

PROGETTO DI PROGRAMMAZIONE AD OGGETTI 2010

Nicola Moretto

SCELTE PROGETTUALI	3
SCELTE DI IMPLEMENTAZIONE	4
ESTENDIBILITÀ.....	4
SEPARAZIONE LOGICA/GRAFICA.....	6
ARCHITETTURA DELLE CLASSI.....	7
INTRODUZIONE.....	7
GAMEWELCOME.....	7
<i>GWPlayer</i>	7
<i>GWOptions</i>	7
GAMECORE.....	7
GAMETABLE.....	8
GTCASELLA.....	9
<i>GTAlbergo</i>	10
<i>GTPlayer</i>	10
CASELLE LOGICHE.....	10
<i>GCCasella</i>	10
<i>GCEdificabile</i>	10
<i>GCIpotecabile</i>	11
<i>GCNonEdificabile</i>	11
<i>GCCameraCommercio</i>	11
<i>GCPolizia</i>	11
<i>GCTribunale</i>	11
<i>GCIstituzione</i>	11
<i>GCSocieta</i>	11
GCPLAYER.....	12
TERMINAZIONE DELLA PARTITA.....	13
VALORI IN GIOCO.....	14

Scelte progettuali



La Banca. Nel Monopoli, la Banca gestisce unicamente il flusso di denaro in uscita (versato ai giocatori) o in entrata (versato dai giocatori, si tratti di pagamento di pedaggi, acquisto di vie e quote societarie, ...); alla luce dell'esiguo ruolo attivo nel gioco, si è deciso di non implementarla ma di permettere qualsiasi transazione economica compatibile con le finanze dei giocatori coinvolti; per farsi un'idea delle dinamiche economiche, si può pensare alla Banca come ad un deposito di liquidità virtualmente illimitato (sia in ingresso che in uscita). Tale scelta risponde inoltre ad un'esigenza di implementazione: come si è detto, la Banca, nel corso di una partita, può arrivare ad incassare dai giocatori cifre molto elevate e questa situazione è soggetta a potenziale *overflow*, con conseguenti problemi nella gestione del denaro depositato. Per approfondire le modalità di gestione delle transazioni economiche, leggere la sezione *GCPlayer*.

Creazione della partita. All'avvio del programma ai giocatori sarà presentata una finestra per la configurazione dei parametri necessari alla creazione della partita: i nomi (almeno 2 caratteri, esclusi gli spazi), le opzioni di gioco disponibili, il capitale iniziale e il numero massimo di turni (per limitare a priori la durata della partita). Mediante dei *tooltip* si forniranno agli utenti informazioni sul significato delle opzioni, sui vincoli al formato dei nomi e sulle altre grandezze da specificare; per il numero di turni e il capitale verranno scelti dei valori di default (per maggiori informazioni, consultare la *Tabella 6*).

Tavolo da gioco. L'interazione dell'utente con il programma avverrà mediante una finestra, che modellerà il tavolo da gioco del Monopoli; sarà possibile visualizzare il percorso di 20 caselle, che si snoderà lungo i bordi esterni, mentre al centro avremo un'area dedicata alle statistiche di gioco (informazioni sul giocatore e sul turno corrente) e una alle azioni disponibili (contestuali al tipo di casella su cui è in sosta il giocatore attivo).



Dinamiche di gioco. La gestione dei turni e di tutto ciò che concerne l'aggiornamento delle statistiche del gioco (caselle, giocatori, ...) verrà affidata ad una classe, che dovrà rivestire il ruolo di *intermediario* tra l'interfaccia grafica e la parte logica, mantenendole aggiornate (ossia garantendo la consistenza delle informazioni) e gestendo opportunamente le richieste dell'utente.

Caselle grafiche. Per implementare al meglio le dinamiche di gioco, caselle grafiche e logiche verranno definite separatamente; in tal modo sarà possibile creare un unico tipo di casella grafica polivalente e modulare, ossia capace – a seconda della casella logica corrispondente – di mostrare solo le informazioni ad essa correlate. L'aspetto assomiglierà alle classiche caselle del Monopoli: nella fascia inferiore gli indicatori della presenza di un giocatore in sosta, al centro il nome e l'eventuale valore¹ (solo per alcune caselle) e nella parte superiore incune informative.

Caselle logiche. Per quanto concerne la parte logica, sarà ragionevole definire una gerarchia dei tipi di caselle (in principio, si possono suddividere le caselle in *edificabili* e *non edificabili*), che sarà rispecchiata nel codice da una gerarchia di classi, modellata mediante ereditarietà; resta da definire come gestire il diverso comportamento che alcuni tipi di caselle assumono qualora vengano selezionate le opzioni disponibili (considerando che possono essere selezionate in qualsiasi combinazione, indipendentemente le une dalle altre); tale *indipendenza* tra le opzioni rappresenta un requisito indispensabile da conservare nella modellazione (anche per ragioni di espandibilità) e perciò si è preferito derivare ulteriormente le classi interessate da tali opzioni, creando delle sottoclassi ad hoc, opportunamente ridefinite per rispecchiare i differenti comportamenti. Questa scelta risulta funzionale, espandibile ed infine perfettamente adattabile alla situazione (potenziale) in cui una certa casella assuma diversi comportamenti, dipendenti dalla combinazione di 2 o più opzioni e quindi difficilmente gestibile altrimenti (siano p le opzioni distinte da cui dipende il comportamento di una casella, allora si avrebbero 2^p combinazioni possibili di comportamenti).

¹ Il valore viene mostrato se e solo se risulta diverso da 0; viene restituito dal metodo *getValore()* in *GCCasella*.

In generale, ogni qualvolta sia concesso o richiesto di compiere scelte legate alla giocabilità del programma o a valori assegnabili arbitrariamente (vedere la sezione *Valori in gioco*), si è cercato di mantenersi il più possibile aderenti alle specifiche della versione da tavolo del Monopoli.

Scelte di implementazione

Estendibilità

Numero di giocatori. Il gioco prevede la partecipazione di 4 giocatori, ma nella progettazione e nella scrittura del codice si è cercato di rendere possibile l'eventuale ampliamento del numero di giocatori con il minor numero di interventi al codice stesso, cercando il miglior compromesso tra la semplicità del codice e la flessibilità desiderata.



Numero di opzioni. Garantire l'estendibilità nel caso delle opzioni è più complesso, poiché ciascuna di esse ha o potrebbe avere requisiti ed effetti diversi sulla giocabilità e pertanto potrebbe richiedere interventi importanti sul codice; la gerarchia delle caselle logiche e lo sfruttamento del polimorfismo rispondono espressamente alla necessità di garantire un comportamento modulare ed espandibile, ossia la possibilità di variare la giocabilità senza dover intervenire nel codice che governa la gestione globale della partita (gestione dei turni, aggiornamento statistiche dei giocatori, ...).

All'avvio del programma, viene creato un oggetto della classe *GameCore* con due parametri: il *numero di giocatori* e il *numero di opzioni*. Il numero di giocatori viene utilizzato per impostare alcune variabili di gioco (numero di giocatori e colori assegnati), gestire i turni e istanziare i giocatori².

Ricevute le informazioni per la configurazione della partita, si procede alla creazione del tavolo da gioco³ virtuale e dei profili dei giocatori (con le informazioni correlate ossia nome, colore e capitale iniziale); la creazione del percorso richiede due fasi distinte: vengono infatti istanziati due percorsi, uno grafico (in *GameTable*) e uno logico (in *GameCore*). Il primo crea caselle identiche – in numero pari alla lunghezza del percorso – che mostreranno alcune o tutte le informazioni disponibili (se è edificabile ci sarà l'icona del proprietario e dell'albergo, altrimenti no e così via), a seconda della tipologia della casella corrispondente.



Figura 1: flusso del controllo all'avvio del programma

Numero e tipologia delle caselle. L'estendibilità – nel caso delle caselle – può riguardare sia il loro *numero* (aggiunta di nuove caselle edificabili, ...) sia la loro *tipologia* (caselle probabilità o imprevisti, ...).

Si è scelto di creare un solo tipo di casella grafica per garantire maggior semplicità ed espandibilità del programma; qualora si aggiungano nuove caselle (in numero e/o in tipo) non sarà necessario apportare importanti modifiche al codice. Il circuito vero e proprio – che tiene conto delle tipologie delle caselle coinvolte e delle opzioni di gioco selezionate dal giocatore in avvio di partita – viene creato e gestito separatamente in *GameCore*; questa porzione del codice risente evidentemente delle caselle disponibili e delle opzioni selezionate, perciò è qui che si concentreranno le modifiche in caso di espansioni.

Per garantire la massima espandibilità si è scelto innanzitutto di modellare le differenti tipologie di caselle con altrettante classi, derivate da una *classe base astratta* (*GCCasella*), al fine di costruirne una *gerarchia* facilmente espandibile; in *GCCasella* sono stati definiti alcuni metodi virtuali (da ridefinire opportunamente nelle sottoclassi), alcuni *signals* (per poter aggiornare la parte grafica, ossia l'istanza di *GameTable*) e attributi comuni a tutte le caselle.

² Per ciascun giocatore, viene creata un'istanza della classe *GCPlayer*, che memorizza tutte le informazioni di gioco ad esso correlate.

³ La classe *GameTable* modella la schermata principale, che accompagna i giocatori nel corso della partita.

Azione	Metodo
aggiornare le azioni disponibili	void aggiornaTask ()
conoscere il valore della casella	int getValore ()
sosta del giocatore	void sosta (GCPlayer*)
transito di un giocatore	void transito (GCPlayer*)
ottenere il nome della casella	const QString getName () const
	Slot
invia comunicazione all'utente	void sendMessage (const QString &)
disabilita il controllo delle quote	void sendQuota (const bool & disabled)
imposta i valori delle quote	void sendQuota (const int &, const int &, const int &)
imposta le azioni disponibili	void sendTask (bool *Task)

Tabella 1: metodi principali in GCCasella

Una volta modellate le classi, si procede – nella fase di **creazione della partita** – a decidere, per ogni casella, con quale classe debba essere modellata (detta altrimenti, di quale sottoclasse di *GCCasella* debba essere istanza), tenendo sempre in considerazione le opzioni di gioco scelte dall'utente (se un'opzione è selezionata, può essere necessario scegliere una classe differente nella gerarchia). La tabella seguente mostra la scelta della classe per ogni possibile combinazione di tipologia della casella e di opzioni selezionate dal giocatore; per maggiori informazioni sulla gerarchia delle classi, consultare la sezione *Caselle logiche*.

	Base	Ipoteca	Legale	Privatizzata	
Caselle edificabili	GCEdificabile	GCIpotecabile	GCEdificabile		
Camera di Commercio	GCCameraCommercio				
Stazione ferroviaria	GCNonEdificabile			GCSocieta	
Società dell'acqua potabile					
Società elettrica					
Municipio	GCNonEdificabile		GCNonEdificabile	GCIstituzione	
Polizia			GCPolizia	GCNonEdificabile	
Prigione			GCNonEdificabile		
Tribunale			GCTribunale	GCNonEdificabile	
Sede provinciale			GCNonEdificabile		GCIstituzione
Sede regionale					

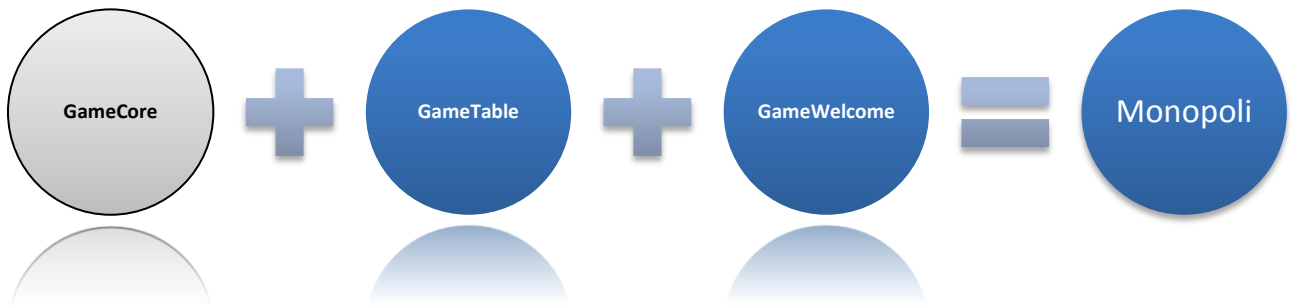
Tabella 2: gerarchia delle classi per le caselle logiche

Completata questa fase fondamentale della modellazione, è possibile definire – nei metodi di *GameCore* che gestiscono le dinamiche della partita – del **codice polimorfo**, che utilizza i metodi virtuali definiti in *GCCasella* e opportunamente ridefiniti nelle sottoclassi per interagire con qualsiasi casella logica; nelle circostanze in cui sia richiesto accesso ai membri propri (metodi pubblici, in generale) delle sottoclassi, si ricorre al **casting dinamico**: è il caso in cui si richiedano di effettuare azioni specifiche della casella (acquistare una casella edificabile, acquistare o rivendere quote di società privatizzabili, ...).

L'**espandibilità** in caso di inclusione di nuove tipologie di caselle è dunque chiara: sarà sufficiente inserire opportunamente la nuova casella nella gerarchia delle caselle logiche (sottoclassi di *GCCasella*), ridefinirne i metodi e aggiungere al circuito la nuova casella nelle posizioni desiderate; essendo la nuova casella sottotipo di *GCCasella*, non si rende necessario alcun intervento nel codice che gestisce il flusso e le dinamiche del gioco.

Separazione logica/grafica

Per garantire la separazione della parte logica da quella grafica, si è costruito il programma intorno a 3 classi fondamentali: *GameCore*, *GameTable* e *GameWelcome*; tutte le classi che compongono l'interfaccia **grafica** hanno il prefisso GW (*GameWelcome*) o GT (*GameTable*), mentre quelle che compongono la parte **logica** hanno prefisso GC (*GameCore*). Per ulteriori informazioni sull'architettura delle classi, consultare la sezione *Architettura delle classi*.



Logica. *GameCore* rappresenta la classe attorno a cui è costruito l'intero programma; all'avvio – come illustra *Figura 1: flusso del controllo all'avvio del programma* – viene creata per prima un'istanza di tale classe, che poi provvede a configurare la partita e a gestirla nella sua intera durata. Ogni qualvolta sia richiesto un aggiornamento della parte grafica (in seguito alla modifica delle statistiche di gioco) o della parte logica (in seguito all'interazione dell'utente con il programma), la richiesta passa attraverso l'istanza di *GameCore*, con il risultato di isolare – nella maniera migliore possibile – la parte logica dalla parte grafica; inoltre *GameCore* contiene le statistiche del gioco (lista giocatori, percorso logico, ...) e gestisce l'avanzamento del turno, la terminazione della partita e ogni altro aspetto della stessa. Per garantire la separazione tra grafica e logica si è cercato inoltre di ridurre al minimo l'impiego – nelle classi logiche – di funzionalità caratteristiche della libreria Qt (ad esempio il meccanismo di *signals* e *slots*, indispensabile per la gestione dell'interazione dell'utente con l'interfaccia grafica, e alcuni tipi di dati).

Grafica. *GameWelcome* modella la schermata che viene presentata all'utente per la configurazione della partita, mentre *GameTable* rappresenta la finestra che accompagna l'utente nell'intero corso della partita e attraverso la quale egli interagisce con il programma; ogni interazione dell'utente viene catturata e gestita da *GameCore*, che provvede ad aggiornare le statistiche ed eseguire le operazioni opportune, sia direttamente sia – è il caso della sosta di un giocatore su una casella – eseguendo le istruzioni specificate dalla casella medesima (sfruttando il polimorfismo e i metodi virtuali). Così, ogni qualvolta un giocatore sosta su una casella, viene invocata la versione di un opportuno metodo⁴ ridefinito in ciascuna delle sottoclassi di *GCCasella*.

Un aspetto fondamentale per il conseguimento dell'indipendenza logica-grafica è la distinzione tra le **caselle grafiche** (visibili nel tavolo da gioco e istanze della medesima classe *GTCasella*) e le **caselle logiche**, che incorporano le informazioni e i comportamenti specifici per ciascuna tipologia di casella (edificabile, non edificabile, ...). Tale distinzione rende possibile modificare le prime senza dover intervenire nel codice delle seconde (e viceversa), a condizione che le modifiche apportate alle une non richiedano (da un punto di vista progettuale) specifici interventi alle altre; ad esempio, l'introduzione di un nuovo tipo di casella modellata da una particolare classe potrebbe richiedere che nelle caselle grafiche sia inclusa una nuova informazione ad esse correlata; tuttavia, in linea generale, non dovrebbero risultare necessarie revisioni del codice.

Il meccanismo signal/slot della libreria Qt è stato impiegato per gestire la terminazione dell'applicazione (*QCoreApplication::quit()*) e l'interazione dell'utente mediante l'interfaccia grafica (*QPushButton::clicked()* per la pressione dei pulsanti).

⁴ *virtual void GCCasella::sosta(GCPlayer*)*

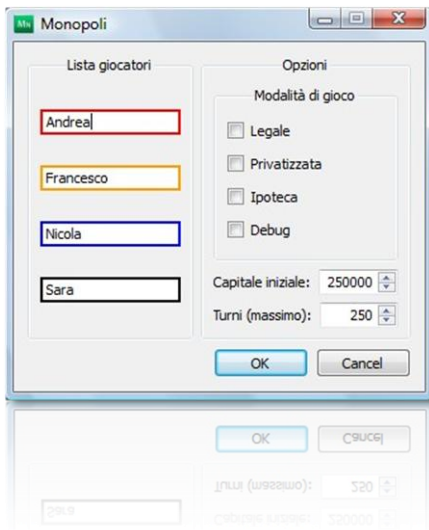
Architettura delle classi

Introduzione⁵

Come si è già spiegato, il programma è costruito su 3 classi fondamentali: *GameCore*, *GameWelcome* e *GameTable* e da un insieme di classi ad esse collegate, ragion per cui i loro nomi hanno prefisso GC, GT o GW ad indicare la classe di riferimento; si sottolinea che tale legame è puramente concettuale e nulla ha a che vedere con eventuali relazioni ereditarie.

GameWelcome

La classe modella la schermata che viene presentata all'avvio del gioco e alla creazione di ogni partita durante una sessione; i pulsanti visibili nella parte inferiore della schermata sono istanza di *QDialogButtonBox*, un contenitore per i pulsanti “standard” che compaiono generalmente nelle finestre di dialogo e dotati di comportamenti predefiniti (evidenti nella denominazione specifica e differenziata dei signal utilizzati per intercettare la pressione da parte dell'utente: *accepted()* per il pulsante OK e *rejected()* per Cancelli).



GWPlayer. Il widget che permette di inserire la lista dei giocatori è definito nella classe *GWPlayer* per ereditarietà dalla classe *QGroupBox*; la classe include 4 istanze della classe *QLineEdit*, che vengono ciascuno personalizzata assegnando al bordo il colore del giocatore (mediante *QStyleSheet*) e un valore di default per il nome del giocatore, inseriti in un layout verticale (*QVBoxLayout*).

GWOptions. La classe *GWOptions* modella il widget che permette di selezionare le **opzioni** di gioco desiderate (4 istanze di *QCheckBox*), impostare il **capitale** assegnato a ciascun giocatore (*QSpinBox* con valore di default 250000, minimo 100000, massimo 1000000) e il **numero massimo di turni** (*QSpinBox*, valore di default 250, minimo 100, massimo 1000).

La pressione del pulsante **OK** determina l'invocazione del metodo *raccoltaDati()*, che si occupa – nell'ordine – di:

1. prelevare i nomi dei giocatori => *GWPlayer::getNome*⁶
2. controllare che i nomi dei giocatori siano formati da almeno 2 caratteri (spazi esclusi), altrimenti viene presentata una finestra di segnalazione dell'errore, che invita gli utenti a controllare che i nomi inseriti rispettino tali requisiti;
3. controllare le opzioni selezionate => *GWOptions::isLegale*, *::isPrivatizzata*, *::isIpoteca*, *::isDebug* restituiscono un valore booleano che comunica se l'opzione corrispondente è stata selezionata;
4. prelevare il capitale e il numero massimo di turni specificati => *GWOptions::getCapitale*, *::getTurni*

Al termine della procedura viene inviato un segnale all'istanza di *GameCore*, cui vengono passati i valori appena raccolti affinché si proceda con la creazione della partita; la pressione del pulsante **Cancel** determina invece la terminazione del programma

GameCore

GameCore viene istanziata all'inizio del gioco con 2 parametri (nomi dei giocatori e opzioni selezionate), con i quali vengono immediatamente settate le variabili della partita (numero di giocatori e opzioni, lunghezza del percorso, turni totali e rimanenti, giocatori rimasti) e viene immediatamente istanziata *GameWelcome*.

⁵ Le schermate sono state catturate in Windows Vista e Ubuntu 10.04 LTS. I nomi dei metodi – per brevità – non includono la lista dei parametri e il tipo di ritorno.

⁶ Il metodo, prima di restituire il nome del giocatore, elimina eventuali spazi superflui presenti all'inizio e alla fine della stringa prelevata.

Ricevute da quest'ultima le informazioni necessarie a configurare la partita, viene invocato il metodo *creaPartita(...)*, cui vengono passati il numero dei giocatori e delle opzioni, il capitale iniziale e il numero massimo di turni.

Il corpo del metodo esegue quindi le seguenti operazioni:

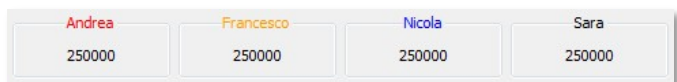
5. vengono reinizializzate alcune variabili della partita (nel caso sia stata creata una nuova partita mentre era in corso un'altra);
6. vengono creati i profili dei giocatori, corrispondenti ad altrettante istanze della classe *GCPlayer*⁷, e vengono memorizzate le opzioni selezionate;
7. viene creato il circuito logico, selezionando le opportune classi da utilizzare per ciascuna casella del percorso, tenendo conto della distribuzione di queste ultime (v. sezione *Valori in gioco*) e delle opzioni selezionate dall'utente (v. *Tabella 2: gerarchia delle classi per le caselle logiche*);
8. viene invocato il costruttore di *GameTable*, richiedente come parametri i nomi dei giocatori, le opzioni scelte, il numero massimo di turni e un puntatore all'istanza di *GameCore*.

Al termine della procedura, i giocatori avranno dinanzi a sé la tavola da gioco virtuale e saranno pronti ad iniziare la partita; da ora in poi ogni interazione dell'utente avverrà mediante tale finestra.

GameTable

La classe *GameTable* rappresenta il tavolo da gioco su cui si svolge la partita; il costruttore – invocato da *GameCore* – provvede a definire modularmente l'interfaccia con cui interagiranno i giocatori nell'intero corso della partita.

GTPlayer. La classe modella il *widget* al centro in basso che mostra il capitale corrente di ciascun giocatore; il costruttore richiede che gli vengano



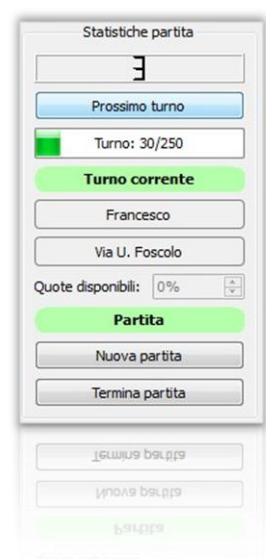
Andrea	Francesco	Nicola	Sara
250000	250000	250000	250000

passati il numero e i nomi dei giocatori e il capitale iniziale di ciascuno, quindi provvede a creare – per ogni giocatore – un'istanza di *QGroupBox* avente come titolo il nome del giocatore (il colore del font corrisponde al colore del giocatore) e un'etichetta (*QLabel*) che mostra il capitale corrente. Tale valore è soggetto ad aggiornamento in occasione di ogni transazione finanziaria dei giocatori; è pertanto disponibile un metodo pubblico - *aggiornaCapitale(GCPlayer* giocatore, const int& somma)* – che permette di aggiornare a *somma* il capitale di *giocatore*.

GTStatistiche. Le informazioni e le statistiche di gioco sono raccolte nel *widget* visibile sulla'area sinistra della finestra principale e raggruppate in tre aree (dall'alto in basso): il progresso nella partita, le informazioni sul turno corrente e la terminazione della partita.

In **alto** è visibile il *widget* che mostra l'ultimo valore uscito del dado virtuale (*QLCDNumber*), lanciato ad ogni pressione del pulsante *Prossimo turno* che, come suggerisce il nome, permette di procedere al turno successivo; al di sotto di esso, è presente una barra dei progressi (*QProgressBar*) che mostra i turni giocati e totali.

Al **centro** sono riportate le informazioni essenziali sul turno corrente: il giocatore attivo, la casella presso cui è in sosta e un selettore del numero di *quote* (utilizzabile solo nelle caselle in cui sia possibile acquistare o vendere quote di società o alberghi); quest'ultimo selettore (*QSpinBox*) viene aggiornato (quando possibile) in modo tale da fornire solo valori validi nel contesto delle azioni disponibili per il giocatore in sosta.

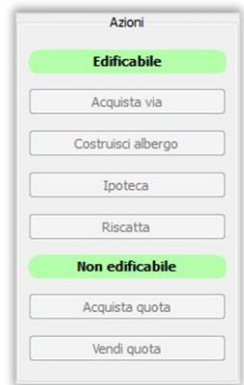


⁷ Il costruttore di *GCPlayer* richiede - come parametri – il nome del giocatore, il capitale iniziale, il colore, il suo numero e il puntatore all'istanza di *GameCore*, per poter interagire con quest'ultima.

In **basso** troviamo due pulsanti (*QPushButon*) che consentono di terminare la partita (“*Termina partita*”) ed eventualmente crearne una nuova (“*Nuova partita*”); in quest’ultimo caso, all’utente verrà presentata nuovamente la schermata iniziale per la configurazione della partita, con gli ultimi valori utilizzati per i parametri (nomi dei giocatori, opzioni di gioco selezionate, capitale iniziale e massimo numero di turni).

Un clic sul pulsante “**Prossimo turno**” provoca l’invocazione del metodo *GameTable::showNumber*, che calcola un numero casuale⁸ compreso tra 1 e 6 (simulando l’esito possibile del lancio di un dado), aggiorna il valore mostrato sul *widget* collocato sopra il pulsante (*GTStatistiche::aggiornaDado*) e invoca il metodo *GameCore::avanzaTurno* (passando in entrambi i casi il valore calcolato come parametro), che gestisce l’avanzamento del turno effettuando le seguenti operazioni:

1. se è rimasto un solo giocatore attivo o se sono esauriti i turni, viene conclusa la partita e dichiarato il vincitore (*GameCore::terminaPartita* – per saperne di più, leggere *Terminazione della partita*);
2. scorre ordinatamente la lista dei giocatori sino a trovare il prossimo, scartando quelli *non attivi* (i.e. esclusi dalla partita) o che abbiano dei *turni di attesa* da scontare⁹ (che vengono decrementati di 1);
3. si calcola la casella di destinazione del giocatore (invocando – per ogni casella in cui transita – il metodo *GCCasella::transito*), si aggiornano di conseguenza le statistiche inerenti la posizione del giocatore sul circuito (*GCPlayer::setPosizione* comunica al giocatore la sua nuova posizione, *GameCore::muoviGiocatore* ne aggiorna l’icona - sul tavolo da gioco – che rivela la casella su cui sosta);
4. vengono aggiornati la barra di avanzamento dei turni (*GTStatistiche::aggiornaPartita*), il nome e la posizione del giocatore mostrati (*GTStatistiche::aggiornaTurno*);
5. viene invocato il metodo *GCCasella::sosta* che – opportunamente ridefinito nelle sottoclassi - determina le azioni disponibili all’utente nel proprio turno secondo il tipo della casella in cui si trova e la situazione della partita;



GTask. Ogni tipo di operazione economica che l’utente intende effettuare nel corso della partita, verrà compiuta attraverso i pulsanti (istanze di *QPushButon*) messi a disposizione da tale *widget*; in ogni turno di gioco, a seconda della *casella* in cui si fermerà il giocatore, delle *opzioni di gioco* abilitate e della *situazione di gioco*, solo alcune azioni (o anche nessuna) potrebbero risultare disponibili. Le azioni sono raggruppate in due categorie, a seconda delle caselle che coinvolgono:

6. **Edificabili:** acquista via, costruisci, ipoteca o riscatta l’albergo (le ultime due disponibili solo se l’opzione *ipoteca* è attiva);
7. **Non edificabili:** acquista o vendi quote delle società (disponibili solo se l’opzione *privatizzata* è attiva);

e – per default – risultano indisponibili; ciascuna tipologia di casella specificherà (nella sottoclasse di *GCCasella* corrispondente) quali azioni debbano essere disponibili ogni qualvolta un giocatore soste su di essa, ridefinendo opportunamente lo slot *void sendTask(bool *Task)*.



Figura 2; sequenza di chiamate per l’aggiornamento delle azioni disponibili

GTCasella

Il circuito su cui si muovono i giocatori è modellato mediante istanze della classe *GTCasella*, che attinge le informazioni necessarie (nome delle caselle, valore, ...) dalle classi logiche mediante *GameCore*; il layout della casella è in 3 livelli (dall’alto in basso):

⁸ Se l’opzione *Debug* è attiva, il valore calcolato è sempre pari a 1.

⁹ Essendoci – alla luce del punto 1 – almeno 2 giocatori attivi, se entrambi hanno turni di attesa pendenti la ricerca proseguirà riducendoli sino a che uno dei giocatori possa procedere.

1. in alto troviamo un'icona rettangolare che assume un'aspetto diverso a seconda dell'informazione che intende comunicare: *invisibile* (default), *parzialmente trasparente* e *opaca*; gli ultimi due stati sono riservati alle caselle edificabili, che grazie ad essi comunicano rispettivamente che un giocatore (individuato dal colore assunto dall'icona medesima) ne è proprietario o vi ha costruito un albergo;
2. il nome e il valore della via (*QLabel*);
3. le icone circolari rivelano la posizione dei giocatori (una per ciascuno, del colore del giocatore);

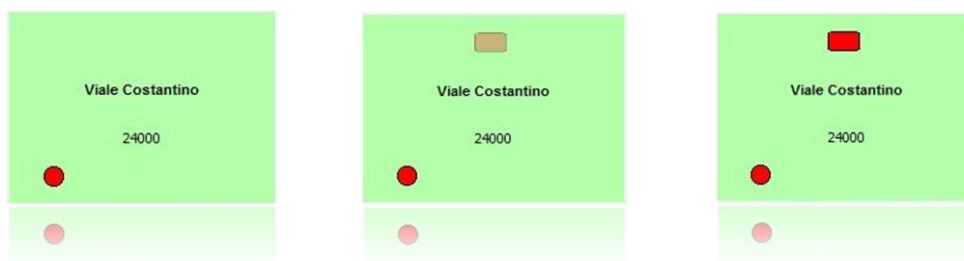


Tabella 3: casella con l'icona in 3 stati (invisibile, trasparente, opaco)

La classe fornisce due metodi pubblici rispettivamente per aggiungere (*GTCasella::addSostaGiocatore*) o rimuovere (*GTCasella::removeSostaGiocatore*) l'icona di sosta di un giocatore e un metodo (*GTCasella::aggiornaStato*) per aggiornare le informazioni per gestire lo stato dell'icona nell'area superiore (istanza della classe *GTAlbergo*).

GTAlbergo. Tale classe modella l'icona a 3 stati visibile nell'area superiore della casella e dispone di alcuni metodi per gestirne lo stato; in particolare, essa necessita di sapere se la casella ha un proprietario (e –in tal caso – il colore che lo contraddistingue) e se è stato anche costruito un albergo. Queste informazioni vengono aggiornate rispettivamente mediante i metodi *GTAlbergo::setProprietario*, *GTAlbergo::setColore* e *GTAlbergo::setAlbergo*.

GTPlayer. Ciascuna delle icone che rappresentano i giocatori in sosta è istanza della classe *GTPlayer* e viene inizializzata con il colore del rispettivo giocatore (in ordine da sinistra a destra); per aggiornare la presenza di un giocatore presso una casella, si invoca il metodo pubblico *GTPlayer::setSosta*, con un parametro booleano posto a *true* (il giocatore è in sosta) o *false* (il giocatore non è in sosta).

Caselle logiche

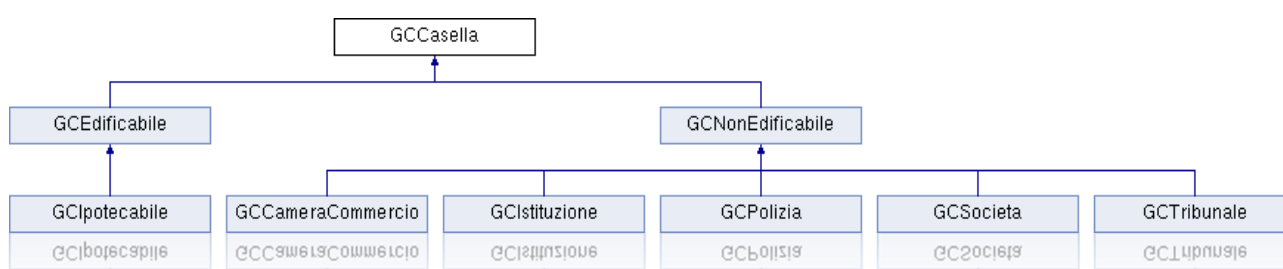


Figura 3: geachia delle sottoclassi di GCCasella

GCCasella. Il circuito logico è formato da istanze di sottoclassi di *GCCasella*, che implementano attributi e metodi specifici della tipologia di casella rappresentata e ridefiniscono i metodi della classe base, che consentono di gestire correttamente il turno di un giocatore sulla casella in questione; tali metodi sono visibili in *Tabella 1: metodi principali in GCCasella*. Il distruttore di *GCCasella* è virtuale così che – quando verranno deallocate le caselle logiche del percorso – venga invocato dinamicamente il corretto distruttore di default della sottoclasse.

GCEdificabile. La classe rappresenta – se l'opzione *Ipoteca* non è attiva – le caselle edificabili presenti sul percorso, che possono avere un proprietario e su cui può essere costruito un albergo (necessariamente in quest'ordine); tale tipologia di casella può rendere disponibili le azioni “*Acquista via*” e “*Costruisci albergo*” (rappresentate dagli omonimi pulsanti sulla finestra principale). Il **pedaggio** – che dipende dalla valore della

via e dell'albergo – può essere ottenuto invocando il metodo pubblico *GCEdificabile::pedaggio*, che lo calcola sommando la rendita della via e dell'albergo (se presente), differenti per ogni casella e riportati nella *Tabella 7*. L'albergo può essere costruito invocando il metodo *GCEdificabile::costruisciAlbergo*; in ogni istante è possibile verificare se sia presente mediante il metodo *GCEdificabile::isAlbergo*.

GCIpotecabile. Se l'opzione *Ipoteca* è attiva, le caselle edificabili sono istanziate dalla classe *GCIpotecabile* che – rispetto alla superclasse diretta *GCEdificabile* – aggiunge la possibilità di **ipotecare e riscattare quote dell'albergo** mediante – rispettivamente – i metodi *GCIpotecabile::ipoteca* e *GCIpotecabile::riscatta*; per effettuare tali operazioni vengono rese disponibili le azioni “*Ipoteca*” e “*Riscatta*”. È possibile riscattare o ipotecare (nello stesso turno e più volte) multipli del 10% del valore dell'albergo (fino al 100%); l'esatto ammontare della percentuale di albergo da ipotecare/riscattare dev'essere indicato mediante il selettore “*Quote disponibili*”, visibile nella parte sinistra della finestra di gioco principale; il valore di default è il minimo, ossia il 10%.

GCNonEdificabile. La casella *GCNonEdificabile* modella le caselle non edificabili – eccezion fatta per la Camera di Commercio – se non sono state selezionate le opzioni di gioco *Legale* e *Privatizzata*; quando un giocatore sosta su una di queste caselle, deve pagare un pedaggio (in alcune è nullo), per conoscere il cui importo (vedi *Tabella 8*) è possibile utilizzare il metodo *GCNonEdificabile::getPedaggio*. Da sottolineare che tali caselle non consentono alcuna interazione da parte dell'utente.

GCCameraCommercio. *GCCameraCommercio* differisce da *GCNonEdificabile* per il comportamento in caso di sosta e transito, trattandosi dell'unica classe che comporti degli effetti in caso di solo **transito** del giocatore, richiedendo quindi di ridefinire il metodo *GCCasella::transito*; inoltre – ad ogni sosta o transito del giocatore – gli viene versata una somma di denaro fissata arbitrariamente (pari a 15000).

Se l'opzione di gioco *Legale* è attivata, le caselle Polizia e Tribunale vengono rispettivamente modellate con le classi *GCPolizia* e *GCTribunale*.

GCPolizia. Se un giocatore sosta sulla casella Polizia, viene simulato il lancio di una moneta per determinare se debba fermarsi per un turno e pagare un ammenda; in caso affermativo, viene aggiornato il numero di turni di sosta mediante il metodo *GCPlayer::setTurniAttesa*.

GCTribunale. Se un giocatore sosta sulla casella Tribunale, viene simulato il lancio del dado per determinare il numero di turni di attesa (aggiornati con *GCPlayer::setTurniAttesa*); dovendo il giocatore ripartire dalla Prigione, la sua posizione sul percorso grafico dev'essere spostata nella nuova casella mediante il metodo *GameCore::muoviGiocatore*, che richiede come parametri il giocatore, la posizione corrente e quella di destinazione (in quest'ordine).

Se l'opzione di gioco *Privatizzata* è attiva, è possibile acquistare e vendere quote delle società; il valore di ciascuna è fissato arbitrariamente e riportato in *Tabella 8*.

GCIstituzione. In occasione della sosta di un giocatore sulla casella, è possibile *acquistare* – se disponibile – una quota della società associata (a multipli del 10%), selezionando la percentuale desiderata mediante il selettore “*Quote disponibili*” (la percentuale massima di quota acquistabile coincide con quella effettivamente disponibile); le quote disponibili e il costo unitario (pari al 10%) vengono comunicati al giocatore all'inizio del turno mediante l'apposita area dei messaggi, posta in alto al centro nella finestra principale.

GCSocieta. Un giocatore che soste su tale casella può – se ne possiede – vendere una percentuale della compartecipazione alla società (a multipli del 10%), indicando la percentuale da cedere mediante il selettore “*Quote disponibili*” (che rispecchia nel valore massimo la percentuale effettivamente posseduta dal giocatore) e premendo il pulsante “*Vendi quota*”; se il giocatore non possiede quote, nessuna interazione è possibile.

Acquista via	<i>GameCore::acquistaVia</i>	GCEdificabile::acquistaVia
Costruisci albergo	<i>GameCore::costruisciAlbergo</i>	GCEdificabile::costruisciAlbergo
Ipoteca	<i>GameCore::ipotecaAlbergo</i>	GCIpotecabile::ipoteca
Riscatta	<i>GameCore::riscattaAlbergo</i>	GCIpotecabile::riscatta
Acquista quota	<i>GameCore::acquistaQuota</i>	GCIstituzione::acquistaQuota
Vendi quota	<i>GameCore::vendiQuota</i>	GCIstituzione::vendiQuota

Tabella 4: gestione delle richieste di transazioni da parte dell'utente

GCPlayer

Ciascun giocatore è rappresentato nel gioco da un'istanza della classe *GCPlayer*, che memorizza le statistiche di gioco relative (aggiornabili e conoscibili mediante metodi pubblici) e gestisce le operazioni di prelievo o deposito del capitale e di acquisto delle caselle edificabili (la classe memorizza la lista dei possedimenti edificabili di cui l'autore è proprietario, aggiungibili mediante il metodo *GCPlayer::addPossedimento*).

Informazione	Conoscenza	Aggiornamento
Capitale ¹⁰	<i>GCPlayer::getCapitale</i>	-
Liquidità	<i>GCPlayer::capitale</i>	<i>GCPlayer::deposita</i> , <i>GCPlayer::preleva</i>
In partita	<i>GCPlayer::attivo</i>	<i>GCPlayer::terminaPartita</i>
Posizione	<i>GCPlayer::getPosizione</i>	<i>GCPlayer::setPosizione</i>
Turni attesa	<i>GCPlayer::getTurniAttesa</i>	<i>GCPlayer::setTurniAttesa</i>

Tabella 5: proprietà fondamentali e metodi disponibili

In occasione di una transizione economica che richieda il deposito o il prelievo di capitale viene rispettivamente invocato il metodo *GCPlayer::deposita* e *GCPlayer::preleva*, con argomento la somma in questione. In caso di prelievo, viene verificato che la somma richiesta sia minore del denaro a disposizione dell'utente, in caso contrario viene sollevata un'eccezione di tipo *ECC_CapitaleInsufficiente*, a segnalare che la transazione è fallita a causa della mancanza del denaro necessario per portarla a compimento.

I due possibili contesti in cui tale eccezione può venir sollevata e catturata sono:

- il giocatore doveva pagare un pedaggio o un'ammenda => viene escluso dalla partita e il suo capitale azzerato (*GCPlayer::terminaPartita*);
- il giocatore ha cercato di effettuare un acquisto (via, albergo, quota societaria, ...) => gli viene notificata l'impossibilità a procedere con la transazione.

Tale scelta permette di gestire uniformemente qualsiasi transizione economica che richieda un prelievo di denaro dal giocatore, intervenendo ad annullare la transizione nel momento esatto in cui viene rilevato che il giocatore non dispone di denaro sufficiente per portarla a compimento.

¹⁰ Il capitale include il denaro liquido e i possedimenti edificabili.

Terminazione della partita

La partita termina se si verificano una o entrambe le seguenti condizioni:

4. rimane un solo giocatore attivo;
5. non rimangono turni disponibili.

Nel primo caso, il vincitore coincide con il solo giocatore rimasto in gioco mentre nel secondo caso – se ci sono diversi giocatori attivi – viene calcolato il patrimonio di ciascuno e viene decretato vincitore colui per cui questo è massimo; in caso di parità, vengono decretati più vincitori *ex-aequo*; il patrimonio di un giocatore include: il capitale, il valore delle caselle edificabili (inclusi gli eventuali alberghi) e le quote di società possedute.

Il metodo che pone fine alla partita – *GameCore::terminaPartita()* – effettua le seguenti operazioni:

1. elimina ogni possibilità di interazione dell'utente con l'interfaccia grafica, disabilitando i pulsanti;
2. per ogni giocatore, procede a calcolare il patrimonio sommando il denaro liquido e il valore dei possedimenti edificabili (*GCPlayer::getCapitale*) agli eventuali¹¹ investimenti nelle società privatizzabili (*GameCore::getInvestimenti*) e lo confronta con il patrimonio massimo riscontrato sui giocatori già considerati;
3. il metodo *GameCore::getInvestimenti* somma il valore delle quote azionarie possedute da un giocatore in ogni società privatizzabile, pari al prodotto della percentuale posseduta (*GCSocieta::getQuotaGiocatore*) per il valore unitario (*GCSocieta::getQuota*);
4. verifica se il giocatore con il massimo patrimonio è solo (decretandolo unico vincitore) o se vi sono più giocatori con identico patrimonio (dichiarandoli tutti vincitori a pari merito);
5. la lista dei vincitori viene comunicata nell'area dei messaggi in alto al centro della schermata principale.

¹¹ Gli investimenti vengono calcolati solo se l'opzione di gioco *Privatizzata* è stata selezionata.

Valori in gioco

	Default	Minimo	Massimo	Incremento
Capitale iniziale	250000	100000	1000000	50000
Numero massimo di turni	250	100	1000	50

Tabella 6: valori dei parametri di configurazione della partita

Nome	Valore	Rendita	Valore albergo	Rendita albergo	# ¹²
Corso Garibaldi	35000	3700	15000	4500	11
Corso Leopardi	30000	3000	13000	3800	6
Corso Napoleone	28000	2700	12000	3300	13
Piazza Dante	25000	2500	11000	3300	18
Piazza Giulio Cesare	36000	3600	17000	4600	2
Via Manzoni	18000	2000	10000	2600	9
Via Roma	23000	2400	11000	3000	14
Via Ugo Foscolo	20000	2000	9000	2600	8
Viale Costantino	24000	2300	12000	3000	3
Viale Traiano	26000	2600	13000	3500	1

Tabella 7: valori assegnati alle caselle edificabili

Nome	Pedaggio	Quota (privatizzata)	Ammenda (legale)	#
Camera di Commercio	15000 ¹³	-	-	0
Stazione ferroviaria	2500	-	-	17
Società dell'acqua potabile	3000	-	-	7
Società elettrica	3200	-	-	4
Tribunale	-	-	-	12
Polizia	-	-	4500	19
Prigione	-	-	-	16
Municipio	-	6000	-	5
Sede regionale	-	5000	-	10
Sede provinciale	-	4500	-	15

Tabella 8: valori assegnati alle caselle non edificabili

¹² Rappresenta la posizione sul circuito (contando a partire dalla casella in alto a sinistra, in senso orario).

¹³ Il pedaggio viene versato al passaggio o alla sosta.