UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Ingegneria del Software**

Software Testing Gamification: Code Defenders

Anno Accademico 2019 - 2020

Candidato:

Michael Carannante

matr. N46003817

Indice

Introduzione

L'attività di test è davvero molto importante durante lo sviluppo di sistemi software perché ha l'obiettivo di produrre del codice di qualità. Ciò significa privo di potenziali errori che possono compromettere la corretta esecuzione del programma. Molto spesso però questa fase non viene affrontata con la dovuta attenzione da parte degli sviluppatori, causando così la produzione di software scadenti caratterizzati da bug e crash frequenti.

In ambito aziendale questo comporta ingenti perdite di denaro, infatti come descritto nel report del 2017 Software Fail Watch si è stimata una perdita di circa 1.7 trilioni di dollari a causa di problemi software che potevano essere risolti con una attività di test più approfondita.

Tutto ciò è spesso dovuto a come il testing viene affrontato in ambito lavorativo e accademico. Nel primo caso per portare a termine il progetto commissionato entro le scadenze previste, si impiega la maggior parte del tempo nella produzione di codice che soddisfi le specifiche andando a trascurare di conseguenza l'aspetto funzionale incappando, successivamente, in errori da risolvere.

Mentre i corsi di Ingegneria del Software spesso non stimolano l'attenzione degli studenti verso questa fase perché appare ripetitiva e poco interessante rispetto alla produzione di nuovo codice. In più a causa della sua natura intrinsecamente euristica, l'apprendimento della stessa risulta più impegnativa e richiede molta più pratica di quanto comunemente insegnato.

Una possibile soluzione molto diffusa è l'uso di strumenti in grado di generare casi di test in modo automatico. Si è però constatato che gli esseri umani tendono a generare casi di test molto più robusti, accurati e con un chiaro significato.

Per misurare quanto è stata proficua una tecnica di test si fa largo uso di metodi poco affidabili come la "line coverage". Questa misura rappresenta quante linee di codice sono state coperte, e quindi eseguite, dalla test suite. Ovviamente è un ottimo indicatore su come stiano evolvendo i test, ciò non toglie che bisogna usarla con cautela perché non garantisce che siano efficienti.

A tal proposito un approccio diverso suggerisce l'uso di una particolare metodologia di test chiamata Mutation Testing. Si basa sull'inserimento di errori artificiali, detti anche "mutanti", con la successiva stima di quanti di essi una test suite riesce a distinguere dal programma originale.

Il numero di mutanti individuati viene riportato come punteggio. Tutto ciò risulta essere molto utile siccome i mutanti che non vengono scovati possono guidare il programmatore a produrre dei casi di test più specifici aumentando così la loro abilità di rilevare errori.

La misura basata sull'analisi delle mutazioni, non solo fornisce la quantità di linee di codice eseguite dai test, ma produce anche una stima di quanto siano efficaci a trovare gli errori.

Ciononostante, l'approccio del Mutation Testing non è molto diffuso nella pratica poiché vi sono diverse ipotesi che lo rendono difficile da utilizzare.

Prima di tutto il numero di mutanti che possono essere introdotti all'interno di un programma non banale può aumentare a dismisura, introducendo quindi problemi di scalabilità.

Secondo poi solo una parte dei mutanti è essenzialmente utile perché molti di essi possono essere facilmente identificati oppure ridondanti. Per questo la loro scoperta da parte di una test suite va ad incrementare il punteggio falsando così la misura dell'efficienza.

Come ultimo punto si ha la presenza di alcuni mutanti di tipo "Equivalent", ovvero hanno un funzionamento uguale al codice originale che dunque non compromette la corretta esecuzione del programma. Sono tali da non poter essere distinti da nessun caso di test.

Nonostante queste limitazioni si è notato che l'attività di Mutation Testing si adatta perfettamente alla Gamefication. Infatti, si è pensato di applicare concetti relativi al Gaming e dunque ai videogiochi, all'attività di test. Ciò ha portato all'uso di elementi come la suddivisione in squadre, assegnazione di punti e composizione di Scoreboard, per spronare al massimo gli utenti facendo leva sulla loro intrinseca voglia di competere per generare dei casi di test sempre più efficaci.

Dall'unione dei due approcci è nato Code Defenders, un gioco utilizzato soprattutto in ambito accademico che affronta l'attività di Mutation Testing in modo creativo.

Capitolo 1: Software Testing

Il gioco è progettato per lavorare su delle classi scritte in linguaggio JAVA.

Per testarle viene quindi utilizzato il framework JUnit 4, appartenente alla famiglia XUnit che è un insieme di framework dedicati al testing per diversi linguaggi di programmazione.

Un framework può essere visto come una struttura logica di appoggio che nel caso specifico consente di agevolare l'attività di test.

Al XUnit appartengono diverse componenti che consentono di scrivere un caso di test e di controllare se è andato tutto a buon fine oppure vi è stato qualche problema.

La prima componente è il metodo setup(). Questo è eseguito prima di ogni caso di test e risulta utile per soddisfare le precondizioni relative al test che si vuole proporre.

Successivamente vi è il metodo teardown() eseguito dopo ogni test. Utile per resettare le post condizioni.

Vi sono inoltre le asserzioni, sono l'elemento fondamentale perché consentono di controllare se i dati in uscita, prodotti da un certo caso di test, risultano soddisfare una particolare condizione booleana.

Ve ne sono diverse e cambiano in base alla condizione che si vuole utilizzare. Di seguito è mostrato un estratto della tabella di tutte le asserzioni disponibili in JUnit, presente all'indirizzo http://junit.sourceforge.net/javadoc_40/org/junit/Assert.html.

Method Summary	
static void	assertArrayEquals(byte[] expecteds, byte[] actuals) Asserts that two byte arrays are equal.
static void	assertArrayEquals(char[] expecteds, char[] actuals) Asserts that two char arrays are equal.
static void	assertArrayEquals(int[] expecteds, int[] actuals) Asserts that two int arrays are equal.
static void	assertArrayEquals(long[] expecteds, long[] actuals) Asserts that two long arrays are equal.
static void	assertArrayEquals(java.lang.Object[] expecteds, java.lang.Object[] actuals) Asserts that two object arrays are equal.
static void	assertArrayEquals(short[] expecteds, short[] actuals) Asserts that two short arrays are equal.
static void	assertArrayEquals(java.lang.String message, byte[] expecteds, byte[] actuals) Asserts that two byte arrays are equal.
static void	assertArrayEquals(java.lang.String message, char[] expecteds, char[] actuals) Asserts that two char arrays are equal.
static void	assertArrayEquals(java.lang.String message, int[] expecteds, int[] actuals) Asserts that two int arrays are equal.
static void	assertArrayEquals(java.lang.String message, long[] expecteds, long[] actuals) Asserts that two long arrays are equal.
static void	assertArrayEquals(java.lang.String message, java.lang.Object[] expecteds, java.lang.Object[] actuals) Asserts that two object arrays are equal.
static void	assertArrayEquals(java.lang.String message, short[] expecteds, short[] actuals) Asserts that two short arrays are equal.

Figura 1.1: Estratto della tabella delle Asserzioni

La più comune è assertEquals che verifica se due dati sono uguali oppure no. In caso affermativo vuol dire che il test è andato a buon fine, in caso contrario si segnalerà il suo fallimento.

In generale vengono inserite alla fine di un caso di test e vengono imposte sui dati prodotti e i valori che si aspetta riscontrare dopo l'esecuzione del programma originale.

Infine, vi è un metodo per ogni caso di test quindi il codice vero e proprio.

Date le componenti, la struttura di un caso di test è la seguente:

- Inizializzazione precondizioni: si settano i valori relativi allo specifico test in modo da soddisfarle
- Inserimento dei valori di input: viene fatto attraverso la chiamata alle funzioni pubbliche del tipo set()
- Codice di test: insieme di istruzioni atte al test della classe
- Valutazione delle asserzioni: è il controllo di espressioni booleane che devono risultare vere se il test dà esito positivo, ovvero quando i dati in uscita e/o le post condizioni risultano equivalenti a quelli attesi

Nel caso di JUnit 4 i test avranno tutti questa struttura, preceduta da un import di metodi e annotazioni del framework. In più sarà presente "@Test" per marcare quel metodo come un caso di test e un Assert alla fine del codice per verificarne la riuscita.

Capitolo 2: Code Defenders

1. Descrizione

Code Defenders è un gioco web-based competitivo focalizzato sull'attività di Mutation Testing Software. Il suo scopo è quello di svolgere l'attività di test su una particolare classe JAVA presa in esame, detta CUT. In generale in ogni partita vengono coinvolti almeno due giocatori, uno nel ruolo di Attaccante e uno in quello di Difensore.

Gli attaccanti, detti anche Attackers, hanno il compito di aggiungere dei mutanti più minuziosi possibile, in modo da renderli difficili da scovare. Mentre i difensori, detti Defenders, dovranno creare dei casi di test più forti possibile per scoprire i mutanti ed ucciderli.

Il gioco, quindi, integra tutte le principali componenti del Mutation Testing quali la creazione di mutanti, la loro uccisione e il controllo sui mutanti Equivalent.

Infatti, nell'ultimo caso se un defender ha il dubbio che un particolare mutante è Equivalent può iniziare un duello con il creatore dello stesso. In base a come si conclude si determinerà se è Equivalent assegnando dei punti extra al vincitore.

Anche se Code Defenders è attualmente solo un prototipo risulta essere molto consono nell'approccio del Mutation Testing perché lo rende divertente e facile da comprendere.

Infatti, questo gioco è spesso utilizzato in ambito accademico per affrontare in modo diverso il Software Testing.

Durante lo svolgimento dell'elaborato si è avuto modo di testare la piattaforma scaricata e installata in locale, ciononostante è anche disponibile online al sito https://code-defenders.org/, analizzando tutti i suoi casi d'uso.

2. Regole di gioco

In generale l'applicativo mette a disposizione tre modalità di gioco: Battleground, Melee e Puzzles. La prima è la modalità classica, ovvero gli utenti sono divisi in due squadre: Attackers e Defenders. Competono cercando di totalizzare una quantità di punti sempre maggiore.

Gli Attackers hanno il compito di mutare il codice, intuitivamente si tratta di piccole modifiche apportate alla classe originale (per esempio la sostituzione di un operatore logico, modifica di condizioni booleane, e così via) che vanno ad alterare la corretta esecuzione del programma. Starà poi ai Defenders creare appositi test per scovare gli errori.

Quindi metaforicamente parlando la CUT è protetta dai Defenders con test sempre più robusti, in modo da respingere tutti i Mutants.

Il gioco, quindi, può essere organizzato in turni dove iniziano sempre gli Attackers. Una volta creato un mutant, il turno passa ai Defenders che avranno il compito di comporre uno o più test. Quando sono pronti essi verranno eseguiti sulla classe e in base ai risultati verranno assegnati i punti.

Vi è inoltre la possibilità di identificare un mutante di tipo Equivalent. Cioè se un defender sospetta che un mutante sia Equivalent allora può sfidare il suo creatore in un duello.

A questo punto l'attacker avrà la possibilità di smentire l'affermazione scrivendo un caso di test che identifichi il mutante portandolo in vantaggio in termini di punti, oppure confermare dando il vantaggio al defender.

Nel seguito si tratterà una simulazione di una partita tra due utenti appartenenti alle due squadre, vedendo nello specifico un duello.

Un'ulteriore modalità sono i Melee games. Sono molto simili alle partite classiche con la differenza che un utente è sia Attaccante che Difensore. Dunque si possono scrivere test per risolvere mutanti inseriti da altri utenti, oppure si possono inserirne di nuovi.

In questa modalità nonostante si possano inserire mutanti e casi di test, nessuno dei giocatori ne può trarre beneficio in termini di punti. Ovvero scrivere un test che va ad uccidere un mutante inserito dallo stesso utente non comporta l'uccisione e dunque il guadagno di punti.

Il test viene inserito nella suite e sarà utilizzato per l'identificazione di mutanti scritti da altri utenti. L'ultima modalità è quella dei Puzzles.

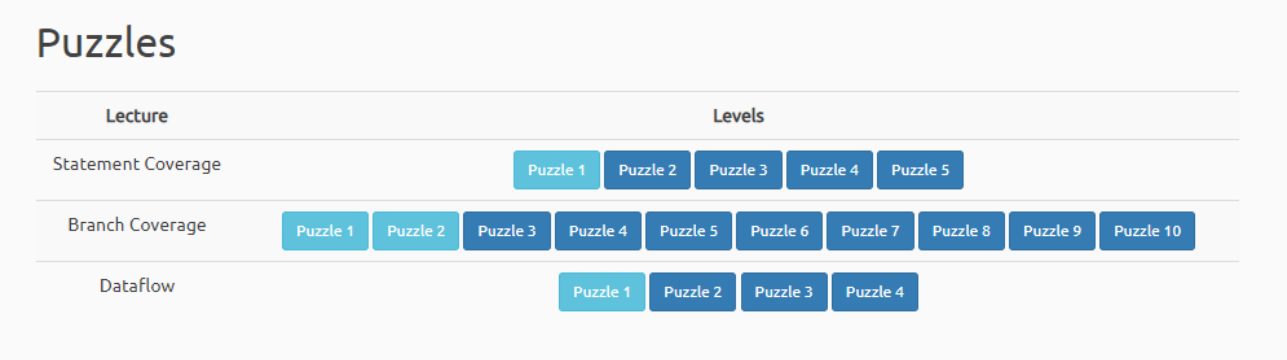


Figura 2.2.1: Schermata principale dei puzzles

Consistono in piccoli mini giochi in cui gli utenti possono iniziare a familiarizzare con l'ambiente, i mutanti e i test che possono essere scritti. Infatti sono delle partite organizzate in livelli di difficoltà, con l'ulteriore suddivisione rispetto alla tecnica di test a cui appartengono.

E' da specificare che sono delle partite di un unico turno, in cui l'utente deve scrivere un caso di test o inserire un mutante che soddisfi le richieste mostrate in alto. In più non si guadagnerà alcun punto per il superamento degli stessi.

Quindi a tutti gli effetti non sono gestiti da un sistema automatizzato che consente lo svolgimento di una partita completa.

Vengono suddivisi rispetto a tre diverse tecniche: Statement Coverage, Branch Coverage e Dataflow.

La prima consiste nello sviluppo di test che coprano almeno una volta tutti gli statement del programma. Infatti si adotta spesso come una possibile metrica per i casi di test.

Le richieste dei puzzles consistono nella scrittura di casi di test che coprano tutte le istruzioni della classe proposta, oppure l'inserimento di mutanti nelle linee non ancora testate.

Figura 2.2.2: Esempio di puzzles Coverage Statement, inserimento mutante

Figura 2.2.3: Esempio di puzzles Coverage Statement, scrittura caso di test

La Branch Coverage si riferisce invece all'esecuzione di tutti i percorsi logici che vi possono essere nel programma. Per esempio se vi sono degli if, i test si devono assicurare di coprire sia il caso in cui la condizione è vera con il relativo set di istruzioni e sia il caso in cui essa è falsa.

Infatti nei puzzles viene chiesto di inserire mutanti nei percorsi logici non ancora coperti, o invece di scrivere dei test tali da provarli tutti.

Figura 2.2.4: Esempio di puzzles Branch Coverage, scrittura caso di test

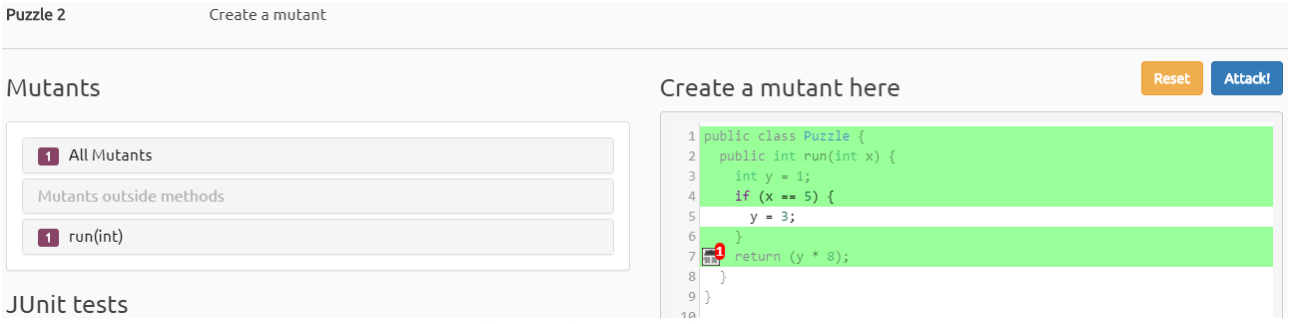


Figura 2.2.5: Esempio di puzzles Branch Coverage, inserimento mutante

Il Dataflow invece consiste nel testare tutti i possibili percorsi logici che vi possono essere in un programma in base ai dati con cui si sta lavorando.

Di conseguenza nei puzzles verrà chiesto di testare una determinata condizione di funzionamento fornita dai dati presenti nella classe. In questo gruppo di mini giochi non è richiesto l'aggiunta di mutanti.

Figura 2.2.6: Esempio di puzzles Dataflow, scrittura caso di test

3. Sistema di Punteggio

Esiste un sistema di punteggio ben definito tale da stimolare gli utenti a creare dei mutanti sempre più fittizi e dei test sempre più robusti.

Nonostante le partite siano organizzate in squadre, i singoli utenti ricevono i punti. Inoltre rispetto alla partita che si sta giocando, il punteggio della squadra è dato dalla somma dei punti dei singoli componenti.

In una generica partita è sempre possibile visualizzare la Scoreboard con il pulsante dedicato posizionato in alto a destra.

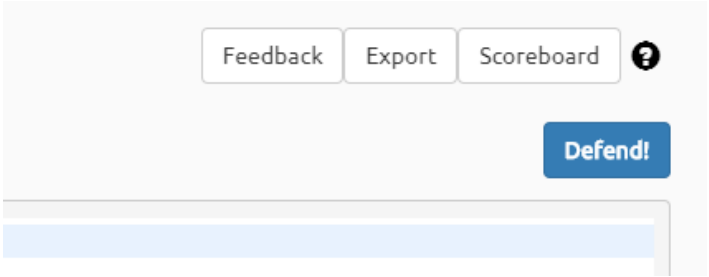


Figura 2.3.1: Pulsante Scoreboard

Nella tabella si trovano non solo quanti punti hanno totalizzato le due squadre, ma anche delle info riguardanti i singoli utenti.

Si ha in rosso la squadra degli attackers. Ogni utente è individuato dal proprio username. Accanto vi è il numero di mutanti inserito e successivamente un resoconto dello stato dei mutanti, cioè quanti di quelli inseriti sono ancora vivi, sono stati uccisi o sono stati contestati come Equivalent.

Vi sarà poi un ulteriore resoconto relativo ai duelli intrapresi, riportando il numero di duelli vinti, persi e ancora in corso. Infine si trova il numero di punti guadagnati.
Alla fine dell'elenco dei giocatori attaccanti vi sarà un'ulteriore riga che riporta le stesse informazioni ma complessive di tutta la squadra.
Allo stesso modo sotto è presente in blu l'elenco dei defenders con il numero di test prodotti e il numero di mutanti uccisi.

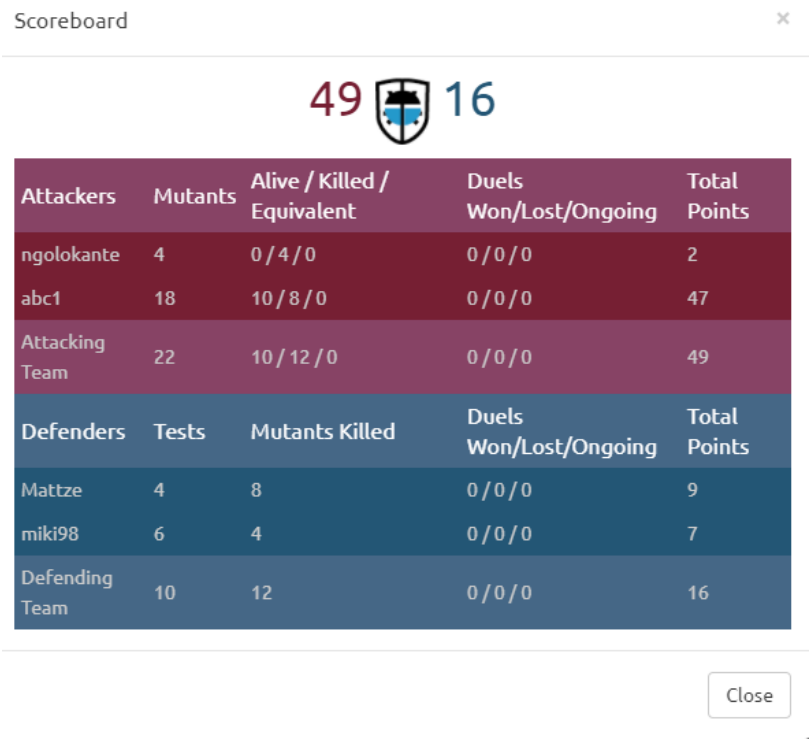


Figura 2.3.2: Esempio Scoreboard

Il sistema di punteggio si basa sull'assegnazione di punti a ogni mutante e test, che varia man mano con l'evoluzione del gioco.
I mutanti guadagnano punti in questo modo:

- Se un mutante quando viene creato è ucciso da un test già esistente, allora non riceve nessun punto.
- Un mutante guadagna un punto per ogni test che copre la linea mutata senza identificarlo. Quindi anche se creati su linee pesantemente testate, rischiosi siccome i test già esistenti possono identificarli, permettono di guadagnare più punti.
- Se un mutante è creato e non ucciso da nessun nuovo caso di test, allora guadagna un punto. Questi in aggiunta a quelli del caso precedente.
- Una volta che il mutante è ucciso, il suo punteggio non è più incrementato.

Invece i test:

- Un test guadagna un numero di punti pari alla quantità di mutanti uccisi più uno. Questo vale per i mutanti creati prima del caso di test e per tutti quelli successivamente inseriti.
- Un test guadagna un punto per ogni mutante appena creato che riesce ad uccidere.
- Quando un mutante viene inserito, viene eseguita sul codice tutta la test suite in ordine di creazione, quindi dai nuovi fino ai più vecchi. Se uno o più test uccidono questo mutante, il punto viene assegnato solo al caso di test più vecchio, quindi all'ultimo che è stato eseguito.

In definitiva il punteggio di un attaccante è definito dalla somma di tutti i punti guadagnati con i mutanti. Mentre il punteggio dei difensori è dato dalla somma dei punti ottenuti con i test.

Con i duelli i punti sono assegnati in questo modo:

- Se un difensore dichiara un mutante Equivalent, l'attaccante può smentire tale affermazione andando a creare un caso di test che lo uccida. In questo caso il mutante è ucciso e l'attaccante ottiene un punto.
- Tuttavia se l'attaccante accetta che il mutante è Equivalent, oppure il gioco termina durante il duello, allora perderà tutti i punti relativi a quel mutante e il difensore guadagnerà un punto.

Durante lo svolgimento dei duelli il mutante supposto Equivalent resta vivo, dunque potrebbe essere ucciso dagli altri difensori con altri casi di test. Se ciò accade il duello è cancellato. In più i mutanti soggetti dei duelli posso continuare a guadagnare punti se nuovi casi di test non riescono ad ucciderlo.

Infine se gli attaccanti inseriscono test che eseguono, ma falliscono l'uccisione dei mutanti Equivalent, allora perdono il duello, il mutante è ucciso e assunto Equivalent.

Un problema che si potrebbe verificare con il sistema dei punteggi per i duelli è che i difensori negli ultimi istanti di gioco potrebbero dichiarare tutti i mutanti Equivalent, lasciando così gli attaccanti senza il tempo di reagire. Lo stesso vale per questi ultimi, siccome potrebbero verso la fine della partita inserire solo mutanti Equivalent senza dare la possibilità ai difensori di verificare e iniziare i possibili duelli.

Tutto ciò però è evitato aggiungendo un "tempo di grazia" (per esempio di un'ora) dopo la conclusione della partita. In questo periodo di tempo nessun nuovo mutante o test può essere inserito perché è completamente dedicato alla risoluzione dei mutanti Equivalent.

Ciò consente di non svantaggiare nessuna delle due squadre.

In più il tempo è diviso in due parti: nella prima i difensori possono reclamare i mutanti come Equivalent, nella seconda gli attaccanti si occupano di smentire o approvare tali affermazioni.

4. Installazione in locale

Anzitutto per installare Code Defenders occorre scaricare le sue componenti principali sul sito GitHub all'indirizzo: https://github.com/imartinezortiz/CodeDefenders/tree/docker-deployment

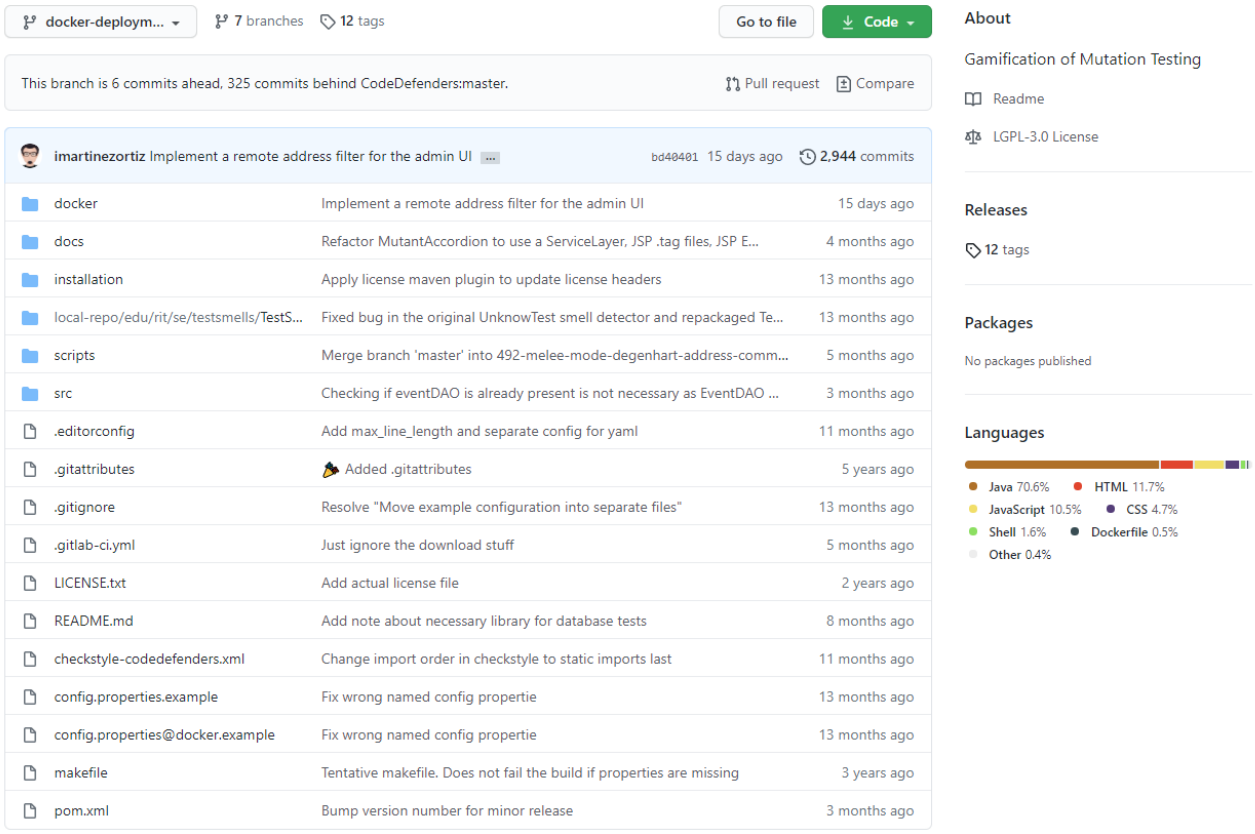


Figura 2.4.1: Schermata del contenuto presente sul link GitHub

GitHub per chi non lo conoscesse è un sito sul quale qualsiasi sviluppatore può caricare i propri software rendendoli disponibili per il download. Infatti Code Defenders è un progetto open source sviluppato e sostenuto alla Chair of Software Engineering II, all'università di Passau e all'università di Leicester, messo a disposizione di tutti su questa piattaforma.

Scaricare tutti i file compresi nella cartella principale, detta anche Master, è molto semplice: basta andare a destra, sul pulsante code e selezionare la voce Download Zip.

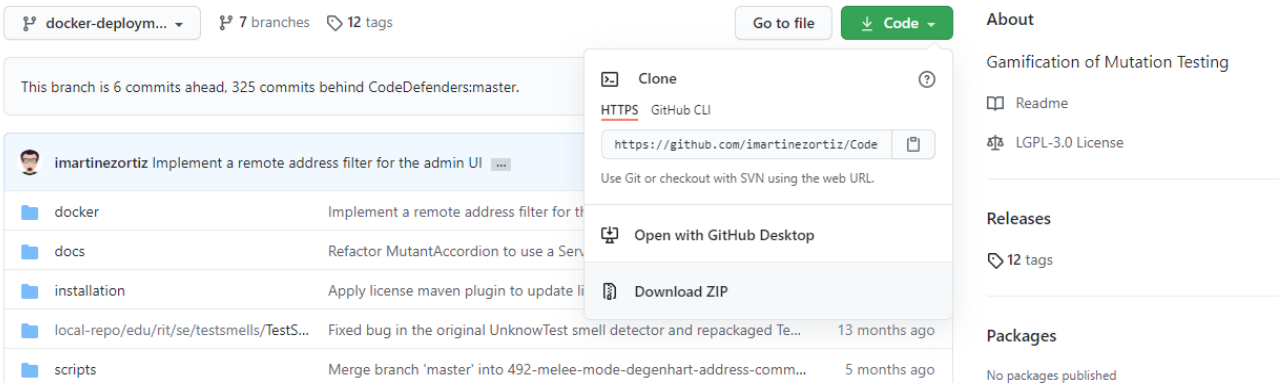


Figura 2. 4.2: Pulsante Download Zip

Una volta terminato il download si possono estrarre e posizionare i file nella cartella che più fa comodo.

Sul sito non solo troviamo tutti i file del gioco, ma anche una guida su come installarlo.

Vi sono due possibilità. Nella pagina principale è presente un primo procedimento in cui si vanno a installare i singoli programmi necessari all'installazione, composizione ed esecuzione del gioco. Ogni componente va configurata rispetto alle proprie esigenze. Ciononostante risulta essere un procedimento lungo e tedioso. Per questo si è seguita una strada diversa.

Tra le cartelle del gioco ve n'è una in particolare chiamata Docker. Essa fa riferimento all'omonimo programma che consente di semplificare di molto l'installazione.

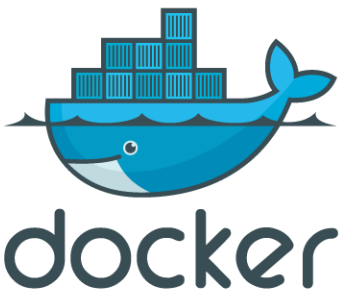


Figura 2.4.3: Logo Docker

Si introduce dunque Docker un progetto open source con lo scopo di automatizzare la distribuzione di applicazioni sotto forma di "contenitori" leggeri, portabili e autosufficienti che possono essere eseguiti in cloud (pubblici o privati) o in locale.

Dunque i contenitori, detti anche Container, sono l'insieme di tutti i file, dipendenze e dati che necessita un'applicazione per essere eseguita. Ad esempio: librerie, file di configurazione, script e così via.

Il processo di distribuzione di un'applicazione è fatto creando un'immagine, cioè un file che contiene tutti i dati precedentemente elencati, usata dal Docker per comporre un container che eseguirà l'applicazione in esso contenuto.

Per questo motivo i container hanno la proprietà di essere autosufficienti, cioè hanno già tutte le necessarie dipendenze dell'applicazione, di conseguenza non richiedono particolari configurazioni sull'host.

In più risultano portabili perché essi sono distribuiti attraverso un formato standard, le immagini, che possono essere lette ed eseguite da qualsiasi server Docker.

Inoltre i container sono anche leggeri perché sfruttano direttamente i servizi del sistema operativo ospitante al posto di richiedere l'esecuzione di un kernel virtualizzato come avviene nel caso delle Virtual Machine.

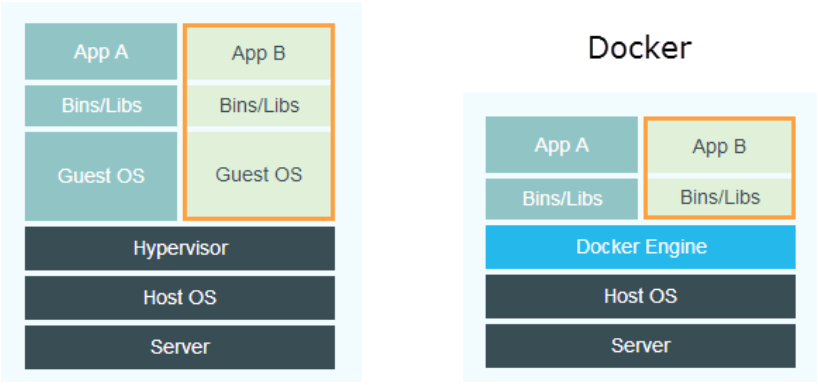


Figura 2.4.5: Docker vs VM

A tal fine si vogliono sfruttare i servizi offerti dal Docker per inserire l'applicazione all'interno di un container ed eseguirla. Per fare ciò occorre innanzitutto scaricare e installare il programma.

Ci si può collegare al sito ufficiale https://www.docker.com/, cliccare su Get Started e selezionare il download di Docker Desktop. Una volta completato basta eseguire il setup e seguire i passaggi.

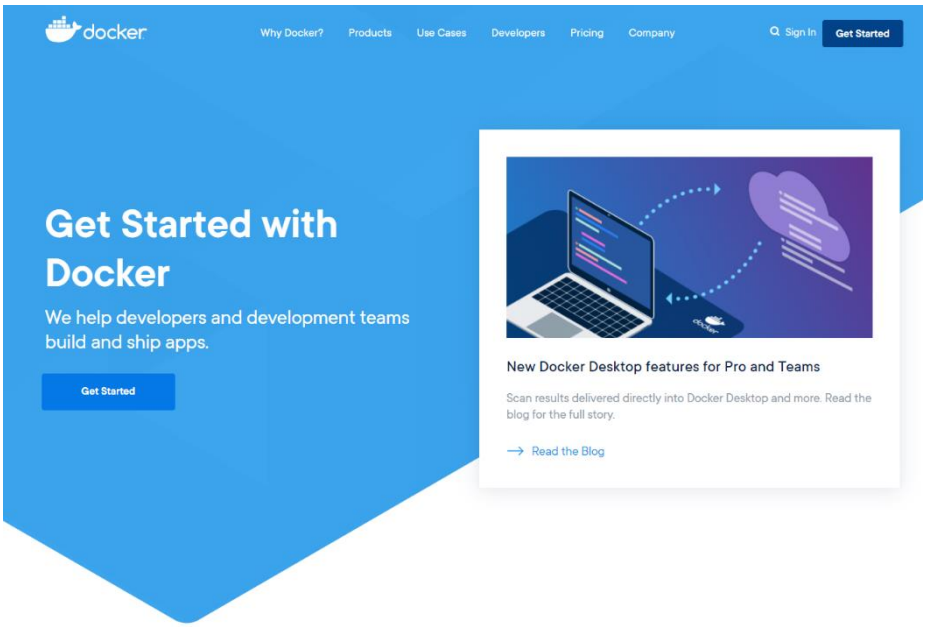


Figura 2.4.6: Home sito ufficiale

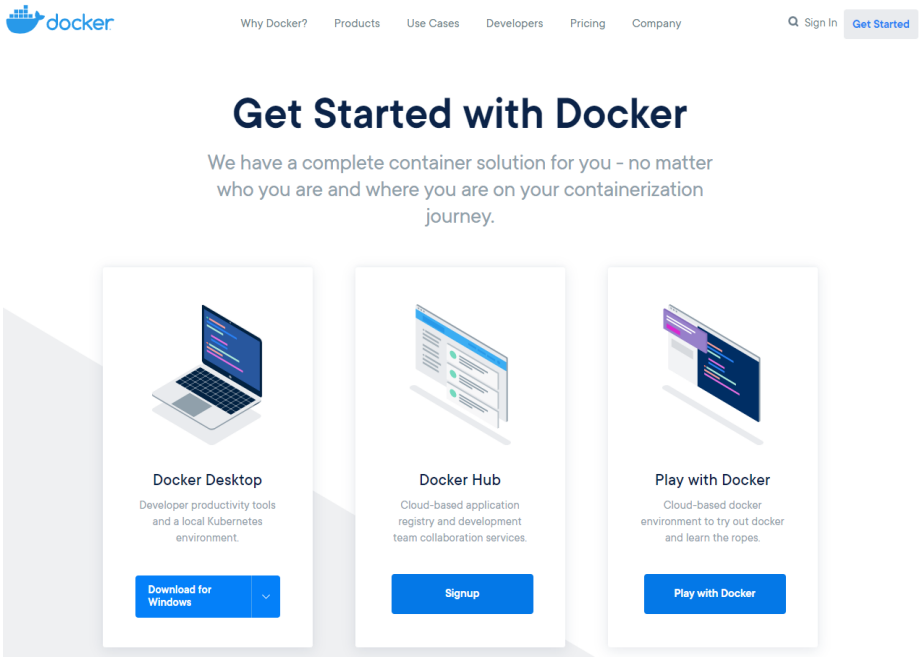
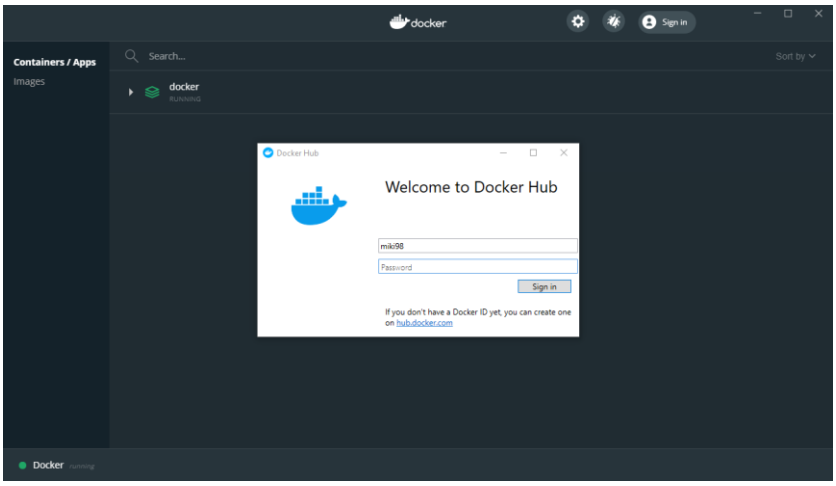


Figura 2.4.7: Schermata download

Conclusa l'installazione Docker sarà già operativo sul pc. Nel caso particolare di Windows è probabile che non si avvii subito, ma riporti un messaggio di errore con un link da cui scaricare un componente aggiuntivo. Bisogna scaricarlo per risolvere dei problemi di compatibilità interna col sistema operativo presente sulla macchina.

In ogni caso seguendo i passi riportati sul sito web del link, si è in grado di installare il componente e una volta fatto Docker sarà perfettamente funzionante.

A questo punto prima di iniziare ad utilizzarlo è bene creare un nuovo account e fare il login sull'applicazione desktop. Ciò è possibile collegandosi di nuovo al sito ufficiale oppure aprendo il programma e cliccando su Sign in seguendo poi i relativi passaggi per la creazione di un nuovo utente.



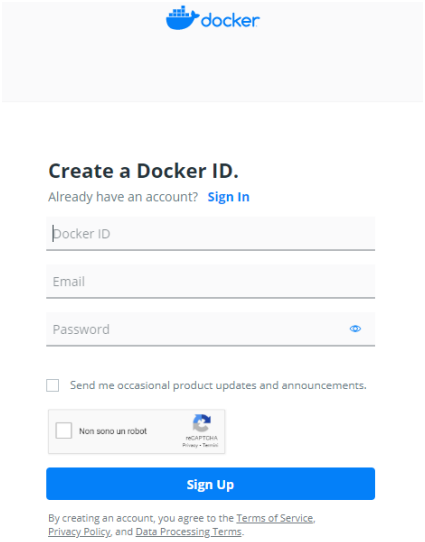


Figura 2.4.8: Login App Desktop

Figura 2.4.9: Login sito ufficiale

Prima di iniziare la procedura di installazione del gioco bisogna aggiungere delle variabili che fungono da parametri per il Docker. L' obiettivo è quello di comporre i container in cui si andrà a mettere tutte le componenti del gioco. Difatti Code Defenders è composto da più container che lavorano simultaneamente. Tutto ciò sarà fatto in maniera autonoma dal Docker eseguendo i file presenti nella cartella del gioco, tuttavia occorre inizializzare alcune variabili di ambiente.

Per farlo si crea un file nella cartella docker del gioco con intestazione ".env".

Può essere creato con qualsiasi editor di testo aggiungendo l'opportuna estensione che sta per "Environment".

Al suo interno si inserisce il seguente contenuto: "RELEASE_VERSION=v1.7".

In questo modo si dirà al Docker quale versione del gioco si sta per installare.

A questo punto si può finalmente utilizzare il Docker per comporre i container. Si procede dunque con l'apertura del prompt dei comandi e ci si posiziona nella cartella di Code Defenders precedentemente installata.

Se non si è familiari col prompt di windows si può utilizzare il comando DIR per visualizzare il contenuto, sotto forma di elenco, della directory corrente e il comando CD per entrare in una specifica cartella.

Raggiunta la cartella Code Defenders occorrerà accedere alla cartella docker e digitare il seguente comando: docker-compose up --no-start.

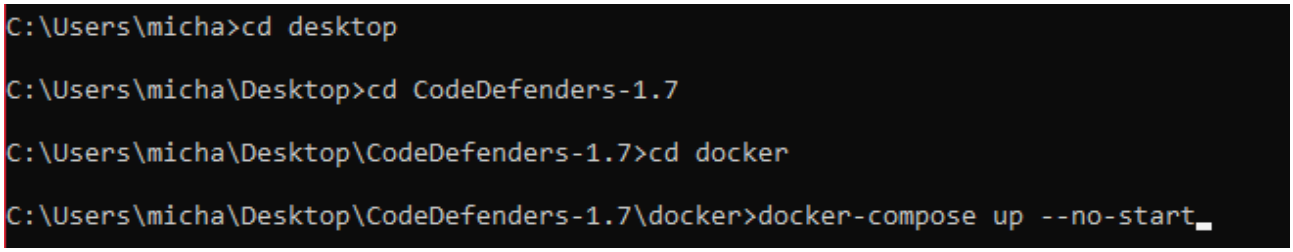


Figura 2.4.10: Schermata CMD

Tale comando ci consentirà di scaricare, costruire, creare e unire tutti i container associati al gioco. In più siccome abbiamo aggiunto l'opzione "--no-start" alla fine del procedimento essi non verranno portati nello stato di esecuzione.

Come anticipato, Code Defenders è composto da tre container che verranno appunto creati e uniti in maniera del tutto autonoma.

Quello che otterremo sul CMD dopo l'inserimento del comando è un report molto accurato sul proseguimento dell'installazione. Si nota che il Docker in automatico installa tutti i programmi di cui ha bisogno per l'esecuzione del gioco, in particolare non sono fatte direttamente sul pc, ma saranno interne al Docker e verranno associate ai contenitori che stiamo creando.

La composizione avrà una durata variabile e dipenderà molto dal pc e dalla rete che si sta utilizzando. In media dovrebbe durare un quarto d'ora.

Al termine il procedimento sarà quasi finito: si dovrà aggiungere un ultimo parametro nel file ".env". Si aggiunge "ALLOWED_REMOTE_ADDR_REGEXP=172.XXX.YYY.ZZZ".

Bisogna sostituire l'indirizzo con quello restituito dal seguente comando eseguito sul CMD, sempre nella cartella docker del gioco: " "docker network inspect docker_default --format "{{(index (index .IPAM.Config) 0).Gateway}}" ".

Adesso per far in modo che il CMD si accorga delle modifiche apportate nel file, bisognerà chiuderlo e riaprirlo andandosi a riposizionare nella cartella docker.

Infine basterà digitare il comando "docker-compose up" che, siccome i container sono stati creati in precedenza, li porterà nello stato di Running.

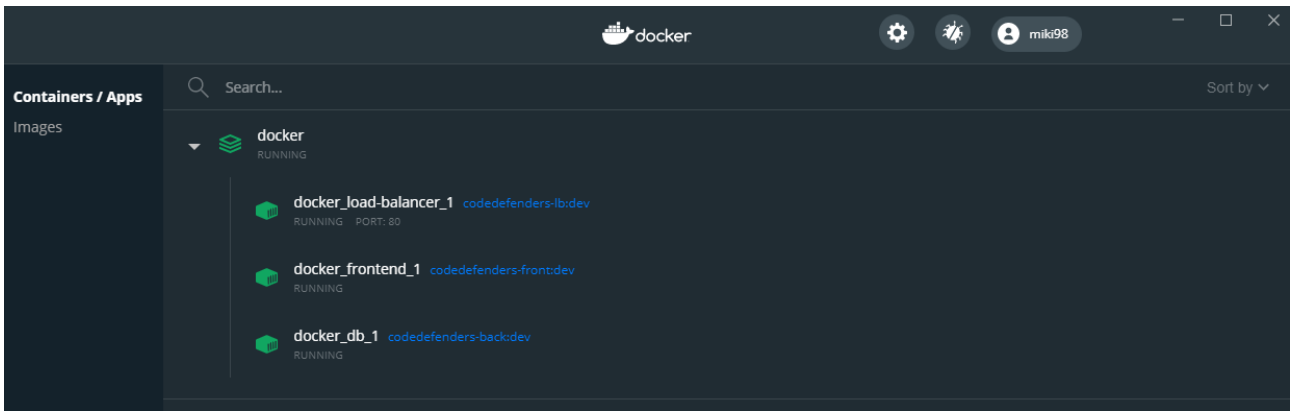


Figura 2.4.11: Containers in stato di Running

Non resta che andare su un qualsiasi browser e digitare "localhost/codedefenders/" e apparirà la schermata iniziale del gioco.

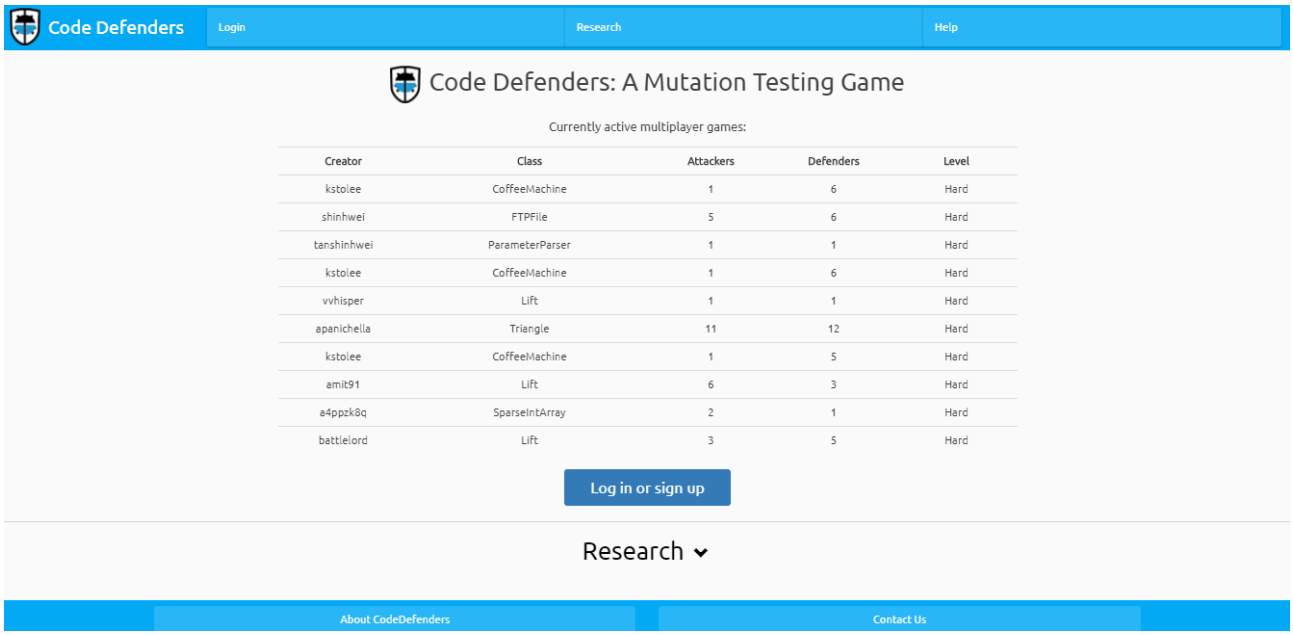


Figura 2.4.12: Home Code Defenders

E' da notare che durante l'installazione viene chiesto di consentire l'accesso a reti pubbliche con un messaggio da parte del firewall di Windows. Accettando tale richiesta renderemo possibile l'accesso alla piattaforma anche da altri host collegati alla stessa rete locale.

Per farlo sarà necessario digitare nel browser l'indirizzo IP della macchina su cui è stato installato Code Defenders seguito da "/codedefenders/".

È stato anche possibile connettersi al gioco attraverso l'internet. Infatti si è seguito il procedimento di installazione su una macchina dell'Università disponibile a un certo indirizzo IP pubblico.

Digitando tale indirizzo nella barra di ricerca del browser seguito da /codedefenders/ si può accedere alla piattaforma e giocare.

5. Descrizione Piattaforma

Prima di poter far uso dell'applicazione occorre registrarsi e fare l'accesso. Ciò è possibile cliccando su "Log in or sign up" inserendo nome utente, e-mail e password. E' da specificare che nel nome utente e password non vanno inseriti caratteri speciali, altrimenti la registrazione non va a buon fine.

Una volta acceduti, il sito conduce alla home. Essa è strutturata mostrando in alto un insieme di pulsanti formanti una barra di navigazione, mentre nel resto della pagina sono presenti l'elenco di tutte le partite a cui l'utente ha partecipato sotto il nome di "My games" e subito dopo l'elenco di tutte le partite attualmente in corso accessibili dall'utente.

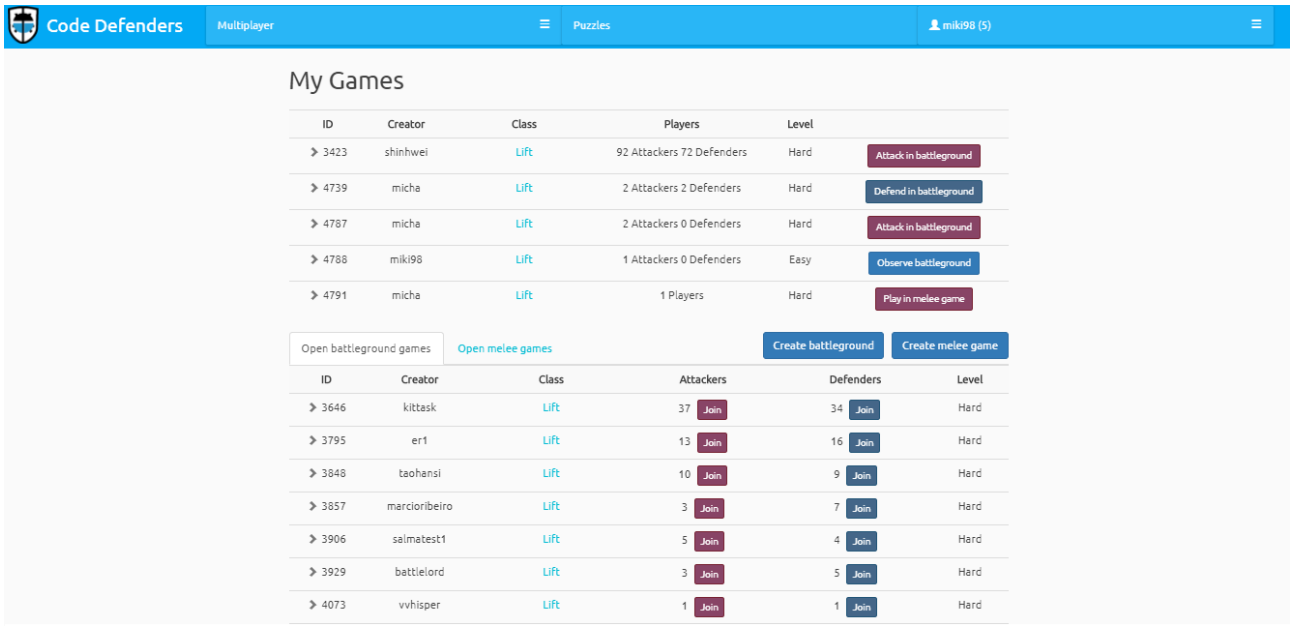


Figura 2.5.1: Home Code Defenders con login, presa dal sito ufficiale

Ogni utilizzatore può decidere di partecipare a una partita oppure crearne una nuova.

Nel primo caso basta dare un'occhiata alla home, scegliere la partita a cui si vuole partecipare e premere sul pulsante join. In particolare per quanto riguarda le partite classiche, i Battleground, esistono due pulsanti perché ci si può unire alla squadra degli Attackers o a quella dei Defenders.

Invece scegliendo la lista dedicata ai Melee games ve ne sarà solo uno per via della loro modalità di gioco.

Figura 2.5.2: Pulsanti join per i Battleground

Figura 2.5.3: Pulsante join per i Melee

Nelle liste genericamente viene indicato un ID, il creatore della partita e il nome della CUT su cui si sta giocando. Infine è presente anche la difficoltà della partita: facile o difficile.
Esse verranno illustrate nel seguito durante la creazione di una partita classica.
A tal proposito si possono creare le partite secondo le modalità Battleground e Melee.
Dopo il settaggio delle varie impostazioni sarà visibile nella home nell' elenco My games. Invece per tutti gli altri utenti sarà disponibile nell'elenco generale.
Osservando la parte superiore della pagina si nota la presenza di pulsanti dedicati alla navigazione del sito.

Figura 2.5.4: Barra di navigazione

Partendo da sinistra si trova il logo di Code Defenders che in qualsiasi momento riporta alla home del gioco. Subito dopo si trova il pulsante Multiplayer. Se premuto visualizza una finestra a tendina con i seguenti collegamenti: Games, History, Leaderboard.
Andando su Games il sito visualizzerà niente altro che la home con l'elenco di tutte le partite. History consente invece di visualizzare l'elenco di tutte le partite a cui si è partecipato e che sono terminate, quindi non più disponibili.
L'ultimo collegamento, Leaderboard, mostra l'elenco dei punteggi di tutti i giocatori presenti su Code Defenders.

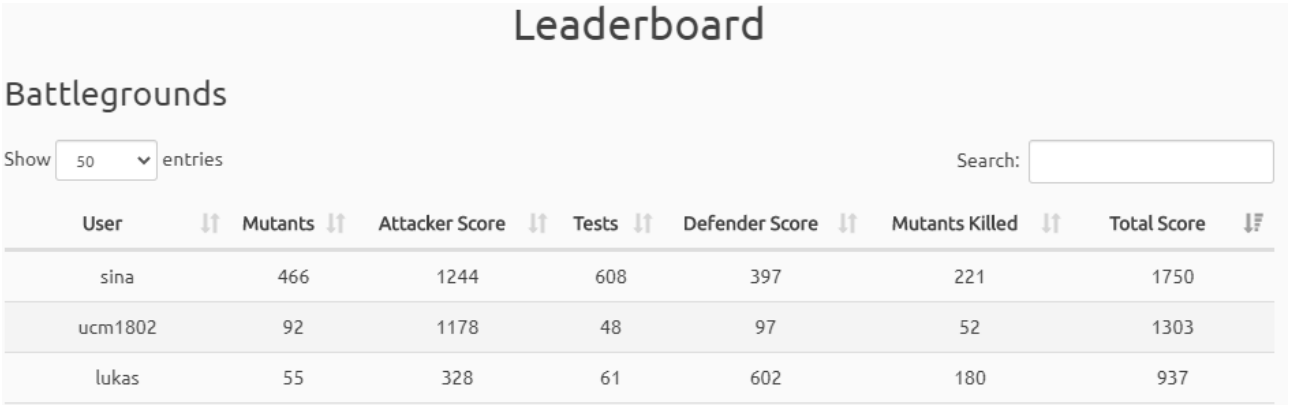


Figura 2.5.5: Leaderboard

Si visualizzano diverse informazioni. La prima è l'username degli utenti che, se si è interessati a uno in particolare, può essere cercato con l'apposita barra di ricerca a destra dell'elenco.

Si prosegue con il numero di mutanti creati e il punteggio complessivo totalizzato nel ruolo di attaccante. Nello specifico tale valore rappresenta tutti i punti che l'utente ha totalizzato in tutte le partite in cui ha assunto il ruolo di attaccante.

Proseguendo si trova il numero di test creati con il relativo punteggio guadagnato in tutte le partite sostenute come difensore. Si riporta infine il numero globale di mutanti uccisi e il punteggio totale dato dalla somma dei punti totalizzati con i mutanti, i test e i duelli.

Ritornando alla barra di navigazione, proseguendo si trova il pulsante per accedere ai Puzzles visti in precedenza e infine un pulsante dedicato all'utente. Premendolo si aprirà un ulteriore finestra a tendina mostrando: Profile, Help, Logout e un piccolo elenco con tutte le azioni fatte dall'utente. Andando su Profile si potrà visualizzare le proprie informazioni personali inserite nel momento della registrazione, dando la possibilità di cambiarle.

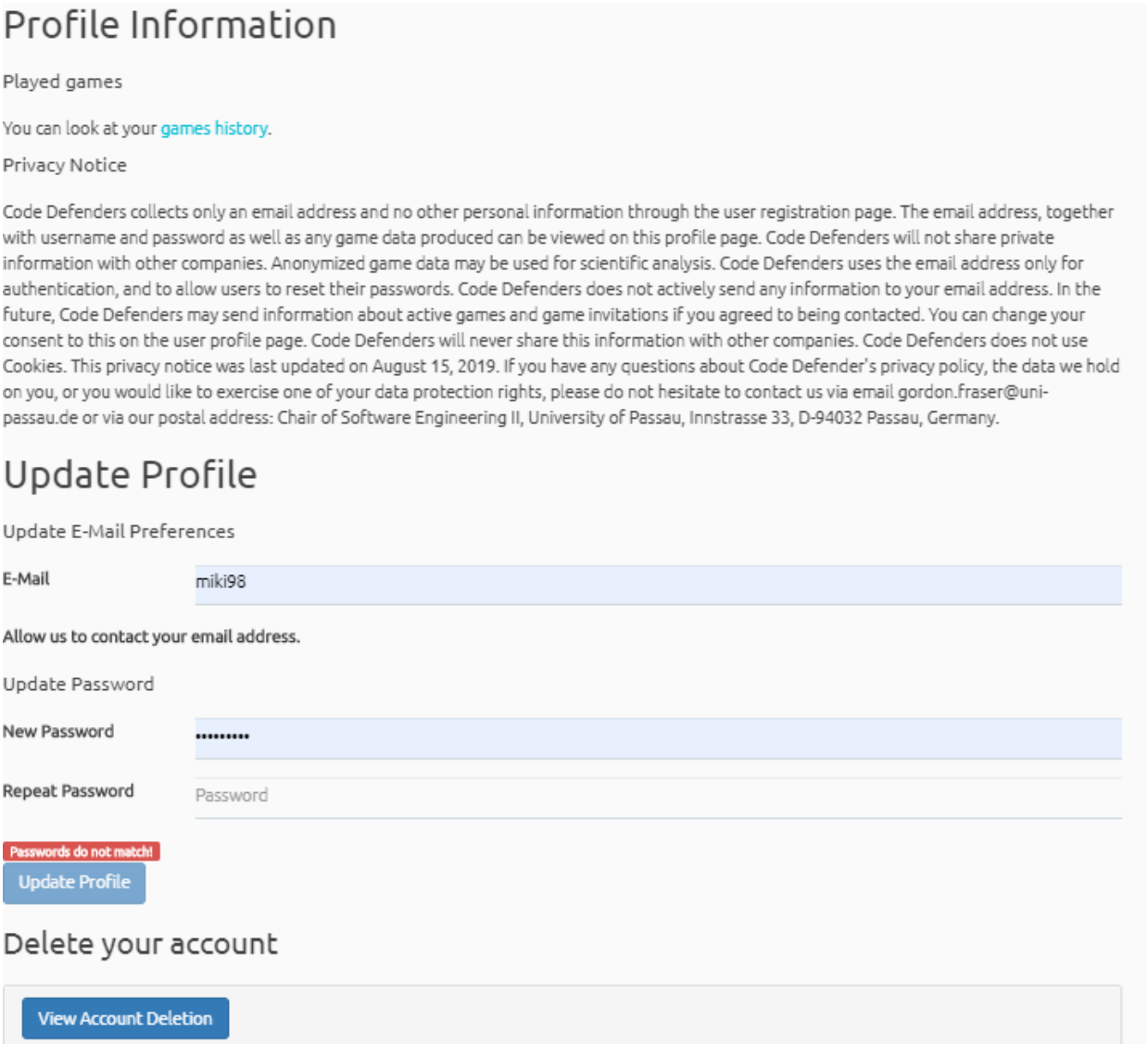


Figura 2.5.6: Schermata Profile

In più sarà presente una nota sulla Privacy in cui si espone che l'applicazione non condividerà le informazioni con altre compagnie, l'indirizzo email viene utilizzato solo per consentire il reset della password e i dati utilizzati ai fini scientifici di ricerca verranno utilizzati in maniera del tutto anonima.

Come ultima opzione vi è la cancellazione dell'account.

Premendo invece su Help, si aprirà una pagina in cui si riepiloga a grandi linee il funzionamento del gioco.

Il Logout in definitiva consentirà di disconnettersi.

6. Creazione Battleground

Creando una partita si necessita il settaggio di diverse impostazioni e dato che sono identiche per le due modalità disponibili, si prenderà in considerazione la creazione di un Battleground.

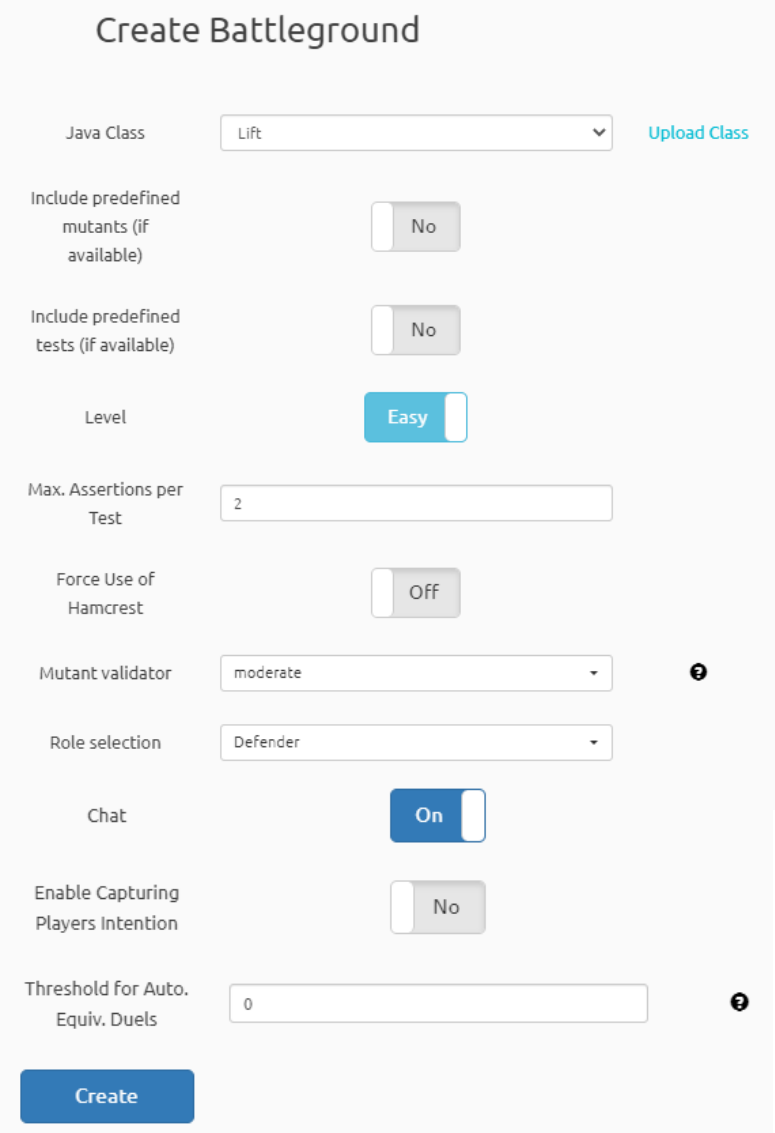


Figura 2.6.1: Impostazioni partita

Come si nota nella figura 2.6.1, si sceglie come prima cosa la classe su cui si vuole giocare. L'applicazione mette a disposizione delle classi caricate di default o la possibilità di aggiungerne una nuova. Nel caso specifico, se il gioco è stato scaricato e installato in locale, esso sarà sprovvisto di queste classi base, dunque si dovrà aggiungerne una dal proprio pc per poter proseguire.

In particolare quando si aggiunge una classe abbiamo a disposizione ulteriori personalizzazioni.

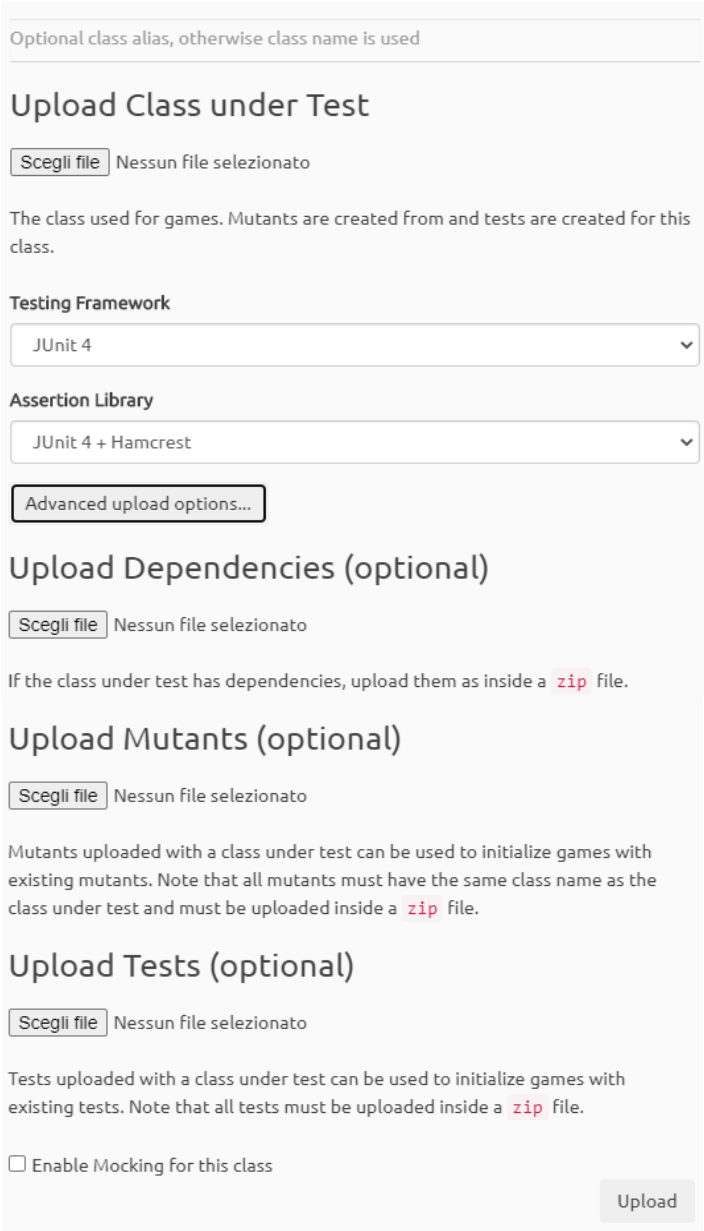


Figura 2.6.2: Impostazioni per inserire una nuova classe JAVA

Si può scegliere il tipo di testing framework che si vuole utilizzare. Per il momento è disponibile solo il framework JUnit 4.

Subito sotto si può scegliere la libreria delle asserzioni, in particolare ve ne sono di più tipi: JUnit 4, 5 o Hamcrest. Oppure si possono includere le JUnit insieme ad Hamcrest.

A seguire si trovano una serie di impostazioni più avanzate che sono del tutto opzionali. Per esempio abbiamo la possibilità di caricare le possibili dipendenze della classe, oltre a dei casi di test o mutanti definiti dall'utente.

Questi risultano utili quando si vuole iniziare la partita con dei mutanti o dei casi di test già presenti.

Alla fine della pagina delle impostazioni si trova un elenco di tutte le classi già caricate sul gioco. Si ha la possibilità di visualizzarle e di copiarle se si ha necessità e inoltre vengono mostrate anche una serie di parametri caratteristici della classe come la presenza di test, dipendenze o mutanti predefiniti e il tipo di libreria utilizzate per i test.
Tornando alla pagina delle impostazioni della partita vi sono dunque due pulsanti che vanno ad abilitare i mutanti e/o i test predefiniti.
Successivamente si ha un ulteriore bottone dedicato alla selezione della difficoltà.
Ve ne sono due: facile o difficile. La differenza sta nel fatto che i giocatori appartenenti alle due squadre vedono cose diverse in base alla difficoltà. Prendendo come riferimento la difficoltà difficile gli attaccanti non vedono qual è il codice dei test che uccidono i mutanti. Possono fare affidamento solo sulla copertura delle istruzioni, mostrata evidenziando in verde le linee della classe che sono state testate. Mentre i difensori non possono vedere il codice dei mutanti uccisi. Come riferimento hanno solo un'indicazione su dove sia presente il bug attraverso un'opportuna icona.
Di contro nella difficoltà facile tutte queste informazioni sono mostrate a tutti gli utenti.
In più c'è da notare che l'icona dei bug può assumere colori diversi in base allo stato del mutante.
Per esempio se il mutante è di colore rosso vuol dire che è ancora vivo e deve essere ucciso.
In questo caso è anche presente un numerino che indica il numero di mutanti che riguardano la stessa linea di codice.
Se il mutante è ucciso, invece, assumerà il colore grigio. Mentre se è supposto equivalente prenderà il colore viola e se esso verrà ucciso nel duello allora diventerà grigio con le ali bianche.
Tutti i colori sono mostrati nell'immagine successiva.

Figura 2.6.3: Colori dei mutanti

Di seguito si può impostare il numero massimo di asserzioni che ogni test può produrre. In genere due si utilizza come valore di default. Ciò vuol dire che in un unico caso di test non si potranno mettere più di sue asserzioni per verificare che sia andato tutto a buon fine.
In merito a questa limitazione, ve ne sono altre riguardanti i mutanti nell'impostazione immediatamente successiva. Si tratta del Mutant Validator e può essere di tre tipi: relaxed, moderate e strict.

La differenza sta nel fatto che gli attaccanti non possono utilizzare determinate istruzioni o costrutti logici per creare i propri mutanti.
Per esempio il tipo relaxed non consente di fare delle call alle librerie System e Random.
La prima può essere utilizzata per visualizzare messaggi o il contenuto di qualche variabile usando il comando system.out.println, mentre la seconda genera valori random.
Nel caso di Moderate i mutanti devono essere privi di commenti, ulteriori operatori logici (per esempio && e ||), operazioni ternarie (simili a if-else) e nuove strutture di controllo (per esempio switch, if, for).
Nell'ultimo caso, Strict, i mutanti saranno privi di Reflection, operazioni sui bit (per esempio shift o and bit a bit) e infine cambiamenti di firma.
Nel caso particolare delle Reflection si tratta di una infrastruttura che permette di ispezionare un oggetto a runtime, cioè di capire quel particolare oggetto a quale classe appartiene, la composizione in termini di attributi, metodi e così via.
Invece il cambiamento della firma di un metodo della classe può riguardare la modifica del numero di parametri, l'ordine con il quale sono messi o la rimozione di parametri non utilizzati.
Le ultime due impostazioni, il numero di asserzioni e il Mutant Validator, riguardano delle limitazioni imposte sull'editor di testo usato per la generazione di mutanti e dei casi di test.
Queste restrizioni esistono per tenere sotto controllo lo sviluppo del codice in modo da evitare la creazione di mutanti troppo difficili da individuare.
Per esempio un attaccante potrebbe inserire un if in cui va a confrontare una certa variabile rispetto a un numero arbitrario che gli attaccanti non potrebbero conoscere, inserendo così un mutante che non può essere individuato senza la conoscenza del codice prodotto. Mentre per quanto riguarda i difensori, hanno un numero massimo di asserzioni. Ciò spinge gli utenti a generare casi di test molto simili a quelli che produrrebbero nella realtà. Infatti avendo un numero di asserzioni illimitato essi potrebbero generare dei casi di test che coprano la maggior parte del codice, quindi molto lunghi e poco pratici da leggere.
A seguire si può scegliere il ruolo che si avrà nella partita: Attaccante, Difensore o Osservatore.
Nell'ultimo caso si potrà assistere alla partita, quindi vedere tutti i casi di test e i mutanti generati dai partecipanti.
Si può inoltre abilitare la chat tra i vari giocatori. Nel caso specifico la chat riguarda sia la squadra che tutti gli utenti partecipanti alla partita. Può essere trovata in basso a destra

contrassegnata con un numero nell'angolo della pagina. Il numero indica i messaggi ancora non letti.

Se premuto si aprono due finestre in cui nella prima si possono mandare messaggi a tutti i membri della partita e nella seconda solo alla propria squadra. Siccome è ancora in fase di sviluppo la chat non funziona in maniera ottimale, se si prova a mandare un messaggio otterremo come echo "Undefined".

La successiva impostazione si chiama Capturing Players Intention. In questo caso si possono specificare delle informazioni aggiuntive prima della creazione di un caso di test o di un mutante.

Nel primo caso l'utente deve specificare la linea di codice della CUT che vuole testare, mentre nel secondo deve notificare se il mutante inserito, secondo lui, può essere ucciso, è di tipo Equivalent oppure non sa se può essere ucciso.

In definitiva l'ultima impostazione è detta Threshold for Auto. Equiv. Duels. Si tratta di un indicatore che permette di attivare in automatico i duelli per un mutante supposto Equivalent.

Cioè si controllano il numero di test che coprono quel mutante, ma non lo uccidono. Se tale numero raggiunge quello specificato in questa impostazione allora il mutante è dichiarato in automatico Equivalent facendo partire il duello.

Capitolo 3: Simulazione di una partita

Si prenda in considerazione una partita in modalità Battleground con due partecipanti, Kenobi nella squadra dei difensori e Vader in quella degli attaccanti.

La simulazione è fatta utilizzando il gioco installato in locale e le impostazioni relative alla partita sono mostrate in figura.

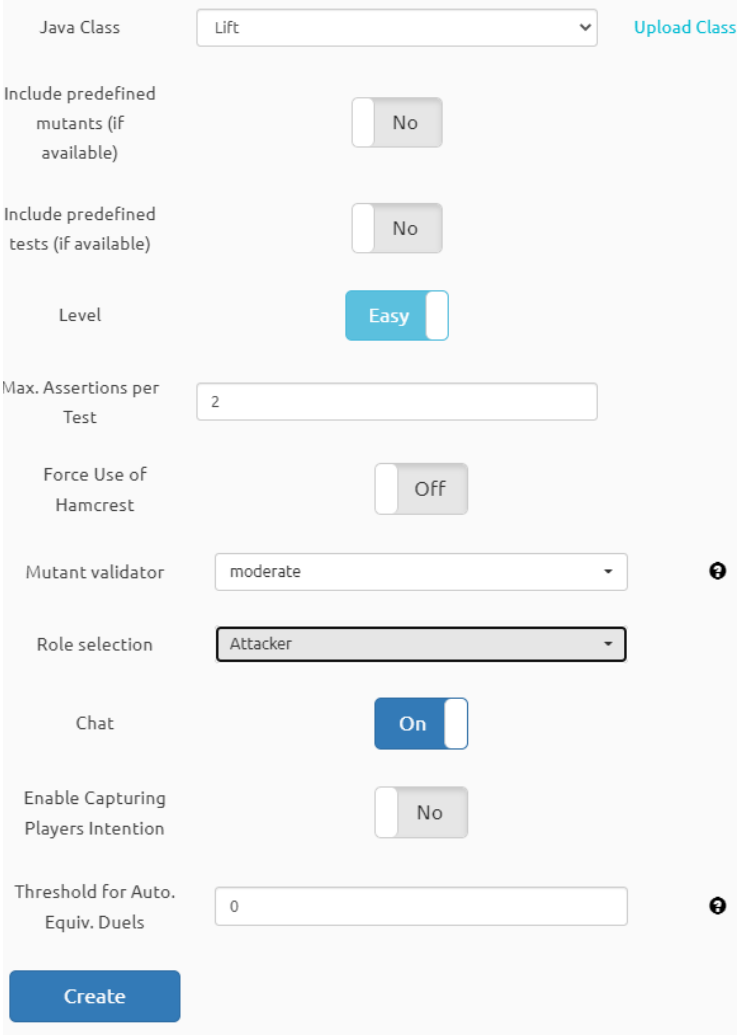


Figura 3.1: Impostazioni partita simulazione

La CUT presa in esame, Lift, è una di default scaricata dalla piattaforma Code Defenders e rappresenta il funzionamento di un ascensore.

Come detto il primo turno è sempre degli Attaccanti. Si mostra dunque la schermata messa a loro disposizione.

Figura 3.2: Schermata Attaccante

Si possono individuare tre finestre. In quelle di sinistra l'attaccante ha a disposizione l'elenco di tutti i metodi della classe in cui verranno segnalati tutti i mutanti e i test che sono stati creati. Selezionando un metodo e cliccando sul relativo test o mutante si può visualizzare il codice. Nel caso specifico dei mutanti si può visualizzare il codice mutato in verde e quello corretto in rosso.

E' da notare che tutte queste informazioni sono visibili solo perché la modalità della partita è impostata su facile. In caso contrario gli attaccanti possono vedere il codice solo dei mutanti generati. Lo stesso vale per i difensori con i casi di test.

La finestra di destra invece mette a disposizione dell'attaccante il codice originale della classe. Potrà modificarlo a piacimento e cliccando poi sul pulsante Attack! potrà immettere il mutante nella CUT.

Supponiamo adesso che Vader vada a modificare il metodo getTopFloor() aggiungendo nel return “+1” e cliccando subito dopo il pulsante di attacco.

```
1  public class Lift {
2
3      private int topFloor;
4      private int currentFloor = 0; // default
5      private int capacity = 10;    // default
6      private int numRiders = 0;    // default
7
8      public Lift(int highestFloor) {
9          topFloor = highestFloor;
10     }
11
12     public Lift(int highestFloor, int maxRiders) {
13         this(highestFloor);
14         capacity = maxRiders;
15     }
16
17     public int getTopFloor() {
18         return topFloor+1;
19     }
```

Figura 3.3: Mutante inserito

In generale se tutto va a buon fine, cioè se i mutanti rispettano tutti i vincoli definiti in precedenza, allora l’applicativo riporterà in alto un messaggio di successo. Siccome il mutante appena inserito è perfettamente conforme alle limitazioni previste otterremo tale messaggio:

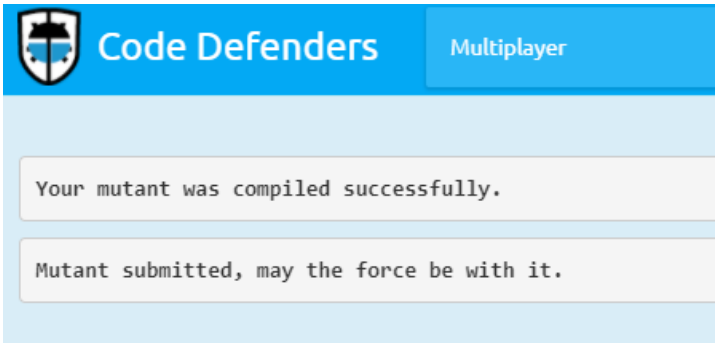


Figura 3.4: Messaggio di successo

“Che la forza sia con lui” è il miglior augurio che un attaccante possa fare al proprio mutante siccome più test riuscirà passare più punti riceverà.
A questo punto il turno passerà ai difensori.

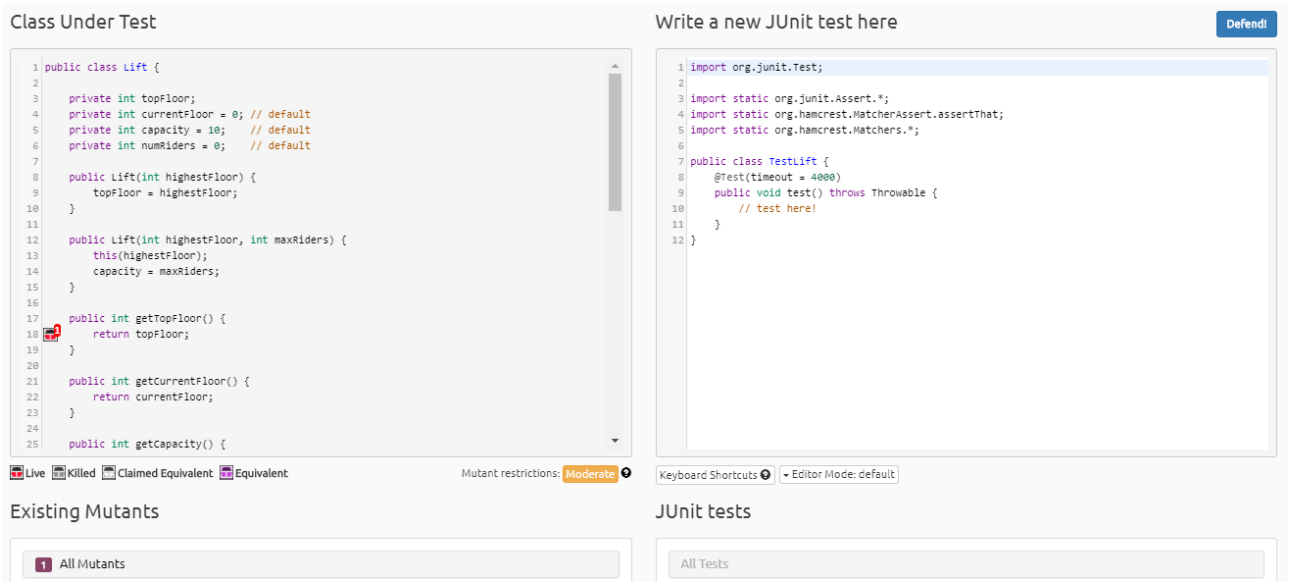


Figura 3.5: Schermata Difensore

Si nota che nonostante il turno sia dei difensori, gli attaccanti possono ugualmente inserire nuovi mutanti, senza restare in attesa del turno successivo.

I difensori avranno a disposizione quattro finestre. Quelle sottostanti hanno lo stesso funzionamento di quelle viste con gli attaccanti.

Quella in alto a sinistra, invece, mostra il codice originale della CUT su cui viene riportata la presenza dei mutanti attraverso il simbolo del bug.

Si può notare che entrambi gli utenti possono vedere quali linee di codice della classe sono state eseguite dai casi di test attraverso una marcatura di colore verde. Essa diventerà sempre più scura man mano che aumentano i casi di test che lavorano su quelle linee.

Invece quella sulla destra è una finestra di editor dove si possono scrivere i casi di test. Essa sarà inizializzata con l'inclusione delle librerie necessarie. La scrittura del codice va fatta dove indicato dal relativo commento.

Ora supponiamo che Kenobi scriva un caso di test tale da uccidere il mutante.

Write a new JUnit test here

Defend!

```
1  import org.junit.Test;
2
3  import static org.junit.Assert.*;
4  import static org.hamcrest.MatcherAssert.assertThat;
5  import static org.hamcrest.Matchers.*;
6
7  public class TestLift {
8      @Test(timeout = 4000)
9      public void test() throws Throwable {
10         Lift prova = new Lift(5);
11         int x = prova.getTopFloor();
12         assertEquals(x,5);
13     }
14 }
```

Figura 3.6: Caso di test per il metodo getTopFloor()

Nel test si è creata una nuova istanza di un oggetto appartenente alla classe Lift. Si è chiamato il costruttore che va a inizializzare il valore della variabile privata topFloor, settando gli altri a quelli di default. Subito dopo per testare il metodo getTopFloor() possiamo dichiarare una variabile intera in cui si inserisce il valore restituito dal metodo.
Infine con un assertEquals confronto il valore di x con il valore che mi aspetto mi ritorni la classe. Nel nostro caso si è chiamato il costruttore con il parametro 5, ciò vuol dire che topFloor deve essere stato inizializzato a quel valore. Dunque il valore di ritorno del metodo get, che restituisce il contenuto della variabile privata, deve essere pari a 5. Dunque si effettua il relativo controllo.
Siccome il mutante aggiunge un "+1" il valore di ritorno sarà diverso da quello che ci si aspetta. Dunque si sarà scovato il mutante e ucciso.
Una volta cliccato sul pulsante Defend! il test verrà eseguito sulla classe e riporterà in alto un messaggio di risposta. In generale se va tutto a buon fine, come in questo caso, avremo un messaggio di successo con sotto riportato se è stato ucciso un mutante oppure no. Invece nel caso in cui ci fossero problemi nel codice avremmo un report con gli errori riscontrati in fase di compilazione.

```
Great! Your test compiled and passed on the original class under test.

Great, your test killed the last mutant!
```

Figura 3.7: Messaggio di successo

```
Your test did not compile. Try again, but with compilable code.

[javac] TestLift.java:11: error: cannot find symbol
[javac]         int x = prova.getTopfloor();
[javac]                         ^
[javac]   symbol:   method getTopfloor()
[javac]   location: variable prova of type Lift
[javac] 1 error
```

Figura 3.8: Messaggio con errore di sintassi

```
7  public class TestLift {
8      @Test(timeout = 4000)
9      public void test() throws Throwable {
10         Lift prova = new Lift(5);
11         int x = prova.getTopfloor();
12         assertEquals(x,5);
13     }
14 }
```

Figura 3.9: Errore di sintassi nel codice, il metodo getTopFloor()

```
Great! Your test compiled and passed on the original class under test.

Your test has not killed any mutants, just yet.
```

Figura 3.10: Messaggio di successo senza uccisione

```
7  public class TestLift {
8      @Test(timeout = 4000)
9      public void test() throws Throwable {
10         Lift prova = new Lift(5);
11         int x = prova.getTopFloor();
12         int y = prova.getCapacity();
13         assertEquals(y,10);
14     }
```

Figura 3.11: Caso di test che copre la linea ma non uccide il mutante

Si nota come nel secondo caso, il test presenti un errore di sintassi siccome è stato chiamato il metodo getTopFloor() con la 'f' minuscola.

Mentre nell'ultimo caso il test è stato compilato con successo, ma non ha ucciso nessun mutante siccome la condizione nell'assert risultava soddisfatta.

Si supponga adesso che l'attaccante inserisca un mutante che non può essere scovato da nessun caso di test, dunque Equivalent. Per esempio possiamo supporre che Vader inserisca il seguente mutante.

```
39     public void addRiders(int numEntering) {
40         if (numRiders + numEntering <= capacity) {
41             numRiders = numRiders + numEntering;
42         } else {
43             numRiders = capacity;
44         }
45     }
```

```
39     public void addRiders(int numEntering) {
40         if (numRiders + numEntering > capacity) {
41             numRiders = capacity;
42         } else {
43             numRiders = numRiders + numEntering;
44         }
45     }
```

Figura 3.12: Confronto tra il codice originale e il mutante Equivalent

Come si vede questo è molto particolare perché il codice prodotto ha lo stesso funzionamento del codice originale. Dal punto di vista logico se il numero di persone nell'ascensore più il numero di nuovi passeggeri supera la capacità, allora non potrà salire

che un numero di persone pari a Capacity. In caso contrario saliranno la somma tra i passeggeri attuali e quelli nuovi.
D'altro canto, se il numero di persone presenti più il numero di nuovi passeggeri è minore o uguale alla capacità, allora salirà un numero di persone pari alla somma. Altrimenti avremo in totale un numero di passeggeri ancora una volta pari alla Capacity.
Dunque il mutate consiste nell'invertire la condizione dell'if e le relative istruzioni presenti nei due casi, ottenendo così un codice che ha lo stesso comportamento di quello originale.
E' chiaro che nessun caso di test riuscirà a scovare il mutante, per questo il difensore potrà dichiararlo Equivalent, andando sull'icona del bug del mutante e premendo sul pulsante Claim Equivalent.
Inizierà dunque il duello. L'attaccante avrà a questo punto due possibilità: creare un caso di test che smentisca l'affermazione oppure accettarla.

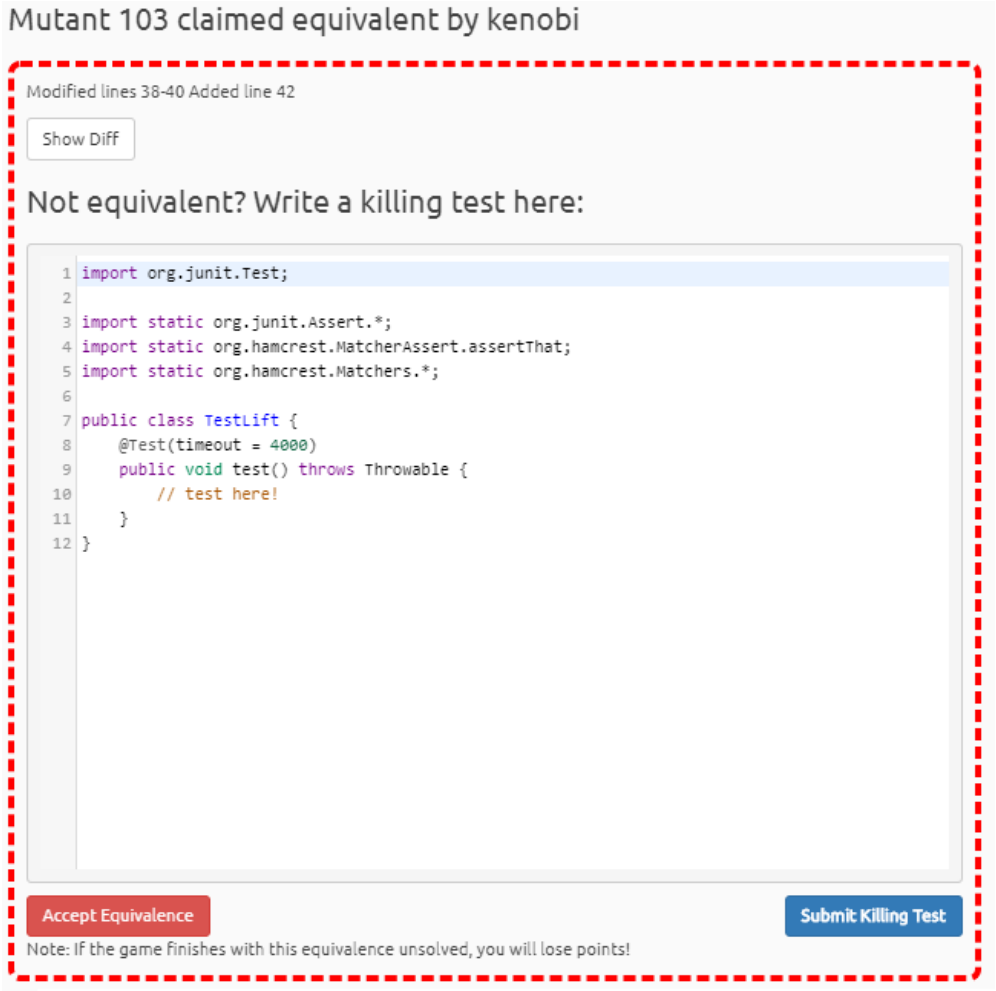


Figura 3.13: Schermata del duello sottoposto all'Attacker

In base a come si svolge il duello vi sarà un'assegnazione diversa dei punti. Se il mutante è accettato come Equivalent allora è ucciso, facendo perdere a Vader tutti i punti associati a

quel mutante e Kenobi guadagnerà un punto. In caso contrario l'attaccante ucciderà il mutante e guadagnerà un altro punto.
Nel caso specifico Vader non sarà in grado di produrre un caso di test che smentisca l'affermazione, di conseguenza accetterà l'equivalenza.
Il gioco poi evolve con il susseguirsi di attacchi e difese da parte degli utenti finché il creatore della partita, Vader, decide di concluderla premendo sul pulsante End Game.

Conclusioni

Da un punto di vista didattico questa piattaforma risulta essere perfetta per quanto riguarda l'apprendimento dello sviluppo di casi di test sempre più robusti, insieme alla creazione di mutanti sempre più fittizi. Tutto ciò consente di far capire l'importanza dell'attività di test durante il ciclo di vita di un software. Tuttavia la piattaforma, essendo un prototipo, presenta delle limitazioni. Per esempio richiede delle conoscenze di base abbastanza forti sul linguaggio JAVA, sui test e le diverse tipologie. Infatti si è notato durante le diverse prove della piattaforma che gli utenti non aventi molta dimestichezza col linguaggio avevano non poche difficoltà. Per questo motivo si potrebbe creare in futuro una pagina dedicata completamente all'introduzione degli utenti al linguaggio di programmazione, un vero e proprio tutorial spiegato passo dopo passo.

Ciò di fatto non dovrebbe essere limitato al linguaggio JAVA. Infatti come ulteriore sviluppo si potrebbe adattare la piattaforma a più linguaggi di programmazione, incorporando per esempio il C/C++ molto diffuso, soprattutto tra gli studenti.

In definitiva come ultimo sviluppo si suggerisce l'implementazione di un bot così da poter sostenere delle partite single player. Difatti il gioco al momento, consente di creare delle partite solo tra più persone, escludendo la possibilità di fare pratica da soli. Si potrebbero aggiungere diversi tipi di difficoltà, in base alla quale un utente automatizzato inserisca dei mutanti via via più difficili da scovare e infine si potrebbe settare la possibilità dell'inserimento di mutanti Equivalent.

In conclusione nell'elaborato si è affrontato la problematica dello sviluppo di casi di test nel modo tradizionale. Siccome ripetitiva e tediosa, si è raggiunto una soluzione che segue un approccio diverso.

Si è introdotta la Gamefication perfetta a tal fine, rendendo l'intero processo di sviluppo di casi di test competitiva e divertente. Comportando così l'implementazione di test suite semplici e accurate.

Dunque si è vista l'installazione e il funzionamento della piattaforma in tutti i suoi aspetti e casi d'uso.

Nonostante sia ancora in fase di sviluppo, risulta avere un livello molto forte di personalizzazione con la possibilità di testare una classe caricata dagli utenti.

Il funzionamento complessivo è molto intuitivo, insieme al sistema di punteggio accurato e bilanciato.

In definitiva si è provata la piattaforma in locale tra due utenti riportando una simulazione che affronta tutte le casistiche.

Tale piattaforma verrà in seguito utilizzata nell'ambito universitario in un corso di testing software per introdurre gli studenti a questo approccio alternativo. A questo punto non resta che installare o andare sul sito ufficiale e iniziare a fare pratica e come il gioco suggerisce: che la forza sia con voi!

Bibliografia

[1] Gordon Fraser, Alessio Gambi, Marvin Kreis, *Gamifying a Software Testing Course with Code Defenders.*

[2] Benjamin S. Clegg, Jose Miguel Rojas, Gordon Fraser, *Teaching Software Testing Concepts Using a Mutation Testing Game.*

[3] Jose Miguel Rojas, Thomas D. White, Benjamin S. Clegg, Gordon Fraser, *Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game*.

[4] Gordon Fraser, Alessio Gambi, José Miguel Rojas, *A Preliminary Report on Gamifying a Software Testing Course with the Code Defenders Testing Game*.

[5] Jose Miguel Rojas, Gordon Fraser, *CODE DEFENDERS: A Mutation Testing Game.*

[6] *CODE DEFENDERS, https://code-defenders.org/.*