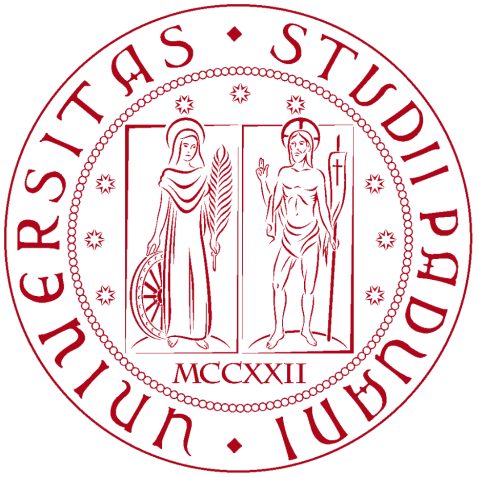


UNIVERSITÀ DEGLI STUDI DI PADOVA

UNIVERSITÀ DEGLI STUDI DI PADOVA
CORSO DI PROGRAMMAZIONE AD OGGETTI
A.A. 2023/2024

Picello Davide Matricola: 2034825

Relazione progetto PaO - Zoo Simulator

Componenti gruppo:

- **Picello Davide 2034825**
- Mahdi Mouad 2044222

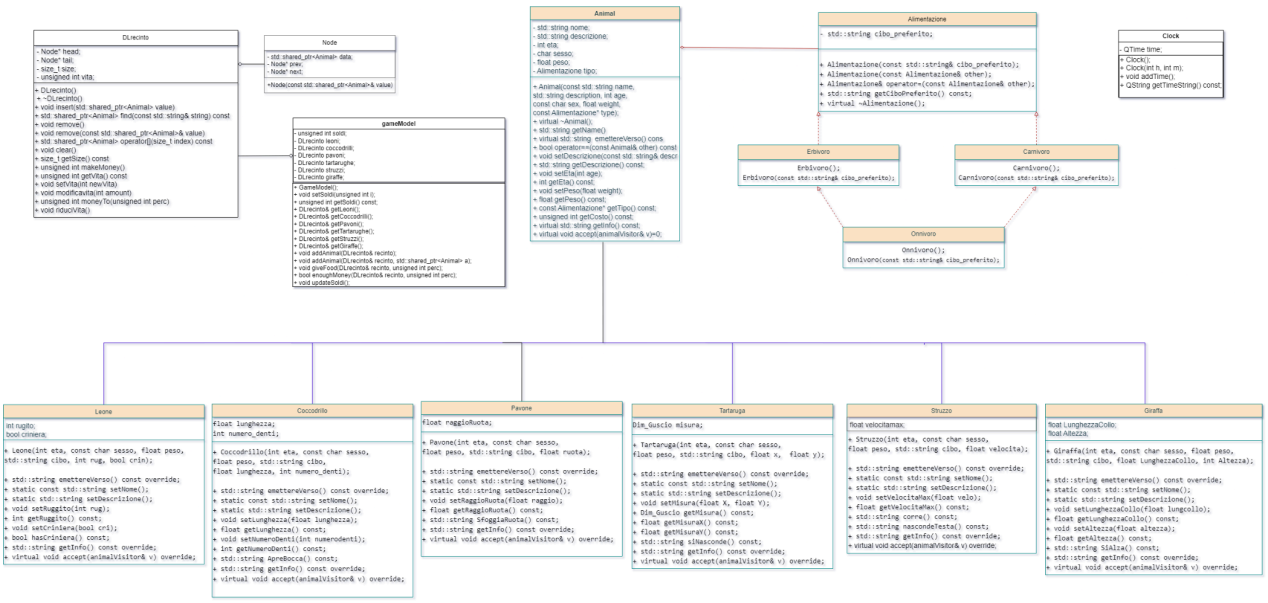
Tema

Il tema del progetto è, come si può intuire dal titolo, un piccolo videogioco a tema Zoo. Quest'idea è stata concepita, e poi successivamente scelta, proprio perché si presta bene all'utilizzo del polimorfismo e dell'ereditarietà.

Lo scopo del gioco è gestire il proprio zoo, potendo aggiungere animali, sfamandoli e potendo vedere alcune informazioni specifiche di ogni animale: alcune sono informazioni comuni come una descrizione riguardo al tipo di animale, ed alcune specifiche per ogni singolo componente, come, il nome ed il peso. Inoltre ogni tipo di animale ha degli attributi propri, per esempio, il leone, ha la potenza del suo ruggito.
Un animale ha un costo specifico per essere comprato, ma genera anche un profitto per ogni aggiornamento del timer di gioco, in quanto si presuppone che più animali attirino più visitatori.

Modello logico

La gerarchia delle classi degli animali ha una classe base virtuale "Animal" che comprende i metodi ed attributi comuni a tutti gli animali. Da questa derivano le classi derivate coccodrillo, giraffa, leone, pavone, struzzo e tartaruga, le quali hanno dei metodi ed attributi propri.
La classe Animal contiene anche una classe Alimentazione, inserita nelle prime fasi del progetto, anche se poi, oltre a dirci che tipo di alimentazione l'animale ha e qual'è il suo cibo preferito,non ha mai trovato una sua attuale utilità.
Infine troviamo anche il recinto, ovvero il container richiesto nelle specifiche del progetto, che è una lista doppiamente collegata dinamica che serve per tener traccia dei singoli animali istanziati e delle relative rimozioni. La scelta di una lista doppiamente collegata non era indispensabile, ma è stato deciso di lasciarla per puro scopo didattico, per differenziarsi da una semplice lista.



L'immagine a qualità originale si può trovare nella cartella.

Utilizzo non banale del polimorfismo

Per il soddisfacimento di questo vincolo abbiamo implementando la creazione dinamica delle varie finestre per ogni animale tramite un **visitor**.
In particolare, nella cartella *visitor*, i file *animalDetails* ed *animalVisitor* (che funzionano anche grazie al metodo virtuale puro in Animal *accept(animalVisitor& v)*, poi implementato in ogni sottoggetto), contengono l'implementazione del visitor, che ha il compito di creare delle finestre diverse in base al tipo di animale con, ognuna, diverso layout e diverse funzioni specifiche per quel tipo.
Per esempio: nel widget relativo al Leone, abbiamo un particolare layout con dei colori che lo simboleggiano come l'arancione, ed una sua funzione specifica, ovvero quella di ruggire, simulata tramite popup con informazioni sulla potenza del ruggito (attributo proprio di ogni istanza di leone); invece con il coccodrillo abbiamo un colore diverso, il verde, ed una funzione diversa, ovvero quella aprire la bocca e sfoggiare la sua dentatura.

Abbiamo provato anche ad integrare dei suoni (per esempio il ruggito del leone quando si attivava il suo ruggito) ma, a quanto abbiamo capito, non disponevamo del modulo di QT necessario (*QT Multimedia)*.

Persistenza dei dati

Il vincolo sulla persistenza dei dati è stato soddisfatto tramite il salvataggio, e caricamento, delle partite tramite file .xml.
Abbiamo una classe "dataManager" che si occupa di salvare, o caricare, i dati relativi ad una partita: prima salva le informazioni generali come la data, l'ora ed i soldi, poi memorizza anche tutti i dati di ogni singolo animale per ogni recinto.

Funzionalità implementate

Gameplay

Si hanno 6 recinti, uno per ogni tipo di animale presente nel gioco: coccodrillo, giraffa, leone, pavone, struzzo e tartaruga.
Questi recinti vengono generati vuoti, e ci si può interagire in un paio di modi:

Sfamare animali

Tramite l'apposito bottone si possono sfamare tutti gli animali del recinto. La fame/salute viene simulata tramite la barra della vita, visibile dalla mappa. Se la salute è a zero verrà rimosso un animale ad ogni aggiornamento del timer di gioco (2 secondi reali/2 minuti del gioco) per simularne la morte.
Si può decidere di quanto sfamare il recinto, scegliendo, tramite un popup, a quanto portare il valore della salute (25%, 50%, 75% o 100%) e quante risorse richiede questo.
Ovviamente, se non si possiedono abbastanza risorse per una determinata opzione, questa non sarà disponibile.

Aggiungere animali

Tramite l'apposito bottone si può aggiungere 1 animale al recinto del suo tipo. Questo richiederà un determinato numero di risorse in base al tipo di animale aggiunto, ma questo valore influenza anche il guadagno che un animale genera, per esempio:

> Un leone richiede €10 per l'acquisto, ma poi genererà, ad ogni aggiornamento del timer di gioco, due volte quello stesso valore. Quindi, per avere un recinto con 4 leoni dovremmo spendere €40, ma questo genererà €80 ogni aggiornamento del timer di gioco.

Come si può notare il ricavo è enorme rispetto alla spesa, questo è fatto appositamente per rendere l'esperienza di gioco facile e quindi avere una maggiore libertà sulle scelte che si possono prendere. Per rendere il gioco più impegnativo basterebbe far scegliere all'utente un livello di difficoltà, e diminuire il guadagno del recinto in base all'opzione scelta.

Interazione con i singoli animali e ricerca

Se si clicca sulla label di un singolo animale, o si cerca per nome tramite la barra di ricerca, verranno mostrati i vari attributi dell'animale che abbiamo selezionato, tramite un'apposita finestra, generata automaticamente in base al tipo di animale scelto.

Generazione dati casuali

Anche grazie alle funzionalità implementate nei file "*generate.h* ed *generate.cpp*" abbiamo creato un meccanismo che permette, quando un animale viene generato, che gli vengano assegnati degli attributi che possano essere quanto più veritieri: il peso di un coccodrillo, per esempio, viene generato scegliendo un valore casuale tra 2 estremi, che saranno ovviamente diversi dai dati usati per i pavoni.

Shotcut

Durante una partita, se si preme “esc” sulla propria tastiera, verrà mostrato un menù con 3 scelte:

- “Close without saving” che chiude la partita senza salvarla
- “Cancel” che chiude solo questo menù appena creato, facendoci tornare alla partita
- “Save” per salvare i progressi della partita in corso
    - Selezionando questa opzione ci verrà chiesto di inserire il nome della partita. Se il nome è già presente, allora, ci verrà chiesto se vogliamo sovrascrivere tale salvataggio

Layout

Il layout è flessibile, essendo tutto inserito in un grande *QGridLayout*, che è flessibile per sua natura. Comunque abbiamo deciso di tenere una dimensione minima per non far rimpicciolire troppo il tutto.

Altre funzionalità

Altre funzionalità interessanti ma di cui si è già parlato precedentemente sono: salvataggio e caricamento delle partite, finestra con dati specifici generata dinamicamente per ogni tipo di animale.

Rendicontazione delle ore

Nonostante il file delle specifiche del progetto indicasse circa 50 ore di lavoro individuale, il totale di ore impiegate effettive è stato ben al di sopra di quella soglia, arrivando a circa 65/70 ore.
C’è anche da dire, però, che l’80% del tempo circa è stato impiegato per la parte grafica fatta con QT, che ha richiesto così tanto tempo per principalmente due motivi:

- QT è stato un framework nuovo per entrambi i componenti del gruppo. Questo ha comportato, quindi, un'importante fase di “autoapprendimento” di questa nuova tecnologia.
- Sviluppare un videogioco, ovviamente, richiede un maggiore impegno per quanto riguarda la parte grafica, e questo progetto non ha assolutamente smentito questa convinzione.

Suddivisione delle attività

Le attività sono state, il più possibile, equamente distribuite tra i due componenti, per esempio: i visitor finali (sei in totale) sono stati sviluppati 3 da me, e 3 dal mio compagno.

Modifiche apportate

Non ci sono state modifiche in quanto è la prima volta che questo progetto viene consegnato

Conclusioni

Lo sviluppo di questo progetto è stato incredibilmente stimolante.
Nonostante abbia richiesto molto più del tempo inizialmente previsto, ritengo che questa sia stata un’esperienza estremamente formativa e, nonostante la considerevole quantità di tempo totale impiegata, molto gradevole. Infatti, se dovessi rifare questo progetto da capo, probabilmente, ora, sarei in grado di diminuire drasticamente le ore impiegate.