Componenti del gruppo:

- Antonino Granata, 20034202
- Cecilia Tasca, 20040354

Relazione del progetto: Proposta 2 - Mondo Robot

Il robot manutentore si muoverà all'interno della mappa di una casa ed avrà la possibilità di effettuare varie operazioni sulla cella in cui sta oppure muoversi all'interno della mappa, spostandosi solo attraverso le celle adiacenti. Oltre al robot, è presente anche un animale domestico che si muove in maniera autonoma sulla mappa. Esistono diverse tipologie di celle, la più semplice è la cella vuota, la quale può essere percorsa dal Robot o dall'animale domestico e può acquisire lo status di bagnata se è vicina ad una cella lavatrice o rubinetto. Sia la lavatrice sia il rubinetto sono due tipologie di celle non percorribili dal Robot o dall'animale domestico, ma se il robot si posiziona in una cella adiacente, può modificarne lo status (ad es. se il rubinetto è aperto, il robot può chiuderlo. Analogamente avviene per la lavatrice). Infine, la mappa presente la cella fornello, non è percorribile dal Robot o dall'Animale Domestico ed il suo stato è modificabile dal Robot, spegnendolo o accendendolo. Il Robot manutentore non solo ha la possibilità di muoversi tra le celle adiacenti, ma se riconosce che la cella in cui si trova è bagnata può asciugarla. Inoltre, può accendere o spegnere un fornelo se si trova nella cella adiacente; mentre, se è nella cella adiacente ad un rubinetto può chiuderlo. Infine, può aggiustare una cella lavatrice, a patto che sia nella cella adiacente.
L'animale domestico sfrutta un Thread per muoversi indipendentemente dal Robot Manutentore, sulla mappa sarà identificato da un cane. Infine, è necessario specificare che il Robot e l'Animale Domestico sono due oggetti distinti che si muovono sulle celle.

Class Diagram:

Per avere una maggiore comprensione del progetto abbiamo deciso di iniziare a descrivere le classi nel dettaglio, per poi arrivare a specificare le interazioni e tutto ciò che ne deriva, quindi ereditarietà, gestione delle eccezioni e uso di una classe annidata. Prima di definire gli oggetti, definiamo il thread dell'acqua: ha lo scopo di allagare le celle asciutte adiacenti alle celle contenenti gli oggetti Lavatrice e Rubinetto.

Il thread dell'acqua, definito dalla classe avviaThPerditaAcqua ha lo scopo di modificare lo status della cella, passando da cella asciutta a cella bagnata ed una condizione per fermare il Thread. All'interno di questa classe è presente una classe annidata di tipo Thread thPerditaAcqua. Per definirlo abbiamo utilizzato il metodo allagaCella(int n, int r, int c) che riceve un intero n, compreso tra 0 e 3, per scegliere in maniera randomica quale cella adiacente bagnare, e la posizione della cella (tramite r e c); inoltre, abbiamo creato un metodo perditaAdiacente(int r, int c), che riceve la posizione della cella, tramite r e c, e rende bagnata quella cella. Infine abbiamo definito il metodo run() che richiama allagaCella() fornendogli un numero randomico tra 0 e 3 (per il movimento) e la posizione della cella, tramite riga e colonna. Il thread ha un tempo di sleep pari a 10 secondi.

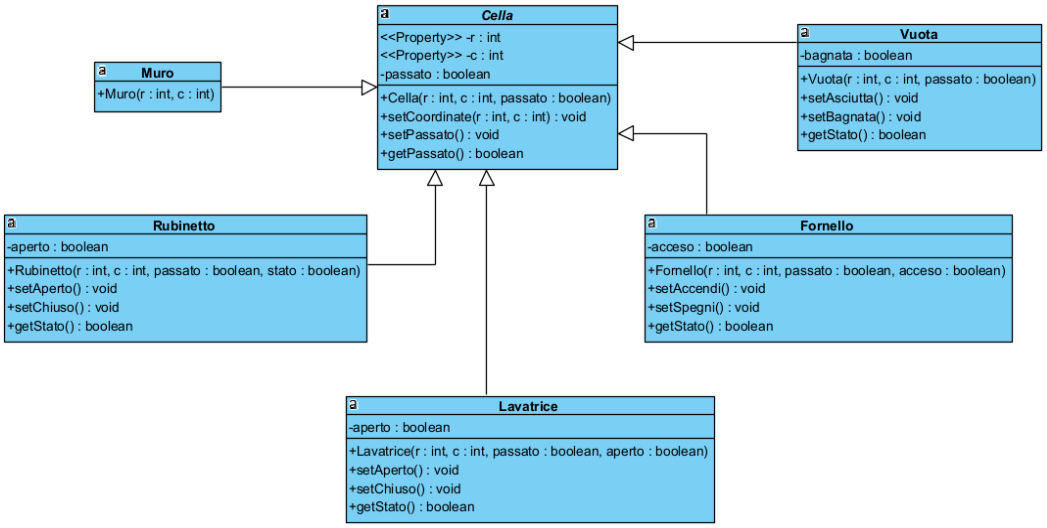
Gli oggetti che costituiscono la mappa sono:

- Fornello, è un'estensione di Cella ed è definita da un costruttore che prende le coordinate, se è già passato da quella cella e lo stato del fornello. Inoltre, presenta i metodi setSpegni(), per spegnere, setAccendi(), per accendere, e getStato(), per ricevere lo stato del fornello
- Lavatrice, è un'estensione di Cella ed è definita da un costruttore che prende le coordinate, se è già passato da quella cella e lo stato della lavatrice. Inoltre, presenta i metodi setSpegni(), per spegnere setAccendi(), per accendere, e getStato(), per ricevere lo stato della lavatrice
- Muro, è un'estensione di Cella, ed è definito da un costruttore che prende le coordinate esterne della mappa, definendo celle che non possono essere attraversate
- Rubinetto, è un'estensione di Cella ed è definita da un costruttore che prende le coordinate, se è già passato da quella cella e lo stato del rubinetto e l'avvio del thread dell'acqua. Inoltre, presenta i

metodi setAperto(), per aprire, setChiuso(), per chiudere, e setStato(), per dare in output lo stato (se è aperto o chiuso) e per fermare il thread dell'acqua

- Vuota, è un'estensione di Cella ed è definita da un costruttore che prende le coordinate e se è stata visitata oppure no. Inoltre, presenta i metodi setAsciutta(), per definire se la cella sia asciutta, setBagnata(), se la cella è bagnata, e getStato(), per definire lo stato della cella (bagnata o asciutta)

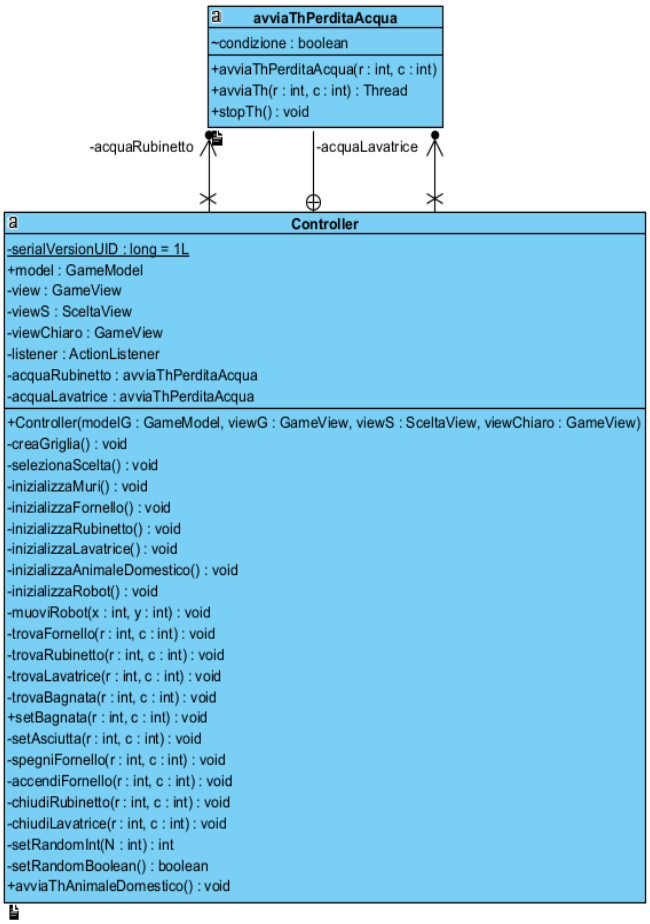
La classe Cella è un'astrazione delle classi precedenti. Il suo costruttore è definito dalle coordinate della cella, dove r rappresenta le righe e c le colonne, e se la visita alla stessa è avvenuto o se deve ancora avvenire. Inoltre, presenta i metodi per definire le coordinate, per capire se la cella è stata visitata oppure no, restituendo un booleano; infine, un ultimo metodo per ricevere le coordinate come valore di ritorno. Di seguito, il Class Diagram:



Gli oggetti Animale Domestico e Robot presentano alcune similitudini, infatti si muovono all'interno della mappa passando da una cella adiacente all'altra, potendo passare su celle bagnate. Inoltre, entrambi presentano i metodi getC(), getR(), setC() e setR(), utilizzati per le coordinate. Infine i metodi per definire il loro movimento sono in Controller.java. Vi sono però alcune differenze: il movimento del robot è gestito dall'utente, che può scegliere in quale cella adiacente spostarlo tramite un click del mouse; mentre l'animale domestico si muove grazie ad un thread. Più precisamente, quest'ultimo, ha il metodo muoviAnimaleDomestico(int x, int y), dove x e y sono le coordinate della posizione effettiva, ed il metodo muoviAnimaleDomestico(int n) che riceve un intero n, compreso tra 0 e 3, per scegliere in maniera randomica quale cella adiacente muoversi. Il thread ha un tempo di sleep pari a 0.5 secondi, quindi l'animale domestico si muove 0.5 sec si muove.

In controller classe annidata: A differenza delle classi annidate statiche, le classi interne non statiche hanno un legame stretto con l'istanza della classe esterna e possono accedere ai membri (variabili e metodi) della classe esterna, inclusi quelli privati.

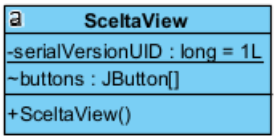
Ora ci occupiamo di definire le classi dell'MVC, ovvero quelle che si occupano della grafica. Sono presenti una classe Model, due classi View ed una classe Controller.

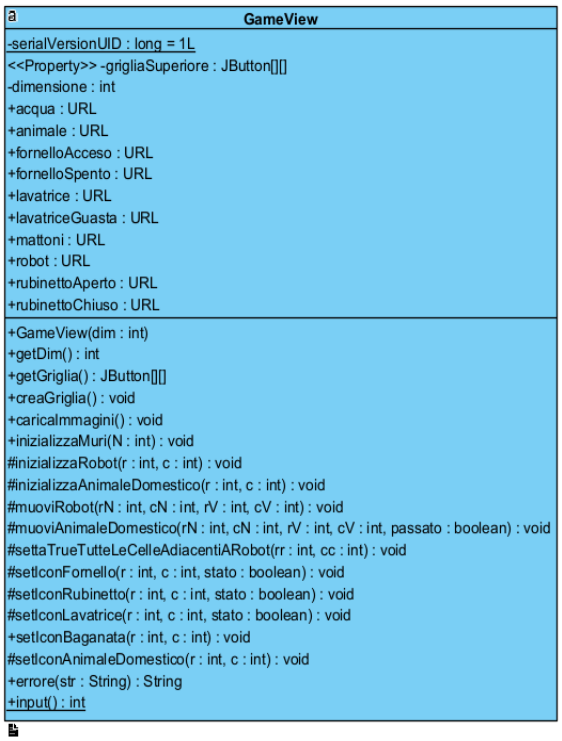


Il Controller.java presenta i thread dell'acqua e dell'animale domestico, già presentati in precedenza, tutti i metodi per definire le posizioni iniziali degli oggetti, quindi: inizializzaMuri(), inizializzaFornello(), inizializzaRubinetto(), inizializzaLavatrice(), inizializzaAnimaleDomestico(), inizializzaRobot(). Presenta i metodi per far sapere al robot se in una delle sue celle adiacenti è presente un oggetto, essi sono: trovaFornello(int r, int c), trovaRubinetto(int r, int c), trovaLavatrice(int r, int c), TrovaBagnata(int r, int c). Infine sono presenti i metodi per poter modificare lo status delle celle: setAsciutta(int r, int c), setBagnata(int r, int c), spegniFornello(int r, int c), accendiFornello(int r, int c), chiudiRubinetto(int r, int c), chiudiLavatrice(int r, int c). Ogni intero r e c serve per definire la posizione della cella.

Le frecce indicano che il modello può essere navigato solo nella direzione opposta della freccia. Il simbolo "+" indica che la classe aggregante, Controller.java, ha una relazione di aggregazione con la classe aggregata, avviaThreadPerdita. Questo significa che la classe aggregante contiene o fa riferimento alla classe aggregata, ma la classe aggregata può esistere indipendentemente dalla classe aggregante

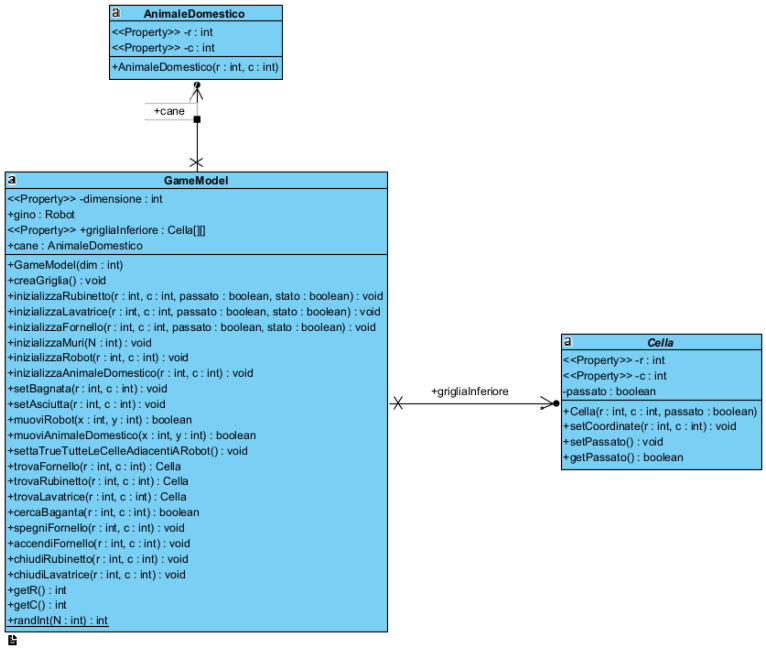
La SceltaView.java presenta una classe SceltaView che estende JPanel e ha lo scopo di definire i bottoni da poter far cliccare all'utente per selezionare un'azione da compiere. L'utente può scegliere tra: Asciuga, Accendi Fornello, Spegni Fornello, Chiudi Rubinetto, Chiudi Lavatrice.





La GameView.java inizialmente crea un pop up per far scegliere all'utente la dimensione N della griglia di gioco, successivamente inizializzaMuri(int N) riceve la dimensione N e crea la griglia definendo il perimetro, usando r e c per le coordinate, di Muri. Successivamente crea due mappe con la dimensione N inserita, la prima presenta la griglia di gioco, dove l'utente può scegliere le mosse del Robot e sceglie cosa fare; al contrario nella seconda griglia vi è una rappresentazione di dove vengono generati randomicamente tutti gli oggetti. Questa GameView presenta il metodo caricaImmagini() con le immagini degli oggetti e dei cambiamenti di stato delle celle o degli oggetti. I metodi setIconFornello(int r, int c, boolean stato), setIconLavatrice(int r, int c, boolean stato), setIconLavatrice(int r, int c, boolean stato), oltre alle coordiante della cella (rappresentate da r e c), presentano lo stato per definire il cambio di stato e, di conseguenza, il cambio di immagine, setIconBaganata(int r, int c), che aggiunge l'Icon con l'acqua se la cella viene bagnata, setIconAnimaleDomestico((int r, int c) specifica in quale cella vi è l'immagine dell'animale domestico.

La GameModel.java comunica direttamente con la GameView.java ed è la parte di codice dove avviene la creazione degli oggetti. In un primo momento riceve la dimensione dall'utente e, tramite creaGriglia(), crea la mappa di gioco. Inoltre, ci sono i metodi per definire ogni oggetto, ad esempio inizializzaRubinetto((int r, int c, boolean passato, boolean stato), dopo aver controllato che la cella sia vuota, definisce un oggetto



di tipo Rubinetto e lo posiziona sulla cella, questo avviene tramite il metodo trovaRubinetto(int r, int c). I metodi inizializzaLavatrice(int r, int c, boolena passato, boolean stato) e inizializzaFornello(int r, int c, boolean passato, boolean stato) hanno un funzionamento analogo. Il metodo inizializzaRobot(int r, int c), dopo aver controllato che la cella sia vuota, richiama il metodo settaTrueTutteLeCelleAdiacentiARobot() con lo scopo di definire le celle adiacenti come attraversabili. Il metodo inizializzaAnimaleDomestico(int r, int c) posiziona, dopo aver controllato che la cella sia vuota, l'animale sulla cella. I metodi muoviRobot(in x, int y)

e muoviAnimaleDomestico() sono pressochè simili, infatti entrambi si muovono sulla cella e hanno un controllo per evitare che i due oggetti si posizionino sulla stessa cella. L'unica differenza è il metodo settaTrueTutteLeCelleAdiacentiARobot() per permettere di settare le celle adiacenti al robot come attraversabili. Infine, sono presenti i metodi spegniFornello(int r, int c), accendiFornello(int r, int c) che hanno lo scopo di spegnere o accendere un oggetto di tipo Fornello, analogamente avviene per chiudiRubinetto(int r, int c) per un oggetto di tipo Rubinetto e per chiudiLavatrice(int r, int c) per un oggetto di tipo Lavatrice.

Funzionalità del progetto

Lo scopo del progetto è creare una mappa, di grandezza superiore a 10x10 decidibile dall'utente, con alcuni oggetti statici, come i fornelli, le lavatrici ed i rubinetti, ed oggetti dinamici, come il Robot, l'animale domestico e l'acqua. Il personaggio principale è il robot, rappresenta le scelte dell'utente, esso può effettuare una singola mossa per turno. Le mosse sono: muoversi sulla mappa avanzando tramite le celle adiacenti oppure può decidere se compiere un'interazione con la cella in cui si trova; infatti, può asciugare una cella bagnata, spegnere il fornello, aggiustare il rubinetto o la lavatrice. Appena la mappa viene generata tutti gli oggetti vengono generati in maniera randomica, l'utente conosce solo la posizione del robot e le sue celle adiacenti. Il suo obiettivo è continuare a muoversi per trovare dove sono posizionati sulla mappa tutti gli altri oggetti. L'utente può scegliere cliccando direttamente la cella in cui vuole che il robot si sposti e, quando avrà finalmente trovato una cella in cui effettuare una operazione, l'utente, può scegliere tramite alcuni bottoni situati di fianco alla mappa. I bottoni sono: asciuga, spegni fornello, accendi fornello, chiudi rubinetto e chiudi lavatrice. L'animale domestico, rappresentato da un cane, si muoverà in maniera autonoma, apparendo sulla mappa solo se si trova nella zona già sbloccata dal Robot. Anche il flusso dell'acqua che perde avviene in maniera autonoma, in base ad una certa quantità di tempo, anziché in base alle mosse del Robot.