

Iterazione Finale

Andrea Belotti, Andrea Tintori, Sergio Fabiani

Marzo 2020

Contents

1	Introduzione	9
1.1	init	9
1.2	Struttura e sezioni dell'elaborato	9
2	Casi d'uso	11
2.1	Requisiti raggiunti	11
2.2	Requisiti Funzionali	12
2.3	Requisiti non Funzionali	12
2.4	Scenari	13
3	Architettura	15
3.1	Design architettura finale	15
4	Algoritmi	19
4.1	Pseudocodice	19
4.2	Calcolo della complessità	21
5	Testing	23
5.1	Analisi dinamica	23
5.2	Analisi statica	28
6	Conclusioni e considerazioni	31
6.1	Implementazioni future	31
6.2	Considerazioni Personali	31
	Glossary	33
	Acronyms	35

List of Figures

2.1	Scenario cambio del nickname	13
2.2	Scenario seleziona livello	14
3.1	Class Diagram Finale	16
3.2	Vista dei casi d'uso implementati	17
3.3	Deployment Diagram Finale	18
5.1	Powershell con gli errori dovuti al test statico presenti nel codice	29
5.2	Powershell che mostra che l'analisi statica è avvenuta con successo	30
5.3	File di output degli errori dell'analisi statica	30

Listings

4.1	pseudocode	19
5.1	testing della classe Player	24
5.2	testing della classe Arrow	27

Capitolo 1

Introduzione

1.1 init

Pdf della documentazione Finale redatta il 25 marzo 2021 del progetto Kokokò, per il corso Informatica III.

1.2 Struttura e sezioni dell'elaborato

[Nel Capitolo 2 - Casi d'uso](#), vengono descritti gli obiettivi raggiunti, i casi d'uso, le caratteristiche funzionali e non dell'iterazione 1 che sono state completate. Oltre a questo, nonostante il progetto sia finito, vengono descritti eventuali obiettivi e passaggi per poter rendere fruibile su PlayStore l'applicazione ed eventuali passi per migliorarne la grafica.

[Nel Capitolo 3 - Architettura](#), viene descritta l'architettura dell'applicazione che è stata raggiunta al termine dell'iterazione precedente, descrivendo così l'architettura finale.

[Nel Capitolo 4 - Algoritmi](#), vengono descritti gli algoritmi utilizzati all'interno del progetto, attraverso lo pseudo codice con la loro complessità.

[Nel Capitolo 5 - Testing](#), viene descritto il metodo di testing dinamico e statico e vengono descritti i risultati.

[Nel Capitolo 6 - Conclusioni e considerazione](#), vengono descritte le conclusioni del progetto.

Capitolo 2

Casi d'uso

In questo capitolo si parla dei requisiti funzionali e non, dei diagrammi UML e scenari completati.

2.1 Requisiti raggiunti

Durante la riunione di brainstorming del 31 marzo 2021 è stato fatto il punto della situazione controllando i risultati raggiunti e i casi d'uso completati. Più nello specifico sono stati completati i seguenti casi d'uso:

- Giocare con l'applicazione.
- SignUp.
- LogIn.
- Modificare le opzioni di gioco.
- Scegliere la modalità di gioco (dato che per ora si può solamente giocare off-line).
- Scegliere il livello nella modalità off-line (dato che per ora i livelli sono sequenziali e non è permesso scegliere livelli già completati).
- Creare una scoreboard così da permettere all'utente di vedere il suo punteggio.
- Rifinire le opzioni di gioco (attivare e disattivare l'audio, scegliere il nickname).

- Possibilità di fare il log-out.
- Mantenere la mail salvata nella schermata di Log-In così da non doverla inserire ogni volta manualmente.

Parlando invece di requisiti funzionali e non, si è potuto creare un'applicazione che funziona su più del 98% dei dispositivi Android, il peso attuale dell'applicazione è di 23,72 MB e la percentuale di crash in ore di utilizzo è inferiore al 2% (per questa percentuale è stata fatta utilizzare l'applicazione a 60 persone per circa 10 minuti l'una). I livelli creati sono 11 ed è stata creata una abilità che permette di fare un dash; è stato creato un algoritmo di match-making descritto nel capitolo sottostante. La velocità di transizione tra un livello è l'altro è inferiore al decimo di secondo. La Ram massima utilizzata durante il gioco è 97 MB, mentre l'uso medio è di 912KB al secondo. L'utilizzo del processore di un dispositivo di media gamma oscilla tra il 27% e il 31%, mentre l'utilizzo di GPU varia tra il 5,6% e il 6% (questi dati sono stati ottenuti tramite GPUWatch 2.0 per Samsung, un Widget in grado di dare specifiche riguardo le risorse utilizzate da un'applicazione).

2.2 Requisiti Funzionali

Sono stati raggiunti tutti i requisiti funzionali proposti nelle iterazioni precedenti.

2.3 Requisiti non Funzionali

I requisiti non-funzionali, ossia i requisiti e vincoli offerti dal sistema che il cliente chiede tramite caratteristiche desiderate, raggiunti sono:

- **Affidabilità:** Vogliamo garantire la massima affidabilità ed efficienza per la gestione dei dati dei clienti, separando in due database i dati sensibili da quelli di gioco. (RAGGIUNTO)
- **Portabilità :** Puntiamo a creare un'applicazione utilizzabile almeno dal 98% degli utenti Android. (*Raggiunto*)
- **Velocità :** Velocità di transizione tra i vari screen(1/10 di secondo) e velocità match-making(si punta ad un massimo di 30 secondi, anche

se dipende dal numero di giocatori connessi in quel momento). (*Raggiunto*)

- **Size** : Essendo un gioco in pixel art fatto ex novo, si punta a non utilizzare troppa memoria (sicuramente sotto i 512MB) e a far in modo che la ram utilizzata dal dispositivo sia la minore possibile. (*Raggiunto*)
- **Facilità d'uso** : Si punta ad avere sia l'installazione che il gioco user-friendly e il più chiaro possibili(Descritta nella guida). (*Raggiunto*)
- **Robustezza** : Puntiamo a non avere crash dell'applicazione e se per caso dovesse occorrere, faremo in modo che il tempo di risposta dell'applicazione sia il minore possibile(inferiore ai 5 secondi). (*Raggiunto*)

2.4 Scenari

Gli scenari descritti nella precedente iterazione (riportati qui sotto) sono stati pienamente "soddisfatti".

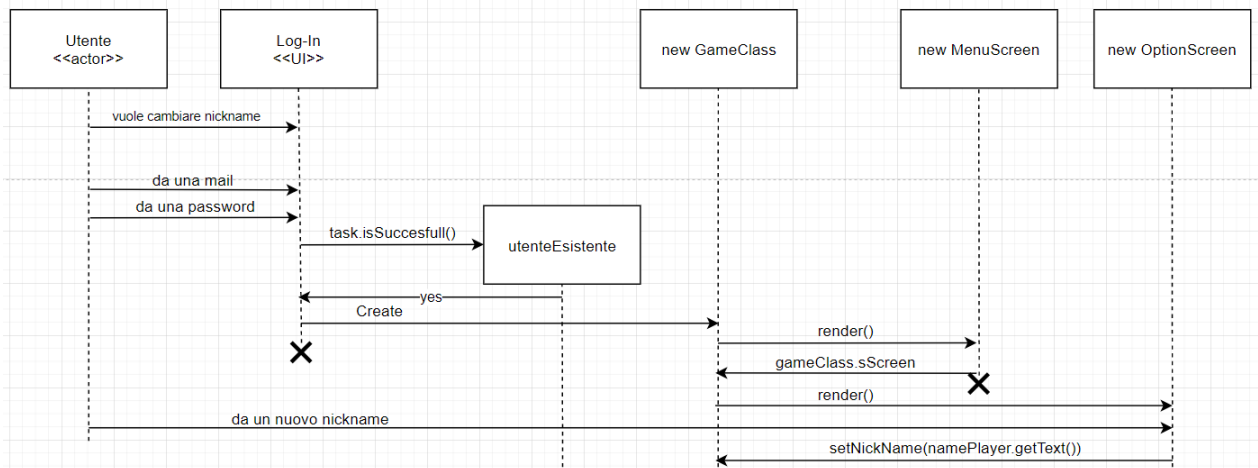


Figure 2.1: Scenario cambio del nickname

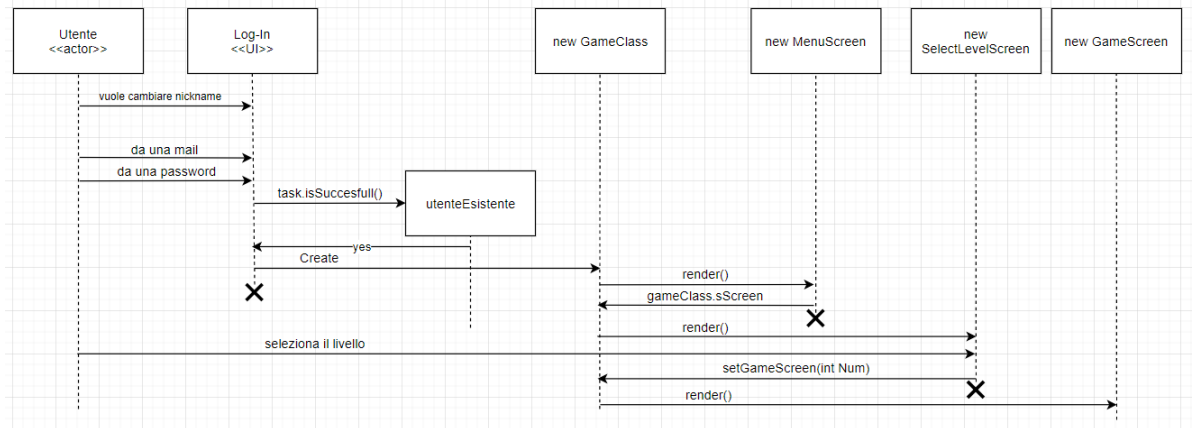


Figure 2.2: Scenario seleziona livello

Capitolo 3

Architettura

In questo paragrafo si propone l'architettura del progetto e alcune immagini che mostrano come si presenta l'applicazione programmata fino ad ora.

3.1 Design architettura finale

Di seguito si riporta il design architettonico del progetto alla fine dell'iterazione 1 completata il 31/03/2020:

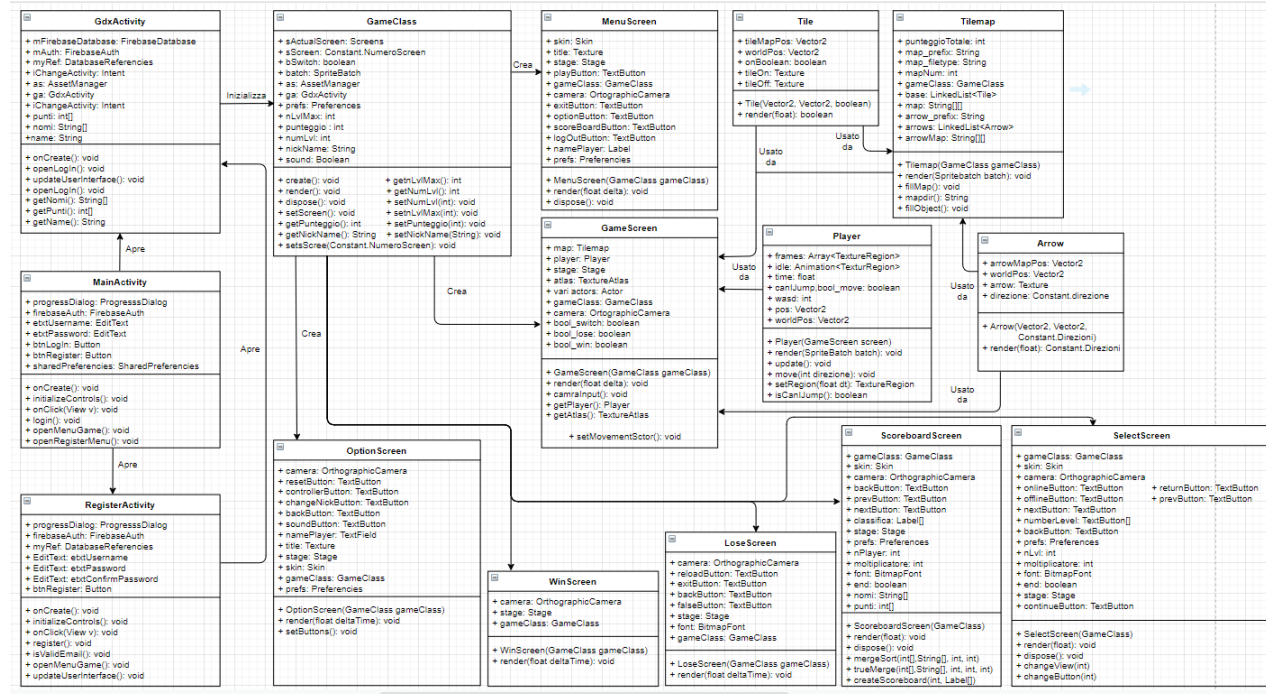


Figure 3.1: Class Diagram Finale

Come si può notare dal class diagram la base del progetto non è cambiata in modo drastico, semplicemente sono state aggiunte delle classi e dei metodi che vanno a implementare tutti i casi d'uso proposti all'interno del documento "Iterazione 1". Si possono notare altri due screen: SelectScreen e ScoreboardScreen che vanno a mostrare a schermo due nuovi diversi screen e altri metodi che permettono di utilizzare quest'ultimi. Di seguito è mostrato come si presentano all'interno del gioco le nuove funzionalità.



(a) Vista di SelectScreen



(b) Vista di ScoreboardScreen



(c) Vista di OptionScreen

Figure 3.2: Vista dei casi d'uso implementati

Infine si propone il deployment diagram del progetto che rispecchia il diagramma definito e pensato all'interno dell'iterazione 1.

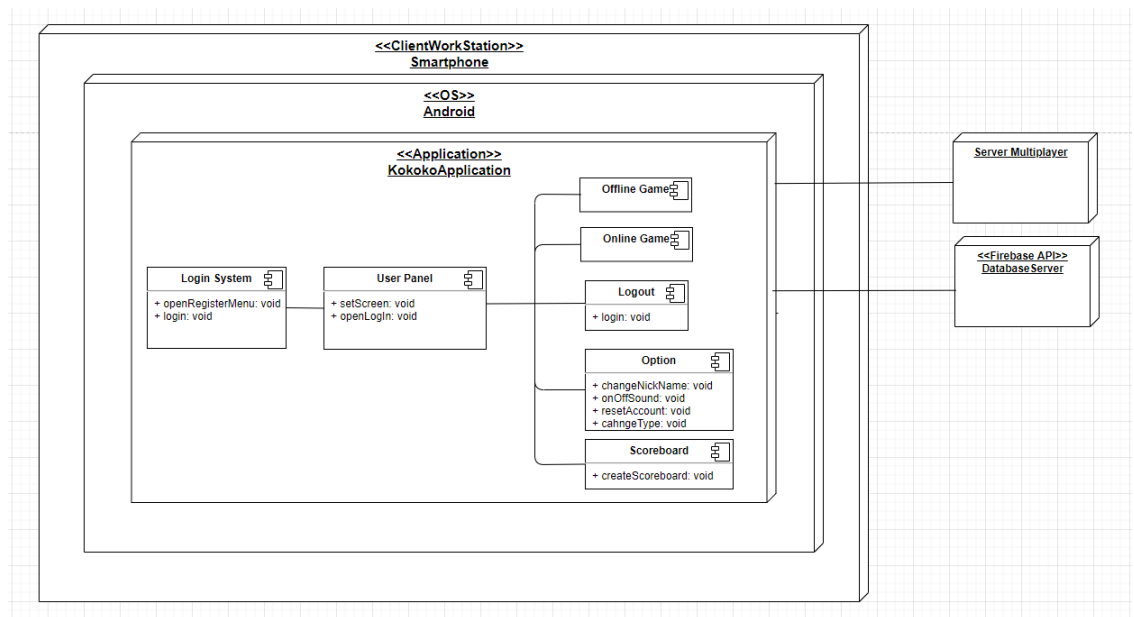


Figure 3.3: Deployment Diagram Finale

Capitolo 4

Algoritmi

In questo capitolo si parla degli algoritmi inseriti all'interno del codice, in particolar modo si è pensato di inserire un algoritmo riguardante il match-making. Il match-making dovrà riunire in una lobby un massimo di 4 giocatori diversi, una volta presenti farà partire la sessione di gioco. I giocatori avere caratteristiche comuni, quali il punteggio, che indica la loro bravura nel gioco, e il luogo di provenienza. Ogni NUMSEC secondi l'algoritmo allargherà il range di questi controlli così da non aver tempi d'attesa troppo elevati.

4.1 Pseudocodice

Di seguito è riportato lo pseudo-codice dell'algoritmo di match-making definito per il progetto:

```
1 sql_query = "SELECT PlayerID, MMR WHERE mmr between 900-1100
  AND ping < 200ms";
2 players := ExecuteQuery(sql_query);
3
4 rooms := roomsNext;
5 roomsNext := new EmptyRoomList;
6
7 for player in players do
8     available = true;
9     for room in rooms do
10         if(abs(player.mmr - room.meanMmr) <= (50 + room.wait
            * 10) && room.players < 3){
```

```

11         room.add(player);
12         sql_query = "DELETE WHERE PlayerID = " + player.
           playerID;
13         ExecuteQuery(sql_query);
14         available = false;
15         break;
16     endif
17 endfor
18 if(available) then
19     rooms.newRoom(player);
20 end if
21 endfor
22
23 for room in rooms do
24     if(room.players = 3) then
25         room.start();
26     else
27         room.wait += 1;
28         roomsNext.add(room);
29     end if
30 endfor

```

Listing 4.1: pseudocode

Questo codice permette di introdurre il matchmaking tramite l'uso di un algoritmo greedy per creare le stanze di gioco.

I giocatori in coda, non ordinati in alcun modo, vengono selezionati con una query al server dedicato ai giocatori che mandano una richiesta di partecipazione al gioco online; selezionati in base a condizioni ben precise (che possono dipendere dal ping, dal punteggio o altro) vengono poi passati all'algoritmo. Per ogni giocatore in attesa viene controllato se c'è una stanza disponibile, per far ciò prova ad abbinarlo ad ogni stanza finché non ne trova una adatta, nel caso nessuna rispetti le condizioni necessarie viene creata una nuova stanza a partire dal giocatore stesso. Il controllo sui requisiti della stanza è effettuato in base al numero di giocatori già presenti e il punteggio dei giocatori, in modo da fornire un adeguato livello di sfida; nel caso in cui una stanza rimanga in attesa, la condizione viene ammorbidita per facilitare l'abbinamento e rendere la coda meno lunga a discapito della rigidità delle regole. Al termine del controllo sui giocatori, tutte le stanze che sono pronte per essere avviate (3 giocatori al loro interno) vengono mandate in esecuzione sul server dedicato alle istanze di gioco, mentre quelle incomplete tornano in lista per il controllo successivo con il tempo trascorso in attesa aumentato.

L'algoritmo pensato evita di ordinare i giocatori poiché per prendere server in accomodato bisogna pagare in base a quanto vengono sfruttati, quindi si è pensato di evitare di utilizzare i server più del previsto così da limitare i costi. Naturalmente questo algoritmo è stato solo ipotizzato e mai provato in quanto: non si possiedono dei server e poiché affittando server ad hoc bisogna comunque adeguare il tutto ai server utilizzati.

4.2 Calcolo della complessità

La complessità di questo algoritmo si calcola tenendo conto del numero dei giocatori online presenti nella query (n) e del numero di stanze create(m).

Il caso **migliore** in cui posso ci si può trovare è che ogni giocatore è inserito nella prima stanza che trova, in questo caso la complessità temporale del nostro algoritmo è dell'ordine di n , ossia $O(n)$.

Il caso **peggiore** invece è quando ci si trova in una situazione di questo tipo: Ci sono 300 giocatori disponibili ($n = 300$), dopo un ciclo ci si trova nelle seguente condizione, 299 giocatori liberi, una stanza creata con un giocatore. Ad ogni ciclo successivo le stanze esistenti non soddisfano le condizioni per inserire il giocatore, quindi bisogna creare una nuova stanza dopo aver controllato tutte le precedenti. Così facendo ad ogni ciclo controllo le stanze precedentemente create arrivando ad avere una complessità temporale $\Omega(n^2)$.

Capitolo 5

Testing

5.1 Analisi dinamica

L'analisi dinamica è il processo di valutazione di un sistema software o di un suo componente basato sull'osservazione del suo comportamento in esecuzione. La preconditione necessaria per poter effettuare un test è la conoscenza del comportamento atteso per poterlo confrontare con quello osservato. L'oracolo è quella componente che conosce il comportamento atteso per ogni caso di test e può essere umano, quindi basato sulla conoscenza delle specifiche e sul giudizio personale, oppure automatico, generato dalla definizione di specifiche formali.

La prima fase di un'analisi dinamica prevede l'esecuzione di tutte le procedure e dei casi di test, prendendo come input sia la configurazione del software che quella dei test. Il testing è stato svolto su tutti i metodi pubblici presenti all'interno del codice, evitando i metodi privati dato che sono richiamati all'interno di quelli pubblici, quindi testando i pubblici, sono stati testati anche loro.

Per fare il testing dinamico è stato utilizzato JUnit4, in particolar modo sono stati utilizzati i Mock e le Asserzioni per i metodi delle classi che non facevano parte delle Activity; per le classi facenti parte delle Activity, quindi aventi una GUI, il testing è stato un po' più complicato ed è stato utilizzato un framework chiamato Espresso che ha semplificato il lavoro di testing.

Il testing delle classi con Activity è rimasto invariato rispetto ai test svolti per la scorsa iterazione, mentre sono stati migliorati alcuni test, ad esempio quello per la classe player. Di seguito sono riportati i test svolti per le classi

Player e Arrow.

```
1 package com.example.kokoko.libgdx;
2
3 import com.badlogic.gdx.graphics.OrthographicCamera;
4 import com.example.kokoko.Constant;
5 import com.example.kokoko.libgdx.Screen.GameScreen;
6 import org.junit.jupiter.api.Test;
7 import org.junit.runner.RunWith;
8 import org.mockito.Mockito;
9 import org.mockito.runners.MockitoJUnitRunner;
10 import static org.junit.jupiter.api.Assertions.*;
11
12
13 @RunWith(MockitoJUnitRunner.class)
14 class PlayerTest {
15
16     private static final Constant.Direzioni BL = Constant.
17     Direzioni.BOTTOMLEFT;
18     private static final Constant.Direzioni TL = Constant.
19     Direzioni.TOPLEFT;
20     private static final Constant.Direzioni TR = Constant.
21     Direzioni.TOPRIGHT;
22     private static final Constant.Direzioni BR = Constant.
23     Direzioni.BOTTOMRIGHT;
24
25
26     @Test
27     public void testMoveBL(){
28         GameScreen gameScreen = Mockito.mock(GameScreen.class);
29
30         Player player = new Player(gameScreen, new
31         OrthographicCamera());
32         int ris = player.move(BL);
33         assertEquals(ris, 3);
34     }
35
36     @Test
37     public void testMoveTR(){
38         GameScreen gameScreen = Mockito.mock(GameScreen.class);
39
40         Player player = new Player(gameScreen, new
41         OrthographicCamera());
42         int ris = player.move(TR);
43         assertEquals(ris, 4);
44     }
45 }
```



```
37     }
38
39     @Test
40     public void testMoveTL(){
41         GameScreen gameScreen = Mockito.mock(GameScreen.class);
42     };
43         Player player = new Player(gameScreen, new
44 OrthographicCamera());
45         int ris = player.move(TL);
46         assertEquals(ris, 1);
47     }
48
49     @Test
50     public void testMoveBR(){
51         GameScreen gameScreen = Mockito.mock(GameScreen.class);
52     };
53         Player player = new Player(gameScreen, new
54 OrthographicCamera());
55         int ris = player.move(BR);
56         assertEquals(ris, 2);
57     }
58
59     @Test
60     public void testCanJump(){
61         GameScreen gameScreen = Mockito.mock(GameScreen.class);
62     };
63         Player player = new Player(gameScreen, new
64 OrthographicCamera());
65         assertTrue(player.isCanJump());
66     }
67
68     @Test
69     public void testVect2(){
70         GameScreen gameScreen = Mockito.mock(GameScreen.class);
71     };
72         Player player = new Player(gameScreen, new
73 OrthographicCamera());
74         assertNotNull(player.getPos());
75     }
76
77     @Test
78     void testUpDate(){
79         GameScreen gameScreen = Mockito.mock(GameScreen.class);
80     };
```

```
72     Player player = new Player(gameScreen, new
    OrthographicCamera());
73     player.update(Mockito.anyFloat());
74     assertTrue(player.time >= 0);
75 }
76 }
```

Listing 5.1: testing della classe Player

```
1 package com.example.kokoko.libgdx;
2
3 import com.badlogic.gdx.graphics.g2d.SpriteBatch;
4 import com.badlogic.gdx.math.Vector2;
5 import com.example.kokoko.Constant;
6 import org.junit.jupiter.api.Test;
7 import static org.junit.jupiter.api.Assertions.*;
8
9 class ArrowTest {
10
11     @Test
12     void render() {
13         Arrow arrow1 = new Arrow(new Vector2(0,0), new
Vector2(0,0), Constant.Direzioni.BOTTOMLEFT);
14         Arrow arrow2 = new Arrow(new Vector2(0,0), new
Vector2(0,0), Constant.Direzioni.TOPLEFT);
15         Arrow arrow3 = new Arrow(new Vector2(0,0), new
Vector2(0,0), Constant.Direzioni.BOTTOMRIGHT);
16         Arrow arrow4 = new Arrow(new Vector2(0,0), new
Vector2(0,0), Constant.Direzioni.TOPRIGHT);
17
18         assertEquals(arrow1.render(new SpriteBatch()),
Constant.Direzioni.BOTTOMLEFT);
19         assertEquals(arrow2.render(new SpriteBatch()),
Constant.Direzioni.TOPLEFT);
20         assertEquals(arrow3.render(new SpriteBatch()),
Constant.Direzioni.BOTTOMRIGHT);
21         assertEquals(arrow4.render(new SpriteBatch()),
Constant.Direzioni.TOPRIGHT);
22
23     }
24 }
```

Listing 5.2: testing della classe Arrow

La copertura raggiunta tramite il testing è del 78%. La percentuale è migliorata ma alcuni metodi privati non sono stati testati, dato che sono nascosti all'utente e quindi solitamente è buona norma ignorarli poiché vengono controllati quando si attuano i test dei metodi pubblici. Il coverage invece attuato tramite Espresso è arrivato al 93% comprendo pressoché tutte le casistiche.

5.2 Analisi statica

Questo tipo di test del software viene eseguito prima di mettere in azione il software. I test statici vengono eseguiti per cercare gli errori negli algoritmi, codici o documenti. Gli errori commessi durante la scrittura del software vengono controllati per la correzione utilizzando test statici.

Per questo codice è stato utilizzato Checkstyle uno strumento di sviluppo per aiutare i programmatori a scrivere codice Java che aderisce a uno standard di codifica. Automatizza il processo di controllo del codice Java per risparmiare agli umani questo compito noioso (ma importante). Ciò lo rende ideale per i progetti che desiderano applicare uno standard di codifica. Checkstyle può controllare molti aspetti del tuo codice sorgente. Può trovare problemi di progettazione di classi, problemi di progettazione di metodi. Ha anche la capacità di controllare il layout del codice e problemi di formattazione. I controlli vengono scritti in .xml e questo lo rende estremamente personalizzabile. Per poterlo utilizzare è stato creato un file checkstyle.xml inserito all'interno del progetto, in questo file sono stati inseriti diversi `<module>` che vanno a definire regole da seguire per avere uno standard comune nella programmazione del codice in Java.

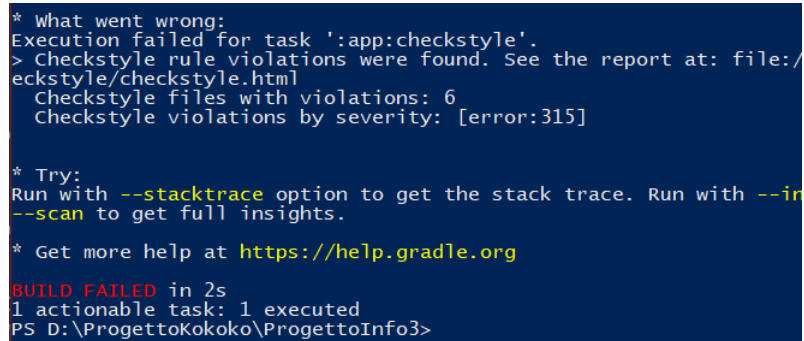
Il file .xml utilizzato è lo stesso della scorsa iterazione.

Sono stati creati diversi casi di analisi statica per il progetto svolto. I principali controlli fatti riguardano:

- Block Checks
- Class Design
- Coding
- Headers
- Imports
- Javadoc Comments
- Metrics
- Miscellaneous
- Modifiers

- Naming Conventions
- Regexp
- Size Violation
- Whitespace

Inizialmente con l'aggiunta delle classi per poter eseguire i casi d'uso richiesti sono stati trovati, tramite il testing dinamico, 315 errori.



```
* What went wrong:
Execution failed for task ':app:checkstyle'.
> Checkstyle rule violations were found. See the report at: file:/
eckstyle/checkstyle.html
Checkstyle files with violations: 6
Checkstyle violations by severity: [error:315]

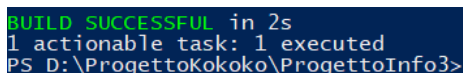
* Try:
Run with --stacktrace option to get the stack trace. Run with --in
--scan to get full insights.

* Get more help at https://help.gradle.org

BUILD FAILED in 2s
1 actionable task: 1 executed
PS D:\ProgettoKokoko\ProgettoInfo3>
```

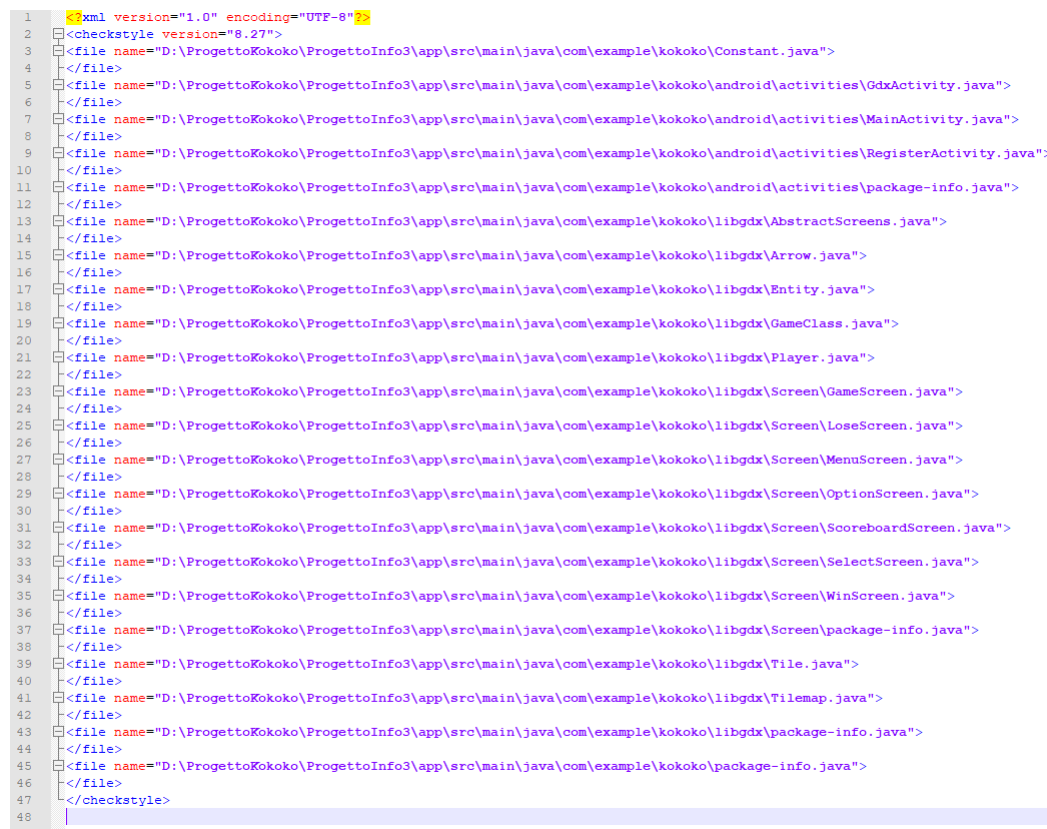
Figure 5.1: Powershell con gli errori dovuti al test statico presenti nel codice

Dopo vari processi il risultato raggiunto è l'assenza di errori nell'analisi statica all'interno del progetto. I principali problemi riscontrati riguardano gli Whitespace, gli Import e la complessità di alcuni controlli poiché sono stati utilizzato troppi if in modo concatenato, aumentando la complessità del codice.



```
BUILD SUCCESSFUL in 2s
1 actionable task: 1 executed
PS D:\ProgettoKokoko\ProgettoInfo3>
```

Figure 5.2: Powershell che mostra che l'analisi statica è avvenuta con successo



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <checkstyle version="8.27">
3   <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\Constant.java">
4   </file>
5   <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\android\activities\GdxActivity.java">
6   </file>
7   <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\android\activities\MainActivity.java">
8   </file>
9   <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\android\activities\RegisterActivity.java">
10  </file>
11  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\android\activities\package-info.java">
12  </file>
13  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\AbstractScreens.java">
14  </file>
15  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Arrow.java">
16  </file>
17  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Entity.java">
18  </file>
19  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\GameClass.java">
20  </file>
21  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Player.java">
22  </file>
23  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\GameScreen.java">
24  </file>
25  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\LoseScreen.java">
26  </file>
27  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\MenuScreen.java">
28  </file>
29  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\OptionScreen.java">
30  </file>
31  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\ScoreboardScreen.java">
32  </file>
33  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\SelectScreen.java">
34  </file>
35  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\WinScreen.java">
36  </file>
37  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Screen\package-info.java">
38  </file>
39  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Tile.java">
40  </file>
41  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\Tilemap.java">
42  </file>
43  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\libgdx\package-info.java">
44  </file>
45  <file name="D:\ProgettoKokoko\ProgettoInfo3\app\src\main\java\com\example\kokoko\package-info.java">
46  </file>
47 </checkstyle>
```

Figure 5.3: File di output degli errori dell'analisi statica

Capitolo 6

Conclusioni e considerazioni

6.1 Implementazioni future

Dopo aver completato il progetto non si è fermato tutto, infatti sono stati pensati altri algoritmi da poter utilizzare per il match-making, altre abilità da creare, altri modi di giocare, altre funzionalità da aggiungere etc. Un esempio è la funzione trova all'interno della classifica che permette al giocatore di cercare un altro giocatore tramite il nickname oppure tramite la posizione in classifica.

6.2 Considerazioni Personali

La preparazione di questo progetto ha contribuito ad un generale aumento della conoscenza personale sull'argomento di studio. L'intero percorso di lettura, ricerca, studio e programmazione e progettazione non solo ha permesso di padroneggiare le nozioni base sull'argomento, ma ha permesso di capire decisamente meglio anche altri argomenti. Oltre a questo mi ha dato una valida motivazione per entrare nel mondo di Latex, che inizialmente mi sembrava ostico ma più la relazione andava avanti più i procedimenti erano spontanei. La soddisfazione per questa opportunità offerta e il lavoro fatto è tanta.

Glossary

algorithm è una strategia che serve per risolvere un problema ed è costituito da una sequenza finita di operazioni (dette anche istruzioni), consente di risolvere tutti i quesiti di una stessa classe. 9, 12, 19, 21

architettura l'insieme dei criteri di progetto in base ai quali è progettato e realizzato un computer, oppure un dispositivo facente parte di esso. Per estensione, descrivere l'architettura di un dispositivo significa, in particolare, elencarne le sottoparti costituenti ed illustrarne i rapporti interfunzionali. 9

casi d'uso tecnica usata nei processi di ingegneria del software per effettuare in maniera esaustiva e non ambigua, la raccolta dei requisiti al fine di produrre software di qualità. 9, 11

crash indica il blocco o la terminazione improvvisa, non richiesta e inaspettata di un programma in esecuzione (sistema operativo o applicazione), oppure il blocco completo dell'intero computer. 12, 13

database un insieme di dati strutturati ovvero omogeneo per contenuti e formato, memorizzati in un computer, rappresentando di fatto la versione digitale di un archivio dati o schedario. 12

iterazione l'atto di ripetere un processo con l'obiettivo di avvicinarsi a un risultato desiderato. 1, 9, 15

match-making designa l'unione di due individui per affinità. 12

off-line usato quale sinonimo della locuzione non in linea o fuori linea. 11

testing indica un procedimento, che fa parte del ciclo di vita del software, utilizzato per individuare le carenze di correttezza, completezza e affidabilità delle componenti software in corso di sviluppo.. 9

Acronyms

UML Unified Modeling Language. 11