselle mostrate sullo schermo dello stesso colore della casella sul tabellone); facceDado raggruppa le PictureBox che rappresentano i 6 esiti del dado da mostrare; nomi raggruppa le etichette coi nomi dei giocatori; dati raggruppa le TextBox coi dati dei giocatori (che verranno aggiornate tramite il metodo **DatiGiocatore()**); infine pulsanti raggruppa tutti i Button (eccetto Regole, che non viene mai disabilitato).

Quando viene caricato il form, attraverso il gestore dell'evento **load()** vengono inizializzati tutti gli attributi. Si hanno vari gestori degli eventi, che sono essenzialmente di due tipi: quelli che gestiscono la pressione dei pulsanti:

- **dado_Click()**, che invoca **LanciaDadoSpostaGiocatore()**
- **turno_Click()**, che invoca **CambiaTurno()**
- **libro_Click()**, che invoca **Compra()** sulla casella corrente
- **appunti_Click()**, che invoca **Potenzia()** sulla casella corrente
- **avviaPartita_Click()**, che invoca **AvviaPartita()**
- **stop_Click()**, che invoca **TerminaPartita()**

e quelli che gestiscono i due timer:

- **veroTimer_Tick()**, per poter aggiornare il label TimerGioco (decrementando il tempo residuo ogni secondo)
- **timerMovimento_Tick()** per poter posticipare un secondo movimento nel caso in cui si peschi una carta di tipo Movimento.

Oltre ai gestori degli eventi, sono stati introdotti i metodi:

- **Muoviti()**, che prende in input la casella sorgente e la casella destinazione e invoca **MovimentoPedina()** sulla pedina del giocatore corrente
- **MostraProprietario()** che permette di mostrare sul tabellone il nuovo proprietario della casella corrente dopo un acquisto
- **TrovalImmagineProprietario()** per trovare la PictureBox che identifica il proprietario della casella corrente
- **MostraCarta()** che viene invocato dopo aver pescato una carta Imprevisto o Probabilità
- **ColoraCarta()** per colorare la carta a seconda che sia Imprevisto o Probabilità
- **StampaInfoGioco()**, che stampa le informazioni di gioco (es. "Pinco Pallino salta il turno") sulla TextBox scrollabile di flusso di gioco.

18) AvviaPartita

AvviaPartita
-gioco: Gioco {readonly}
-tg: TavoloGioco {readonly}
+AvviaPartita(gioco: Gioco, tg: TavoloGioco)
+Conferma_Click(sender: object, e: EventArgs)
+ValidaGiocatore(nome: string, ruolo: string): bool
+AggiornaDati(indice: int, immagine: Bitmap, nome: string, dati: string)

La classe AvviaPartita eredita da Form, e consente l'immissione (tramite TextBox e ComboBox) del nome e del ruolo dei giocatori, che vengono aggiunti al gioco, previa validazione, attraverso il gestore dell'evento **Conferma_Click()**. Una volta aggiunti i giocatori, i dati vengono visualizzati sul tavolo da gioco.

19) Regole

Regole
+Regole()
+Istruzioni {get; }(): PictureBox

La classe Regole eredita anch'essa da Form, e serve a visualizzare le regole di gioco sotto forma di PictureBox. Essa viene creata una volta premuto il pulsante Regole che si trova sul Tavolo da Gioco.

20) Pedina

Pedina
-tg: TavoloGioco {readonly}
-gioco: TavoloGioco {readonly}
-immaginePedina: PictureBox
+DISTANZA_PEDINE: int = 20 {readonly}
+DISTANZA_CASELLE: int = 55 {readonly}
-bassoDestra: Point {readonly}
-bassoinistra: Point {readonly}
-altoSinistra: Point {readonly}
-altoDestra: Point {readonly}
+Pedina(tg: TavoloGioco, gioco: Gioco)
+ImmaginePedina {set; }(): PictureBox
+MovimentoPedina(prec: Casella, corr: Casella)
+PosizionaPedinaAngolo(punto: Point)

È stata introdotta una classe Pedina per implementare la logica del movimento della pedina sul tabellone. Il metodo **MovimentoPedina()** determina lo spostamento della PictureBox immaginePedina dalla casella precedente alla casella corrente avanzando di una casella alla volta. Perciò si è reso necessario un attributo che indichi la distanza tra le caselle sul tabellone. Poiché gli angoli hanno un'ampiezza diversa rispetto alle altre caselle, è stato introdotto un metodo **PosizionaPedinaAngolo()** che viene invocato quando si incontrano le caselle margine del tabellone: le pedine vengono posizionate al punto indicato dai vari attributi di tipo Point (viene sfruttato l'attributo di distanza delle pedine per fare in modo che non si sovrappongano).

Infine abbiamo la classe Program, col metodo **Main()** che non fa altro che creare il Form TavoloGioco.

Si riporta il diagramma delle classi UML completo (suddiviso in due immagini per questioni di leggibilità).

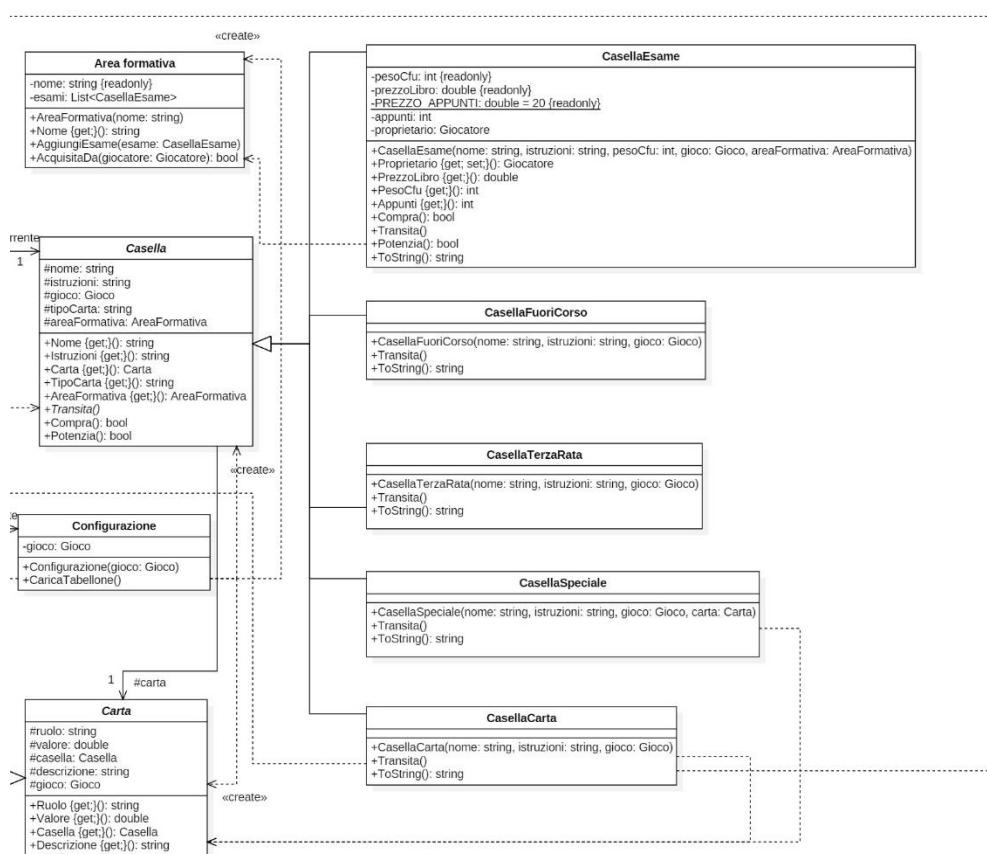
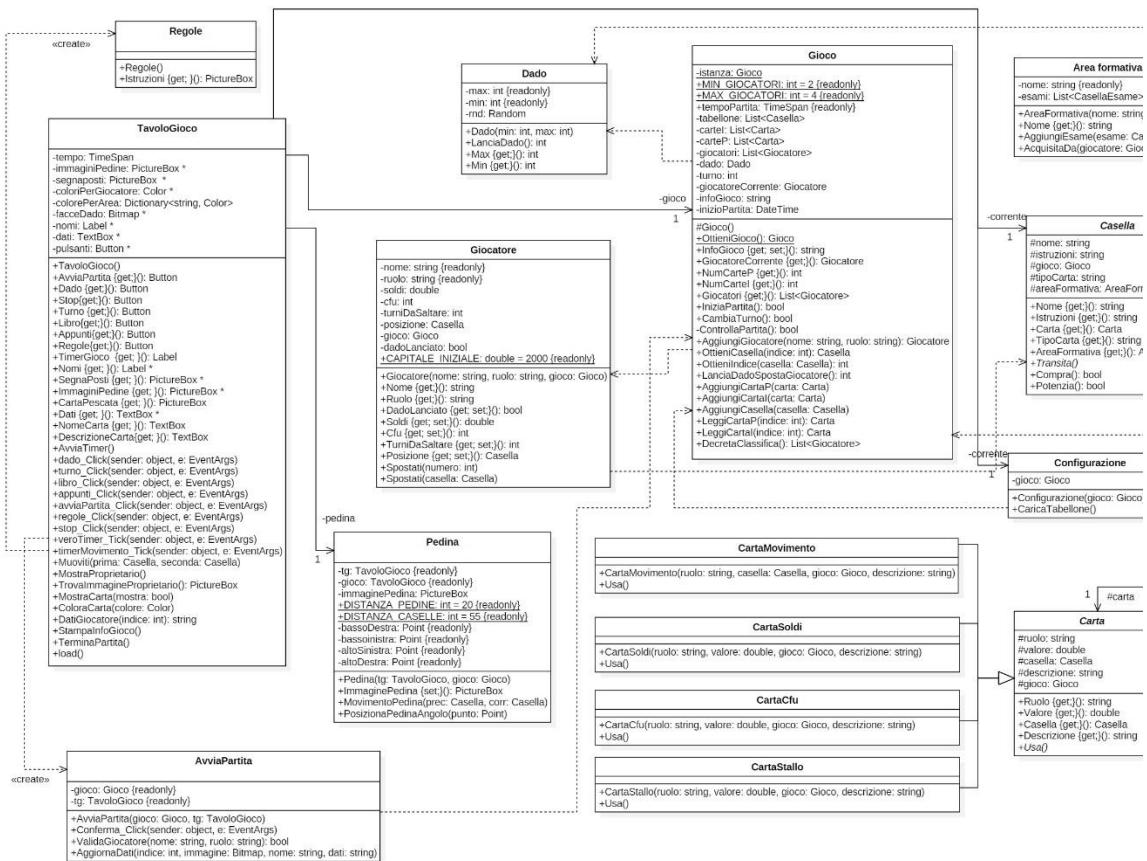


Diagramma di sequenza

I diagrammi di sequenza servono a mostrare un aspetto interessante delle interazioni tra oggetti/classi. Si è scelto quindi di rappresentare la sequenza di chiamate meno intuitiva, cioè quella che riguarda il lancio di un dado che porta ad una Casella Carta che a sua volta determina l'estrazione di una carta di tipo Movimento che conduce la pedina del giocatore ad una nuova casella.

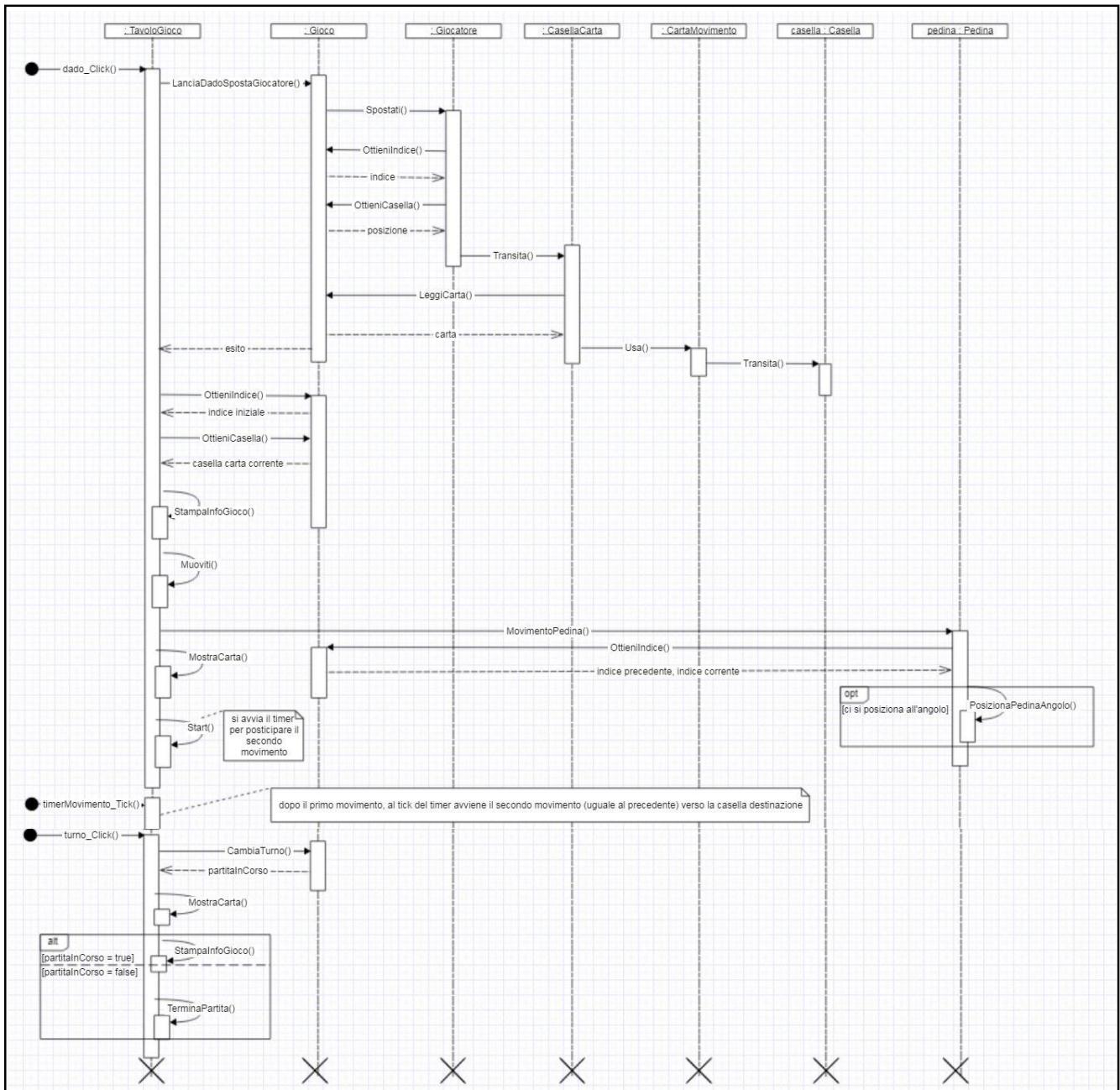
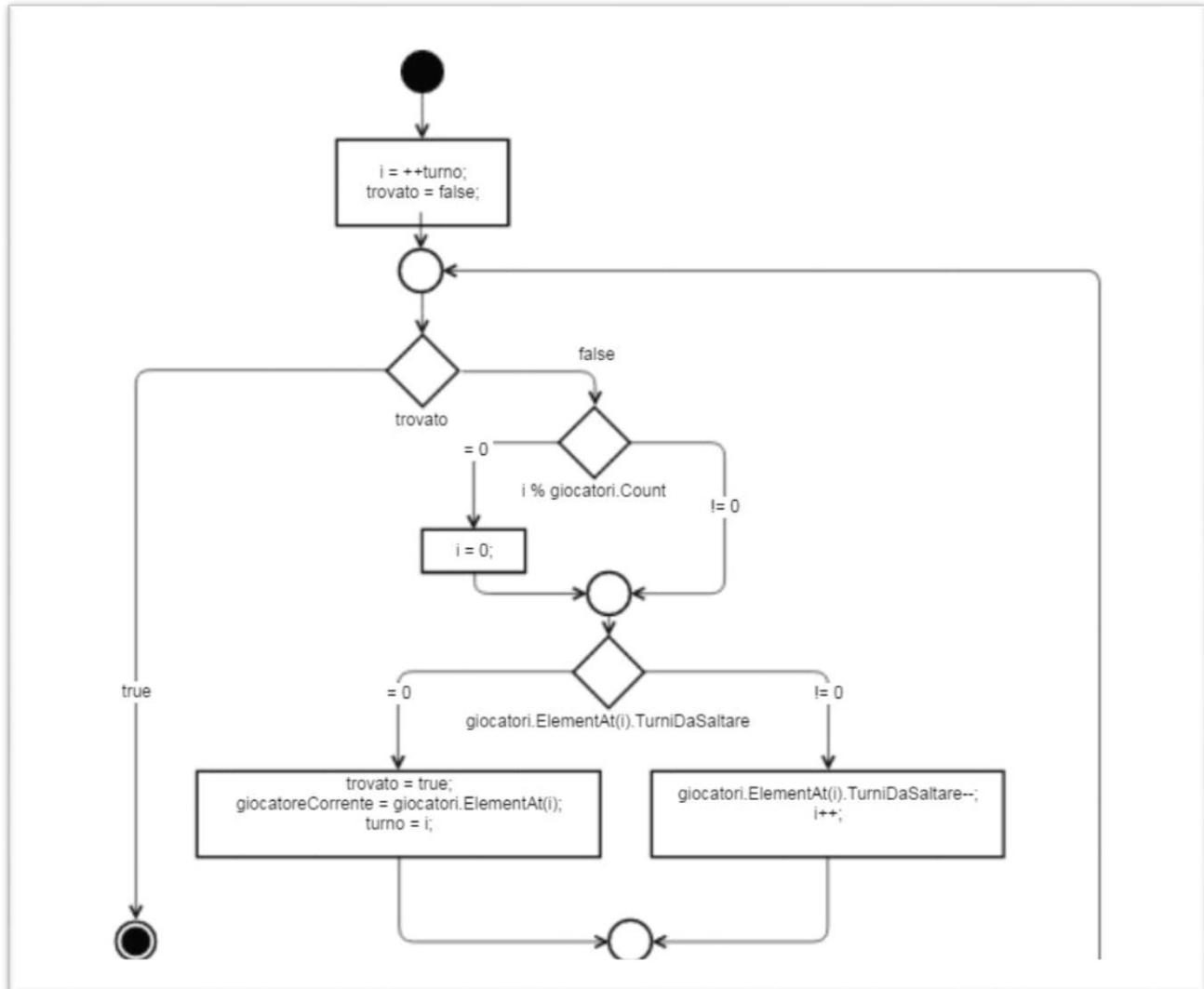


Diagramma di attività

Poiché l'algoritmo di cambio turno era particolarmente articolato, si è deciso di rappresentarlo sotto forma di diagramma di attività per meglio comprenderne il funzionamento.



4. IMPLEMENTAZIONE

Si riporta di seguito il codice dell'applicativo sviluppato.

LibreriaUnipoly.dll

1) Gioco

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //classe che rappresenta il gioco corrente
    public class Gioco
    {
        /*poiché viene usato il pattern Singleton, serve
         un attributo statico che rappresenti l'unica istanza*/
        private static Gioco istanza = null;

        public const int MIN_GIOCATORI = 2;      //numero minimo di giocatori
        public const int MAX_GIOCATORI = 4;       //numero massimo di giocatori
        public readonly TimeSpan tempoPartita;   //durata massima della partita
        private List<Casella> tabellone;        //lista di tutte le caselle
        private List<Carta> carteI;              //mazzo di carte Imprevisto
        private List<Carta> carteP;              //mazzo di carte Probabilità
        private List<Giocatore> giocatori;       //lista di giocatori
        private Dado dado;                      //dado
        private int turno;                      //turno corrente
        private Giocatore giocatoreCorrente;   //giocatore corrente
        private string infoGioco;               //informazioni di gioco
        private DateTime inizioPartita;         //data di inizio partita

        //costruttore protetto per implementare Singleton
        protected Gioco()
        {
            //inizializzazione degli attributi
            this,tempoPartita = new TimeSpan(2, 0, 0); //2 ore
            this,giocatori = new List<Giocatore>();
            this,dado = new LibreriaUnipoly.Dado(1,6); //dado a 6 facce
            this,infoGioco = null;
            this,tabellone = new List<Casella>();
            this,carteI = new List<Carta>();
            this,carteP = new List<Carta>();
        }

        //metodo statico da invocare per creare il gioco
        public static Gioco OttieniGioco()
        {
            if (istanza == null)
                istanza = new Gioco();
            return istanza;
        }

        //proprietà per accedere a tutte le proprietà e impostare le info di gioco
        public string InfoGioco { get { return this.infoGioco; } 
                                  set { this.infoGioco = value; } }
        public Giocatore GiocatoreCorrente { get { return this.giocatoreCorrente; } }
        public int NumCarteP { get { return this.carteP.Count; } }
        public int NumCarteI { get { return this.carteI.Count; } }
        public List<Giocatore> Giocatori { get { return this.giocatori; } }
```

```

public int Turno { get { return this.turno; } }

//metodo per iniziare una nuova partita.
//Restituisce true se l'avvio è andato a buon fine
public bool IniziaPartita()
{
    bool esito = false;
    //controlla se il numero di giocatori rientra nel range
    if (this.giocatori.Count >= MIN_GIOCATORI &&
        this.giocatori.Count <= MAX_GIOCATORI)
    {
        //inizializza parametri di gioco
        this.inizioPartita = new DateTime();
        this.inizioPartita = DateTime.Now;
        this.turno = 0;
        this.giocatoreCorrente = giocatori.ElementAt(0);

        //posiziona i giocatori sul Via
        foreach(Giocatore g in giocatori)
            g.Posizione = OttieniCasella(0);

        esito = true;
    }
    return esito;
}//end IniziaPartita()

//metodo per passare il turno al prossimo giocatore
//Restituisce true se la partita è ancora in corso
public bool CambiaTurno()
{
    bool esito = true;

    //se la partita è terminata, non si può passare il turno
    if (ControllaPartita() == false)
        esito = false;
    //altrimenti, se il giocatore ha lanciato il dado
    else if (giocatoreCorrente.DadoLanciato == true)
    {
        //si attua il passaggio
        this.giocatoreCorrente.DadoLanciato = false;
        //si incrementa il turno
        int i = ++turno;

        /*si aggiorna giocatore corrente cercando il primo non in stallo
         e aggiornando il numero di turni da saltare di quelli in stallo*/
        bool trovato = false;
        while (trovato == false)
        {
            //se tutti hanno giocato, si ritorna al primo
            if (i % this.giocatori.Count == 0)
                i = 0;

            if (this.giocatori.ElementAt(i).TurniDaSaltare == 0)
            {
                trovato = true;
                this.giocatoreCorrente = this.giocatori.ElementAt(i);
                turno = i;
            }
            else
            {
                infoGioco += this.giocatori.ElementAt(i).Nome +
                            " salta il turno.\r\n";
                this.giocatori.ElementAt(i).TurniDaSaltare--;
                i++;
            }
        }
    }
}

```

```

        }
    } //end while
} //end else if
//rimane il caso in cui il giocatore non abbia lanciato il dado
else
{
    //lanciamo un'eccezione
    infoGioco += "Lancia il dado prima\r\n";
    throw new Exception();
}

return esito;
}//end metodo CambiaTurno()

//metodo che restituisce true se la partita è ancora in corso
private bool ControllaPartita()
{
    bool esito = true;
    //controlla lo stato della partita
    foreach (Giocatore g in giocatori)
    {
        if (esito == true)
        {
            //se c'è almeno un laureato, allora la partita è finita
            if (g.Cfu >= 180)
            {
                infoGioco += "Il giocatore " + g.Nome + " si è laureato!\r\n";
                DecretaClassifica();
                esito = false;
            }
            //se sono trascorse almeno 2 ore, allora la partita è finita
            else if ((DateTime.Now - this.inizioPartita) >= tempoPartita)
            {
                infoGioco += "Tempo scaduto!\r\n";
                DecretaClassifica();
                esito = false;
            }
        }
    }
    return esito;
}//end metodo ControllaPartita()

/*metodo per aggiungere giocatori alla partita
restituisce il giocatore creato o null se la creazione
non è andata a buon fine*/
public Giocatore AggiungiGiocatore(string nome, string ruolo)
{
    Giocatore nuovoGiocatore = null;
    //controlla se il giocatore può essere aggiunto
    if(this.giocatori.Count < MAX_GIOCATORI)
    {
        nuovoGiocatore = new Giocatore(nome, ruolo, this);
        this.giocatori.Add(nuovoGiocatore);
    }
    return nuovoGiocatore;
}//end metodo AggiungiGiocatore()

//metodo che restituisce la casella all'indice in input
public Casella OttieniCasella(int indice)
{
    return this.tabellone.ElementAt(indice % this.tabellone.Count);
}

```

```

//metodo che restituisce l'indice della casella in input
public int OttieniIndice(Casella casella)
{
    return this.tabellone.IndexOf(casella);
}

//metodo per lanciare il dado e spostare il giocatore.
//Restituisce l'esito del dado se ha avuto successo, zero altrimenti
public int LanciaDadoSpostaGiocatore()
{
    int diffIndice = 0;      //esito del lancio
    /*se il dado non è già stato lanciato, il giocatore
     viene spostato di *diffIndice* caselle*/
    if (this.giocatoreCorrente.DadoLanciato == false)
    {
        diffIndice = this.dado.LanciaDado();
        this.giocatoreCorrente.Spostati(diffIndice);
        this.giocatoreCorrente.DadoLanciato = true;
    }
    else
    {
        infoGioco += "Non puoi rilanciare il dado.\r\n";
        throw new Exception();
    }
    return diffIndice;
}//end metodo LanciaDadoSpostaGiocatore()

// metodi per aggiungere carte, caselle, e leggere carte
public void AggiungiCartaP(Carta carta) { this.carteP.Add(carta); }
public void AggiungiCartaI(Carta carta) { this.carteI.Add(carta); }
public void AggiungiCasella(Casella casella) { this.tabellone.Add(casella); }
public Carta LeggiCartaP(int indice) { return this.carteP.ElementAt(indice-1); }
public Carta LeggiCartaI(int indice) { return this.carteI.ElementAt(indice-1); }

// metodo per stilare la classifica finale
public List<Giocatore> DecretaClassifica()
{
    List<Giocatore> classifica = new List<Giocatore>();
    //tutti i giocatori vengono aggiunti in classifica
    foreach (Giocatore g in giocatori)
        classifica.Add(g);
    //la classifica viene ordinata prima secondo i cfu, poi secondo i soldi
    classifica = classifica.OrderByDescending(Giocatore =>
        Giocatore.Cfu).ThenByDescending(Giocatore => Giocatore.Soldi).ToList();
    infoGioco += "Classifica:" + "\r\n";
    int i = 1;
    foreach (Giocatore giocatore in classifica)
    {
        infoGioco += i + ") " + giocatore.Nome + "\r\n";
        i++;
    }
    return classifica;
}//end metodo DecretaClassifica()
}//end classe
}

```

2) Giocatore

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //classe che rappresenta i giocatori
    public class Giocatore
    {
        //attributi
        private readonly string nome; //nome
        private readonly string ruolo; /*ruolo del giocatore:
                                        può essere in sede, fuori sede,
                                        straniero, pendolare*/
        private double soldi; //capitale
        private int cfu; //cfu acquisiti
        private int turniDaSaltare; //numero di turni da saltare
        private Casella posizione; //posizione corrente
        private Gioco gioco; //gioco a cui si riferisce
        private bool dadoLanciato; /*booleano che indica se il
                                    Giocatore ha già lanciato il dado*/
        public const double CAPITALE_INIZIALE = 2000; //capitale di partenza

        //costruttore
        public Giocatore(string nome, string ruolo, Gioco gioco)
        {
            //inizializzazione degli attributi
            this.nome = nome;
            this.ruolo = ruolo;
            this.soldi = CAPITALE_INIZIALE;
            this.cfu = 0;
            this.turniDaSaltare = 0;
            this.gioco = gioco;
            this.dadoLanciato = false;
        } //end costruttore

        //proprietà read-only per accedere agli attributi
        public string Nome { get { return this.nome; } }
        public string Ruolo { get { return this.ruolo; } }

        //proprietà per leggere e aggiornare gli altri attributi

        public bool DadoLanciato
        {
            get { return this.dadoLanciato; }
            set { this.dadoLanciato = value; }
        }
        public double Soldi
        {
            get { return this.soldi; }
            set
            {
                this.soldi = value;
                //se è in rosso, gli si dimezzano i cfu
                if (this.soldi < 0)
                    this.cfu = this.cfu / 2;
            }
        }
        public int Cfу
        {
            get { return this.cfu; }
```

```

        set
    {
        //il numero di cfu non può essere negativo
        this.cfu = value;
        if (this.cfu < 0)
            this.cfu = 0;
    }
}
public int TurniDaSaltare
{
    get { return this.turniDaSaltare; }
    set
    {
        //il numero di turni non può essere negativo
        this.turniDaSaltare = value;
        if (this.turniDaSaltare < 0)
            this.turniDaSaltare = 0;
    }
}
public Casella Posizione
{
    get { return this.posizione; }
    set { this.posizione = value; }
}

//metodo per spostare il giocatore di *numero* caselle
public void Spostati(int numero)
{
    /*si calcola l'indice della posizione corrente, si aggiunge *numero* e
     si calcola la nuova posizione*/
    this.posizione = gioco.OttieniCasella(gioco.OttieniIndice(posizione) + numero);
    //si transita sulla casella
    this.posizione.Transita();
}

//metodo per spostare il giocatore alla casella in input
public void Spostati(Casella casella)
{
    //si aggiorna la posizione
    this.posizione = casella;
    //si transita sulla casella
    this.posizione.Transita();
}
}//end classe
}

```

3) Dado

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //classe che rappresenta il dado da lanciare
    public class Dado
    {
        //attributi
        private readonly int max;    //numero massimo che può uscire
        private readonly int min;    //numero minimo che può uscire
        private Random rnd;         //numero random

```

```

public Dado(int min, int max)
{
    //inizializzazione degli attributi
    this.max = max;
    this.min = min;
    this.rnd = new Random();
}

//metodo che implementa il lancio del dado e restituisce l'esito
public int LanciaDado()
{
    return this.rnd.Next(min, max + 1);
}

//proprietà read-only per accedere ai valori del dado
public int Max { get { return this.max; } }
public int Min { get { return this.min; } }
}
}

```

4) Configurazione

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //classe di configurazione delle caselle e delle carte del gioco
    public class Configurazione
    {
        private Gioco gioco;           //gioco a cui si riferisce

        //costruttore
        public Configurazione(Gioco gioco)
        {
            //inizializzazione dell'attributo
            this.gioco = gioco;
        }

        //metodo per configurare il tabellone
        public void CaricaTabellone()
        {
            //creazione delle caselle speciali con relative carte
            Carta c1 = new CartaSoldi(null, -500, this.gioco, "Prima rata! -500 euro");
            Casella cas_1 = new CasellaSpeciale("VIA", "Paghi una tassa", this.gioco, c1);
            Carta c2 = new CartaSoldi(null, -400, this.gioco, "Seconda rata! -400 euro");
            Casella cas_2 = new CasellaSpeciale("Seconda rata", "Paghi una tassa",
this.gioco, c2);
            Casella cas_3 = new CasellaTerzaRata("Terza rata", "Paghi in base al tuo
capitale", this.gioco);
            Casella cas_4_1 = new CasellaFuoriCorso("Fuori corso", "Perdi il 10% del tuo
capitale", this.gioco);
            Casella cas_4_2 = new CasellaFuoriCorso("Fuori corso", "Perdi il 10% del tuo
capitale", this.gioco);
            Carta c5 = new CartaSoldi(null, 500, this.gioco, "Borsa di studio! +500 euro");
            Casella cas_5 = new CasellaSpeciale("Borsa di studio", "Ottieni una borsa di
studio", this.gioco, c5);
            Carta c6 = new CartaStallo(null, 2, this.gioco, "Resta fermo due turni");
        }
    }
}

```

```

        Casella cas_6 = new CasellaSpeciale("Stallo", "Sei in stallo", this.gioco, c6);
        Carta c7 = new CartaMovimento(null, cas_6, this.gioco,
                                         "Vai in stallo: hai pagato in ritardo le
tasse");
        Casella cas_7 = new CasellaSpeciale("Vai in stallo", "Salvi alla casella di
stallo", this.gioco, c7);
        Carta c8 = new CartaStallo(null, 1, this.gioco, "Resta fermo un turno");
        Casella cas_8_1 = new CasellaSpeciale("Vacanza pasquale", "Stallo per un turno",
this.gioco, c8);
        Casella cas_8_2 = new CasellaSpeciale("Vacanza di Natale ", "Stallo per un
turno", this.gioco, c8);
        Casella cas_8_3 = new CasellaSpeciale("Vacanza estiva", "Stallo per un turno",
this.gioco, c8);
        Carta c9 = new CartaCfu(null, -24, this.gioco, "Perdi 24 cfu per il giovedì
universitario");
        Casella cas_9 = new CasellaSpeciale("Giovedì universitario", "Perdi il
libretto", this.gioco, c9);
        Carta c10 = new CartaCfu(null, 24, this.gioco, "Guadagni 24 cfu per l'Erasmus");
        Casella cas_10 = new CasellaSpeciale("Erasmus", "Bonus per l'erasmus",
this.gioco, c10);

        //creazione delle carte probabilità
        this.gioco.AaggiungiCartaP(new CartaCfu(null, 30, this.gioco, "Seminario IBM ->
+30 cfu"));
        this.gioco.AaggiungiCartaP(new CartaCfu(null, 20, this.gioco, "Seminario
intelligenza artificiale -> +20 cfu"));
        this.gioco.AaggiungiCartaP(new CartaCfu("in sede", 15, this.gioco, "Fai l'ECDL
(se sei studente in sede) -> +15 cfu"));
        this.gioco.AaggiungiCartaP(new CartaCfu("straniero", 12, this.gioco,
                                         "Convalida esami sostenuti all'estero (se sei straniero)
-> + 12 cfu"));
        this.gioco.AaggiungiCartaP(new CartaCfu(null, 15, this.gioco, "Workshop -> +15
cfu"));
        this.gioco.AaggiungiCartaP(new CartaSoldi(null, 200, this.gioco, "Regalo dei
parenti -> +200 euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi(null, 400, this.gioco, "Borsa di studio
per merito -> +400 euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi("pendolare", 50, this.gioco,
                                         "Trovi dei soldi a terra alla stazione (se sei pendolare) ->
+50 euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi("fuori sede", 150, this.gioco,
                                         "Tua nonna pensa che non stai mangiando abbastanza e " +
                                         "ti fa una ricarica (se sei fuori sede) -> +150 euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi("straniero", 200, this.gioco,
                                         "Fai lezioni di lingua ai tuoi colleghi (se sei studente
erasmus) -> +200 euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi(null, 50, this.gioco,
                                         "Fai ripetizioni agli studenti delle superiori -> +50
euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi(null, 75, this.gioco, "Sviluppi un'app
di successo -> +75 euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi(null, 20, this.gioco, "Ripari il pc di
un amico -> +20 euro"));
        this.gioco.AaggiungiCartaP(new CartaSoldi(null, 300, this.gioco, "Vinci alla
lotteria -> +300 euro"));
        this.gioco.AaggiungiCartaP(new CartaMovimento(null, cas_5, this.gioco, "Vai sulla
casella borsa di studio"));
        this.gioco.AaggiungiCartaP(new CartaMovimento(null, cas_10, this.gioco, "Vai in
Erasmus"));

```

```

//creazione delle carte imprevisti
this.gioco.AaggiungiCartaI(new CartaCfu(null, -15, this.gioco,
                                         "Uno dei professori è tuo vicino di casa e ti odia -> -15
cfu"));
this.gioco.AaggiungiCartaI(new CartaCfu(null, -16, this.gioco,
                                         "Perdi il libretto universitario -> -16 CFU"));
this.gioco.AaggiungiCartaI(new CartaCfu(null, -10, this.gioco,
                                         "Copi il progetto da internet -> -10 cfu"));
this.gioco.AaggiungiCartaI(new CartaCfu(null, -12, this.gioco,
                                         "C'è un bug nel progetto che non riesci a sistemare -> -
12 cfu"));
this.gioco.AaggiungiCartaI(new CartaSoldi("pendolare", -50, this.gioco,
                                         "Perdi dei soldi alla stazione (se sei pendolare) -> -50
euro"));
this.gioco.AaggiungiCartaI(new CartaSoldi("fuori sede", -150, this.gioco,
                                         "Torni a casa un weekend (se sei fuori sede) -> -150
euro"));
this.gioco.AaggiungiCartaI(new CartaSoldi("straniero", -200, this.gioco, "Se sei
studente erasmus -> -200 euro"));
this.gioco.AaggiungiCartaI(new CartaSoldi(null, -100, this.gioco,
                                         "La sessione ti fa impazzire e devi andare dallo psichiatra
-> -100 euro"));
this.gioco.AaggiungiCartaI(new CartaSoldi(null, -10, this.gioco, "Ti ammali e
devi pagare il ticket -> -10 euro"));
this.gioco.AaggiungiCartaI(new CartaSoldi(null, -70, this.gioco,
                                         "Vieni derubato durante il giovedì universitario -> -70
euro"));
this.gioco.AaggiungiCartaI(new CartaSoldi(null, -30, this.gioco,
                                         "Si laurea un tuo collega e devi fargli il regalo -> -30
euro"));
this.gioco.AaggiungiCartaI(new CartaMovimento(null, cas_4_1, this.gioco, "Vai
sulla casella fuori corso"));
this.gioco.AaggiungiCartaI(new CartaMovimento(null, cas_8_1, this.gioco, "Vai
sulla casella vacanza pasquale"));
this.gioco.AaggiungiCartaI(new CartaMovimento(null, cas_8_3, this.gioco, "Vai
sulla casella vacanza estiva"));
this.gioco.AaggiungiCartaI(new CartaMovimento(null, cas_8_2, this.gioco, "Vai
sulla casella vacanza di Natale"));
this.gioco.AaggiungiCartaI(new CartaMovimento(null, cas_6, this.gioco, "Vai in
stallo"));

//caselle imprevisti e probabilità
Casella cas1 = new CasellaCarta("Imprevisto", "Pesca una carta imprevisto.",
this.gioco);
Casella cas2 = new CasellaCarta("Imprevisto", "Pesca una carta imprevisto.",
this.gioco);
Casella cas3 = new CasellaCarta("Imprevisto", "Pesca una carta imprevisto.",
this.gioco);
Casella cas4 = new CasellaCarta("Probabilità", "Pesca una carta probabilità.",
this.gioco);
Casella cas5 = new CasellaCarta("Probabilità", "Pesca una carta probabilità.",
this.gioco);
Casella cas6 = new CasellaCarta("Probabilità", "Pesca una carta probabilità.",
this.gioco);

//creazione delle aree formative
AreaFormativa af1 = new AreaFormativa("Formazione matematico-fisica");
AreaFormativa af2 = new AreaFormativa("Formazione informatica di base");
AreaFormativa af3 = new AreaFormativa("Discipline Informatiche");
AreaFormativa af4 = new AreaFormativa("Programmazione");
AreaFormativa af5 = new AreaFormativa("Esami di curriculum");
AreaFormativa af6 = new AreaFormativa("Esami a Scelta");
AreaFormativa af7 = new AreaFormativa("Tesi");
AreaFormativa af8 = new AreaFormativa("Lingua Inglese");

```

```

        AreaFormativa af9 = new AreaFormativa("Tirocini formativi");

        //creazione delle caselle che rappresentano gli esami
        CasellaEsame cas7 = new CasellaEsame("Analisi Matematica", "Casella esame", 12,
this.gioco, af1);
        CasellaEsame cas8 = new CasellaEsame("Matematica Discreta", "Casella esame", 6,
this.gioco, af1);
        CasellaEsame cas9 = new CasellaEsame("Fisica I", "Casella esame", 6, this.gioco,
af1);
        CasellaEsame cas10 = new CasellaEsame("Fisica II", "Casella esame", 6,
this.gioco, af1);
        CasellaEsame cas11 = new CasellaEsame("Probabilità e Statistica", "Casella
esame", 6, this.gioco, af1);
        CasellaEsame cas12 = new CasellaEsame("Algoritmi e Strutture Dati", "Casella
esame", 12, this.gioco, af2);
        CasellaEsame cas13 = new CasellaEsame("Architettura degli Elaboratori", "Casella
esame", 12, this.gioco, af2);
        CasellaEsame cas14 = new CasellaEsame("Sistemi Operativi", "Casella esame", 12,
this.gioco, af3);
        CasellaEsame cas15 = new CasellaEsame("Basi di Dati", "Casella esame", 12,
this.gioco, af3);
        CasellaEsame cas16 = new CasellaEsame("Reti di Calcolatori", "Casella esame",
12, this.gioco, af3);
        CasellaEsame cas17 = new CasellaEsame("Programmazione Procedurale e Logica",
"Casella esame", 12, this.gioco, af4);
        CasellaEsame cas18 = new CasellaEsame("Programmazione a Oggetti e Ingegneria del
Software", "Casella esame", 12, this.gioco, af4);
        CasellaEsame cas19 = new CasellaEsame("Linguaggi di Programmazione e Verifica
del Software", "Casella esame", 12, this.gioco, af4);
        CasellaEsame cas20 = new CasellaEsame("Elaborazione di Segnali e Immagini",
"Casella esame", 12, this.gioco, af5);
        CasellaEsame cas21 = new CasellaEsame("Simulazione Numerica", "Casella esame",
6, this.gioco, af5);
        CasellaEsame cas22 = new CasellaEsame("Economia e Gestione delle Imprese",
"Casella esame", 6, this.gioco, af6);
        CasellaEsame cas23 = new CasellaEsame("Piattaforme Digitali per la Gestione del
Territorio ", "Casella esame", 6, this.gioco, af6);
        CasellaEsame cas24 = new CasellaEsame("Tesi", "Casella esame", 6, this.gioco,
af7);
        CasellaEsame cas25 = new CasellaEsame("Lingua Inglese", "Casella esame", 6,
this.gioco, af8);
        CasellaEsame cas26 = new CasellaEsame("Tirocinio in azienda", "Casella esame",
3, this.gioco, af9);
        CasellaEsame cas27 = new CasellaEsame("Tirocinio in università", "Casella
esame", 3, this.gioco, af9);

        //inserimento degli esami nelle aree formative
af1.AggiungiEsame(cas7);
af1.AggiungiEsame(cas8);
af1.AggiungiEsame(cas9);
af1.AggiungiEsame(cas10);
af1.AggiungiEsame(cas11);
af2.AggiungiEsame(cas12);
af2.AggiungiEsame(cas13);
af3.AggiungiEsame(cas14);
af3.AggiungiEsame(cas15);
af3.AggiungiEsame(cas16);
af4.AggiungiEsame(cas17);
af4.AggiungiEsame(cas18);
af4.AggiungiEsame(cas19);
af5.AggiungiEsame(cas20);
af5.AggiungiEsame(cas21);
af6.AggiungiEsame(cas22);
af6.AggiungiEsame(cas23);

```

```

af7.AggiungiEsame(cas24);
af8.AggiungiEsame(cas25);
af9.AggiungiEsame(cas26);
af9.AggiungiEsame(cas27);

//inserimento delle caselle nel tabellone
this.gioco.AggiungiCasella(cas_1);
this.gioco.AggiungiCasella(cas8);
this.gioco.AggiungiCasella(cas_5);
this.gioco.AggiungiCasella(cas7);
this.gioco.AggiungiCasella(cas4);
this.gioco.AggiungiCasella(cas_8_2);
this.gioco.AggiungiCasella(cas12);
this.gioco.AggiungiCasella(cas1);
this.gioco.AggiungiCasella(cas13);
this.gioco.AggiungiCasella(cas22);
this.gioco.AggiungiCasella(cas_6);
this.gioco.AggiungiCasella(cas23);
this.gioco.AggiungiCasella(cas_10);
this.gioco.AggiungiCasella(cas20);
this.gioco.AggiungiCasella(cas21);
this.gioco.AggiungiCasella(cas_9);
this.gioco.AggiungiCasella(cas11);
this.gioco.AggiungiCasella(cas_2);
this.gioco.AggiungiCasella(cas9);
this.gioco.AggiungiCasella(cas10);
this.gioco.AggiungiCasella(cas_8_1);
this.gioco.AggiungiCasella(cas17);
this.gioco.AggiungiCasella(cas2);
this.gioco.AggiungiCasella(cas18);
this.gioco.AggiungiCasella(cas19);
this.gioco.AggiungiCasella(cas_4_1);
this.gioco.AggiungiCasella(cas15);
this.gioco.AggiungiCasella(cas14);
this.gioco.AggiungiCasella(cas5);
this.gioco.AggiungiCasella(cas16);
this.gioco.AggiungiCasella(cas_7);
this.gioco.AggiungiCasella(cas_3);
this.gioco.AggiungiCasella(cas26);
this.gioco.AggiungiCasella(cas_8_3);
this.gioco.AggiungiCasella(cas27);
this.gioco.AggiungiCasella(cas6);
this.gioco.AggiungiCasella(cas3);
this.gioco.AggiungiCasella(cas25);
this.gioco.AggiungiCasella(cas_4_2);
this.gioco.AggiungiCasella(cas24);

}//end CaricaTabellone()
}//end classe
}

```

5) Casella

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //classe astratta da cui erediteranno tutte le caselle
    public abstract class Casella
    {
        //attributi

        protected string nome;          //nome della casella
        protected string istruzioni;    //dettagli della casella
        protected Gioco gioco;          //gioco a cui si riferisce

        /*alcune caselle hanno una carta associata che descrive le
         conseguenze del transito*/
        protected Carta carta;
        /*nel caso in cui la casella sia una CasellaCarta,
         va specificato se la carta è imprevisto o probabilità*/
        protected string tipoCarta;
        /*nel caso in cui la casella rappresenti un esame,
         va specificata l'area formativa dell'esame*/
        protected AreaFormativa areaFormativa;

        //proprietà read-only per accedere agli attributi
        public string Nome { get { return this.nome; } }
        public string Istruzioni { get { return this.istruzioni; } }
        public Carta Carta { get { return this.carta; } }
        public string TipoCarta { get { return this.tipoCarta; } }
        public AreaFormativa AreaFormativa { get { return this.areaFormativa; } }

        //metodo astratto che attua le conseguenze del transito
        public abstract void Transita();

        /*metodo virtuale per comprare la casella.
         Solo le caselle che faranno l'overriding potranno implementare
         l'acquisto, le altre lanceranno un'eccezione*/
        public virtual bool Compra()
        {
            gioco.InfoGioco += "Azione non consentita.\r\n";
            throw new Exception();
        }

        /*metodo virtuale per aggiungere dei potenziamenti alla casella.
         Solo le caselle che faranno l'overriding potranno implementare
         il potenziamento, le altre lanceranno un'eccezione*/
        public virtual bool Potenzia()
        {
            gioco.InfoGioco += "Azione non consentita.\r\n";
            throw new Exception();
        }
    }//end classe
}
```

6) CasellaEsame

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    /*classe che rappresenta una casella di tipo esame. È una sottoclasse di Casella*/
    public class CasellaEsame : Casella
    {
        //attributi

        private readonly int pesoCfu;           //cfu dell'esame
        private readonly double prezzoLibro;    //prezzo del libro richiesto
        private const double PREZZO_APPUNTI = 20; //prezzo degli appunti associati
        private int appunti;                   //numero di appunti associati
        private Giocatore proprietario;        //proprietario del libro

        //costruttore
        public CasellaEsame(string nome, string istruzioni, int pesoCfu,
                            Gioco gioco, AreaFormativa areaFormativa)
        {
            //inizializzazione degli attributi ereditati
            this.nome = nome;
            this.istruzioni = istruzioni;
            this.gioco = gioco;
            this.areaFormativa = areaFormativa;

            //inizializzazione degli attributi aggiunti
            this.pesoCfu = pesoCfu;
            this.prezzoLibro = this.pesoCfu * 10;    //il prezzo del libro dipende dall'esame
            this.appunti = 0;
            this.proprietario = null;
        } //end costruttore

        //proprietà per leggere e aggiornare il proprietario della casella
        public Giocatore Proprietario
        {
            get { return this.proprietario; }
            set { this.proprietario = value; }
        }

        //proprietà read-only per accedere agli altri attributi
        public double PrezzoLibro { get { return this.prezzoLibro; } }
        public int PesoCfu { get { return this.pesoCfu; } }
        public int Appunti { get { return this.appunti; } }

        /*metodo per comprare il libro richiesto e diventare proprietari
        della casella. Restituisce true se l'acquisto ha avuto successo*/
        public override bool Compra()
        {
            bool comprato = false;      //esito dell'acquisto

            //se il giocatore ha abbastanza soldi e la casella non è di nessuno
            if (gioco.GiocatoreCorrente.Soldi >= prezzoLibro && proprietario == null)
            {
                //pagamento e aggiornamento delle informazioni
                gioco.GiocatoreCorrente.Soldi -= prezzoLibro;
                this.proprietario = gioco.GiocatoreCorrente;
                this.proprietario.Cfu += this.pesoCfu;
                comprato = true;
                gioco.InfoGioco += "Libro comprato!\r\n";
            }
        }
    }
}
```

```

        }
    else if (this.proprietario != null)
    {
        //se la casella ha già un altro proprietario
        if (this.proprietario != gioco.GiocatoreCorrente)
            gioco.InfoGioco += "Non puoi comprare il libro perché"
                + " è posseduto già da " + proprietario.Nome + "\r\n";
        //se il proprietario della casella è il giocatore stesso
        else
            gioco.InfoGioco += "Possiedi già quest'esame.\r\n";
    }
    //rimane il caso in cui il giocatore non abbia abbastanza soldi
    else
        gioco.InfoGioco += "Non hai abbastanza soldi, vendi un rene.\r\n";

    return comprato;
} //end metodo Compra()

//metodo che implementa il transito sulla casella
public override void Transita()
{
    //aggiornamento delle informazioni di gioco
    if (proprietario != null)
        gioco.InfoGioco += "Casella posseduta da " + proprietario.Nome + ".\r\n";
    /*se la casella è proprietà di un altro giocatore */
    if (this.proprietario != null && gioco.GiocatoreCorrente != this.proprietario )
    {
        //il giocatore paga al proprietario sia il libro sia gli appunti
        gioco.GiocatoreCorrente.Soldi -= this.prezzoLibro +
            this.appunti * PREZZO_APPUNTI;
        this.proprietario.Soldi += this.prezzoLibro + this.appunti * PREZZO_APPUNTI;
        gioco.InfoGioco += "Devi pagare " + (this.prezzoLibro +
            this.appunti * PREZZO_APPUNTI) + ".\r\n";
    }
    //se il giocatore possiede quella casella
    else if (gioco.GiocatoreCorrente == this.proprietario)
        gioco.InfoGioco += "Possiedi quest'esame nel libretto.\r\n";
    //resta il caso in cui la casella non abbia proprietario
    else
        gioco.InfoGioco += "Puoi comprare il libro a "
            + this.prezzoLibro + " euro, se vuoi.\r\n";
} //end metodo Transita()

//metodo che implementa l'acquisto dei potenziamenti: in questo caso, gli appunti
public override bool Potenzia()
{
    bool aggiunti = false;      //esito del potenziamento

    //si controlla se il giocatore è proprietario di tutte le caselle di quell'area
    bool esito = areaFormativa.AcquisitaDa(gioco.GiocatoreCorrente);

    //se lo è, e ha abbastanza soldi per comprare gli appunti, vengono aggiunti
    if (esito == true && gioco.GiocatoreCorrente.Soldi >= PREZZO_APPUNTI)
    {
        gioco.GiocatoreCorrente.Soldi -= PREZZO_APPUNTI;
        this.appunti++;
        aggiunti = true;
        gioco.InfoGioco += "Appunti aggiunti!\r\n";
    }
    //se non ha tutti gli esami dell'area formativa
    else if (aggiunti == false)
        gioco.InfoGioco += "Non puoi aggiungere appunti perché "
            + "non possiedi tutti gli esami dell'area formativa\r\n";
    //rimane il caso in cui non abbia abbastanza soldi
}

```

```

        else
            gioco.InfoGioco += "Non hai abbastanza soldi, vendi un rene.\r\n";
    }

    return aggiunti;
}//end metodo Potenzia()

//metodo che ridefinisce la descrizione della classe
public override string ToString()
{
    string descrizione;
    descrizione = "Tipo: " + this.istruzioni + "\r\n" +
                  "Area formativa: " + this.areaFormativa.Nome + "\r\n" +
                  "CFU: " + this.pesoCfu + "\r\n" +
                  "Proprietario: ";
    if (proprietario == null)
        descrizione += "nessuno" + "\r\n";
    else
        descrizione += this.proprietario.Nome + "\r\n";

    descrizione += "Prezzo libro: " + this.prezzoLibro + "\r\n" +
                  "Prezzo appunti: " + PREZZO_APPUNTI + "\r\n" +
                  "Numero appunti associati: " + this.appunti + "\r\n";

    return descrizione;
} //end metodo ToString()
}//end classe
}

```

7) CasellaSpeciale

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //sottoclasse di Casella con conseguenze di transito particolari
    public class CasellaSpeciale : Casella
    {
        //costruttore
        public CasellaSpeciale(string nome, string istruzioni, Gioco gioco, Carta carta)
        {
            //inizializzazione degli attributi ereditati
            this.nome = nome;
            this.carta = carta;
            this.istruzioni = this.carta.Descrizione;
            this.gioco = gioco;
        }

        //metodo che implementa il transito sulla casella
        public override void Transita()
        {
            //vengono eseguite le istruzioni della carta associata
            this.carta.Usa();
        }

        //metodo che ridefinisce la descrizione della classe
        public override string ToString()
        {
            return this.istruzioni;
        }
    }
}

```

```
    }
}
```

8) CasellaCarta

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //sottoclasse di Casella che rappresenta le caselle Imprevisti e Probabilità
    public class CasellaCarta : Casella
    {
        //costruttore
        public CasellaCarta(string nome, string istruzioni, Gioco gioco)
        {
            //inizializzazione degli attributi ereditati
            this.nome = nome;
            this.istruzioni = istruzioni;
            this.gioco = gioco;
        }

        //metodo che implementa il transito sulla casella
        public override void Transita()
        {
            //si pesca una carta
            Carta c;

            //si pesca dal mazzo degli imprevisti
            if (this.nome.Equals("Imprevisto"))
            {
                Dado mioDado = new Dado(0, this.gioco.NumCarteI - 1);
                int esito = mioDado.LanciaDado();
                c = this.gioco.LeggiCartaI(esito);
                base.carta = c;
                base.tipoCarta = "Imprevisto";
            }
            //si pesca dal mazzo delle probabilità
            else
            {
                Dado mioDado = new Dado(0, this.gioco.NumCarteP - 1);
                int esito = mioDado.LanciaDado();
                c = this.gioco.LeggiCartaP(esito);
                base.carta = c;
                base.tipoCarta = "Probabilità";
            }
            //si eseguono le istruzioni riportate
            c.Usa();
        }//end classe Transita()

        //metodo che ridefinisce la descrizione della classe
        public override string ToString()
        {
            return this.istruzioni;
        }
    }//end classe
}
```

9) CasellaFuoriCorso

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    /*classe che rappresenta una sottoclasse di Casella*/
    public class CasellaFuoriCorso : Casella
    {
        //costruttore
        public CasellaFuoriCorso(string nome, string istruzioni, Gioco gioco)
        {
            //inizializzazione degli attributi
            this.nome = nome;
            this.istruzioni = istruzioni;
            this.gioco = gioco;
        }

        //metodo che implementa il transito sulla casella
        public override void Transita()
        {
            //se si va fuori corso, il proprio capitale viene ridotto del 10%
            gioco.GiocatoreCorrente.Soldi -=
                ((10 * gioco.GiocatoreCorrente.Soldi) / 100);
        }

        //metodo che ridefinisce la descrizione della classe
        public override string ToString()
        {
            return this.istruzioni;
        }
    }
}

```

10) CasellaTerzaRata

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    /*classe che rappresenta una sottoclasse di Casella*/
    public class CasellaTerzaRata : Casella
    {
        //costruttore
        public CasellaTerzaRata(string nome, string istruzioni, Gioco gioco)
        {
            //inizializzazione degli attributi ereditati
            this.nome = nome;
            this.istruzioni = istruzioni;
            this.gioco = gioco;
        }

        //metodo che implementa il transito sulla casella
        public override void Transita()
        {
            //la terza rata si paga in base al proprio capitale
        }
}

```

```

        gioco.GiocatoreCorrente.Soldi -=
            (100 + this.gioco.GiocatoreCorrente.Soldi / 3);
    }

    //metodo che ridefinisce la descrizione della classe
    public override string ToString()
    {
        return this.istruzioni;
    }
}
}

```

11) Carta

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //classe astratta da cui erediteranno tutte le carte
    public abstract class Carta
    {
        //attributi

        /*alcune carte hanno effetto solo su giocatori con un
        determinato ruolo*/
        protected string ruolo;
        /*alcune carte comportano una variazione o di cfu o di soldi
        o di turni da saltare*/
        protected double valore;
        /*alcune carte comportano uno spostamento del giocatore fino
        ad una determinata casella*/
        protected Casella casella;

        protected string descrizione;    //descrizione della carta
        protected Gioco gioco;          //gioco a cui si riferisce

        //proprietà read-only per accedere a tutti gli attributi
        public string Ruolo { get { return this.ruolo; } }
        public double Valore { get { return this.valore; } }
        public Casella Casella { get { return this.casella; } }
        public string Descrizione{ get { return this.descrizione; } }

        //metodo astratto che attua le istruzioni della carta
        abstract public void Usa();
    }
}

```

12) CartaSoldi

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    /*classe che rappresenta le carte che variano il capitale
    dei giocatori che le pescano. È una sottoclassificazione di Carta */
    public class CartaSoldi : Carta
    {
        //costruttore
        public CartaSoldi(string ruolo, double valore, Gioco gioco, string descrizione)
        {
            //inizializzazione degli attributi ereditati
            this.ruolo = ruolo;
            this.valore = valore;
            this.gioco = gioco;
            this.descrizione = descrizione;
        }

        //metodo che implementa il comportamento della carta
        public override void Usa()
        {
            /*se la carta agisce su tutti i ruoli o se il ruolo del giocatore
            coincide con quello richiesto dalla carta, allora si incrementa
            (o si decrementa, nel caso in cui il valore sia negativo) il
            capitale del giocatore*/
            if (ruolo == null || gioco.GiocatoreCorrente.Ruolo.Equals(ruolo))
                gioco.GiocatoreCorrente.Soldi += this.valore;
        }
    }
}
```

13) CartaMovimento

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    /*classe che rappresenta le carte che determinano uno spostamento
    del giocatore. È una sottoclassificazione di Carta */
    public class CartaMovimento : Carta
    {
        //costruttore
        public CartaMovimento(string ruolo, Casella casella,
                               Gioco gioco, string descrizione)
        {
            //inizializzazione degli attributi ereditati
            this.ruolo = ruolo;
            this.casella = casella;
            this.gioco = gioco;
            this.descrizione = descrizione;
        }

        //metodo che implementa il comportamento della carta
        public override void Usa()
    }
```

```

    {
        /*se la carta agisce su tutti i ruoli o se il ruolo del giocatore
        coincide con quello richiesto dalla carta, allora il giocatore
        viene spostato sulla casella indicata e viene aggiornata la
        sua posizione*/
        if (ruolo == null || gioco.GiocatoreCorrente.Ruolo.Equals(ruolo))
        {
            this.casella.Transita();
            gioco.GiocatoreCorrente.Posizione = this.casella;
        }
    }
}

```

14) CartaCfu

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    /*classe che rappresenta le carte che variano il numero di cfu
    dei giocatori che le pescano. È una sottoclasse di Carta */
    public class CartaCfu : Carta
    {
        //costruttore
        public CartaCfu(string ruolo, double valore, Gioco gioco, string descrizione)
        {
            //inizializzazione degli attributi ereditati
            this.ruolo = ruolo;
            this.valore = valore;
            this.gioco = gioco;
            this.descrizione = descrizione;
        }

        //metodo che implementa il comportamento della carta
        public override void Usa()
        {
            /*se la carta agisce su tutti i ruoli o se il ruolo del giocatore
            coincide con quello richiesto dalla carta, allora si incrementa
            (o si decrementa, nel caso in cui il valore sia negativo) il
            numero di cfu del giocatore*/
            if (ruolo == null || gioco.GiocatoreCorrente.Ruolo.Equals(ruolo))
                gioco.GiocatoreCorrente.Cfu += (int)this.valore;
        }
    }
}

```

15) CartaStallo

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    /*classe che rappresenta le carte che fanno saltare dei turni al
     giocatore. È una sottoclasse di Carta */
    public class CartaStallo : Carta
    {
        //costruttore
        public CartaStallo(string ruolo, double valore, Gioco gioco, string descrizione)
        {
            //inizializzazione degli attributi ereditati
            this.ruolo = ruolo;
            this.valore = valore;
            this.gioco = gioco;
            this.descrizione = descrizione;
        }

        //metodo che implementa il comportamento della carta
        public override void Usa()
        {
            /*se la carta agisce su tutti i ruoli o se il ruolo del giocatore
             coincide con quello richiesto dalla carta, allora si imposta
             il numero di turni da saltare*/
            if (ruolo == null || gioco.GiocatoreCorrente.Ruolo.Equals(ruolo))
            {
                gioco.GiocatoreCorrente.TurniDaSaltare = (int)this.valore;
            }
        }
    }
}

```

16) AreaFormativa

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibreriaUnipoly
{
    //classe che raggruppa diversi esami per tipologia
    public class AreaFormativa
    {
        //attributi
        private readonly string nome;      //nome dell'area formativa
        private List<CasellaEsame> esami; //lista di esami appartenenti all'area

        //costruttore
        public AreaFormativa(string nome)
        {
            //inizializzazione degli attributi
            this.nome = nome;
            this.esami = new List<CasellaEsame>();
        }
    }
}

```

```
//proprietà read-only per accedere al nome
public string Nome { get { return this.nome; } }

//metodo per aggiungere esami all'area formativa
public void AggiungiEsame(CasellaEsame esame)
{
    this.esami.Add(esame);
}

/*metodo che restituisce true se il giocatore in input
possiede tutti gli esami dell'area formativa*/
public bool AcquisitaDa(Giocatore giocatore)
{
    bool acquisita = false;
    int esamiAcquisiti = 0;

    //ciclo per contare il numero di esami dell'area posseduti
    foreach (CasellaEsame c in esami)
    {
        if (c.Proprietario == giocatore)
            esamiAcquisiti++;
    }

    /*se il numero di esami posseduti coincide col numero di
    esami in lista, viene restituito true */
    if (esamiAcquisiti == esami.Count)
        acquisita = true;

    return acquisita;
} //end metodo
} //end classe
```

UnipolyGUI

Nota: non vengono riportati i frammenti di codice generati automaticamente dall'ambiente di sviluppo (cioè quelli relativi alle classi parziali Designer).

17) TavoloGioco

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

//si importa la libreria del gioco
using LibreriaUnipoly;

namespace UnipolyGUI
{
    //classe di tipo Form che mostra il tavolo da gioco
    public partial class TavoloGioco: Form
    {
        //attributi
        private Gioco gioco;                                //gioco a cui ci si riferisce
        private Configurazione config;                      //configurazione del gioco
        private Casella corrente;                           //casella corrente
        private Pedina pedina;                             //pedina corrente
        private TimeSpan tempo;                            //tempo rimanente
        private PictureBox[] immaginiPedine;              //immagini delle pedine
        private PictureBox[] segnaposti;                  //segnaposti accanto ai giocatori
        private Color[] coloriPerGiocatore;               //colori delle pedine
        private Dictionary<string, Color> colorePerArea; //colori delle aree formative
        private Bitmap[] facceDado;                       //facce del dado
        private Label[] nomi;                             //nomi dei giocatori
        private TextBox[] dati;                           //dati dei giocatori
        private Button[] pulsanti;                        //pulsanti di gioco

        //costruttore
        public TavoloGioco()
        {
            //inizializzazione dei componenti
            InitializeComponent();
        }

        //gestore dell'evento di caricamento del form
        private void load(object sender, EventArgs e)
        {
            //creazione del gioco
            gioco = Gioco.OttieniGioco();
            config = new Configurazione(gioco);
            config.CaricaTabellone();
            //inizializzazione degli attributi
            tempo = gioco[tempoPartita];
            pedina = new Pedina(this, gioco);
            coloriPerGiocatore = new Color[] { Color.Red, Color.Blue,
                                              Color.Green, Color.Yellow };
            colorePerArea = new Dictionary<string, Color>()
            {
                {"Formazione matematico-fisica", Color.MediumOrchid },
                {"Formazione informatica di base", Color.SeaGreen },
                {"Discipline Informatiche", Color.Olive },
                {"Programmazione", Color.Red },
            }
        }
    }
}
```

```

        { "Esami di curriculum", Color.DarkOrange },
        { "Esami a Scelta", Color.Brown },
        { "Tesi", Color.DimGray },
        { "Lingua Inglese", Color.Navy },
        { "Tirocini formativi", Color.ForestGreen }
    };
nomi = new Label[] { nome1, nome2, nome3, nome4 };
dati = new TextBox[] { dati1, dati2, dati3, dati4 };
pulsanti = new Button[] { dado, libro, appunti, turno, avviaPartita, stop };
facceDado = new Bitmap[6];
immaginiPedine = new PictureBox[] { pedina1, pedina2, pedina3, pedina4 };
segnaposti = new PictureBox[] { colore1, colore2, colore3, colore4 };

//proviamo a caricare le immagini
string[] nomeFile = new string[] { "dado1", "dado2", "dado3",
                                    "dado4", "dado5", "dado6",
                                    "palla1", "palla2", "palla3", "palla4", "unipoly_template" };
try
{
    for (int i = 0; i < 6; i++)
        this.facceDado[i] =
    (Bitmap)Properties.Resources.ResourceManager.GetObject(nomeFile[i]);
    for (int i = 0; i < 4; i++)
    {
        this.immaginiPedine[i].Image =
    (Bitmap)Properties.Resources.ResourceManager.GetObject(nomeFile[i + 6]);
        this.segnaposti[i].Image =
    (Bitmap)Properties.Resources.ResourceManager.GetObject(nomeFile[i + 6]);
    }
    this.tabellone.BackgroundImage =
    (Bitmap)Properties.Resources.ResourceManager.GetObject(nomeFile[10]);
}
/*se non ci riusciamo, il metodo GetObject lancia un'eccezione e
quindi chiudiamo il form*/
catch (Exception)
{
    MessageBox.Show("Problema caricoamento immagine");
    this.Close();
}
}//end metodo load()

//proprietà per accedere ai controlli sul tavolo da gioco
public Button AvviaPartita { get { return this.avviaPartita; } }
public Button Dado { get { return this.dado; } }
public Button Stop { get { return this.stop; } }
public Button Turno { get { return this.turno; } }
public Button Libro { get { return this.libro; } }
public Button Appunti { get { return this.appunti; } }
public Button Regole { get { return this.regole; } }
public Label TimerGioco { get { return this.timerGioco; } }
public Label[] Nomi { get { return this.nomi; } }
public PictureBox[] Segnaposti { get { return this.segnaposti; } }
public PictureBox[] ImmaginiPedine { get { return this.immaginiPedine; } }
public PictureBox CartaPescata { get { return this.cartaPescata; } }
public TextBox[] Dati { get { return this.datি; } }
public TextBox NomeCarta { get { return this.nomeCarta; } }
public TextBox DescrizioneCarta { get { return this.descrizioneCarta; } }

//metodo per avviare il timer
public void AvviaTimer() { this.veroTimer.Enabled = true; }

//gestore dell'evento di click sul pulsante di lancio del dado
private void dado_Click(object sender, EventArgs e)
{

```

```

//LanciaDadoSpostaGiocatore() lancia un'eccezione se il dado è
//già stato lanciato
try
{
    //salviamo la posizione attuale
    Casella precedente = gioco.GiocatoreCorrente.Posizione;
    int esito = gioco.LanciaDadoSpostaGiocatore();

    //disabilitiamo il pulsante
    dado.Enabled = false;
    //mostriamo l'esito del dado
    this.esitoDado.Visible = true;
    this.esitoDado.Image = facceDado[esito - 1];

    //calcoliamo la nuova casella
    int indiceAttuale = (gioco.OttieniIndice(precedente) + esito) % 40;
    this.corrente = gioco.OttieniCasella(indiceAttuale);

    //aggiorniamo le informazioni di gioco, scegliamo la pedina e la muoviamo
    StampaInfoGioco();
    PictureBox immaginePedina = immaginiPedine[gioco.Turno];
    pedina.ImmaginePedina = immaginePedina;
    Muoviti(precedente, this.corrente);

    //se la casella corrente è imprevisto o probabilità, mostriamo la carta
    if (corrente is CasellaCarta)
        MostraCarta(true);
    //se la casella corrente è di tipo esame, diamo le possibilità di acquisto
    if (this.corrente is CasellaEsame)
    {
        libro.Enabled = true;
        appunti.Enabled = true;
    }
    /*se dalla casella siamo arrivati ad un'altra casella, facciamo partire
     un timer che ci consenta di leggere la carta pescata prima che la
     pedina si muova verso la nuova casella*/
    if (this.corrente.Carta != null && this.corrente.Carta is CartaMovimento)
    {
        timerMovimento.Enabled = true;
        timerMovimento.Start();
    }
    //abilitiamo di nuovo il pulsante di terminazione del turno
    else
        turno.Enabled = true;
} //end try
catch
{
    MessageBox.Show("Hai già tirato il dado.");
}
} //end metodo dado_Click()

//gestore dell'evento di click sul pulsante di terminazione del turno
private void turno_Click(object sender, EventArgs e)
{
    //CambiaTurno() lancia un'eccezione se non è ancora stato lanciato il dado
    try
    {
        //resettiamo il bordo del segnaposto
        segnaposti[gioco.Turno].BorderStyle = BorderStyle.None;
        /*disabilitiamo i pulsanti di avvio partita, di acquisto di libri e appunti
         e di terminazione del turno, resettiamo l'immagine del dado e nascondiamo la
         carta*/
        for (int i = 0; i < 4; i++)
            pulsanti[i].Enabled = false;
    }
}
```

```

        esitoDado.Visible = false;
        MostraCarta(false);
        //aggiorniamo le informazioni sui giocatori
        for (int i = 0; i < gioco.Giocatori.Count; i++)
            Dati[i].Text = DatiGiocatore(i);

        //passiamo il turno al prossimo giocatore
        bool partitaInCorso = gioco.CambiaTurno();

        //se il passaggio ha avuto successo
        if (partitaInCorso == true)
        {
            //mettiamo in evidenza il segnaposto del giocatore corrente,
            //aggiorniamo le info e abilitiamo il pulsante di lancio del dado
            segnaposti[gioco.Turno].BorderStyle = BorderStyle.FixedSingle;
            StampaInfoGioco();
            Dado.Enabled = true;
        }
        //altrimenti la partita è terminata, quindi informiamo
        //l'utente e aggiorniamo il tavolo
        else
        {
            MessageBox.Show("Partita terminata!");
            TerminaPartita();
        }
    } //end try
    catch (Exception)
    {
        MessageBox.Show("Tira il dado prima.");
    }
} //end metodo turno_Click()

//gestore dell'evento di click sul pulsante di acquisto libri
private void libro_Click(object sender, EventArgs e)
{
    //Compra() lancia un'eccezione se la casella non è comprabile
    try
    {
        //proviamo a comprare il libro, se ci riusciamo contrassegniamo
        //la casella col colore del giocatore
        bool esito = gioco.GiocatoreCorrente.Posizione.Compra();
        if (esito == true)
            MostraProprietario();
        //disabilitiamo il pulsante e stampiamo le info di gioco
        libro.Enabled = false;
        StampaInfoGioco();
    }
    catch
    {
        MessageBox.Show("Casella non comprabile.");
    }
} //end libro_Click()

//gestore dell'evento di click sul pulsante di acquisto libri
private void appunti_Click(object sender, EventArgs e)
{
    //Compra() lancia un'eccezione se la casella non è potenziabile
    try
    {
        //proviamo a comprare gli appunti e nel caso stampiamo le info di gioco
        bool esito = gioco.GiocatoreCorrente.Posizione.Potenzia();
        //se è stato possibile, mettiamo un bordo per far notare la differenza
        if (esito == true)
            TrovaImmagineProprietario().BorderStyle = BorderStyle.FixedSingle;
    }
}

```

```

        StampaInfoGioco();
    }
    catch
    {
        MessageBox.Show("Casella non potenziabile.");
    }
}//end appunti_Click()

//gestore dell'evento di click sul pulsante di avvio della partita
private void avviaPartita_Click(object sender, EventArgs e)
{
    //mostriamo il form per aggiungere i giocatori alla partita
    AvviaPartita ap = new AvviaPartita(gioco, this);
    ap.Show();
}

//gestore dell'evento di click sul pulsante di visualizzazione delle regole
private void regole_Click(object sender, EventArgs e)
{
    //mostriamo il form con le regole di gioco
    Regole regole = new Regole();
    regole.Show();
}

//gestore dell'evento di click sul pulsante di terminazione della partita
private void stop_Click(object sender, EventArgs e) { TerminaPartita(); }

//gestore dell'evento di tick del timer di gioco
private void veroTimer_Tick(object sender, EventArgs e)
{
    //aggiorniamo il timer sottraendo un secondo al tempo rimanente
    tempo = tempo - (new TimeSpan(0, 0, 1));
    this.TimerGioco.Text = tempo.Hours + ":" + tempo.Minutes + ":" +
                           tempo.Seconds;
}

/*gestore dell'evento di tick del timer necessario per posticipare il secondo
movimento della pedina nel caso in cui venga pescata una carta movimento*/
private void timerMovimento_Tick(object sender, EventArgs e)
{
    //la pedina si muove verso la casella finale e si resetta tutto
    Casella conseguente = gioco.GiocatoreCorrente.Posizione;
    Muoviti(this.corrente, conseguente);
    this.timerMovimento.Stop();
    this.timerMovimento.Enabled = false;
    MostraCarta(false);
    turno.Enabled = true;
}

//metodo per spostare il giocatore
public void Muoviti(Casella prima, Casella seconda)
{
    //viene mossa la pedina
    pedina.MovimentoPedina(prima, seconda);

    //vengono mostrate le informazioni sulla casella e aggiornato il colore
    nomeCasella.Text = seconda.Nome;
    descrizioneCasella.Text = "\r\n" + seconda;
    if (seconda is CasellaEsame)
        nomeCasella.BackColor = colorePerArea[corrente.AreaFormativa.Nome];
    else
        nomeCasella.BackColor = Color.Gold;
}//end metodo Muoviti()

```

```

//metodo per mostrare un segno sulle caselle che ne indichi il proprietario
public void MostraProprietario()
{
    PictureBox proprietario = TrovaImmagineProprietario();
    //viene impostato il colore corrispondente al giocatore
    proprietario.BackColor = coloriPerGiocatore[gioco.Turno];
    proprietario.Visible = true;
}//end MostraProprietario()

//metodo che restituisce la PictureBox che indica il proprietario della casella
public PictureBox TrovaImmagineProprietario()
{
    /*si cerca la PictureBox nel tabellone che abbia come tag l'indice della
    casella corrente.*/
    PictureBox proprietario = null;
    bool trovato = false;
    while (trovato == false)
    {
        foreach (PictureBox p in this.tabellone.Controls)
        {
            if (p.Tag != null)
            {
                //si converte il tag a string e si estrae il numero
                int tag = int.Parse((string)p.Tag);
                if (tag == gioco.OttieniIndice(corrente))
                {
                    proprietario = p;
                    trovato = true;
                }
            }
        }
    }
    return proprietario;
}//end TrovaImmagineProprietario()

//metodo per mostrare (o nascondere, a seconda dell'input) la carta pescata
public void MostraCarta(bool mostra)
{
    nomeCarta.Visible = mostra;
    descrizioneCarta.Visible = mostra;
    cartaPescata.Visible = mostra;
    if (mostra == true)
    {
        //vengono aggiornate le info della carta
        nomeCarta.Text = corrente.TipoCarta;
        descrizioneCarta.Text = "\r\n" + corrente.Carta.Descrizione;
        //la carta viene colorata a seconda del tipo
        if (corrente.Nome.Equals("Probabilità"))
            ColoraCarta(Color.SandyBrown);
        else
            ColoraCarta(Color.PaleGreen);
    }
}//end metodo MostraCarta()

//metodo per colorare la carta a seconda del colore in input
public void ColoraCarta(Color colore)
{
    nomeCarta.BackColor = colore;
    cartaPescata.BackColor = colore;
    descrizioneCarta.BackColor = colore;
}

//metodo che restituisce i dati del giocatore corrispondente all'indice in input
public string DatiGiocatore(int indice)

```

```

    {
        return "Soldi: " + (int)gioco.Giocatori.ElementAt(indice).Soldi + "\r\n" +
            "CFU: " + gioco.Giocatori.ElementAt(indice).Cfu + "\r\n" +
            "Ruolo: " + gioco.Giocatori.ElementAt(indice).Ruolo;
    }

    //metodo per stampare le informazioni di gioco nella TextBox
    public void StampaInfoGioco()
    {
        if (gioco.InfoGioco != null)
        {
            infoGioco.Text = gioco.InfoGioco;
            //la TextBox viene scrollata automaticamente
            infoGioco.SelectionStart = infoGioco.Text.Length;
            infoGioco.ScrollToCaret();
        }
    }

    //metodo per terminare la partita
    public void TerminaPartita()
    {
        //vengono disabilitati i pulsanti e fermato il timer
        foreach (Control c in pulsanti)
            c.Enabled = false;
        veroTimer.Stop();
        //viene stampata la classifica
        gioco.DecretaClassifica();
        StampaInfoGioco();
    }
}

}//end classe
}

```

18) AvviaPartita

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

//si importa la libreria del gioco
using LibreriaUnipoly;

namespace UnipolyGUI
{
    //classe di tipo Form per aggiungere i giocatori
    public partial class AvviaPartita: Form
    {
        private readonly Gioco gioco;          //gioco a cui ci si riferisce
        private readonly TavoloGioco tg;       //tavolo da gioco da inizializzare

        //costruttore
        public AvviaPartita(Gioco gioco, TavoloGioco tg)
        {
            //inizializzazione di componenti e attributi
            InitializeComponent();
            this.gioco = gioco;
            this.tg = tg;
        }
    }
}

```

```

//gestore dell'evento di click sul tasto Conferma
private void Conferma_Click(object sender, EventArgs e)
{
    //vengono contati i giocatori validi inseriti
    int numGiocatori = 0;
    if (ValidaGiocatore(nome1.Text, (string)ruolo1.SelectedItem))
        numGiocatori++;
    if (ValidaGiocatore(nome2.Text, (string)ruolo2.SelectedItem))
        numGiocatori++;
    if (ValidaGiocatore(nome3.Text, (string)ruolo3.SelectedItem))
        numGiocatori++;
    if (ValidaGiocatore(nome4.Text, (string)ruolo4.SelectedItem))
        numGiocatori++;

    //se i giocatori sono nel range [2,4]
    if (numGiocatori >= 2 && numGiocatori <= 4)
    {
        //vengono aggiunti i primi 2 giocatori
        Giocatore gioc1 = gioco.AggiungiGiocatore(nome1.Text,
                                                    (string)ruolo1.SelectedItem);
        Giocatore gioc2 = gioco.AggiungiGiocatore(nome2.Text,
                                                    (string)ruolo2.SelectedItem);
        //vengono inizializzati e resi visibili: pedine, nomi, dati
        AggiornaDati(0, (Bitmap)tg.ImmaginiPedine[0].Image,
                      gioc1.Nome, tg.DatiGiocatore(0));
        AggiornaDati(1, (Bitmap)tg.ImmaginiPedine[1].Image,
                      gioc2.Nome, tg.DatiGiocatore(1));
        //viene messo in evidenza il segnaposto del primo giocatore
        tg.Segnaposti[0].BorderStyle = BorderStyle.FixedSingle;
        //se c'è anche un terzo giocatore
        if (numGiocatori == 3 || numGiocatori == 4)
        {
            //viene aggiunto il terzo giocatore
            Giocatore gioc3 = gioco.AggiungiGiocatore(nome3.Text,
                                                        (string)ruolo3.SelectedItem);
            //vengono inizializzati e resi visibili: pedine, nomi, dati
            AggiornaDati(2, (Bitmap)tg.ImmaginiPedine[2].Image,
                          gioc3.Nome, tg.DatiGiocatore(2));
        }
        //se c'è anche un quarto giocatore
        if (numGiocatori == 4)
        {
            //viene aggiunto il quarto giocatore
            Giocatore gioc4 = gioco.AggiungiGiocatore(nome4.Text,
                                                        (string)ruolo4.SelectedItem);
            //vengono inizializzati e resi visibili: pedine, nomi, dati
            AggiornaDati(3, (Bitmap)tg.ImmaginiPedine[3].Image,
                          gioc4.Nome, tg.DatiGiocatore(3));
        }
        //si disabilita il pulsante di avvio partita
        tg.AvviaPartita.Enabled = false;
        //si abilitano il pulsante di lancio del dado e quello di stop
        tg.Dado.Enabled = true;
        tg.Stop.Enabled = true;
        //si chiude la finestra
        this.Close();
        //si inizia la partita
        gioco.IniziaPartita();
        //si avvia il timer di gioco
        tg.AvviaTimer();
    }
    //altrimenti viene mostrata una finestra di warning
    else
    {

```

```

        MessageBox.Show("Errore, numero di giocatori inseribili non valido.",
                        "Errore di formato", MessageBoxButtons.OK,
                        MessageBoxIcon.Warning);
    }
}//end Conferma_Click()

//metodo di validazione di nome e ruolo dei giocatori inseriti
public bool ValidaGiocatore(string nome, string ruolo)
{
    bool esito = false; //true se giocatore è valido

    if ((nome.Equals("")) == false && ruolo != null &&
        (ruolo.Equals("in sede") ||
        ruolo.Equals("fuori sede") ||
        ruolo.Equals("pendolare") ||
        ruolo.Equals("straniero")))
        esito = true;

    return esito;
}//end ValidaGiocatore()

//metodo per inizializzare le informazioni sui giocatori sul tavolo da gioco
public void AggiornaDati(int indice, Bitmap immagine, string nome, string dati)
{
    tg.ImmaginiPedine[indice].Image = immagine;
    tg.ImmaginiPedine[indice].Visible = true;
    tg.Nomi[indice].Text = nome;
    tg.Nomi[indice].Visible = true;
    tg.Dati[indice].Text = dati;
    tg.Dati[indice].Visible = true;
    tg.Segnaposti[indice].Visible = true;
}
}//end classe
}

```

19) Regole

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace UnipolyGUI
{
    //classe di tipo Form per visualizzare le regole
    public partial class Regole : Form
    {
        //costruttore
        public Regole()
        {
            InitializeComponent();
            try
            {
                this.immagineRegole.Image =
                    (Image)Properties.Resources.ResourceManager.GetObject("RegolePic");
            }catch
            {
                MessageBox.Show("Immagine non caricata correttamente");
                this.Close();
            }
        }
    }
}

```

```

        }
    }

    //proprietà per accedere all'immagine delle regole
    public PictureBox Istruzioni
    {
        get { return this.immagineRegole; }
    }
}//end classe
}

```

20) Pedina

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;

//si importa la libreria del gioco
using LibreriaUnipoly;

namespace UnipolyGUI
{
    //classe che rappresenta la pedina che si muoverà sul tabellone
    class Pedina
    {
        //attributi
        private readonly TavoloGioco tg;                                //tavolo a cui si riferisce
        private readonly Gioco gioco;                                    //gioco a cui si riferisce
        private PictureBox immaginePedina;                             //immagine della pedina
        public const int DISTANZA_PEDINE = 20;                         //distanza tra le pedine
        public const int DISTANZA_CASELLE = 55;                         //distanza tra le caselle

        //posizione dei 4 angoli
        private readonly Point bassoDestra;                            //angolo inferiore destro
        private readonly Point bassoSinistra;                           //angolo inferiore sinistro
        private readonly Point altoSinistra;                            //angolo superiore sinistro
        private readonly Point altoDestra;                             //angolo superiore destro

        //costruttore
        public Pedina(TavoloGioco tg, Gioco gioco)
        {
            //inizializzazione degli attributi
            this.tg = tg;
            this.gioco = gioco;
            bassoDestra = new Point(588, 590);
            bassoSinistra = new Point(10, 576);
            altoSinistra = new Point(38, 4);
            altoDestra = new Point(610, 30);
        }

        //proprietà per cambiare l'immagine della pedina
        public PictureBox ImmaginePedina { set { immaginePedina = value; } }

        //metodo per spostare la pedina sul tabellone
        //prende in input la casella di partenza e quella di arrivo
        public void MovimentoPedina(Casella prec, Casella corr)
        {
            //si calcolano gli indici di partenza e di arrivo
            int indicePrec = gioco.OttieniIndice(prec);
            int indiceCorr = gioco.OttieniIndice(corr);

            //fino a quando non si arriva sull'indice giusto

```

```

        while (indiceCorr != indicePrec)
    {
        //sleep per visualizzare meglio i salti
        System.Threading.Thread.Sleep(70);

        //se siamo sulla riga inferiore, ci spostiamo a sinistra
        if (indicePrec >= 0 && indicePrec < 9)
            immaginePedina.Location = new Point(immaginePedina.Location.X
                - DISTANZA_CASELLE,
                immaginePedina.Location.Y);

        //alla fine della riga ci posizioniamo sull'angolo
        else if (indicePrec == 9)
            PosizionaPedinaAngolo(bassoSinistra);

        //se siamo sulla riga a sinistra, ci spostiamo in alto
        else if (indicePrec >= 10 && indicePrec < 19)
            immaginePedina.Location = new Point(immaginePedina.Location.X,
                immaginePedina.Location.Y
                - DISTANZA_CASELLE);

        //alla fine della riga ci posizioniamo sull'angolo
        else if (indicePrec == 19)
            PosizionaPedinaAngolo(altoSinistra);

        //se siamo sulla riga superiore, ci spostiamo a destra
        else if (indicePrec >= 20 && indicePrec < 29)
            immaginePedina.Location = new Point(immaginePedina.Location.X
                + DISTANZA_CASELLE,
                immaginePedina.Location.Y);

        //alla fine della riga ci posizioniamo sull'angolo
        else if (indicePrec == 29)
            PosizionaPedinaAngolo(altoDestra);

        //se siamo sulla riga a destra, ci spostiamo in basso
        else if (indicePrec >= 30 && indicePrec < 39)
            immaginePedina.Location = new Point(immaginePedina.Location.X,
                immaginePedina.Location.Y
                + DISTANZA_CASELLE);

        //alla fine della riga ci posizioniamo sull'angolo
        else if (indicePrec == 39)
            PosizionaPedinaAngolo(bassoDestra);

        //ci siamo spostati di una casella
        indicePrec++;
        //se abbiamo percorso tutto il tabellone, torniamo alla prima casella
        if (indicePrec == 40)
            indicePrec = 0;
    }
}//end MovimentoPedina()

//metodo per posizionarsi ad uno degli angoli in input.
//è necessario poiché gli angoli sono più grandi delle altre caselle
public void PosizionaPedinaAngolo(Point punto)
{
    if (gioco.Turno == 0)
        tg.ImmaginiPedine[0].Location = punto;
    else if (gioco.Turno == 1)
        tg.ImmaginiPedine[1].Location = new Point(punto.X + DISTANZA_PEDINE,
            punto.Y);
    else if (gioco.Turno == 2)
        tg.ImmaginiPedine[2].Location = new Point(punto.X, punto.Y
            + DISTANZA_PEDINE);
    else
        tg.ImmaginiPedine[3].Location = new Point(punto.X + DISTANZA_PEDINE,
            punto.Y + DISTANZA_PEDINE);
}

```

```
        } //end PosizionaPedinaAngolo()
    } //end classe
}

Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace UnipolyGUI
{
    static class Program
    {
        /// <summary>
        /// Punto di ingresso principale dell'applicazione.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            //apriamo la finestra di tavolo da gioco
            Application.Run(new TavoloGioco());
        }
    }
}
```

5. TEST

La fase di testing è molto importante nello sviluppo di software. Ovviamente non è possibile assicurarsi che il software sia completamente privo di bug, soprattutto se esso è di dimensioni rilevanti. Ciò che si fa di solito è cercare di coprire il maggior numero di casi possibili attraverso due tecniche: il testing white box e quello black box.

White Box Test

Il collaudo white box usa una prospettiva interna del sistema per progettare casi di test basati sulla struttura interna. Ogni caso di test è associato ad un diverso percorso logico, cioè una sequenza di istruzioni logiche o procedurali nel flusso di esecuzione. Quindi si basa sull'analisi di copertura del codice:

- copertura istruzioni -> si seleziona un insieme di test tali che ogni istruzione di processo (cioè blocco d'azione) sia eseguito almeno una volta. In alcuni casi sono necessari più insiemi di valori per coprire tutte le istruzioni.
- copertura rami -> si seleziona un insieme di test tali che ogni ramo del flusso di controllo sia eseguito almeno una volta.
- copertura condizioni -> si seleziona un insieme di test tali che ogni condizione base sia vera o falsa almeno una volta e ogni elemento di una condizione composta sia vero o falso almeno una volta.

Poiché si è adottata una strategia di tipo Extreme Programming, il collaudo di tipo white box è stato continuo: attraverso la stringa infoGioco si è potuto verificare il corretto funzionamento dei componenti (unit test) man mano che venivano sviluppati.

Per verificare che il transito sulle caselle avvenisse correttamente, però, si è costruita un'applicazione Console che accelerasse il collaudo. Essa è di seguito riportata:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

using LibreriaUnipoly;

namespace Testing
{
    class Program
    {
        static void Main(string[] args)
        {
            Test t = new Test();
            t.testaPartita();
            Console.ReadKey();
        }

        class Test
        {
            Gioco g;
            Configurazione config;
            Casella intermedia;
            Casella finale;

            public Test()
            {
                g = Gioco.OttieniGioco();
                config = new Configurazione(g);
                config.CaricaTabellone();
                g.AggiungiGiocatore("Paola", "pendolare");
                g.AggiungiGiocatore("Giorgia", "pendolare");
            }
        }
    }
}
```

```

        g.IniziaPartita();
    }
    public void testaPartita()
    {
        Console.WriteLine("TESTING\r\n\r\n" + "Via: ");

        for(int i = 0; i < 20; i++)
        {
            g.GiocatoreCorrente.Posizione = g.OttieniCasella(0);
            intermedia = g.OttieniCasella(i);
            g.GiocatoreCorrente.Spostati(i);
            finale = g.GiocatoreCorrente.Posizione;

            if (intermedia is CasellaEsame ||
                intermedia is CasellaFuoriCorso ||
                intermedia is CasellaTerzaRata)
                Console.WriteLine(i + ": " + intermedia.Nome + "\r\n"
                                  + intermedia + "\r\n");
            else if (intermedia is CasellaSpeciale)
            {
                Console.WriteLine(i + ": Casella Speciale. " + intermedia + "\r\n");
                if(intermedia.Carta is CartaMovimento)
                    Console.WriteLine("Passiamo da questa casella a " +
                                      finale + "\r\n");
            }
            else if (intermedia is CasellaCarta)
            {
                Console.WriteLine(i + ": Casella Carta. " + intermedia +
                                  "\r\nDescrizione carta: " +
                                  intermedia.Carta.Descrizione + "\r\n");
                if (intermedia.Carta is CartaMovimento)
                    Console.WriteLine("Passiamo da questa casella a " +
                                      finale + "\r\n");
            }
            Console.WriteLine("Dati giocatore corrente: \r\nRuolo: "
                            + g.GiocatoreCorrente.Ruolo + "\r\nCfu: " +
                            g.GiocatoreCorrente.Cfu + "\r\nSoldi: " +
                            g.GiocatoreCorrente.Soldi + "\r\nTurni da saltare:
                            " + g.GiocatoreCorrente.TurniDaSaltare + "\r\n");
            Console.WriteLine("*****ALTRA TEST*****");
        }
    }
}
}

```

Il risultato è stato il seguente:

TESTING

Via: 0: Casella Speciale. Prima rata! -500 euro

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 0

Soldi: 1500

Turni da saltare: 0

*****ALTR0 TEST*****

1: Matematica Discreta

Tipo: Casella esame

Area formativa: Formazione matematico-fisica

CFU: 6

Proprietario: nessuno

Prezzo libro: 60

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 0

Soldi: 1500

Turni da saltare: 0

*****ALTR0 TEST*****

2: Casella Speciale. Borsa di studio! +500 euro

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 0

Soldi: 2000

Turni da saltare: 0

*****ALTR0 TEST*****

3: Analisi Matematica

Tipo: Casella esame

Area formativa: Formazione matematico-fisica

CFU: 12

Proprietario: nessuno

Prezzo libro: 120

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 0

Soldi: 2000

Turni da saltare: 0

*****ALTR0 TEST*****

4: Casella Carta. Pesca una carta probabilità..

Descrizione carta: Vai in Erasmus

Passiamo da questa casella a Guadagni 24 cfu per l'Erasmus

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 0

*****ALTRO TEST*****

5: Casella Speciale. Resta fermo un turno

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 1

*****ALTRO TEST*****

6: Algoritmi e Strutture Dati

Tipo: Casella esame

Area formativa: Formazione informatica di base

CFU: 12

Proprietario: nessuno

Prezzo libro: 120

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 1

*****ALTRO TEST*****

7: Casella Carta. Pesca una carta imprevisto..

Descrizione carta: Uai in stallo

Passiamo da questa casella a Resta fermo due turni

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 2

*****ALTRO TEST*****

8: Architettura degli Elaboratori

Tipo: Casella esame

Area formativa: Formazione informatica di base

CFU: 12

Proprietario: nessuno

Prezzo libro: 120

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 2

*****ALTRO TEST*****

9: Economia e Gestione delle Imprese

Tipo: Casella esame

Area formativa: Esami a Scelta

CFU: 6

Proprietario: nessuno

Prezzo libro: 60

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 2

*****ALTR0 TEST*****

10: Casella Speciale. Resta fermo due turni

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 2

*****ALTR0 TEST*****

11: Piattaforme Digitali per la Gestione del Territorio

Tipo: Casella esame

Area formativa: Esami a Scelta

CFU: 6

Proprietario: nessuno

Prezzo libro: 60

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 2

*****ALTR0 TEST*****

12: Casella Speciale. Guadagni 24 cfu per l'Erasmus

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 48

Soldi: 2000

Turni da saltare: 2

*****ALTR0 TEST*****

13: Elaborazione di Segnali e Immagini

Tipo: Casella esame

Area formativa: Esami di curriculum

CFU: 12

Proprietario: nessuno

Prezzo libro: 120

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 48

Soldi: 2000

Turni da saltare: 2

*****ALTR0 TEST*****

14: Simulazione Numerica

Tipo: Casella esame

Area formativa: Esami di curriculum

CFU: 6

Proprietario: nessuno

Prezzo libro: 60

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 48

Soldi: 2000

Turni da saltare: 2

*******ALTRA TEST*******
15: Casella Speciale. Perdi 24 cfu per il giovedì universitario

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 2

*******ALTRA TEST*******

16: Probabilità e Statistica

Tipo: Casella esame

Area formativa: Formazione matematico-fisica

CFU: 6

Proprietario: nessuno

Prezzo libro: 60

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 2000

Turni da saltare: 2

*******ALTRA TEST*******

17: Casella Speciale. Seconda rata! -400 euro

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 1600

Turni da saltare: 2

*******ALTRA TEST*******

18: Fisica I

Tipo: Casella esame

Area formativa: Formazione matematico-fisica

CFU: 6

Proprietario: nessuno

Prezzo libro: 60

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 1600

Turni da saltare: 2

*******ALTRA TEST*******

19: Fisica II

Tipo: Casella esame

Area formativa: Formazione matematico-fisica

CFU: 6

Proprietario: nessuno

Prezzo libro: 60

Prezzo appunti: 20

Numero appunti associati: 0

Dati giocatore corrente:

Ruolo: pendolare

Cfu: 24

Soldi: 1600

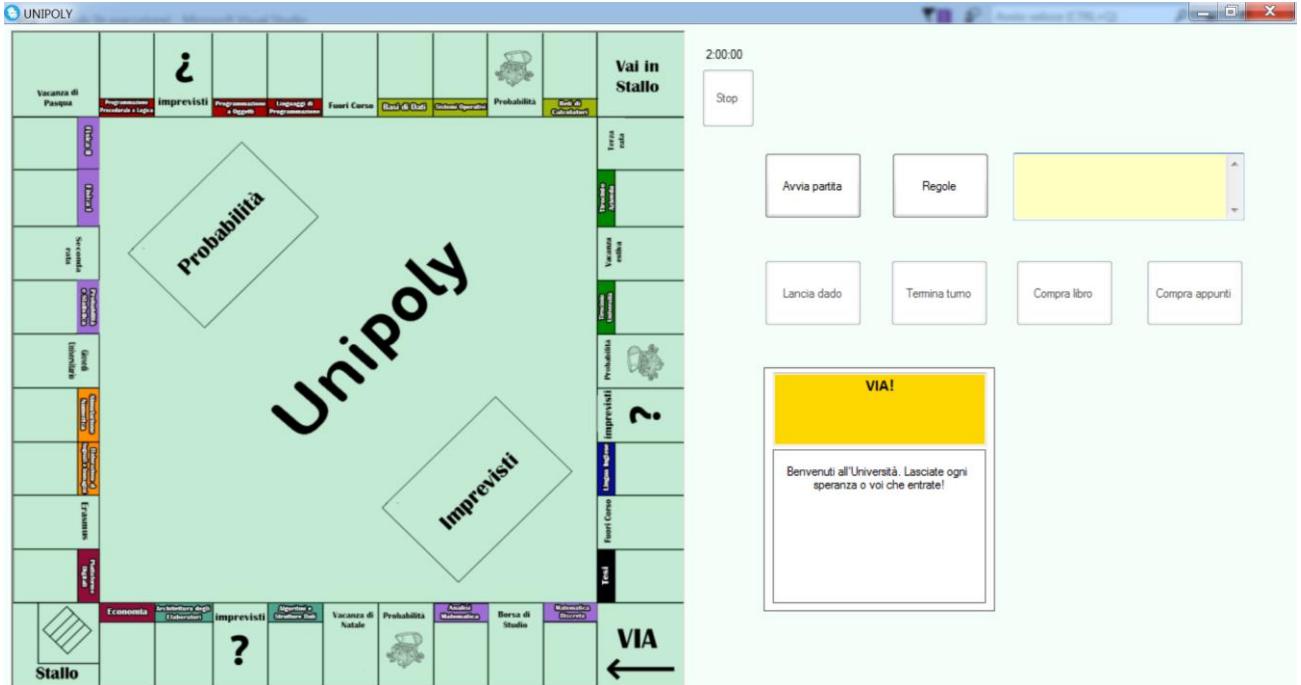
Turni da saltare: 2

Black Box Test

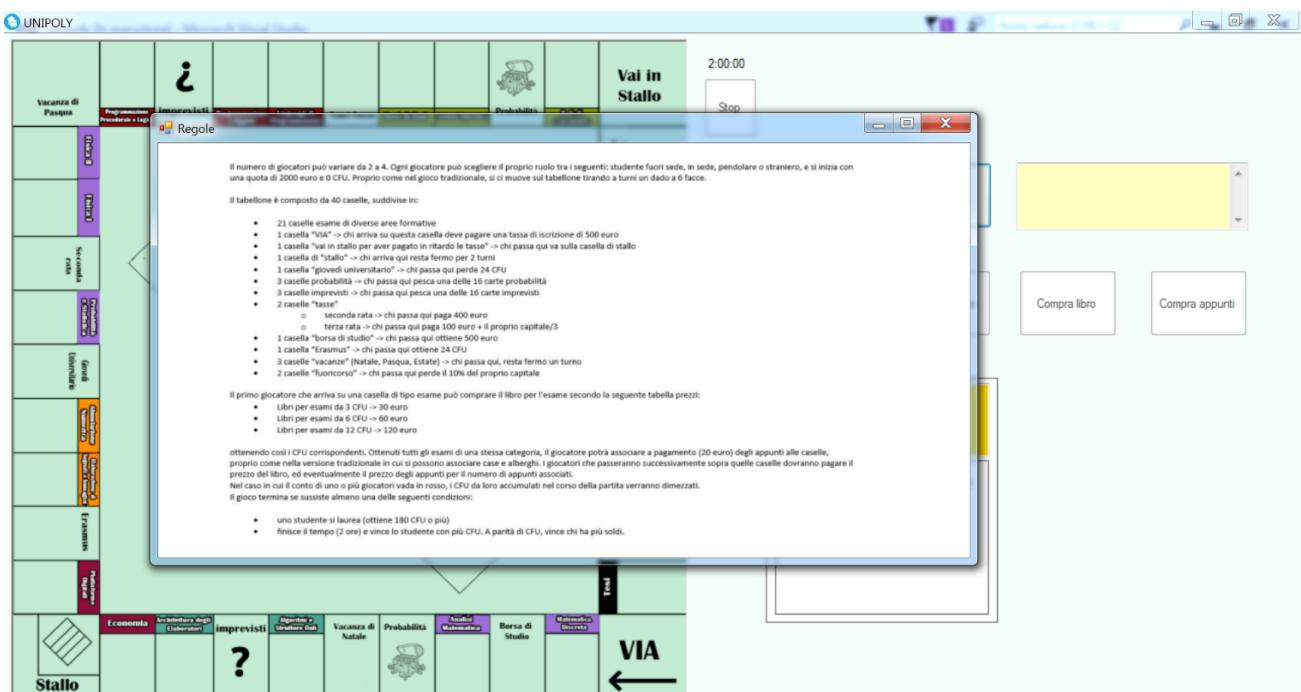
Il collaudo black box si concentra sui requisiti funzionali del software. Permette di derivare, attraverso il partizionamento d'equivalenza, set di condizioni di input che mettono alla prova tutti i requisiti funzionali del programma.

Trova errori nei seguenti casi: funzioni assenti o incorrette; errori di interfaccia con l'utente; errori comportamentali o di performance; errori di inizializzazione o terminazione.

1. Viene testata innanzitutto la schermata iniziale dal tavolo da gioco:

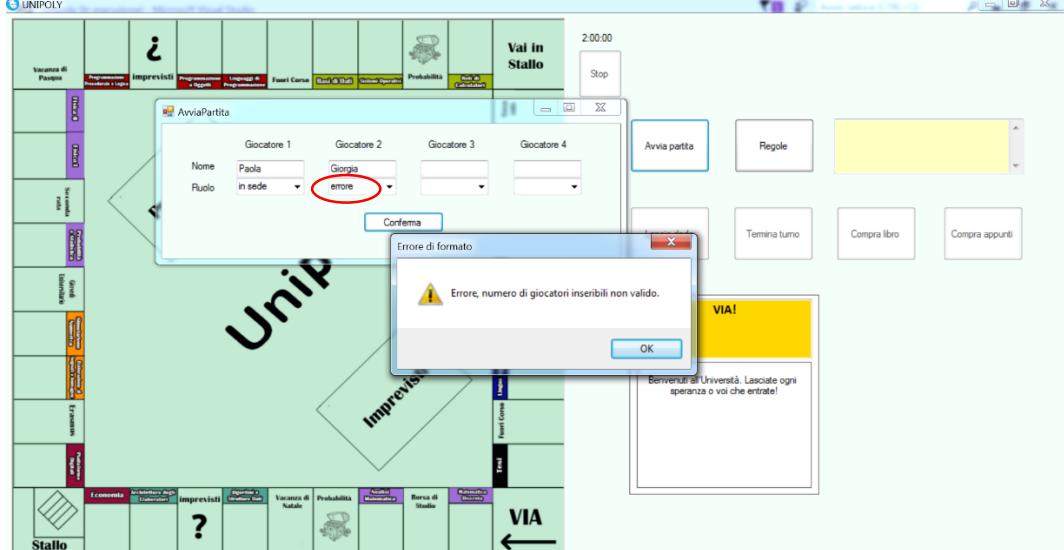
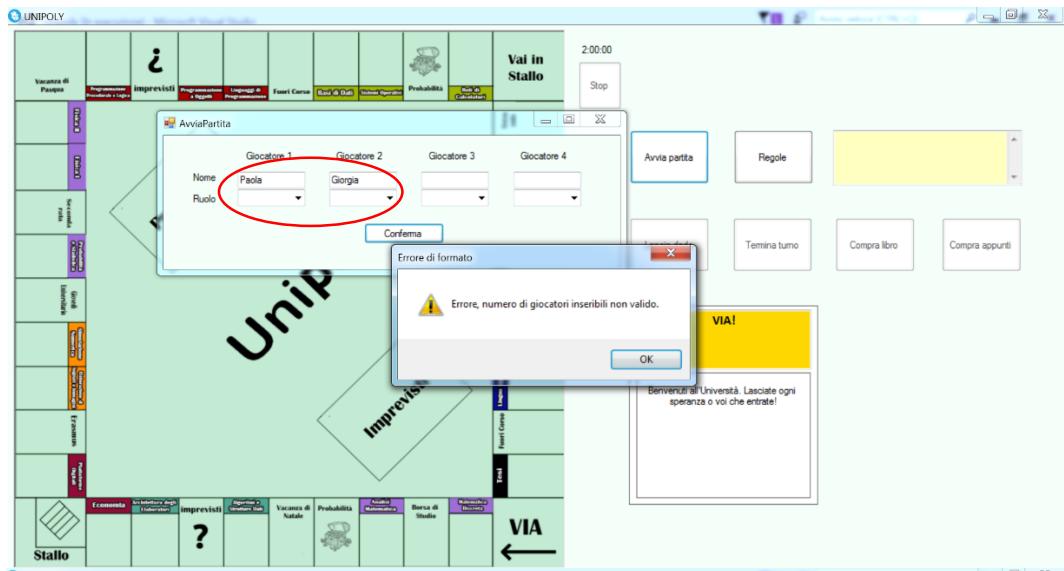


2. Viene testata la corretta visualizzazione del form di regole di gioco:

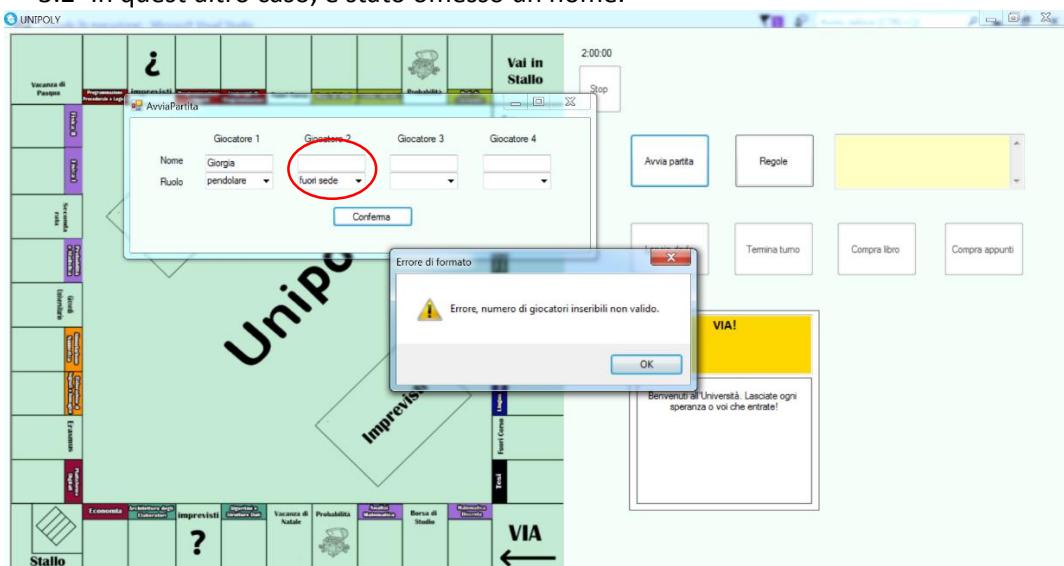


3. Viene testato l'inserimento errato dei giocatori.

3.1 In questi due casi, non sono stati inseriti correttamente i ruoli e quindi viene mostrato un messaggio di errore.



3.2 In quest'altro caso, è stato omesso un nome.



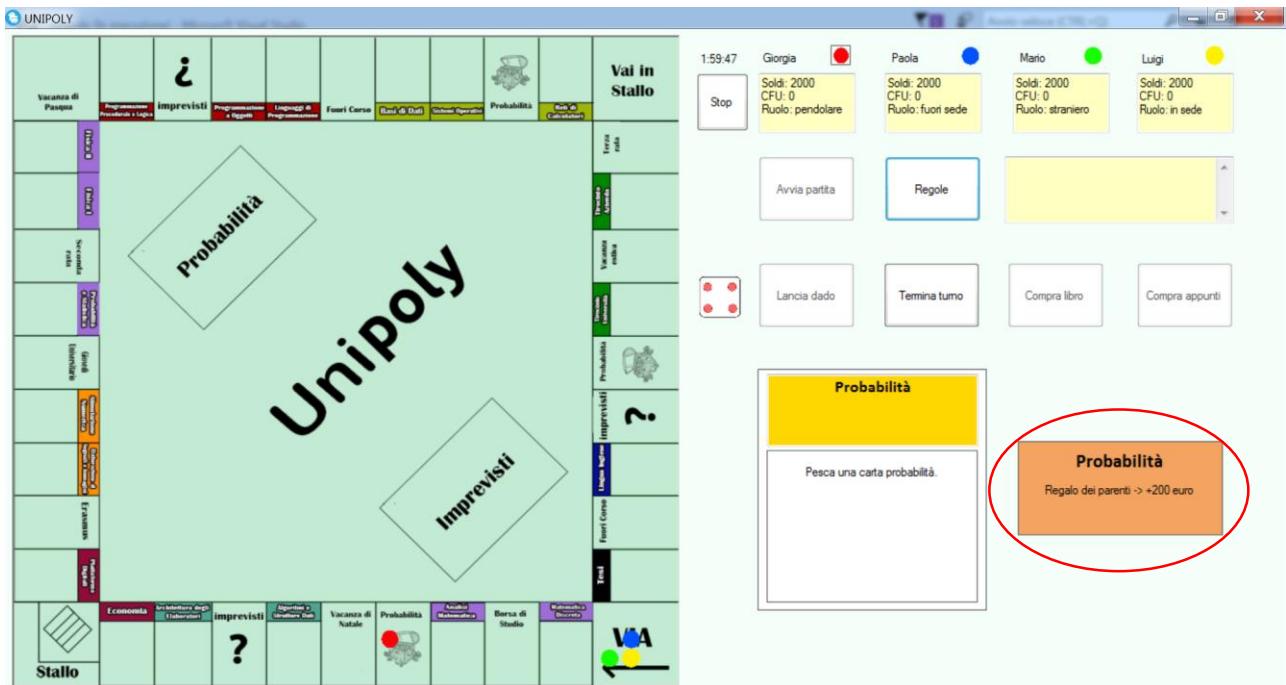
4. Si testa adesso il corretto inserimento di 2, 3 e 4 giocatori.

The screenshots show the Unipoly board game interface with three different player configurations:

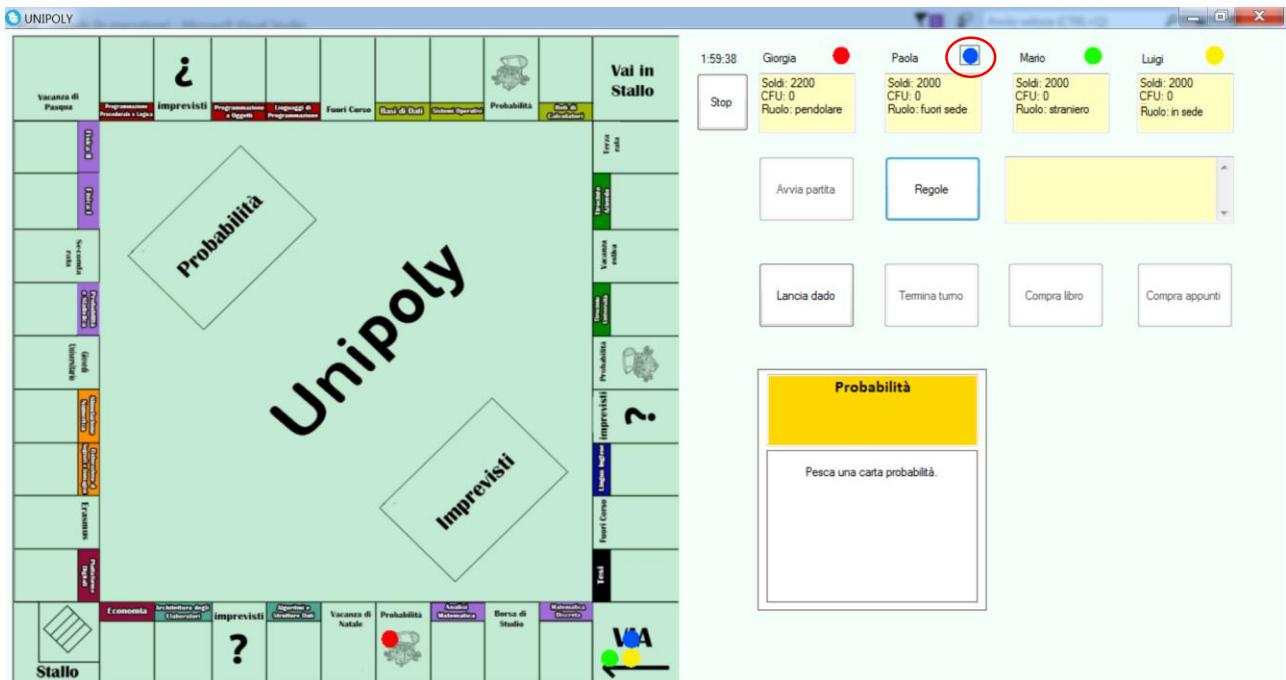
- 2 players:** Paola (red dot) and Giorgia (blue dot). Both have 2000 soldi and 0 CFU. Paola's role is "fuori sede" (off-site), while Giorgia's is "in sede" (on-site).
- 3 players:** Giorgia (blue dot), Paola (red dot), and Mario (green dot). All three have 2000 soldi and 0 CFU. Mario's role is "straniero" (foreigner).
- 4 players:** Giorgia (blue dot), Paola (red dot), Mario (green dot), and Luigi (yellow dot). All four have 2000 soldi and 0 CFU. Luigi's role is "in sede" (on-site).

The board features large diamond-shaped tokens labeled "Probabilità" (Probability) and "Imprevisti" (Unexpected Events) in the center. The right side of the screen displays a menu with options: Stop, Avvia partita (Start game), Regole (Rules), Lancia dado (Roll dice), Termina turno (End turn), Compra libro (Buy book), and Compra appunti (Buy notes). A yellow box labeled "VIA!" (Way!) contains the text: "Benvenuti all'Università. Lasciate ogni speranza o voi che entrate!" (Welcome to the University. Leave all hope or you who enter!).

5. Si testa il transito su una casella di tipo Probabilità. Si noti che la pedina si posiziona avanti di 4 posizioni (concordemente all'esito del dado) e che la carta viene mostrata correttamente.



6. Si noti che, dopo aver premuto sul pulsante Termina turno, vengono aggiornate le informazioni dei giocatori, e che il turno passa al giocatore successivo.



7. Viene testato ora il transito su una casella di tipo Esame. Si noti che viene mostrato un prompt nella sezione flusso di gioco. Si testa in seguito l'acquisto del libro e il tentativo di acquisto di appunti.

The screenshots show a sequence of interactions on a board space labeled "Analisi Matematica" (Analysis of Mathematics) which is identified as an exam space (Casella esame). The board features large diamond-shaped tokens labeled "Probabilità" (Probability) and "Imprevisti" (Unexpected Events).

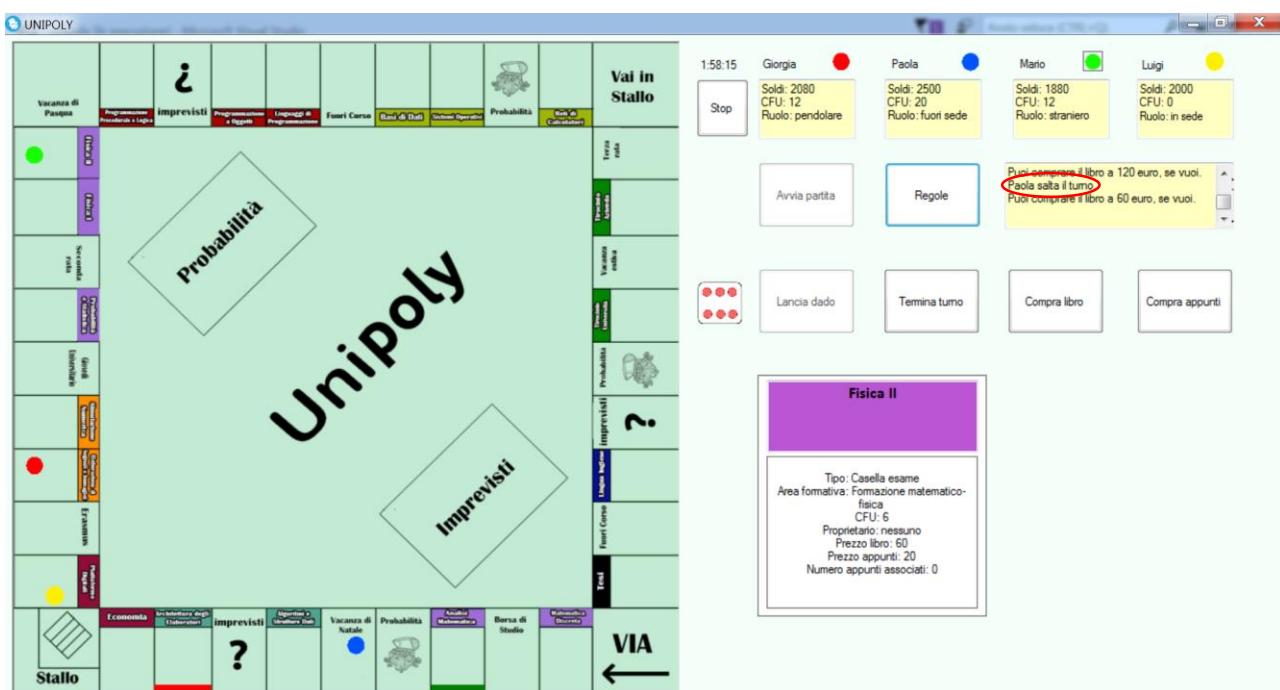
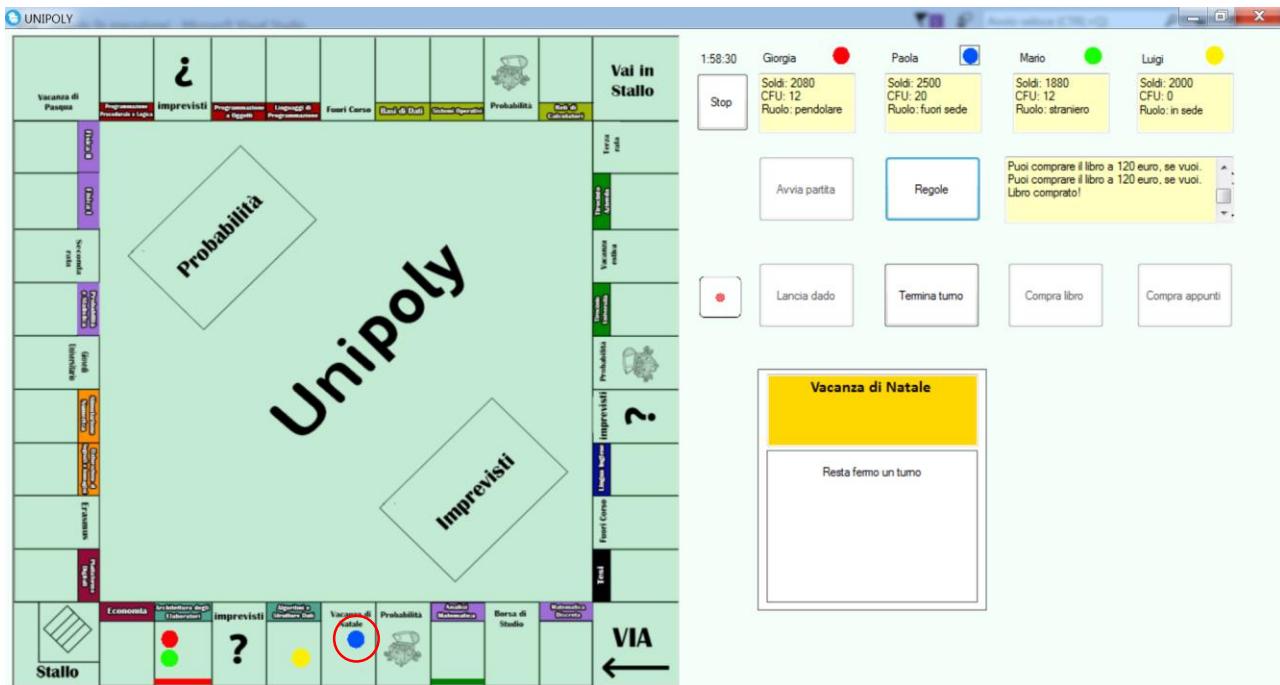
Screenshot 1: The player has the option to "Buy book" (Buy libro) for 120 euros. A red oval highlights the message "Puri comprare il libro a 120 euro, se vuoi." (Feel free to buy the book for 120 euros, if you want.)

Screenshot 2: The player has bought the book, and the message "Libro comprato!" (Book bought!) is displayed. A red oval highlights this message.

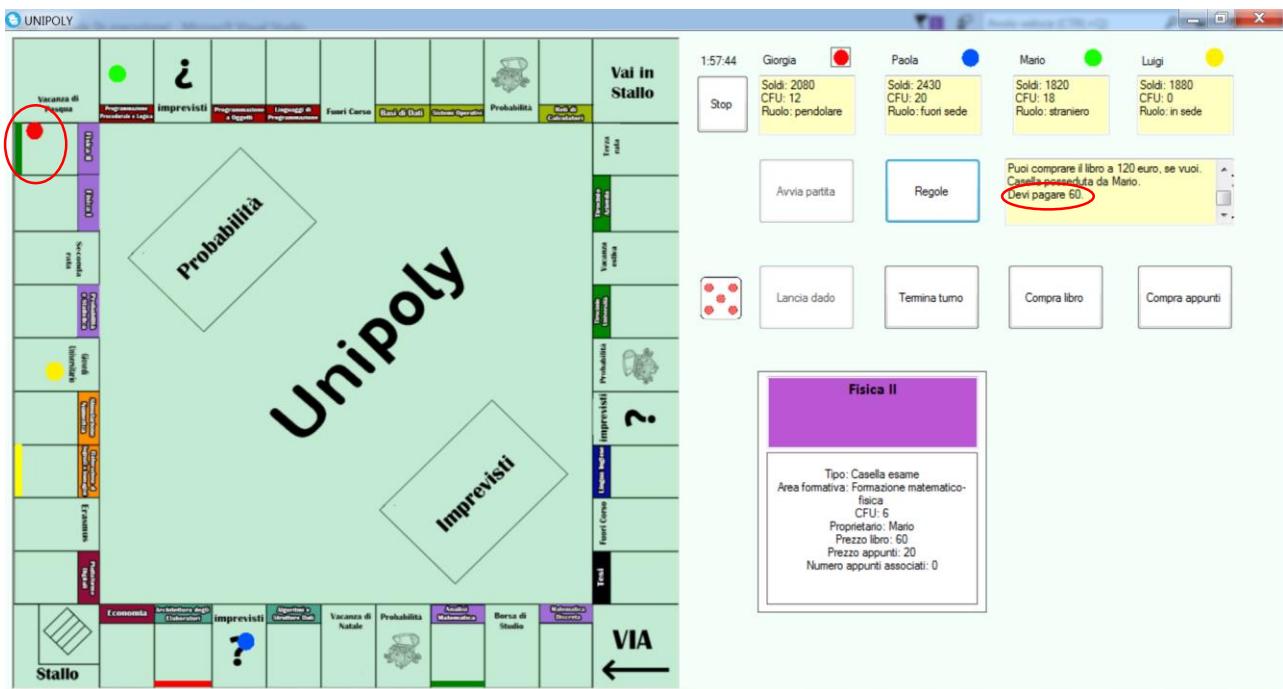
Screenshot 3: The player attempts to buy notes ("Buy appunti") but receives a message stating "Non puoi aggiungere appunti perché non possiedi tutti gli esami dell'area formativa" (You cannot add notes because you do not own all the exams in the area of formation). A red oval highlights this message.

The board also includes other elements like "Vai in Stallo" (Go to the Toilet), "Avvia partita" (Start game), "Regole" (Rules), "Lancia dado" (Roll dice), "Termina turno" (End turn), "Compr libro" (Buy book), "Compr appunti" (Buy notes), and a "Stop" button.

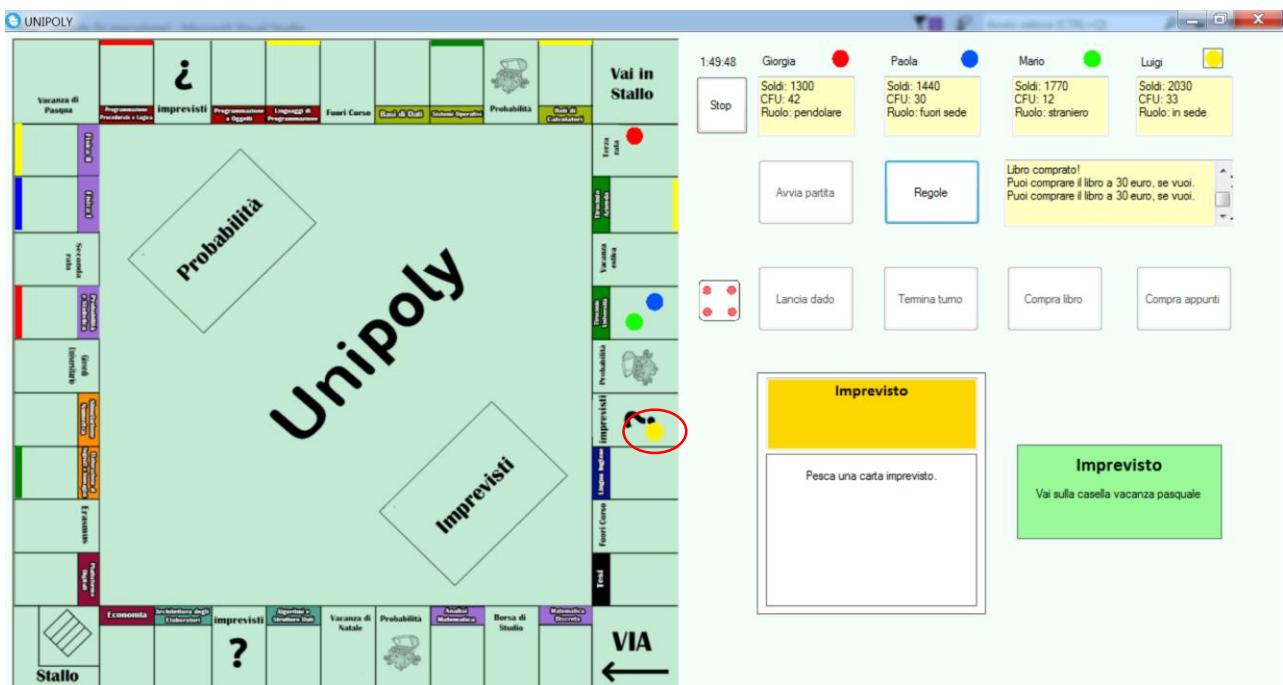
8. Si testa adesso il transito su una casella Stallo. Si noti (attraverso la sezione di flusso di gioco) che effettivamente al giro successivo il giocatore in stallo salta il turno.

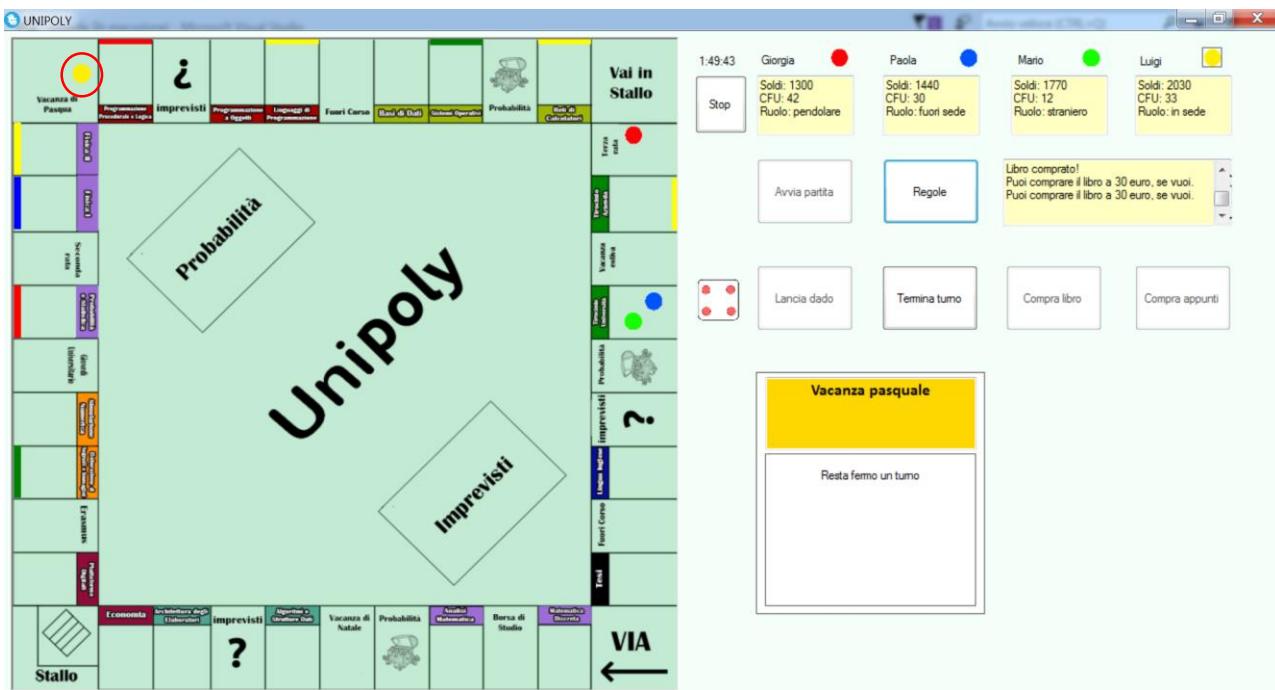


9. Si testa adesso cosa accade nel caso in cui un giocatore capiti sulla casella di qualcun altro. Si noti che dovrà pagare il prezzo del libro (l'aggiornamento avviene al termine del turno).

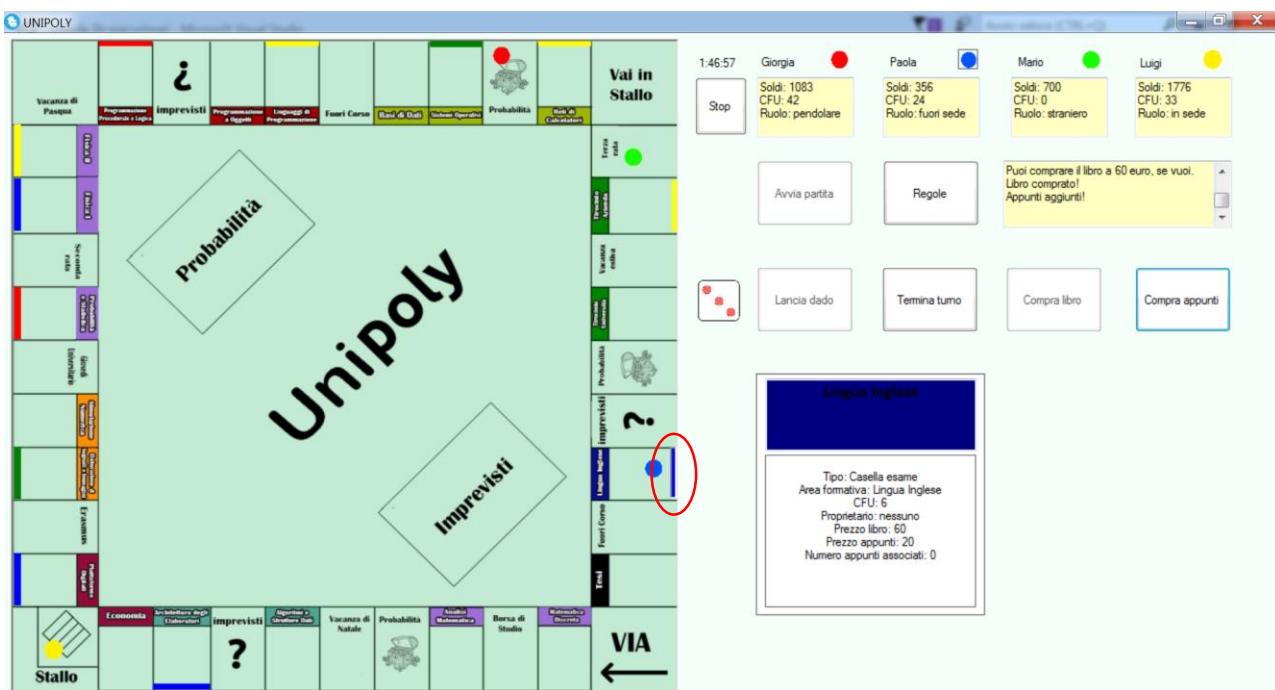


10. Ora si verifica cosa avviene quando si pesca una carta di tipo "Vai sulla casella *": compare la carta, viene avviato il timer movimento per dare al giocatore il tempo di leggerne la descrizione, infine la pedina viene ulteriormente spostata fino alla casella indicata dalla carta.

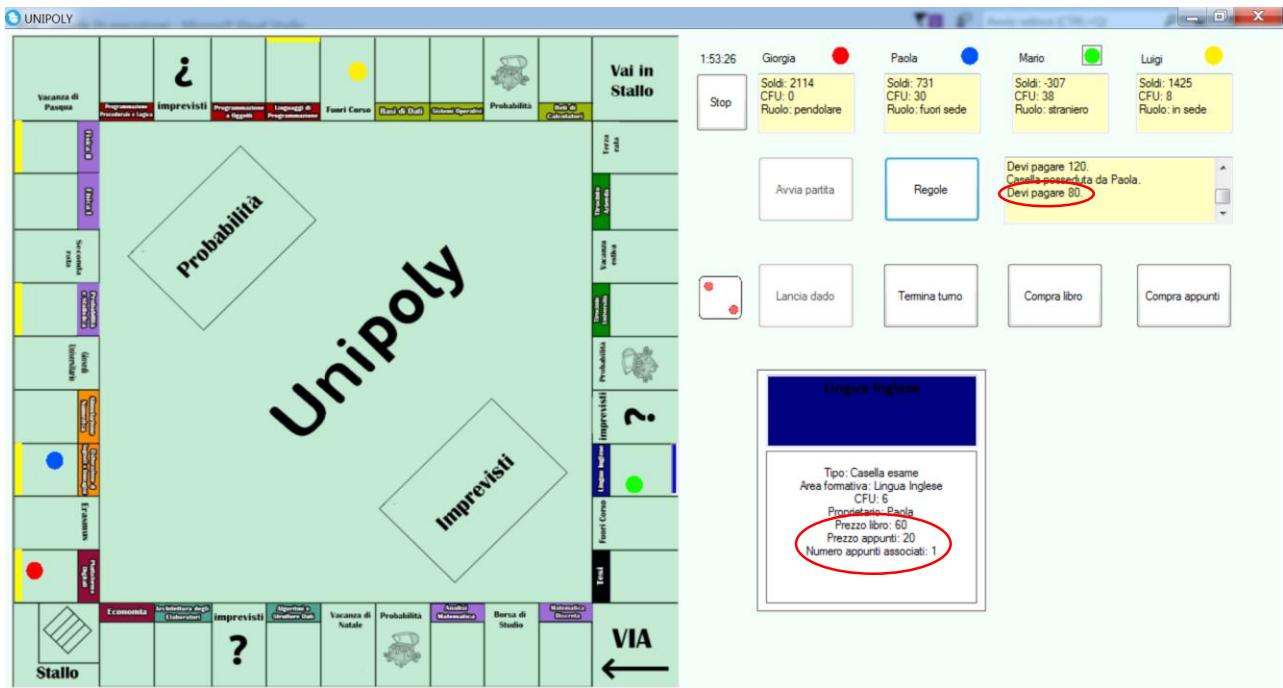




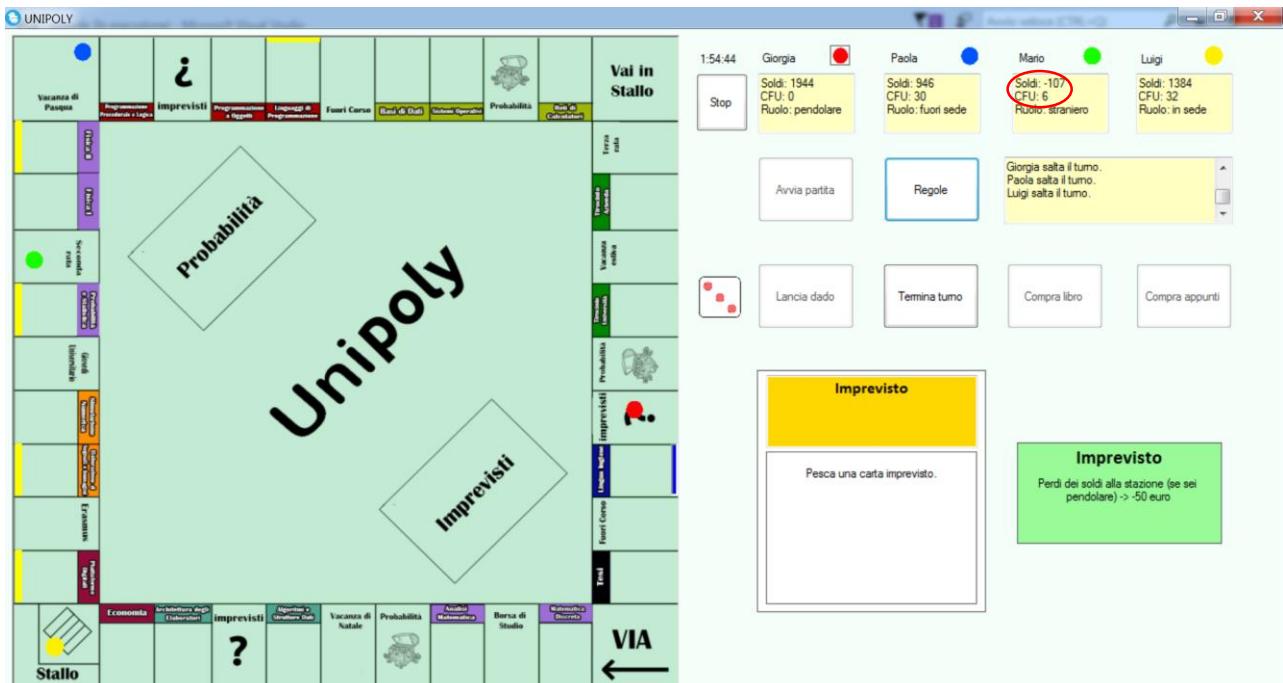
11. Si testa l'aggiunta di appunti ad una casella Esame nel caso in cui si posseggano tutti gli esami della stessa area formativa (in questo caso Lingua Inglese). La casella viene evidenziata modificandone il bordo inferiore. Il numero di appunti associati viene aggiornato premendo su "Termina Turno".

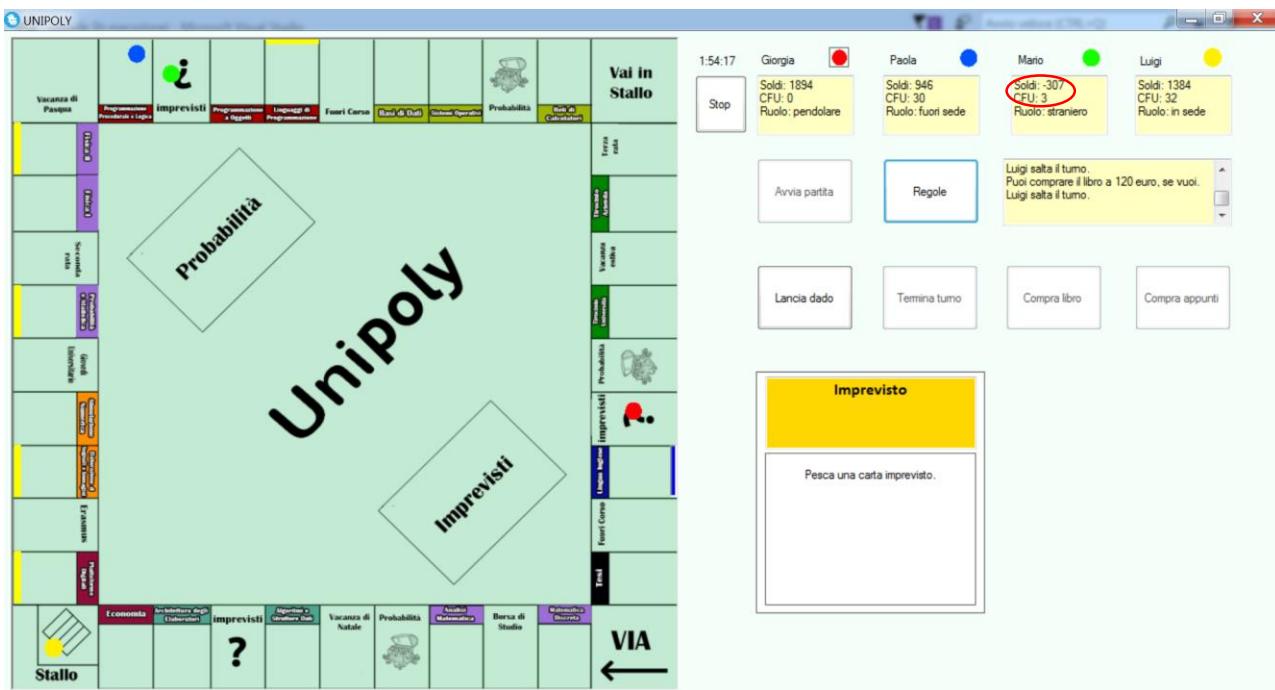


12. Si noti che adesso, se un altro giocatore transita su quella casella, dovrà pagare sia il prezzo del libro sia il prezzo degli appunti al proprietario della casella.



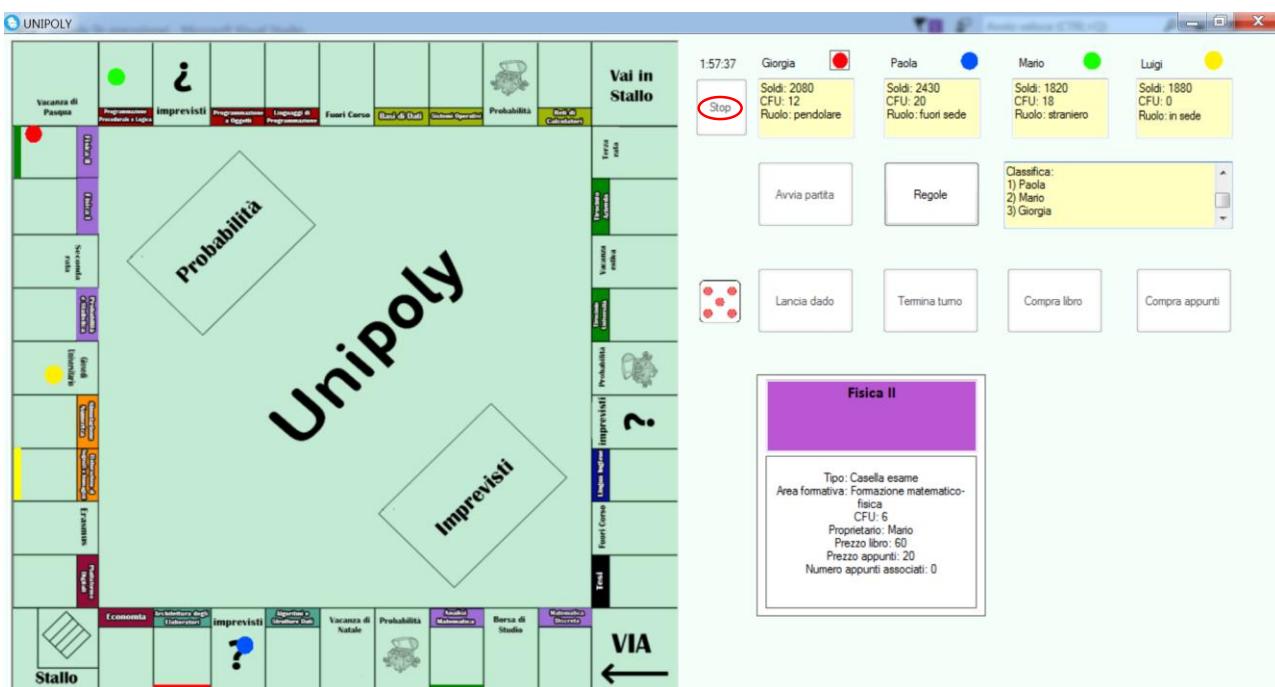
13. Si testa adesso cosa succede se un giocatore va in rosso: si noti come i CFU di Mario passano da 6 a 3 (vengono dimezzati).



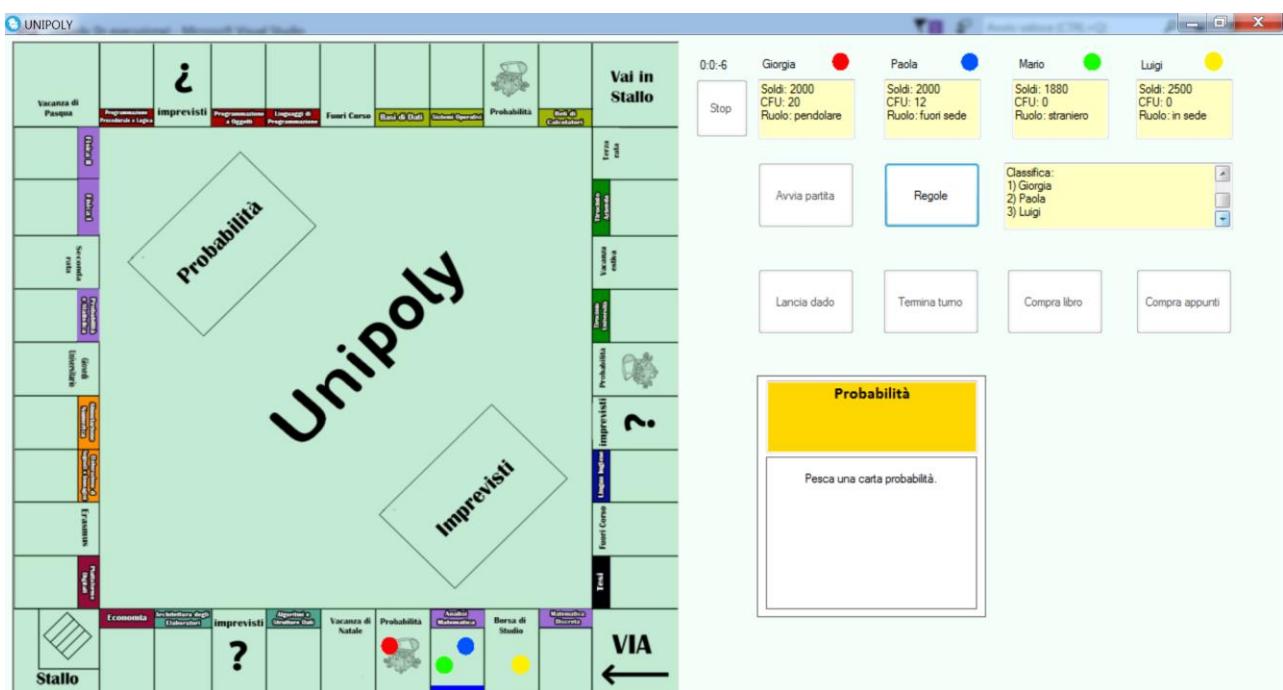
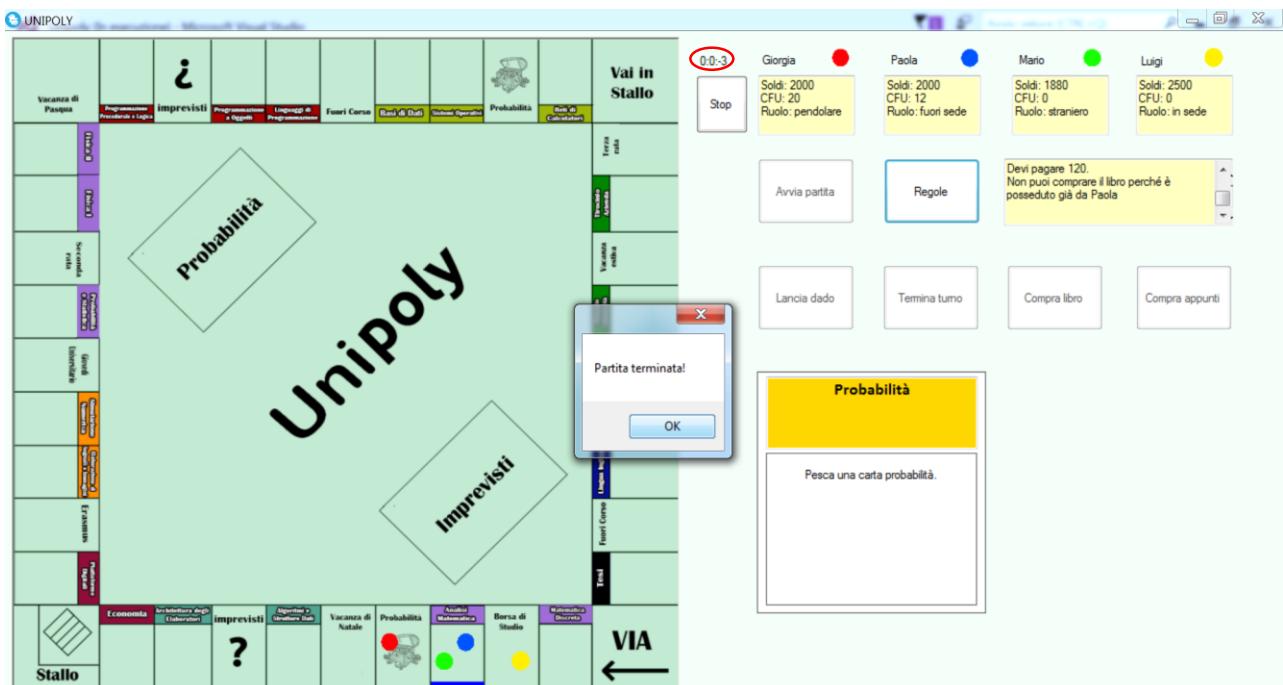


14. Si collauda adesso la terminazione della partita. In ogni caso verranno disabilitati tutti i pulsanti (eccetto “Regole”), verrà mostrata una MessageBox e la classifica sulla sezione di flusso di gioco. La terminazione può avvenire in vari modi:

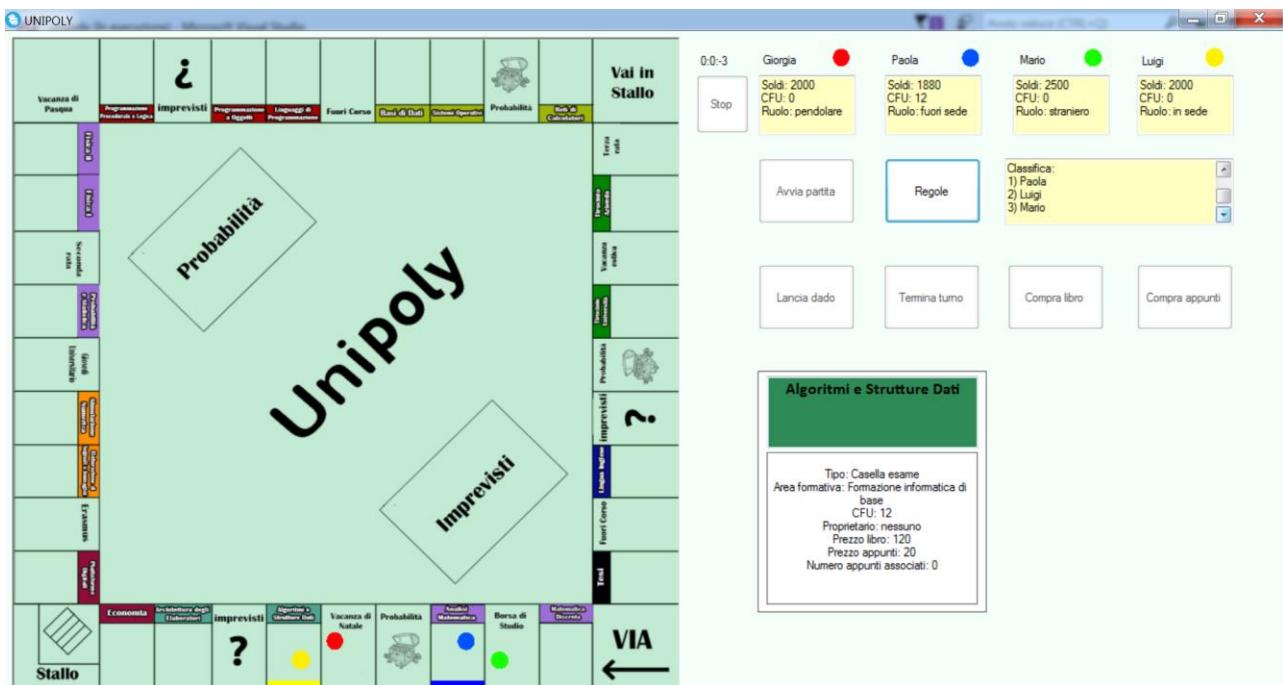
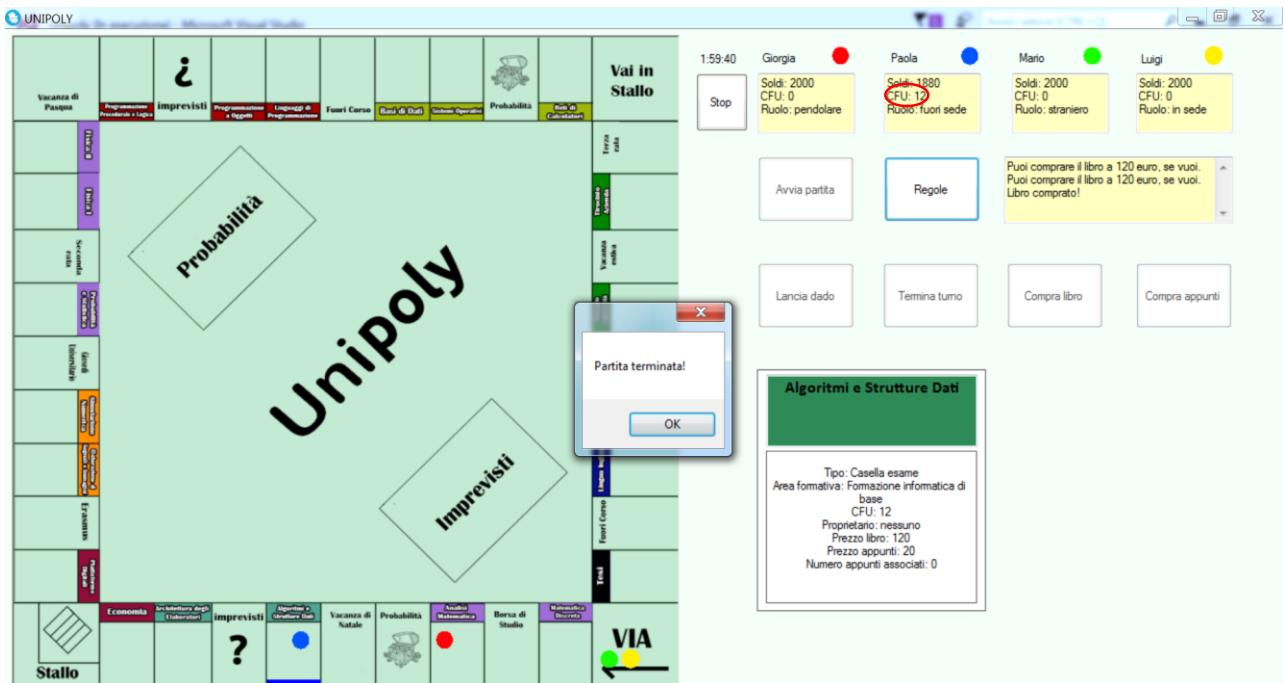
14.1 Si preme sul pulsante “Stop”.



14.2 Scade il tempo. Poiché il controllo sulla partita avviene al premere del pulsante "Termina Turno", il timer potrà andare in negativo (come in questo caso).



14.3 Almeno un giocatore si laurea (ottiene più di 180 CFU). In questo caso, per accelerare il testing, abbiamo assunto che i CFU richiesti per la laurea siano solamente 12.



6. COMPILAZIONE ED ESECUZIONE

L'ambiente di sviluppo utilizzato è **Visual Studio 2015 Community** - framework .NET 4.5.2.

Sarà possibile compilare il gioco seguendo queste istruzioni:

- Estrarre la cartella Unipoly dal file .rar Unipoly.rar
- Doppio click sul file con estensione .sln (Microsoft Visual Studio Solution)
- Cliccare sul menù a tendina Compila -> Compila soluzione

Dopo aver terminato la compilazione sarà possibile eseguire l'applicativo raggiungendo il file eseguibile in questo modo:

- Navigare in **Unipoly/UnipolyGUI/bin/Release**
- Doppio click sull'eseguibile UnipolyGUI.exe

Requisiti minimi:

- Processore 1 GHz o superiore con architettura x86 o x86_64
- 512 Mb di memoria principale
- 600 Mb di spazio su disco per sistemi a 32 bit
- 1.5 Gb di spazio su disco per sistemi a 64 bit
- Sistema operativo Microsoft Windows XP Home Edition o successivo
- Piattaforma .NET 4.5.2 o superiore

In particolare il software è stato eseguito e collaudato sulle seguenti macchine con hardware e software differenti:

Sistema Operativo	RAM	Processori logici	Architettura
Windows 7	16 Gb	8	64 bit
Windows 10	8 Gb	4	64 bit
Windows 10	6 Gb	4	64 bit
Windows 10	4 Gb	4	64 bit