

newMonopoly

M.Martinelli, J.Paglione, D.Tamborrino

04 Febbraio 2018

Indice

1 Strumenti	3
2 Requisiti	3
2.1 Requisti funzionali	3/4
2.2 Requisiti non funzionali	4
3 Casi d'uso	4/5/6/7/8
4 Pattern Usati	9
5 Diagramma dei casi d'uso	10
6 Diagramma delle attività	11
7 Diagramma degli stati	12
8 Diagramma di sequenza	13
9 Diagramma delle classi a livello di dominio	14
10 Diagramma degli stati a livello di progetto	15
11 Istruzioni per la lettura di sonar	16/17/18

1 Strumenti

Unity <https://unity3d.com/>

StarUML <http://staruml.io/>

Ganttproject <http://www.ganttproject.biz/>

Texmaker <http://www.xmlmath.net/texmaker/>

2 Requisiti

2.1 *REQUISITI FUNZIONALI:*

- Il sistema deve permettere ad un minimo di 2 ad un massimo di 6 utenti di giocare una partita a monopoli sul dispositivo sul quale il sistema è in funzione.
- Il sistema deve implementare un'interfaccia utente "Menu" nel quale l'utente potrà scegliere se avviare una nuova partita o uscire.
- Il sistema deve permettere di uscire in qualsiasi momento della partita.
- Il sistema deve permettere ad ogni giocatore, all'inizio di ogni partita di scegliere una pedina che lo rappresenti durante la suddetta.
- Il sistema deve permettere a ogni giocatore di lanciare i dadi all'inizio del proprio turno, tale risultato permetterà l'avanzare del giocatore sul percorso del sistema.
- Il sistema deve, a seconda del dove il giocatore si sia fermato implementare un sistema che preveda i seguenti casi:

Il caso in cui il giocatore si trovi su una casella libera, il giocatore possa scegliere di acquistare la proprietà sulla casella.

Il caso in cui il giocatore si trovi su la proprietà di un altro giocatore si deve pagare un pedaggio a quest'ultimo.

Il caso in cui si trovi sulla casella di via o la si abbia passata, il giocatore deve ricevere un bonus di 500\$.

Il caso in cui si trovi sulla casella "vai in prigione" il giocatore che ha tirato dovrà essere spostato sulla zona "Prigione".

Il caso in cui il giocatore si trovi su una casella probabilità ad esso verrà assegnata una "probabilità" casuale.

Il caso in cui il giocatore si trovi su una casella imprevisto ad esso verrà assegnata una

"imprevisto" casuale.

Il caso in cui il giocatore si sia fermato su una casella tasse, ad esso verranno sottratti tanti soldi quanti indicato nella casella.

- Il sistema deve implementare un sistema di trattativa tra 2 giocatori.

- Il sistema deve implementare un numero minimo di 20 carte probabilità e 20 carte imprevisto.
- Il sistema deve implementare un sistema di “Prigione” definito poi nei casi d’uso.
- Il sistema deve permettere ad ogni turno, prima del passaggio agli altri giocatori di poter entrare in fase “Costruzione”.
- Il sistema deve implementare un’interfaccia che mostri al giocatore le informazioni della proprietà o stazione sulla quale si è fermato.
- Il sistema deve implementare un’interfaccia che mostri quando l’utente fa un tiro doppio, e deve notificargli quando con il prossimo tiro doppio andrà in prigione.
- Il sistema deve implementare un’interfaccia che mostri l’effetto delle carte imprevisto e probabilità.

2.2 REQUISITI NON FUNZIONALI:

- Il sistema sarà sviluppato in Unity e su linguaggio C Sharp.
- Il sistema deve implementare delle pedine da far scegliere al giocatore.
- Il sistema deve implementare un campo da gioco da monopoli, con 40 caselle divise tra proprietà (terreni, stazioni), 2 caselle tasse, 3 caselle probabilità, 3 caselle di bonus comune, 1 della prigione, 1 di vai in prigione, 1 del via.
- Il sistema deve implementare 28 proprietà con relativi costi di terreno, costi di costruzione case e alberghi e costo pedaggio in base alle costruzioni su di esso.
- Il sistema deve permettere lo spostamento del giocatore che ha tirato i dadi sommando alla sua posizione attuale il numero risultante dal lancio dei dadi, quella è la sua nuova posizione.
- Il sistema deve far partire ogni giocatore con 2500 \$.

3 Casi d’uso

INIZIO NUOVA PARTITA:

DESCRIZIONE: Caso d’uso base del sistema, coinvolge il giocatore.

SCENARIO: Il giocatore selezione dal menù nuova partita e il sistema avvierà una nuova partita da 0.

USCIRE:

DESCRIZIONE: Il giocatore in ogni momento della partita potrà scegliere se uscire premendo l’apposito tasto.

SCENARI DI SUCCESSO: Il giocatore esce dalla partita in corso.

SCELTA NUMERO GIOCATORI:

DESCRIZIONE: Caso d’uso incluso nell’inizio della partita, ogni giocatore a turno sceglie la propria pedina.

SCENARIO DI SUCCESSO: I giocatori scelgono ognuno una pedina diversa.
Il sistema impedirà ai giocatori successivi di scegliere la stessa pedina di un altro giocatore.

TIRO DADI://FORMALE

LIVELLO: Obiettivo utente.

ATTORE PRIMARIO: Sistema.

ATTORE SECONDARIO: Giocatore di turno.

PARTI INTERESSATO: Sistema e giocatore di turno.

PRE-CONDIZIONI: Il turno del giocatore deve essere appena iniziato.

SCENARIO DI SUCCESSO: 1 Il giocatore di turno tira i dadi e ottiene due numeri diversi, tale giocatore avanzerà sulla mappa di gioco e effettuerà il suo turno normalmente.

- 1.1 Il giocatore di turno tira i dadi e ottiene 2 numeri uguali, tale giocatore avanzerà sulla mappa di gioco e effettuerà il suo turno normalmente e al termine effettuerà un ulteriore turno ripartendo dal lancio dei dadi.
- 2 Il giocatore di turno tira i dadi e ottiene due numeri diversi, tale giocatore avanzerà sulla mappa di gioco e effettuerà il suo turno normalmente.
- 2.1 Il giocatore di turno tira i dadi e ottiene 2 numeri uguali, tale giocatore avanzerà sulla mappa di gioco e effettuerà il suo turno normalmente e al termine effettuerà un ulteriore turno ripartendo dal lancio dei dadi.
- 3 Il giocatore di turno tira i dadi e ottiene due numeri diversi, tale giocatore avanzerà sulla mappa di gioco e effettuerà il suo turno normalmente.
- 3.1 Il giocatore di turno tira i dadi e ottiene 2 numeri uguali, tale giocatore verrà spostato sulla casella della prigione, entrerà nello stato “in prigione” e potrà solamente fare azioni da fine turno .

FREQUENZA DI RIPETIZIONE: Ogni volta all’inizio del turno del giocatore e massimo altre 2 volte all’interno del turno di un giocatore.

ACQUISTO PROPRIETA’ LIBERA:

DESCRIZIONE: Caso d’uso che estende il TIRO DADI, coinvolge il giocatore di turno che si è fermato su una proprietà libera.

SCENARIO IDEALE DI SUCCESSO: Il giocatore di turno si ferma su una proprietà libera e decide di acquistarla.

SCENARI ALTERNATIVI DI SUCCESSO: Il giocatore si ferma su una stazione libera e decide di acquistarla.

Il giocatore che decide di acquistare una proprietà libera non ha abbastanza soldi pur ipotecando tutte

le sue proprietà l’acquisto non andrà a buon fine.

PAGAMENTO PEDAGGIO:

DESCRIZIONE: Caso d'uso che estende il **TIRO DADI** , coinvolge il giocatore di turno e il giocatore proprietario.

SCENARIO IDEALE DI SUCCESSO : Il giocatore di turno si ferma su una proprietà di un altro giocatore, il giocatore dovrà pagare un pedaggio a seconda della proprietà sulla quale si è fermato. **SCENARI ALTERNATIVI DI SUCCESSO:** Il giocatore di turno si ferma sulla proprietà di un altro giocatore, non ha abbastanza soldi per pagare il pedaggio, il giocatore di turno finirà nel caso d'uso FALLIMENTO.

COSTRUIRE:

DESCRIZIONE: Caso d'uso che il giocatore può ripetere più volte nel suo turno, solo quando ha acquistato tutti i terreni di un certo colore.

SCENARIO IDEALE DI SUCCESSO: Il giocatore di turno sceglie di costruire una casa su una sua proprietà, ha abbastanza soldi, ha al massimo una casa in meno sugli altri territori dello stesso colore, la costruzione riesce.

SCENARI ALTERNATIVI DI SUCCESSO: Il giocatore di turno sceglie di costruire una casa su una sua proprietà, non ha abbastanza soldi pur avendo ipotecato ogni sua altra proprietà, ha al massimo una casa in meno sugli altri territori dello stesso colore, la costruzione gli viene impedita dal sistema.

Il giocatore di turno sceglie di costruire una casa su una sua proprietà, ha abbastanza soldi, non ha al massimo una casa in meno sugli altri territori dello stesso colore, la costruzione gli viene impedita dal sistema.

Il giocatore di turno sceglie di costruire un albergo su una sua proprietà, ha abbastanza soldi, ha al massimo una casa in meno sugli altri territori dello stesso colore, la costruzione riesce.

Il giocatore di turno sceglie di costruire un albergo su una sua proprietà, non ha abbastanza soldi pur avendo ipotecato ogni sua altra proprietà, ha al massimo una casa in meno sugli altri territori dello stesso colore, la costruzione gli viene impedita dal sistema.

Il giocatore di turno sceglie di costruire un albergo su una sua proprietà, ha abbastanza soldi, non ha al massimo una casa in meno sugli altri territori dello stesso colore, la costruzione gli viene impedita dal sistema.

INIZIARE TRATTATIVA:

DESCRIZIONE: Caso d'uso che coinvolge più giocatori, un giocatore è interessato alla proprietà di un altro giocatore e, durante il suo turno, offre una cifra in denaro o delle sue proprietà per ottenere tale proprietà.

SCENARIO IDEALE DI SUCCESSO: Il giocatore proprietario accetta l'offerta, e la proprietà, o la stazione, compresa di case e alberghi passa al giocatore che aveva iniziato la trattativa.

SCENARIO ALTERNATIVO: Il giocatore proprietario fa una controfferta chiedendo al giocatore che ha iniziato la trattativa richiedendo più soldi o delle proprietà da aggiungere alla trattativa.

Il giocatore proprietario rifiuta l'offerta e la trattativa viene annullata.

PASSAGGIO DALLA CASELLA VIA:

DESCRIZIONE: Caso d'uso che coinvolge il giocatore di turno, Il giocatore di turno passa o si ferma sulla casella del VIA e ottiene 500\$ bonus.

PESCA CARTA IMPREVISTO:

DESCRIZIONE: Caso d'uso che estende il **TIRO DADI**, coinvolge il giocatore di turno che si ferma su una casella "imprevisto".

SCENARIO IDEALE DI SUCCESSO: Al giocatore viene imposto di pagare una tassa o riscuotere dei soldi.

SCENARI ALTERNATIVO DI SUCCESSO: Al giocatore viene imposto di pagare una tassa, se esso non ha abbastanza soldi si entrerà nel caso d'uso TRATTATIVA.

Al giocatore viene imposto di pagare una tassa, se esso non ha abbastanza soldi, pur avendo venduto ogni sua proprietà finirà nel caso d'uso **FALLIMENTO**.

Il giocatore pesca la carta "VAI IN PRIGIONE", entrerà nel caso d'uso **USCIRE DI PRIGIONE**.

PESCA CARTA PROBABILITA':

DESCRIZIONE: Caso d'uso che estende il **TIRO DADI**, coinvolge il giocatore di turno che si ferma su una casella "probabilità".

DESCRIZIONE: Al giocatore di turno verrà applicato un effetto casuale, tra cui ricevere denaro, ottenere "uscita gratis di prigione", essere spostato sul campo da gioco su una determinata casella.

VAI IN PRIGIONE:

DESCRIZIONE: Caso d'uso che estende il **TIRO DADI** e **PESCA CARTA IMPREVISTO**, coinvolge il giocatore di turno, il giocatore finirà sulla casella prigione e entrerà nello stato "**USCIRE DI PRIGIONE**".

SCENARIO IDEALE DI SUCCESSO: Il giocatore di turno si ferma sulla casella "vai in prigione", viene spostato sulla casella "prigione" e entrerà nello stato "in prigione".

SCENARI ALTERNATIVI DI SUCCESSO: Il giocatore di turno era nel caso d'uso TERZO TIRO, ottiene un numero doppio, viene spostato sulla prigione e entrerà nello stato "in prigione".

Il giocatore di turno era nel caso d'uso IMPREVISTO O PROBABILITA' e ottiene la carta "VAI IN PRIGIONE", viene spostato sulla casella prigione e entrerà nello stato "in prigione".

USCIRE DI PRIGIONE:

DESCRIZIONE: Caso d'uso che è incluso nella prigione, coinvolge il giocatore di turno.

SCENARIO PRINCIPALE DI SUCCESSO: Il giocatore "in prigione" all'inizio del proprio turno tira i dadi, se fa un numero doppio può uscire da "in prigione" e giocare normalmente.

SCENARI ALTERNATIVI DI SUCCESSO: Il giocatore di turno "in prigione" possiede la carta "esci gratis di prigione" che gli verrà rimossa, uscirà da "in prigione" e continuerà normalmente il turno. Il giocatore decide di pagare 125\$ e uscirà da "in prigione" continuando normalmente il turno. Dopo 3 turni che il giocatore è "in prigione" sarà costretto a pagare 125\$ e uscire da "in prigione" continuando normalmente il turno.

PAGAMENTO TASSE:

DESCRIZIONE: Caso d'uso che estende il **TIRO DADI**, coinvolge il sistema e il giocatore di turno. **SCENARIO IDEALE DI SUCCESSO:** Il giocatore di turno si ferma su una casella tasse e ha abbastanza soldi per pagare la tassa.

SCENARIO ALTERNATIVO DI SUCCESSO: Il giocatore di turno si ferma su una casella tasse, non ha abbastanza soldi per pagare la tassa, allora dovrà vendere tramite **TRATTATIVA** le sue proprietà fino a che non avrà abbastanza soldi per pagare la tassa. Il giocatore di turno si ferma su una casella tasse, ma non ha abbastanza soldi per pagare la tassa pur avendo venduto tutte le sue proprietà, tale giocatore finirà in **FALLIRE**.

FALLIRE:

DESCRIZIONE: Caso d'uso che coinvolge il giocatore di turno, il giocatore non riesce a sostenere una spesa pur avendo venduto ogni sua proprietà, il giocatore andrà in fallimento e perderà la partita.

TERMINARE PARTITA:

DESCRIZIONE: Caso d'uso che estende il fallimento, la partita finirà quando tutti i giocatori tranne uno saranno falliti.

4. PATTERN USATI:

STATE: usato per la realizzazione dello StateController, dato che le operazioni hanno grandi istruzioni condizionali che dipendono dallo stato dell'oggetto. **Singleton:** per la realizzazione del cambioScena.

Strategy: per la realizzazione di Casella, la quale è la generalizzazione di ogni possibile tipo di casella all'interno del gioco.

Lazy initialization: nella realizzazione delle proprietà, per la creazione delle case e degli alberghi solo quando necessario.

5. Diagramma dei casi d'uso

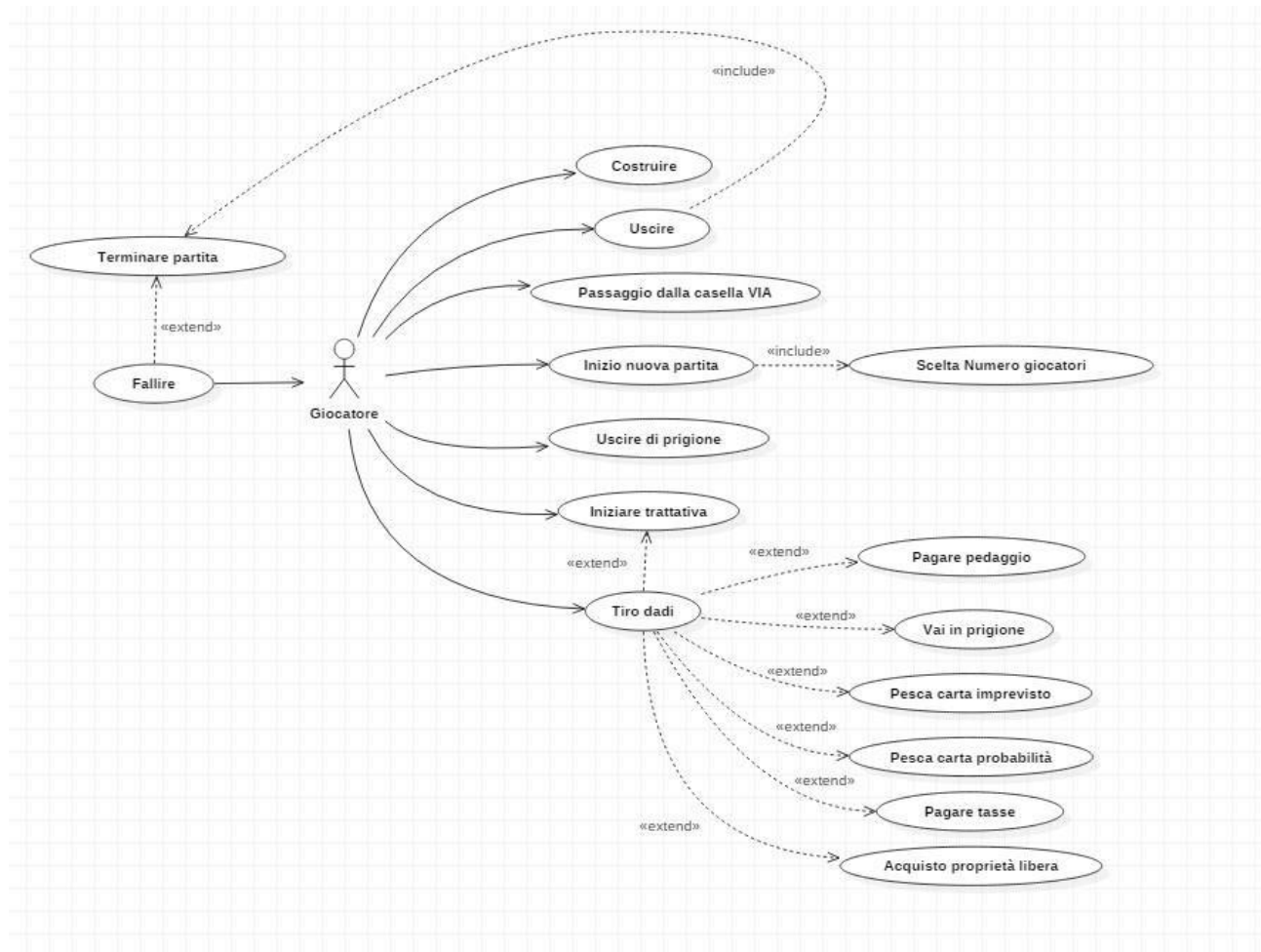


Diagramma 6 delle attività

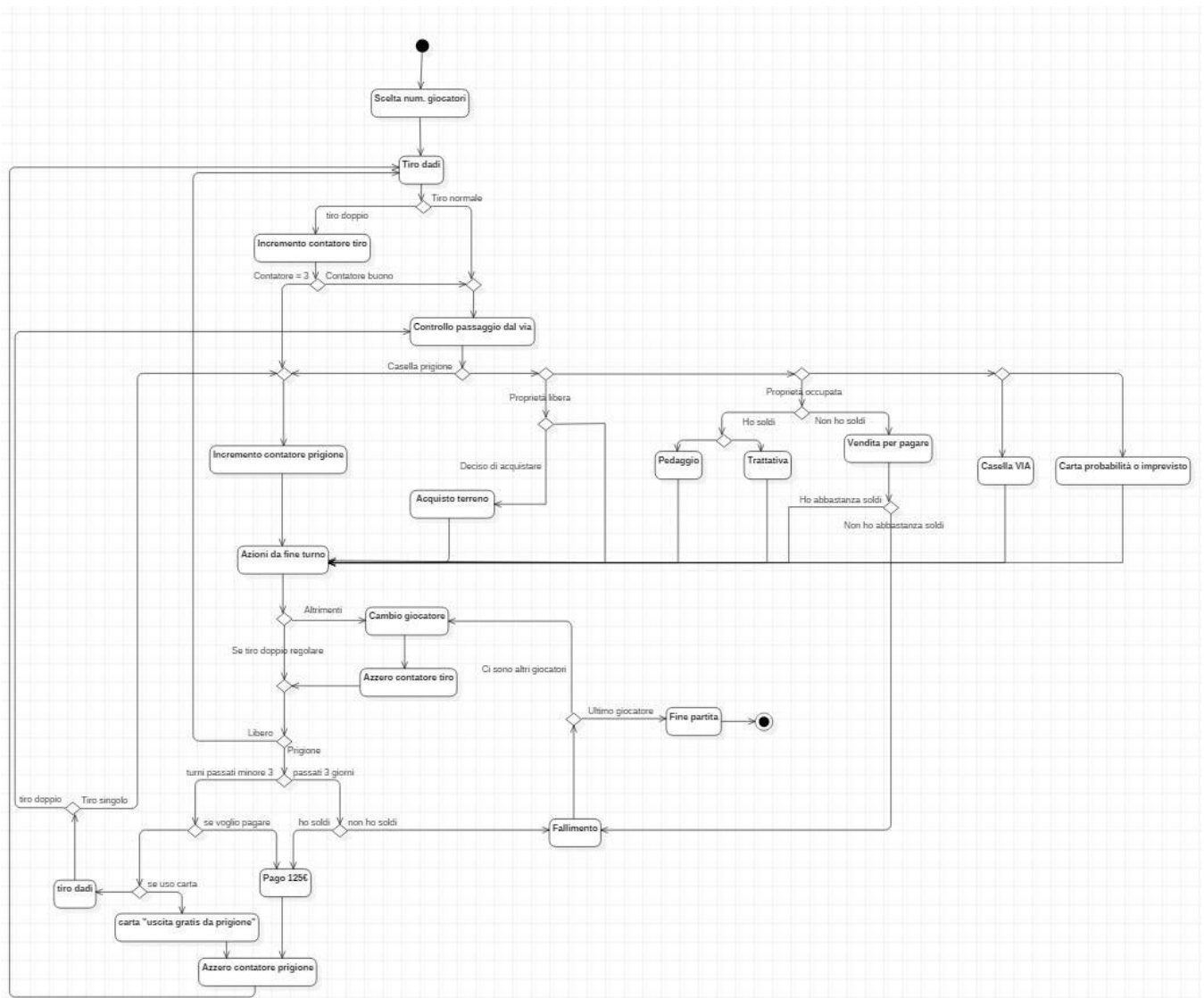
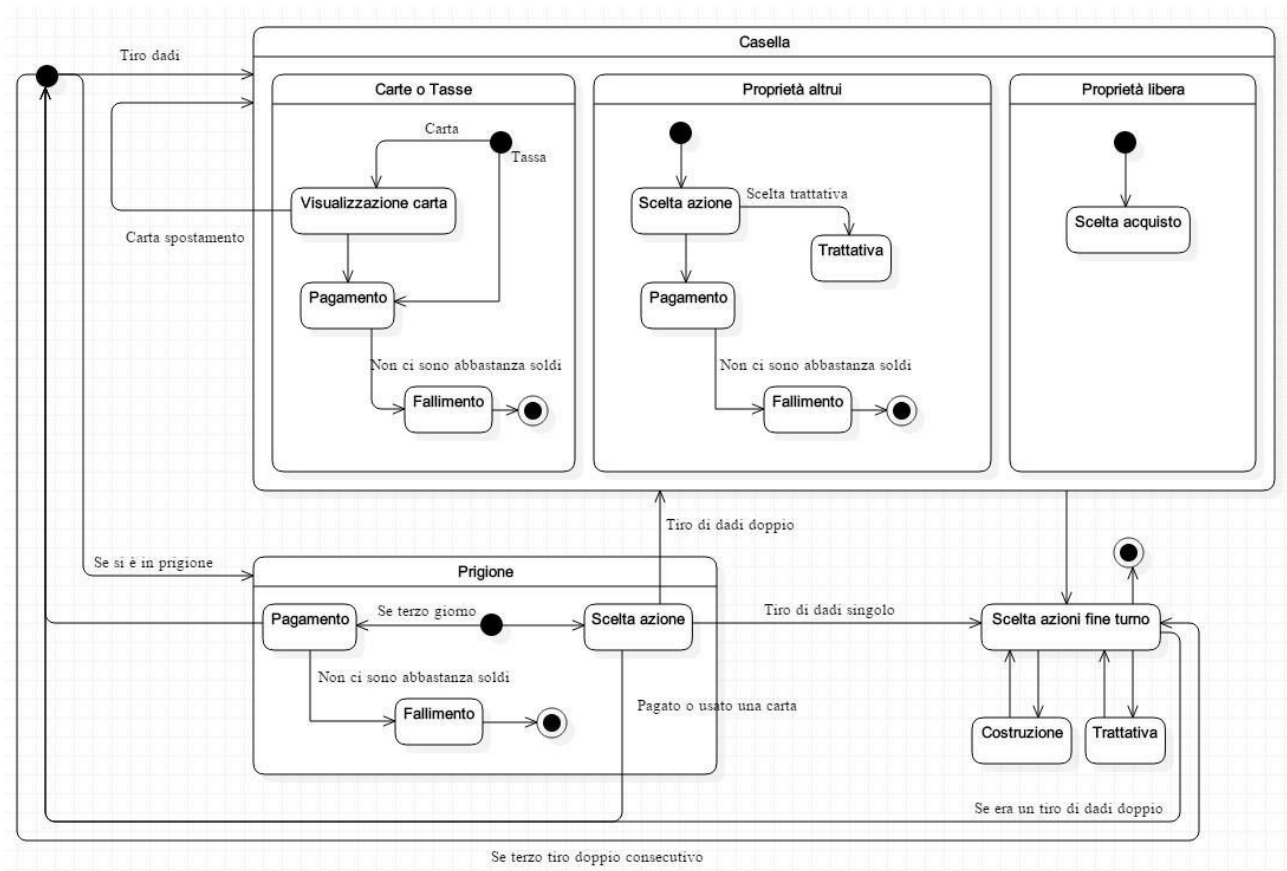


Diagramma 7 degli stati



8 Diagramma di sequenza

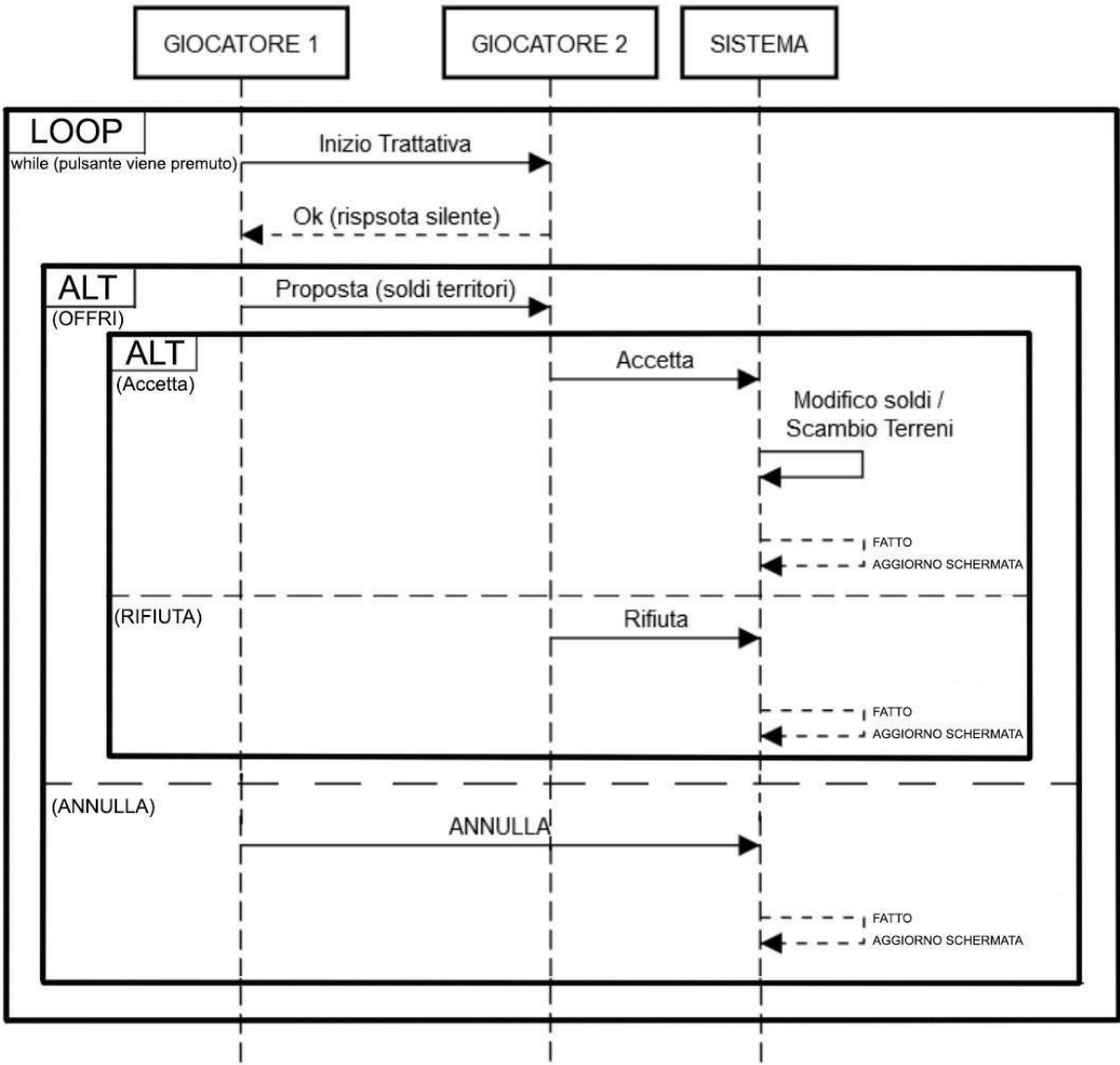


Diagramma 9 delle classi a livello di dominio

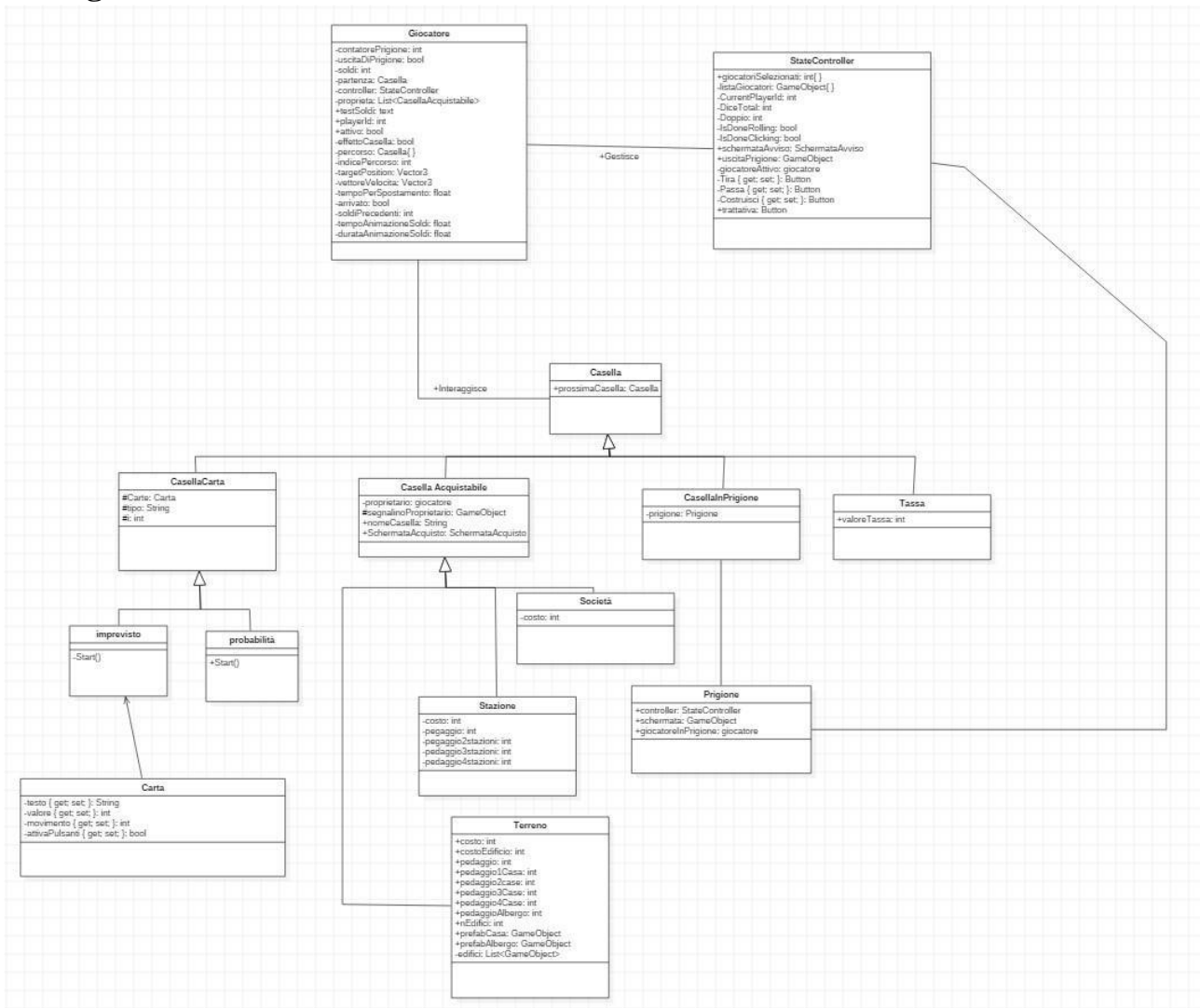
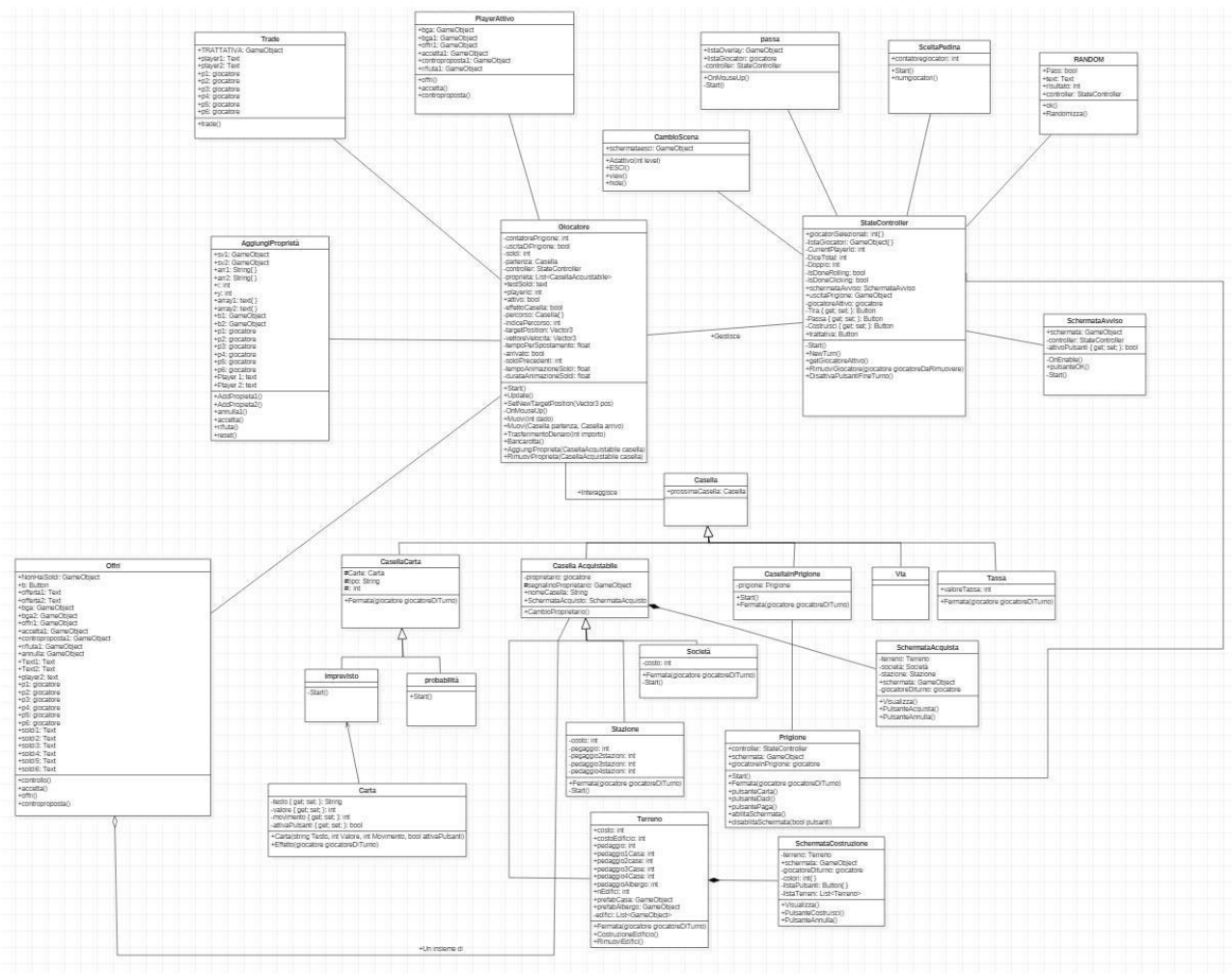


Diagramma 10 delle classi a livello di progetto



11 Istruzioni per la lettura di sonar

La guida descrive il perché Sonar riscontra degli alert sul progetto NewMonoply, dando un punteggio generale di B al nostro codice.

1. Il problema delle variabili public.

The screenshot displays the SonarQube interface with several code snippets and their associated alerts. Each alert is a red box with a lock icon, a 'Vulnerability' label, a 'Minor' severity, and an 'Open' status. The alerts are for public variables that should be made private and encapsulated. The alerts are dated 2 months ago, 7 days ago, and 6 days ago. The code snippets are as follows:

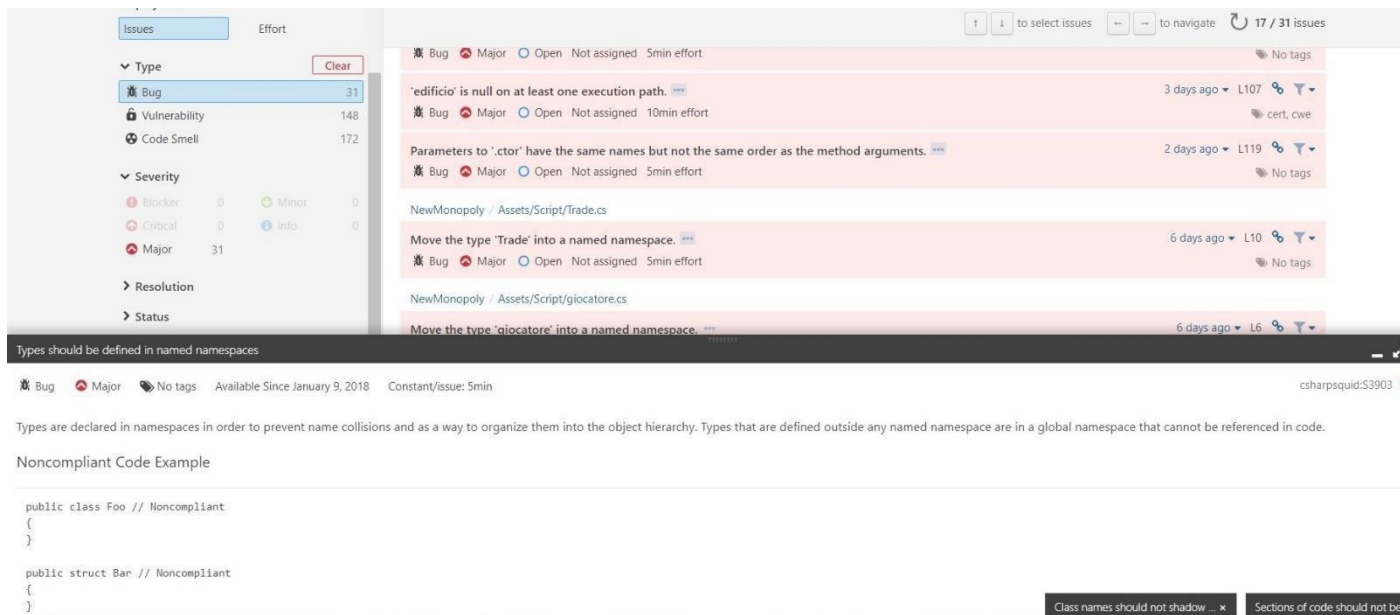
```
am...  
public Text[] text;  
  
Make this field 'private' and encapsulate it in a 'public' property. ***  
Vulnerability Minor Open Not assigned 10min effort cwe  
2 months ago L10  
  
public int risultato = 0; // Ogni volta che si tira si azzera il punteggio dei dadi  
  
Make this field 'private' and encapsulate it in a 'public' property. ***  
Vulnerability Minor Open Not assigned 10min effort cwe  
2 months ago L11  
  
ag...  
ag...  
public int doppio = 0;  
  
Make this field 'private' and encapsulate it in a 'public' property. ***  
Vulnerability Minor Open Not assigned 10min effort cwe  
7 days ago L13  
  
ag...  
public StateController theStateController;  
  
Make this field 'private' and encapsulate it in a 'public' property. ***  
Vulnerability Minor Open Not assigned 10min effort cwe  
6 days ago L14  
  
am...  
// Use this for initialization  
void Start () {
```

Data la natura di Unity, che in ogni momento ha bisogno delle informazioni di ogni oggetto presente sulla scena ci è stato impossibile rendere la maggior parte delle variabili usate da più gameObjects private, per questo motivo Sonar riscontra un centinaio di problem Minor sulla Vulnerability.

The screenshot displays the SonarQube interface with code snippets and their associated alerts. The alerts are red boxes with a lock icon, a 'Vulnerability' label, a 'Minor' severity, and an 'Open' status. The alerts are for public variables that should be made private and encapsulated. The alerts are dated 4 days ago. The code snippets are as follows:

```
9 m.mar... // public int player;  
10 m.mar... public GameObject sv1;  
  
Make this field 'private' and encapsulate it in a 'public' property. ***  
Vulnerability Minor Open Not assigned 10min effort cwe  
4 days ago L10  
  
11  
public GameObject sv2;  
  
Make this field 'private' and encapsulate it in a 'public' property. ***  
Vulnerability Minor Open Not assigned 10min effort cwe  
4 days ago L11  
  
12 m.mar... //public List<string> NameList = new List<string>();  
13  
14 private string[] arr1 = new string[21];  
15 private string[] arr2 = new string[21];  
16
```


2. Il problema del Namespace e dei null



The screenshot displays the SonarQube interface with a list of 31 bugs. The left sidebar shows filters for Type (Bug: 31, Vulnerability: 148, Code Smell: 172) and Severity (Blocker: 0, Critical: 0, Major: 31, Minor: 0, Info: 0). The main panel lists several bugs, including:

- 'edificio' is null on at least one execution path. (Bug, Major, Open, Not assigned, 10min effort)
- Parameters to '.ctor' have the same names but not the same order as the method arguments. (Bug, Major, Open, Not assigned, 5min effort)
- Move the type 'Trade' into a named namespace. (Bug, Major, Open, Not assigned, 5min effort)
- Move the type 'giocatore' into a named namespace. (Bug, Major, Open, Not assigned, 5min effort)

Below the bug list, a section titled "Types should be defined in named namespaces" explains that types are declared in namespaces to prevent name collisions. It includes a "Noncompliant Code Example" showing C# code for a class and a struct without namespaces:

```
public class Foo // Noncompliant
{
}

public struct Bar // Noncompliant
{
}
```

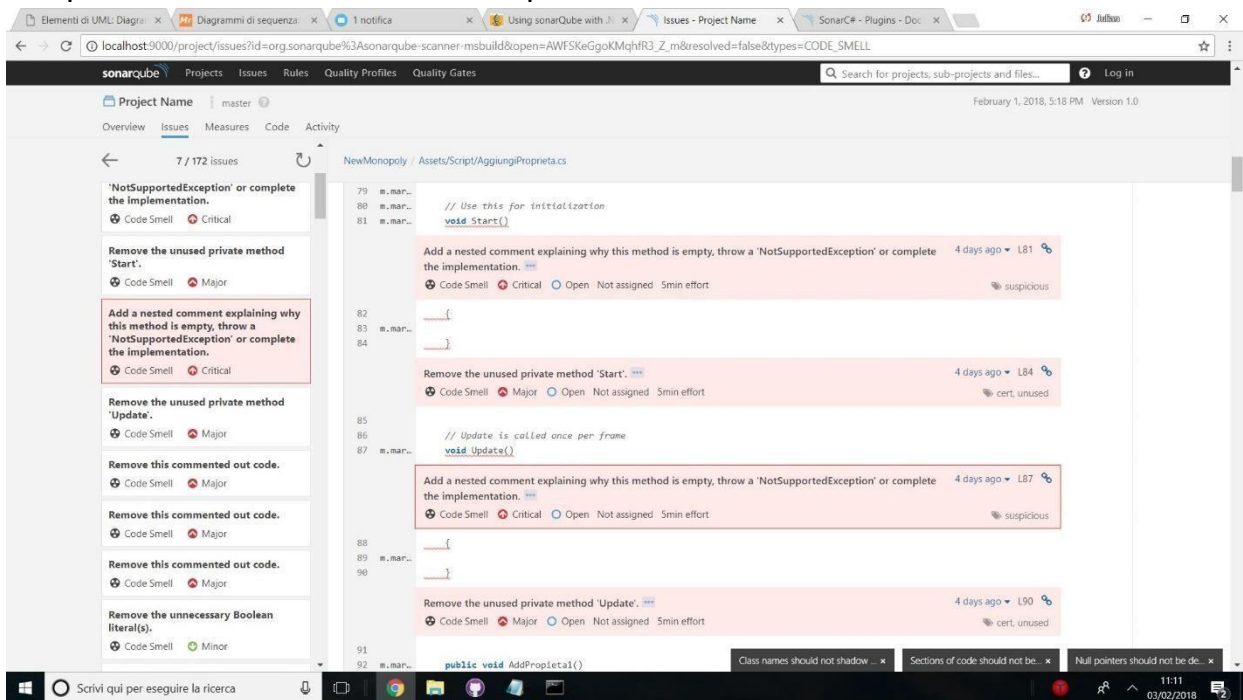
Two warnings are visible at the bottom right: "Class names should not shadow ..." and "Sections of code should not be ...".

Con Unity non possiamo fornire un Namespace, e non ne abbiamo neanche la necessità in quanto ogni script sarà associato ad un `gameObject` e non corriamo il rischio di name collisions .

Mentre per i Null, è Unity che fa un controllo sull'esistenza del `gameObject` e quindi Sonar riscontra una mancanza del controllo a null.

I 31 Bugs sono per queste ragioni, dovrebbero quindi essere.

3 Il problema dei metodi Start e Update



I metodi Start e Update sono dei metodi standard di Unity e non possono essere tolti in quanto descrivono cosa succede frame per frame allo script associato al gameObject : Start è chiamato per primo, per inizializzare lo script.

Update viene eseguito ogni frame per aggiornare la scena di gioco.

Non tutti gli script hanno bisogno di questi due metodi ma negli script dove dovranno per forza essere usati Sonar riscontrerà un errore