

# Dino Run and Jump

1	Introduzione .....	4
1.1	Informazioni sul progetto.....	4
1.2	Abstract.....	4
1.3	Scopo.....	4
2	Analisi.....	5
2.1	Analisi del dominio .....	5
2.2	Analisi e specifica dei requisiti .....	5
2.3	Use case .....	8
2.4	Pianificazione .....	9
2.5	Analisi dei mezzi .....	10
3	Progettazione .....	11
3.1	Design dell'architettura del sistema .....	11
3.2	Design dei dati e database .....	11
3.3	Design delle interfacce.....	12
3.3.1	Design interfacce telefono: .....	12
3.3.2	Design interfacce computer .....	14
3.4	Design procedurale .....	16
4	Implementazione .....	18
4.1	Refactoring.....	18
4.2	Struttura cartelle del progetto.....	19
4.3	docs/css/bootstrap.css.....	19
4.4	docs/css/game.css.....	19
4.5	docs/js/Bootstrap.bundle.min.js .....	20
4.6	docs/js/game.js .....	20
4.6.1	blockInput .....	20
4.6.2	handleMotion .....	20
4.6.3	requestPermission .....	20
4.7	docs/js/index.js.....	20
4.7.1	writeMedals.....	20
4.7.2	connectToGame .....	20
4.7.3	generateGuestId .....	20
4.7.4	jump .....	21
4.7.5	registerNewUser .....	21
4.7.6	loginUser.....	21
4.7.7	logoutUser .....	21
4.7.8	openUserInformation .....	21
4.7.9	generateSession.....	21
4.7.10	changeDinoColor .....	21
4.7.11	saveDinoColor .....	21
4.7.12	showUserInformation.....	22
4.7.13	firebase.auth().onAuthStateChanged((user) => .....	22
4.7.14	checkLoggedUser.....	22
4.7.15	getIsTouchingDown.....	22
4.7.16	Assenza di watchGame .....	22
4.8	docs/Game/js/game.js .....	23
4.8.1	Aggiunta dinamica dell'utente.....	23
4.8.2	setSettingsPhaser.....	24
4.8.3	La lobby .....	24
4.8.4	La Game scene .....	28
4.8.5	La leaderboard.....	29
4.9	docs/Game/js/medaglie.js.....	30
4.10	docs/Game/js/phaser.js e phaser-arcade-physics.js.....	30
4.11	docs/Game/index.html.....	30
4.12	docs/GUI/login.html .....	31
4.13	docs/GUI/collegamentoPartita.html .....	31
4.14	docs/GUI/game.html.....	32
4.15	docs/GUI/paginaUtente.html .....	32
4.16	docs/GUI/personalizzaDino.html .....	33

4.17	docs/GUI/bacheca.html .....	33
4.18	La configurazione di Firebase .....	34
5	Test .....	35
5.1	Protocollo di test .....	35
5.2	Risultati test .....	42
5.3	Mancanze/limitazioni conosciute .....	42
6	Consuntivo .....	43
7	Conclusioni.....	44
7.1	Sviluppi futuri .....	44
7.2	Considerazioni personali.....	45
7.2.1	Michea .....	45
7.2.2	Nadia .....	45
7.2.3	Thomas.....	45
8	Sitografia .....	46
9	Allegati .....	46

## Indice delle figure

Figura 1	Use case .....	8
Figura 2	Gantt preventivo.....	9
Figura 3	Architettura di sistema.....	11
Figura 4	Progettazione DB.....	11
Figura 5	Home page con pop-up di login .....	12
Figura 6	Home page con utente loggato .....	12
Figura 7	Pagina per collegamento .....	12
Figura 8	Pagina utente .....	12
Figura 9	Bacheca medaglie.....	13
Figura 10	Pagina personalizzazione utente .....	13
Figura 11	Pagina di gioco.....	13
Figura 12	Home page.....	14
Figura 13	Creazione di una partita .....	14
Figura 14	Collegamento ad una partita .....	15
Figura 15	Classifica.....	15
Figura 16	Diagramma di flusso .....	16
Figura 17	Esempio leaderboard .....	30
Figura 18	Esempio di partita .....	31
Figura 19	Home page con un utente loggato.....	31
Figura 20	Pagina di collegamento ad una partita.....	32
Figura 21	Pagina di gioco.....	32
Figura 22	Pagina utente .....	32
Figura 23	Pagina personalizzazione dinosauro .....	33
Figura 24	Bacheca .....	33
Figura 25	Gantt consuntivo .....	43

## 1 Introduzione

### 1.1 Informazioni sul progetto

- Allievi coinvolti nel progetto: Michea Colautti, Nadia Fasani, Thomas Sartini.
- Classe: I3AA/BB/BC Scuola Arti e Mestieri Trevano.
- Docenti responsabili: Geo Petrini
- Data inizio: 27 gennaio 2022.
- Data di fine: 05 maggio 2022.
- Linguaggio: JavaScript

### 1.2 Abstract

*We all know the famous Chrome Dino, that little black dinosaur that jumps over a lot of cactuses endlessly and tells us that we are not connected to the internet. We've all hated him and loved him at some point in our lives. With this project the Chrome Dino is taken to another level: starting from a previously created project, we have improved the user experience and added a new interesting game mechanic. With this project, which is even multiplayer, the players jump over cactuses, this time not by pushing a button, but by their own real movements. Using the phone's sensors the player's dinosaur will jump over obstacles. New features have also been added, such as the possibility to customize the dinosaur and earn rewards. But the project is also available to those who cannot, for one reason or another, jump. For this reason, the phone can also be used: by pressing a button they control their own dinosaur.*

### 1.3 Scopo

Lo scopo del progetto è quello di creare una versione multiplayer del famoso *Chrome Dino* dove diversi utenti si possono connettere ad una partita e possono giocare tutti insieme. Il numero di giocatori è quindi variabile, da un minimo di uno ad un massimo di dieci. Ci sarà dunque un utente “host” che si occupa di creare la partita e di mostrarla agli altri utenti, idealmente su uno schermo sufficientemente grande. Man mano che si aggiungono giocatori i loro dinosauri appariranno sullo schermo principale.

L'idea del progetto è il fatto che i giocatori fanno saltare il proprio dinosauro muovendosi essi stessi con un salto. Grazie ai sensori di movimento del telefono e alla struttura a sessioni del programma, i salti vengono trasmessi al server, che comunica alla pagina l'evento. Tuttavia, per quelle persone che non possiedono un telefono con i giusti sensori, oppure sono impossibilitate nel saltare, bisogna introdurre una modalità di gioco basilare, che permette di far saltare il dinosauro cliccando su un tasto.

Gli utenti devono poter inoltre creare un account tramite il quale potranno personalizzare l'aspetto del proprio dino e, una volta giocata una o più partite, guadagnarsi delle medaglie che poi saranno visibili in una pagina dedicata. Per coloro che non vogliono effettuare il login, saranno creati degli utenti ospiti che verranno eliminati quando l'utente esce dal sito.

Ci deve essere infine la possibilità di vedere la partita come spettatori, senza interagire con il gioco vero e proprio.

## 2 Analisi

### 2.1 Analisi del dominio

Non dovremo sviluppare questo progetto da zero. Come base avremo infatti il *Chrome Dino* realizzato da Manuel Grosso (vedi sitografia), nel corso del primo semestre dell'anno scolastico 2021/2022. Questo progetto e quello precedente, hanno in comune l'aspetto multiplayer, anche se per la versione sviluppata da Manuel, il numero di giocatori era impostato a 4. Noi dovremmo rendere questo aspetto dinamico, in modo da permettere maggiore flessibilità. Inoltre, per questo progetto la meccanica di salto cambia notevolmente: infatti, se nella precedente versione del gioco il dinosauro saltava in seguito al comando dato premendo su un tasto, ora i giocatori per far saltare il dinosauro dovranno anch'essi saltare veramente con il loro corpo. Attualmente non ci sono progetti simili al nostro. Esso non risolve un problema vero e proprio, ma non per questo è poco rilevante; troviamo infatti molto interessante questa meccanica di gioco, in quanto rappresenta un'evoluzione di un gioco di per sé semplice. Il progetto si rivolge a tutti coloro che hanno voglia di provare qualcosa di nuovo e -avendo introdotto una meccanica di salto alternativa- non esclude coloro che non possono fisicamente saltare o hanno un telefono privo di sensori adeguati.

### 2.2 Analisi e specifica dei requisiti

ID: REQ-01	
<b>Nome</b>	I giocatori devono poter creare una partita con URL o sessione
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	

ID: REQ-02	
<b>Nome</b>	È possibile registrarsi
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	

ID: REQ-03	
<b>Nome</b>	È possibile eseguire il login
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	

ID: REQ-04	
<b>Nome</b>	I giocatori devono potersi unire alla partita
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	

ID: REQ-05	
<b>Nome</b>	Funzioni aggiuntive di login
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>Note</b>	
Sotto Requisiti	
<b>001</b>	Personalizzazione del dinosauro salvata
<b>002</b>	Punteggio salvato nel profilo
<b>003</b>	Bacheca per visualizzare le medaglie

ID: REQ-06	
<b>Nome</b>	La GUI deve essere responsive
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>Note</b>	

ID: REQ-07	
<b>Nome</b>	Possibilità di giocare come ospite
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>Note</b>	

ID: REQ-08	
<b>Nome</b>	Il dinosauro deve essere personalizzabile prima della partita
<b>Priorità</b>	3
<b>Versione</b>	1.0
<b>Note</b>	

ID: REQ-09	
<b>Nome</b>	Il dinosauro deve saltare sfruttando i sensori del telefono
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>Note</b>	

<b>Sotto Requisiti</b>	
<b>001</b>	Alternativa di gioco in caso di handicap o assenza di sensori.

<b>ID: REQ-10</b>	
<b>Nome</b>	Il numero di giocatori deve essere dinamico: da uno a dieci.
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	

<b>ID: REQ-11</b>	
<b>Nome</b>	Il nome dei giocatori deve apparire a schermo
<b>Priorità</b>	3
<b>Versione</b>	1.0
<b>Note</b>	

<b>ID: REQ-12</b>	
<b>Nome</b>	Alla fine del gioco deve essere visualizzata una classifica
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>Note</b>	
<b>Sotto Requisiti</b>	
<b>001</b>	Punteggi in ordine decrementale
<b>002</b>	Visualizzare la/le medaglie direttamente nella classifica

<b>ID: REQ-13</b>	
<b>Nome</b>	Al vincitore viene assegnata una medaglia
<b>Priorità</b>	3
<b>Versione</b>	1.0
<b>Note</b>	
<b>Sotto Requisiti</b>	
<b>001</b>	Medaglia generata con un algoritmo per rendere il più univoche possibili

<b>ID: REQ-14</b>	
<b>Nome</b>	Possibilità di vedere la partita da remoto
<b>Priorità</b>	3

Versione	1.0
Note	

## 2.3 Use case

Ecco lo *use case* da noi definito:

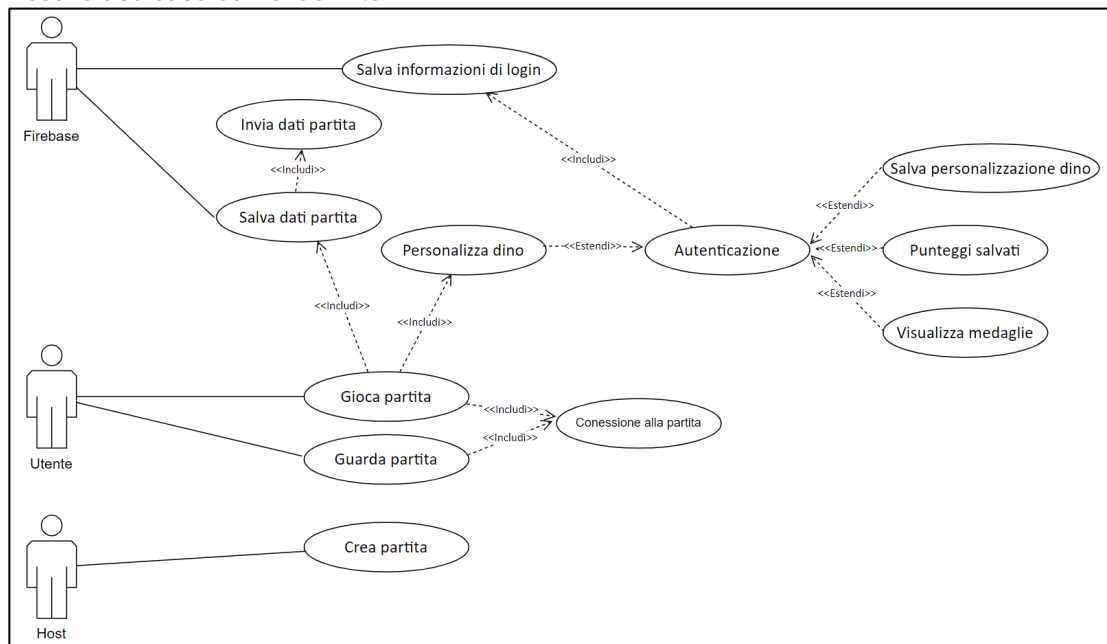


Figura 1 Use case



## 2.4 Pianificazione

Per quanto riguarda la pianificazione alleghiamo il diagramma di Gantt iniziale. Per lo sviluppo del progetto abbiamo deciso di utilizzare un approccio *Waterfall*.

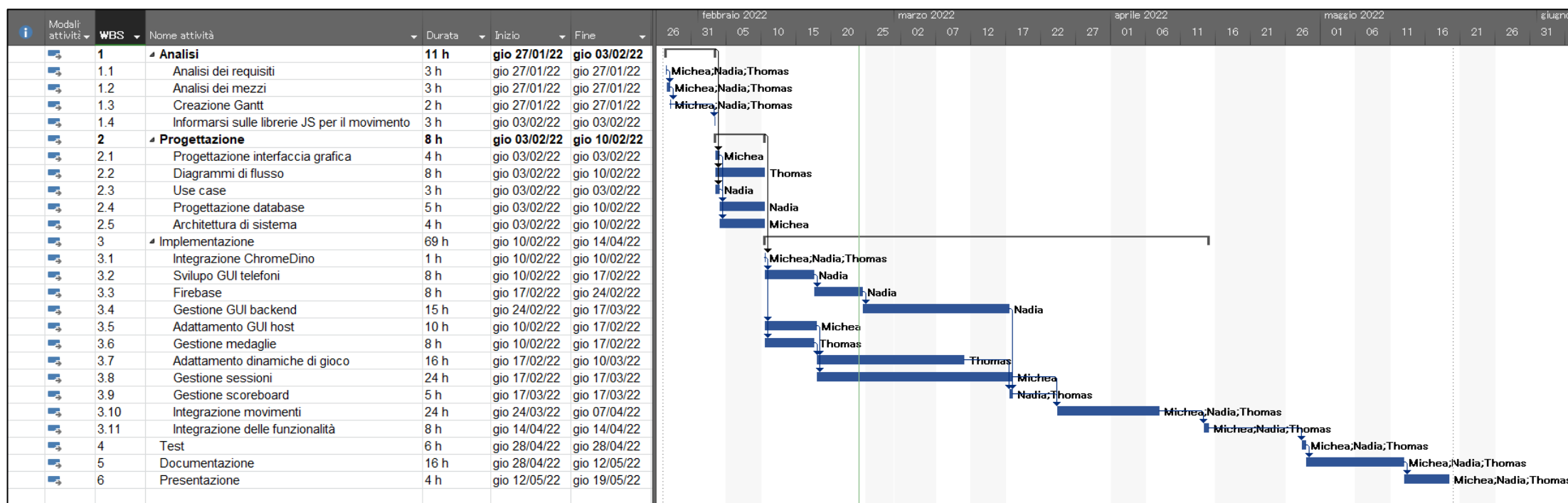


Figura 2 Gantt preventivo

## 2.5 Analisi dei mezzi

### Software

- Firebase 8.2.1
- MockFlow 1.4.7
- Draw.io
- Visual Studio Code 1.65.2
- GIMP 2.10.24
- GitHub
- GitHub Desktop 2.9.12

### Hardware

- Laptop personali
- PC scolastici

Il progetto è scritto in JavaScript, sarà quindi eseguibile da tutti i sistemi operativi. Per quanto riguarda la possibilità di saltare tramite i sensori di movimento, occorre specificare che sui terminali che eseguono IOS sarà possibile usufruire di questa funzione ma, invece di un salto vero e proprio, occorrerà far compiere al telefono un movimento dal basso in alto tramite movimento del polso. Ciò a causa delle diverse impostazioni dei sensori implementate da Apple.

### 3 Progettazione

Essendo consapevoli che la progettazione è una fase importante di ogni progetto, abbiamo voluto dedicare il tempo necessario ad essa, definendo tutti gli aspetti ai quali abbiamo pensato.

#### 3.1 Design dell'architettura del sistema

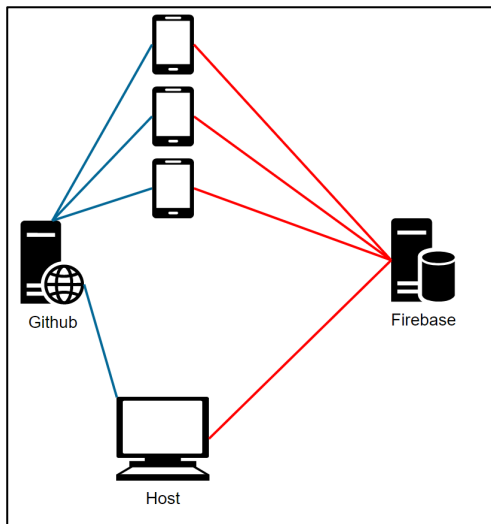


Figura 3 Architettura di sistema

Per l'architettura di sistema abbiamo voluto adottare una struttura abbastanza semplice ma funzionale. Il primo passo della comunicazione, in ordine cronologico, è l'host che crea una partita e la mostra agli utenti. Tutta la parte di gestione del server e delle connessioni, è gestita grazie ai server di GitHub. Infatti questa piattaforma offre un servizio chiamato "GitHub Pages" che svolge la funzione di Web server, togliendo così l'incombenza all'utente.

Contemporaneamente alla creazione della partita, l'host comunica al server Firebase le istruzioni necessarie per il buon funzionamento della stessa. Una volta che la partita è stata creata, gli utenti si collegano alla pagina, collegandosi quindi ai server GitHub, ma instaurano anche una comunicazione con il server Firebase.

#### 3.2 Design dei dati e database

Per il database non abbiamo scelto, come accennato, SQL; ma Firebase. Questo è un software molto utile per la creazione di software e la comunicazione da una piattaforma all'altra. La particolarità di questo software è che si possono creare istanze sul DB direttamente dal codice JS. Ecco il nostro database finale.

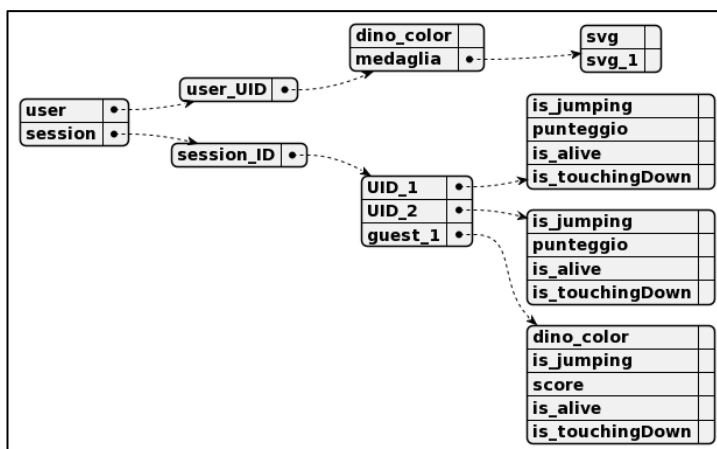


Figura 4 Progettazione DB

I nodi principali sono 2: user e session.

##### User:

Questa istanza contiene l'*uid* degli utenti registrati. Infatti il resto dei dati possono essere recuperati in un altro punto del DB. L'utente contiene però sia il colore del suo dinosauro che una lista di medaglie.

##### Session

Quest'altra istanza contiene invece le partite attive.

Al suo interno ci possono essere fino a 10 utenti, quindi 11 figli contando il *session\_ID*. Tutti gli utenti possiedono gli stessi attributi, utili al buon funzionamento del gioco.

### 3.3 Design delle interfacce

Per la progettazione delle interfacce abbiamo deciso di dividere i *mockups* in 2 famiglie: quelle pensate per il telefono e quelle per il computer. Per le interfacce che dovevano essere visualizzate da entrambi i terminali abbiamo realizzato entrambe le versioni.

#### 3.3.1 Design interfacce telefono:

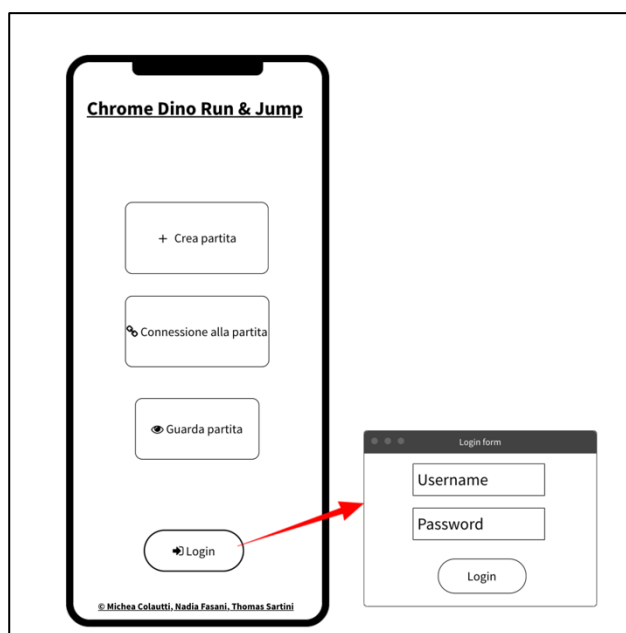


Figura 5 Home page con pop-up di login

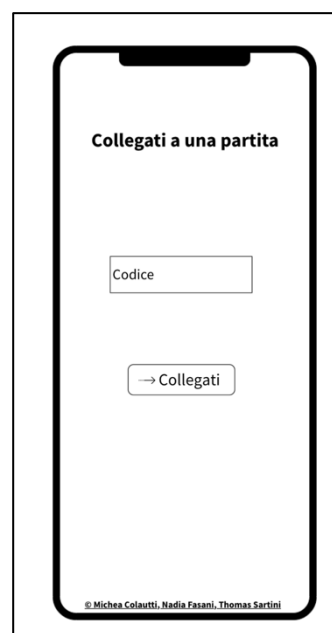


Figura 7 Pagina per collegamento ad una partita



Figura 6 Home page con utente loggato

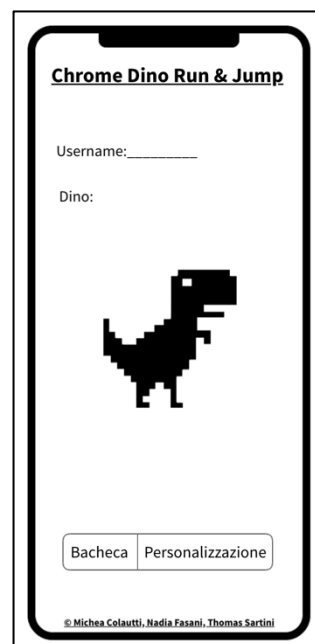


Figura 8 Pagina utente

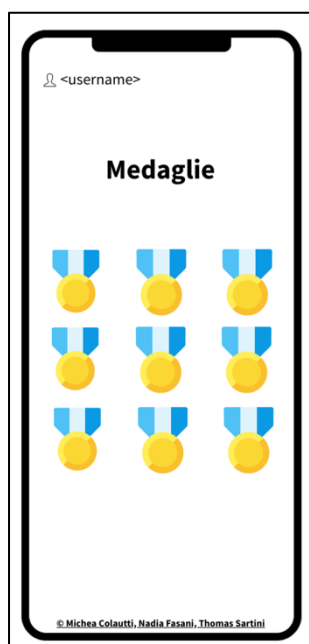


Figura 9 Bacheca medaglie

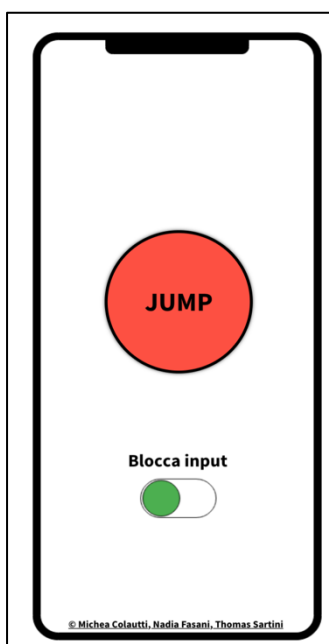


Figura 10 Pagina personalizzazione utente

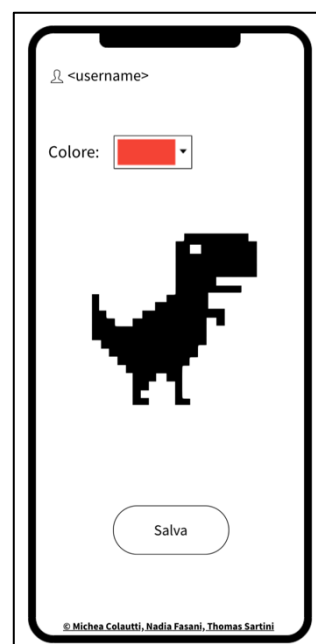


Figura 11 Pagina di gioco

### 3.3.2 Design interfacce computer

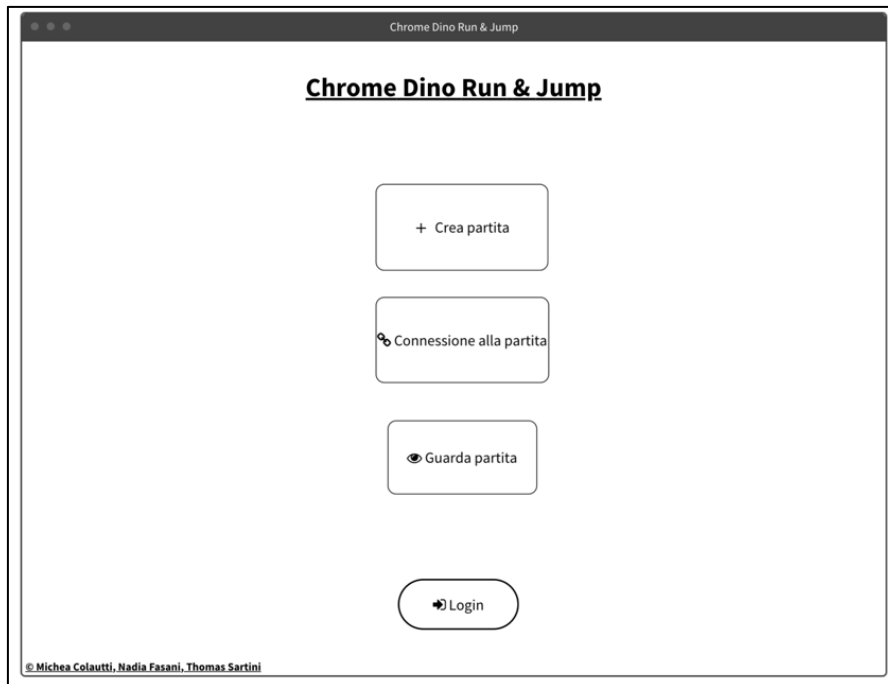


Figura 12 Home page

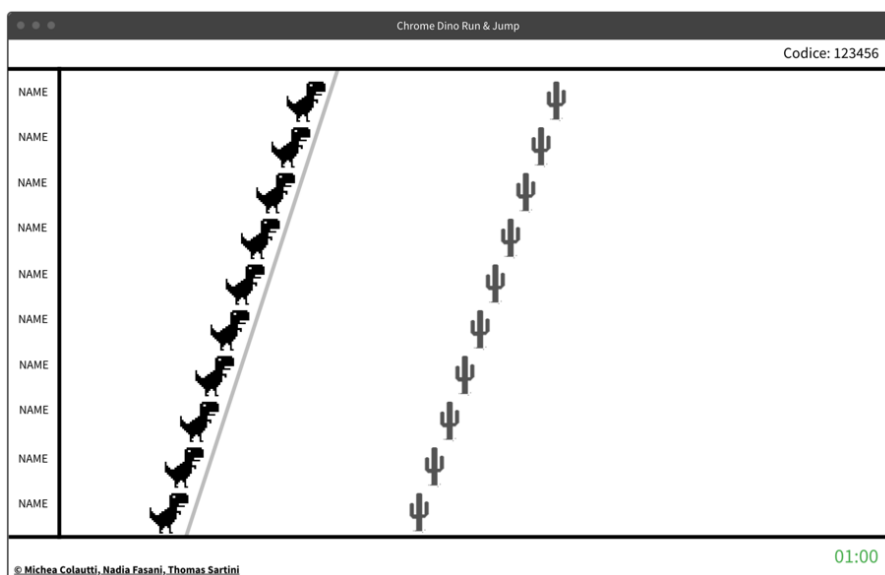


Figura 13 Creazione di una partita

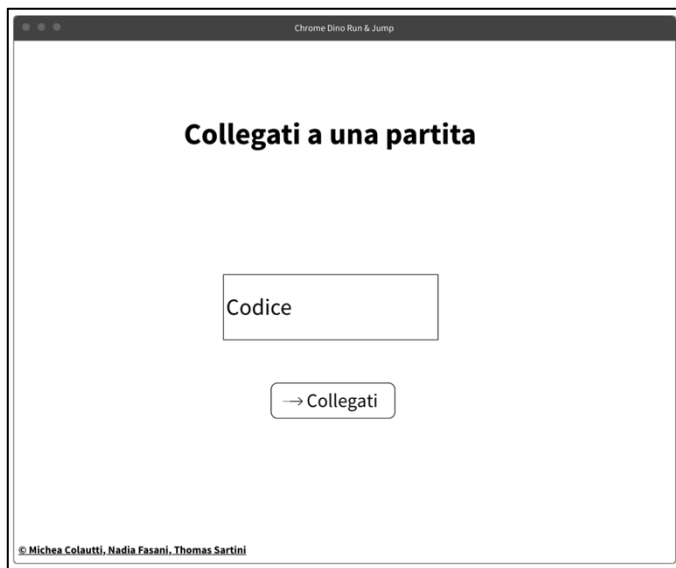


Figura 14 Collegamento ad una partita

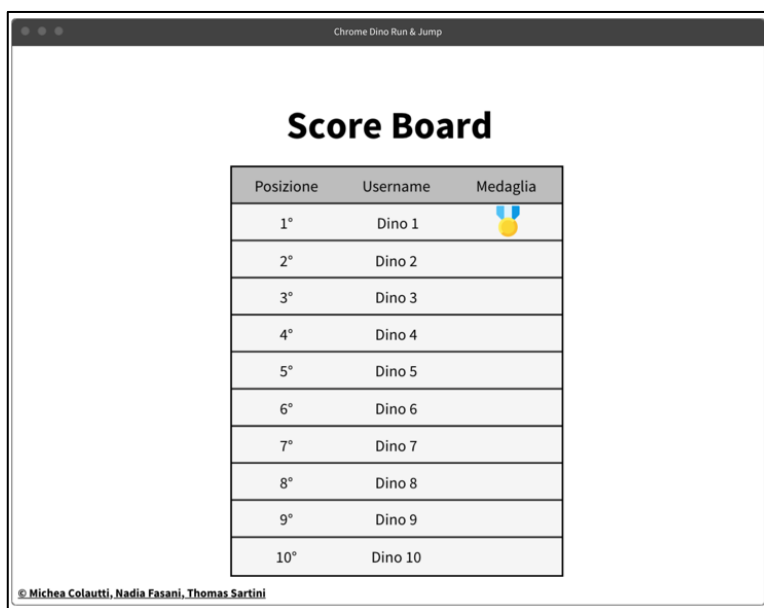


Figura 15 Classifica

### 3.4 Design procedurale

Per quanto riguarda il design procedurale, alleghiamo il diagramma di flusso da noi pensato.

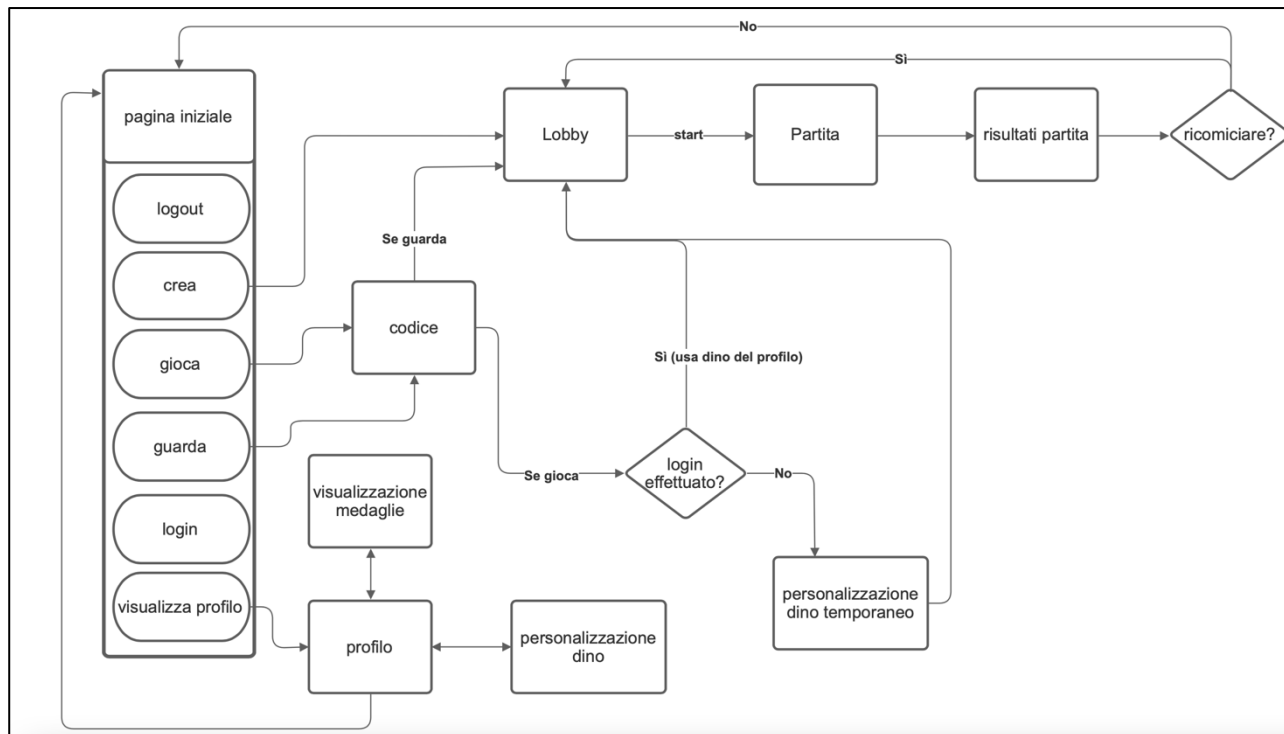



Figura 16 Diagramma di flusso

Lo schema del design è di per sé abbastanza chiaro. Tuttavia sarà maggiormente comprensibile con le seguenti precisazioni. Questo design ci è stato molto utile in fase di implementazione, poiché ci ha permesso di strutturare meglio il codice. Il punto di partenza è la *pagina iniziale*: da qui si può arrivare alla pagina di login oppure fare il logout. Nella *pagina iniziale* sono inoltre presenti altre funzionalità, che sono descritte di seguito:

- Creare una partita
  - Questo collegamento apre la partita nella lobby dove gli utenti si possono connettere.
  - In seguito, quando viene fatta partire la partita, il gioco comincia.
  - Una volta finito il gioco viene mostrata la classifica, e si può ricominciare a giocare o terminare.
- Giocare una partita
  - Questa parte del gioco prevede come prima cosa l'inserimento del codice della partita.
  - Se a giocare è un *guest* si viene indirizzati su una pagina di personalizzazione del dinosauro, altrimenti appare la lobby in attesa di iniziare la partita.
- Guardare una partita
  - Questo collegamento aprirà la stessa pagina di inserimento del codice usato per chi vuole giocare.
  - Una volta inserito il codice si viene portati alla partita.



	<b>SAMT – Sezione Informatica</b>	Pagina 17 di 46
	<b>Esempio di documentazione</b>	

- Visualizzare il proprio profilo.
  - Questa funzione invece permette all'utente di vedere il profilo, e contiene due sotto-funzioni:
    1. Visualizzare la bacheca con le medaglie
    2. Personalizzare il proprio dinosauro.

## 4 Implementazione

### 4.1 Refactoring

La prima parte del progetto è stata, vista la situazione iniziale, capire e modificare il codice di Manuel Grosso. Per farlo abbiamo adottato un approccio sistematico. Il codice originale era stato concepito per una versione del *Chrome Dino* con 4 giocatori, fissi. Quasi tutte le variabili e i metodi riguardanti i dinosauri, i cactus, ed altri elementi erano stati copiati e incollati 4 volte, cambiando solo il nome delle variabili. In pratica ogni dinosauro era una variabile, ogni dinosauro aveva il suo array di cactus, ogni terreno era una variabile, e così via. Abbiamo quindi preso ciò che di buono c'era, ovvero la presenza di Phaser e il funzionamento abbastanza buono del gioco in generale, per partire con il refactoring.

Abbiamo inizialmente tolto il codice JS dal file HTML e creato un file JS esterno, che ci sembrava più pulito, poi abbiamo iniziato a scorrere tutto il codice trasformando tutte le variabili in array. Di seguito alleghiamo solo qualche esempio, poi il resto dell'implementazione del nostro codice sarà spiegata meglio nel capitolo 4.8.

#### Prima

```
var terreni = [2];
var montagne = [2];
var nuvola;
var dino1;
var dino2;
var dino3;
var dino4;
var cactusDino1 = [5];
var cactusDino2 = [5];
var cactusDino3 = [5];
var cactusDino4 = [5];
```

```
//Settaggio difficoltà
if(punteggioDino1 > 4 && supVelocita[0]
|| punteggioDino2 > 4 && supVelocita[0]
|| punteggioDino3 > 4 && supVelocita[0]
|| punteggioDino4 > 4 && supVelocita[0]){

    velocitaSfondo+=1;
    supVelocita[0]=false;
    distanzaMinima += 30;
}
//ripetuto 5 volte con varianti
```

#### Dopo

```
var NUM_DINI = 0;
const NUM_GROUNDS = 3;
const NUM_MOUNTAINS = 10;
const NUM_CACTUS = 5;
...

var grounds;
var mountains;
var cloud;
var dini;
var cactus;

...

grounds = new Array(NUM_GROUNDS);
mountains = new Array(NUM_MOUNTAINS);
dini = new Array(NUM_DINI);
cactus = new Array(NUM_DINI);
```

```
function setDifficulty() {
    backgroundSpeed += 0.2 / NUM_DINI;
    minimumDistance += 6 / NUM_DINI;
}

//Example

function updateTerreni() {
    //grounds translation
    for (var i = 0; i < grounds.length; i++)
    {
        grounds[i].x -= backgroundSpeed;
    }
    ...
}
```

Come si può vedere dagli esempi riportati non ci siamo limitati solo al trasformare le variabili in array, ma abbiamo fatto un ulteriore lavoro di refactor.

Infatti una volta che il gioco era funzionante con gli array abbiamo inserito tutto il codice in dei metodi. Se prima c'erano enormi porzioni di codice direttamente nelle scene di gioco, di cui parleremo dopo, noi abbiamo fatto sì di avere solo metodi in queste parti, in modo da dare più ordine al codice e di renderlo più leggibile.

Sarebbe stato possibile anche utilizzare gli oggetti per eseguire il refactoring, ma per come era costruito il codice di Manuel era più intuitivo e veloce utilizzare gli array.

Oltre agli array abbiamo definito una serie di costanti atte allo specificare tutti i dati del gioco, come lo spazio tra i dinosauri, le grandezze di cactus, le coordinate di partenza, la translazione degli elementi, ecc.

## 4.2 Struttura cartelle del progetto

```
docs/
├── css/
│   ├── bootstrap.css
│   └── game.css
├── js/
│   ├── bootstrap.bundle.min.js
│   ├── game.js
│   └── index.js
├── GUI/
│   ├── bacheca.html
│   ├── collegamentoPartita.html
│   ├── dino.png
│   ├── game.html
│   ├── login.html
│   ├── paginaUtente.html
│   └── personalizzaDino.html
└── Game/
    ├── img
    ├── js/
    │   ├── game.js
    │   ├── medaglie.js
    │   ├── phaser.js
    │   └── phaser-arcade-physics.js
    └── index.html
...

```

## 4.3 docs/css/bootstrap.css

È la libreria di bootstrap per la gestione del CSS. Questa libreria ci è servita nelle pagine html per la gestione della grafica e per realizzare un sito responsive.

## 4.4 docs/css/game.css

Questo file contiene il codice per la grafica dello switch per l'interfaccia di gioco dal telefono.

## 4.5 docs/js/Bootstrap.bundle.min.js

È la libreria di bootstrap per la gestione degli elementi tramite JavaScript, serve per esempio per l'apertura dinamica dei modal di bootstrap.

## 4.6 docs/js/game.js

### 4.6.1 blockInput

Metodo per disabilitare il bottone "jump" per permettere all'utente di saltare con il telefono in tasca senza che venga cliccato per sbaglio.

### 4.6.2 handleMotion

Il metodo viene richiamato da un listener al movimento del dispositivo. Controlla se il dino dell'utente sta toccando terra e nel caso in cui l'accelerazione rilevata dal sensore fosse maggiore di 10 richiama il metodo jump che si trova nel file **docs/js/index.js**

### 4.6.3 requestPermission

Questo metodo serve a richiedere i permessi per utilizzare i sensori del telefono. Viene usato solo sui terminali IOS, poiché questo è l'unico sistema operativo che lo richiede. Infatti Apple ha imposto, per ragioni di privacy, che prima di utilizzare i sensori del telefono l'utente deva scatenare manualmente un evento, ad esempio cliccare su un pulsante, e poi acconsentire all'uso dei sensori tramite il pop-up che appare in automatico.

## 4.7 docs/js/index.js

### 4.7.1 writeMedals

Metodo per mostrare da Firebase tutte le medaglie ottenute dall'utente con un account. Fa riferimento al percorso **user/<user UID>/medals** e per ogni medaglia presente mostra gli svg aggiungendoli ad una tabella.

### 4.7.2 connectToGame

Questa funzione legge il valore della sessione inserita dall'input dall'utente nel file **docs/GUI/collegamentoPartita.html**. Poi viene creata una nuova variabile nel *localStorage* per salvare il numero della sessione a cui l'utente vuole accedere.

A questo punto verifichiamo che nella sessione scelta non ci siano più di 10 giocatori. Se la verifica va a buon fine l'utente può procedere, altrimenti viene ritornato un messaggio d'errore. Se l'utente non ha eseguito il login viene eseguito il metodo *generateGuestId* per poter avere un identificativo univoco per tutti gli utenti. In seguito l'utente viene aggiunto su Firebase all'interno della sessione chiesta dall'utente se esiste e viene aperta la pagina **docs/GUI/game.html**. Se invece l'utente ha eseguito il login, viene automaticamente controllato se la sessione esiste e in seguito l'utente viene aggiunto alla sessione tramite l'*uid* e viene aperta la pagina **docs/GUI/game.html**.

### 4.7.3 generateGuestId

Il metodo genera un id randomico per gli utenti guest nel seguente formato: "guest\_XXXXXX" e in seguito apre la pagina **docs/GUI/personalizzaDino.html** per permettere anche ai guest la possibilità di scegliere il colore del dino. Viene anche creata una nuova variabile *localStorage* per salvare localmente l'id appena creato.

#### 4.7.4 jump

La funzione accede all'id della sessione presente nel *localStorage* e percorre tutti i nodi presenti e trova il nodo che corrisponde al guest o all'utente che ha eseguito l'accesso. Poi esegue un update sull'attributo *is\_jumping* che imposta a true.

#### 4.7.5 registerNewUser

Il metodo legge le informazioni degli input inseriti dagli utenti e crea un nuovo account con email e password. Viene chiesto un nickname all'utente; in seguito viene aggiunto all'input dell'utente un dominio per far accettare a Firebase il nuovo account. Il formato della stringa che viene inviata a Firebase è "<nickname>@dino.ch". In caso di errori, c'è un elemento html che mostra il messaggio d'errore restituito da Firebase.

#### 4.7.6 loginUser

Il metodo legge le informazioni degli input inseriti dagli utenti e autentica l'utente con email e password come per il metodo precedente. Poi mostra alcuni elementi html non visibili per gli utenti guest. In caso di errori nel login c'è un elemento html che mostra il messaggio d'errore ritornato da Firebase.

#### 4.7.7 logoutUser

La funzione disconnette l'utente corrente e ricarica la pagina.

#### 4.7.8 openUserInformation

Il metodo apre la pagina **docs/GUI/paginaUtente.html**.

#### 4.7.9 generateSession

La funzione crea un numero randomico di 6 cifre e crea su Firebase un nuovo *child* sotto il ramo session. Poi apre la pagina **docs/Game/index.html**.

#### 4.7.10 changeDinoColor

Questo metodo viene usato per cambiare il colore al dinosauro nella pagina di personalizzazione. Il colore esadecimale passato come argomento viene infatti assegnato all'elemento con id *dino*

#### 4.7.11 saveDinoColor

Questo è il metodo che, invece, salva il colore del dino.

Si differisce dal metodo precedente poiché al posto di cambiarlo solo a livello grafico, questa parte di codice lo salva su Firebase. Tuttavia il metodo non fa solo questo: infatti il nostro sistema prevede che questo metodo, invocato per altro solitamente in seguito al metodo *changeDinoColor*, venga invocato in due casi distinti: il primo è quando l'utente cambia il colore del dinosauro dalla pagina di personalizzazione, il secondo è quando un utente guest accede ad una partita e può modificare temporaneamente il dinosauro. Per questo è stato introdotto un controllo che, in pratica, verifica che l'utente corrente sia un utente registrato o meno; nel primo caso si limita a salvare il colore su Firebase, accedendo all'istanza corretta, ed a riportare l'utente alla pagina personale (**docs/GUI/paginaUtente.html**). Nel secondo invece il metodo salva il colore e poi crea l'istanza di utente guest nella sessione corretta. Per fare tutti questi passaggi vengono usati i differenti *localStorage* che sono stati o meno settati. Poi porta l'utente alla pagina di gioco **docs/GUI/game.html**

#### 4.7.12 showUserInfo

Questo metodo ci permette di leggere da Firebase tutte le informazioni dell'utente corrente, come il nome o il punteggio massimo.

#### 4.7.13 firebase.auth().onAuthStateChanged((user) =>

Questo è un metodo molto particolare: infatti non è propriamente un metodo ma un listener. Questa parte di codice è stata implementata quando ci siamo accorti che spesso, soprattutto in caso di pagine ricaricate o simili, il codice JS e HTML era più veloce ad eseguirsi rispetto alle richieste in Firebase. Un esempio può essere il login, infatti se un utente si fosse loggato sarebbe stato portato alla home page. Qui però, se avesse cliccato sulla sua pagina personale, si sarebbe presentato un errore. Infatti abbiamo scoperto che l'unico motivo per cui apparivano i link alla pagina utente stava nel fatto che avevamo fatto in modo che dopo il login questi controlli apparissero sempre, senza preoccuparsi di prendere prima i dati da Firebase. Non sapendo noi quando questi dati arrivano abbiamo implementato questo listener. Ora prima di caricare la home page dopo il login aspettiamo di ottenere i dati da Firebase: quando lo stato dell'utente loggato cambia viene invocato questo listener, che salva l'*uid* in un *localStorage*, e in seguito modifica la home page, nascondendo il pulsante di login e mostrando quello per la pagina utente. Questo metodo viene usato, oltre che per il login, per caricare il dinosauro nella pagina utente. Anche qui avevamo lo stesso problema di ritardo dei dati. Il listener è usato assieme al metodo per prendere le informazioni dell'utente, citato poco fa, da mostrare nelle pagine.

Se alcune volte il sito sembra mutare in ritardo oppure caricare elementi della pagina dopo la pagina stessa è "colpa" quindi del ritardo dato da Firebase, che abbiamo cercato di arginare con questo metodo.

#### 4.7.14 checkLoggedInUser

Questa funzione serve a sapere se l'utente è loggato e, di conseguenza, sapere se mostrare o meno il collegamento alla pagina personale. Questa funzione viene richiamata al caricamento della home page.

#### 4.7.15 getIsTouchingDown

Questa funzione permette al gioco di sapere se l'utente sta toccando terra; il motivo di questo metodo è semplice. Se l'utente, saltando fisicamente, dovesse saltare abbastanza in alto, sarebbe possibile che al momento di superare un cactus il nostro sistema di rilevamento dell'accelerazione prenda due salti al posto di uno. Questo potrebbe influire negativamente sul gioco, in quanto il dino compie due salti di fila, ed è già successo nei test che l'utente non faccia poi in tempo a saltare il cactus successivo. Per questo motivo la funzione in questione impedisce al dinosauro di saltare a meno che non stia toccando terra.

#### 4.7.16 Assenza di watchGame

In questo capitolo sarebbe dovuta andare l'implementazione della funzione per guardare una partita, anche detta *ghost*. Purtroppo a circa metà dell'implementazione ci siamo accorti di un problema critico con questa funzione: il database e la struttura da noi scelti non erano funzionali allo sviluppo del *ghost*. Quando siamo arrivati all'implementazione dell'utente *ghost* abbiamo inizialmente pensato ad un semplice modo di farlo: al posto di essere aggiunto come utente registrato o come guest, l'utente avrebbe avuto un id speciale, qualcosa tipo "*ghost\_XXXXXX*". Il gioco, leggendo un id simile, avrebbe ignorato completamente questo giocatore e non avrebbe aggiunto il dinosauro per lui, appunto perché era solo uno spettatore. Avevamo pensato che bastasse, come utente *ghost*, leggere i dati da Firebase per sapere quando la partita iniziava o finiva o quando i dinosauri saltavano. In pratica il *ghost* avrebbe visto la stessa cosa dell'utente *host*. La partita vista del *ghost* sarebbe stata come una partita normale quindi, ma al posto di controllarla gli utenti con il telefono l'avrebbe controllata Firebase stesso. Pensavamo di non doverci preoccupare, essendo

la logica di gioco sul computer del *ghost*, di cose come le tempistiche di gioco o la posizione dei dinosauri, in quanto corrono e saltano tutti allo stesso modo.

Tuttavia non abbiamo considerato i cactus. Se è vero che tutti i dati della partita e dei dinosauri sono su Firebase, ciò non si può dire per gli ostacoli. Infatti essi vengono generati **casualmente** per ogni partita avviata. Perciò anche se il *ghost* si fosse inserito nella partita giusta non sarebbe mai riuscito a vedere la stessa cosa dell'*host*: i dinosauri avrebbero saltato allo stesso momento di quelli originali certo, ma sarebbero quasi sicuramente andati a sbattere poiché la posizione dei cactus sarebbe stata differente rispetto all'originale.

Abbiamo pensato di mettere la posizione dei cactus su Firebase, ma due fattori ci hanno fermato:

1. La mole di dati: Dai nostri test risultava che, a meno che non ci fosse una connessione a internet abbastanza veloce, il dinosauro saltava con un ritardo che oscillava tra 0.5 e 1 secondi di ritardo. Se, oltre allo scambio di informazioni già in atto, avessimo introdotto un ulteriore scambio di coordinate con una frequenza di, almeno, 0.5 secondi; avremmo bloccato l'applicazione. Questo dipende dalla connessione, dalla latenza del sistema, ma anche dal fatto che abbiamo adottato un protocollo *text-based*, quindi molto pesante.
2. L'incognita del non sapere se avrebbe funzionato: Pur provando a implementare un sistema simile non eravamo certi di risolvere il problema, anzi. Avevamo paura di rovinare ciò che di buono era stato fatto fino a quel momento.

Per questi fattori, e dopo una discussione con il mandante, abbiamo deciso di accantonare questa funzione, in quanto non implementabile a quel punto dello sviluppo del progetto.

## 4.8 docs/Game/js/game.js

Per questo capitolo nello specifico la documentazione procederà in maniera differente: verrà fatta una breve introduzione a Firebase e in seguito passeremo a spiegare il codice ma non funzione per funzione, bensì seguendo la logica di gioco. In questo modo eviteremo spiegazioni inutili, alcuni metodi sono pressoché identici uno all'altro, e speriamo di semplificare la comprensione del funzionamento del gioco.

Questa è forse il file js più importante di tutto il nostro progetto: da questa funzione ha origine tutto il gioco vero e proprio, dalla *lobby* allo *scoreboard*. Per svilupparla, come detto nel capitolo riguardante il refactoring, abbiamo utilizzato come base il codice di Manuel Grosso.

### 4.8.1 Aggiunta dinamica dell'utente

L'aggiunta di un utente ad una partita viene fatta tramite un listener: conoscendo l'id della sessione possiamo porre un controllo sui cambiamenti dell'istanza Firebase. Ogni volta che viene aggiunto un *child* il sistema verifica che non ci siano troppi utenti nella partita e, se l'accesso va a buon fine, vengono settate le variabili globali del gioco. Per aggiungere dati ai vari array utilizziamo il metodo *push()*, prendendo i dati da Firebase. Ecco un esempio:

```
diniNicknames.push(snapshot.key);
```

#### 4.8.2 setSettingsPhaser

Inizialmente abbiamo quindi diviso il gioco in 3 *scene* distinte. Infatti tutta la logica di Phaser –il framework che abbiamo usato per le dinamiche del gioco, per la gestione delle collisioni, della fisica ecc.– sfrutta la un sistema a scene. Ogni scena è composta da 3 fasi:

3. Il **preload**: qui si svolgono operazioni iniziali e essenziali, come ad esempio la definizione degli assets.
4. Il **create**: qui viene creata, ad esempio, la scena iniziale in sé. Infatti nel nostro caso abbiamo inserito nella pagina tutti gli elementi esterni, come le montagne, il terreno o le nuvole.
5. L'**update**: questa è quella che possiamo definire la scena effettiva del gioco, viene richiamata 60 volte al secondo e permette di far muovere, interagire, ecc. tutti gli elementi del gioco.

```
var sceneLobby = {
  key: 'sceneLobby',
  preload: preloadGame,
  create: createGame,
  update: updateLobby
};
```

Questo è un esempio di scena, nel nostro caso la lobby. Oltre alla chiave che identifica le varie scene ci sono le 3 fasi sopracitate e il metodo da eseguire per ogni fase.

Oltre alle scene in questa funzione viene settata anche la variabile *config*. Questa variabile, che viene definita con la sintassi di un simil-json, permette di specificare una serie di attributi come lo sfondo del gioco, la sequenza delle scene, la gestione delle collisioni ecc.

#### 4.8.3 La lobby

Appena viene creata una partita tramite l'apposito pulsante si viene catapultati nella lobby. Qui ci sono, come detto, 3 fasi.

##### 4.8.3.1 preloadGame

Questa funzione permette semplicemente di caricare gli assets che verranno usati nel gioco, ecco un esempio, dove carichiamo il terreno di gioco.

```
this.load.image('terreno', host + 'img/terreno3.png');
```

Tramite la *keyword* terreno sarà possibile utilizzare quest'immagine.

##### 4.8.3.2 CreateGame

Questa funzione è quella responsabile di disporre nella pagina html tutti gli elementi del gioco, per farlo usiamo una lista di metodi:



### 1. setStartValues

Questa funzione ci per impostare tutti i valori di partenza: qui assegnamo agli array/matrici che si serviranno in seguito la lunghezza appropriata. Per esempio abbiamo un array di dini lungo quanto il nuemro di giocatori presenti nella partita. Tuttavia in questa funzione settiamo anche dei valori come il margine tra un dino e l'altro, la velocità, ecc.

Inoltre la funzione richiama un metodo fondamentale al funzionamento del jump, ovvero la funzione *createListener*.

### 2. createListener

Questa funzione ci permetti di creare i listener per catturare il salto dell'utente.

In pratica la funzione ha biosgno di ottenre l'indice dal dino che ha sltato, in modo da poter mettere a true l'indice corretto nell'array responsabile del tracciamento dei salti. In seguito, quando il sistema verificherà se il dinosauro sta saltando, farà saltare il dinosauro a livello grafico, grazie al metodo *checkJump*.

Per ottenere quest'infomrazione il metodo utilizza un listener, che verrà richiamato ad ogni cambiamento di ogni child nella sessione corrente. Tramite Firebase otteniamo uno alla volta tutti gli oggetti, gli utenti, che in quel momento sono nella sessione e, ottenendo il loro nickname tramite gli "attributi" dell'oggetto stesso, possiamo risalire all'indice del dinosauro. In seguito verifichiamo che l'attributo cambiato sia effettivamente "is\_jumping" e, in caso, aggiorniamo il valore dell'array. Questo controllo è motlo imporante, in quanto il listener viene richiamato ad ogni cambiamento di un qualunque *child* dell'utente nella sessione corrente; di conseguenza capitava che il dinosauro saltava anche quando non era stato scatenato un evento.

```
//selezione dell'utente
db.ref("session/" + localStorage.getItem("sessionId") + "/"
  + diniNicknames[i]).on("child_changed", function(data) {

  //controllo sull'atributo
  var index = diniNicknames.indexOf((data.ref_.path.pieces_)[2]);
  var player_jump = data.val();
  if (player_jump && (data.ref_.path.pieces_)[3] == "is_jumping") {
    diniJumps[index] = true;
  }
});
```

### 3. setColliderLines

Questa funzione ci permette di creare la fisica del terreno che i dinosauri useranno per non precipitare nel vuoto. La prima cosa che facciamo è creare un *gruppo di collisione* di Phaser. Questo ci peremette di radunare più oggetti in un unico costruito e aggiungerli una fisica comune. Nel nostro caso creiamo il "pavimento" per ogni dinosauro. Infatti conoscendo il numero di dini, con un ciclo for e con delle altezze pre-impostate possiamo creare tutte le line di fondo.

### 4. setGrounds

Questo metodo permette di disporre i terreni per ogni dino. Per il gioco usiamo una tecnica che consiste nel avere un numero predefinito di terreni, 2 nel nostro caso, e di continuare a traslarli una volta che il personaggio li supera; ad esempio: Abbiamo 2 terreni lunghi 2000 pixel, una volta che il dinosauro supera i primi 2000 pixel -il primo terreno- subentra il secondo. Il primo invece viene spostato dalla posizione 0 alla posizione 4000, in modo da accogliere il dinosauro una volta che superà anche il secondo terreno. Ecco il codice:

```
for (var i = 0; i < grounds.length; i++) {
    grounds[i] = gameRef.physics.add.image(counter, 368, 'ground').setOrigin(0, 0);
    grounds[i].setImmovable(true); //fissa i terreni
    grounds[i].body.allowGravity = false; // toglie la gravità
    counter += 2000;
}
```

È necessario togliere la gravità ai terreni poiché altrimenti sarebbero loro stessi a precipitare.

#### 5. setMountains

Questo metodo funziona pressapoco come quello precedente, l'unica differenza sta nel valore della traslazione.

#### 6. setCloud

Questo metodo aggiunge semplicemente un'unica nuvola al gioco, senza traslarla o duplicarla.

#### 7. setCactus

Questo metodo è leggermente più complesso rispetto ai precedenti, infatti in questo caso i cactus non sono un array, ma una matrice. La variabile che tiene traccia dei cactus infatti non si limita a contenere quelli di una sola linea, ma li contiene tutti. Oltre ad avere, ad esempio, 4 cactus in orizzontale la variabile conterrà anche tanti cactus "in verticale" quanti sono i giocatori presenti nel gioco.

Oltre a cambiare la logica delle variabili cambia però anche la logica di posizionamento: infatti non è sempre fissa ma quasi randomica. Infatti il valore della distanza è compreso in un range predefinito, per motivi di giocabilità. Alleghiamo un piccolo estratto del codice.

```
for (var i = 0; i < cactus.length; i++) {
    for (var j = 0; j < cactus[i].length; j++) {
        ...
        var distance = Math.floor(Math.random() * 200) + 70;
        cactus[i][j] = gameRef.physics.add.image(x, START_HEIGHT +
            (i * HEIGHT_SPACE) - HEIGHT_CACTUS, 'cactus').setOrigin(0, 0);

        cactus[i][j].setImmovable(true);
        cactus[i][j].body.allowGravity = false;
    }
}
```

#### 8. setAnimations

Questa funzione permette di impostare le animazioni valide per tutto il gioco. Firebase lavora utilizzando degli speciali file png che contengono le diverse animazioni. Nel nostro caso un file in cui sono rappresentati i vari stati del dino: fermo, mentre sta correndo con la zampa destra o sinistra alzata, morto. Ciò semplifica molto la gestione delle animazioni; ciò che bisogna fare è specificare da quale immagine iniziare e con quale finire e il *frame-rate*.

#### 9. setDini

C'è poi il settaggio dei dini, potrebbe sembrare difficile ma in realtà è più semplice di altri metodi che abbiamo visto precedentemente. È simile in realtà al metodo per i terreni e per i cactus: anche qui ci basiamo su un array, lungo quanto il numero di dini, che ci permette di disporli lungo l'asse verticale. Impostiamo poi il colore ai dinosauri e gli impostiamo anche le collisioni con i bordi.

```
for (var i = 0; i < dini.length; i++) {
    dini[i] = gamescene.physics.add.sprite(START_DISTANCE_DINI +
        (i * TRANSLATION), 0, 'dinoSprite').setOrigin(0, 0);
    dini[i].setTintFill(diniColor[i], diniColor[i], diniColor[i], diniColor[i]);
    dini[i].setCollideWorldBounds(true); //Collisions between border and dini
    colliderDini[i] = gamescene.physics.add.collider(dini[i], linesGroup.getChildren()[i]);
    dini[i].play("run");
}
```

#### 10. setColliderCactusDini

Questa funzione viene eseguita solo se la variabile *ignoreCollisions* è a false.

Quello che fa è, tramite i metodi di Phaser e due cicli *for*, impostare le collisioni tra i dinosauri e i cactus.

```
...
gamescene.physics.add.overlap(dini[i], cactus[i][j], collideCactus, null, gamescene);
...
```

La collisione è di tipo *overlap* in quanto l'effetto desiderato è che il dinosauro vada propriamente a sbattere contro il cactus e poi cada nel nulla, se avessimo usato invece *collide* il dinosauro sarebbe rimbalzato contro il cactus.

La funzione *collideCactus* invece mette a -250 la velocità del dinosauro e rimuove le sue collisioni con il terreno, facendolo precipitare nel vuoto.

#### 11. setDiniNicknames

Infine l'uso di questa funzione si può evincere dal nome: usando l'array di nicknames il gioco scrive i nomi degli utenti prima dei loro dinosauri. Durante il gioco i nomi avanzeranno con i loro personaggi.

#### 4.8.3.3 updateLobby

Questa scena invece permette l'aggiornamento continuo della lobby. Al suo interno sono presenti le invocazioni ai metodi *setTouchingDown* e *checkJump*.

Inoltre è presente il controllo per sapere quando passare alla scena successiva.

Per passare alla scena successiva, viene fatto un controllo su di un attributo booleano che può essere modificato solo premendo il pulsante "Start" sulla GUI dell'host.

Una volta che la partita è stata fatta paritre la scena cambia con:

```
this.scene.switch('sceneGame');
```

Altrimenti viene fatta ripartire con:

```
rif.scene.restart();
```

#### 1. setTouchingDown

Questo metodo permette di cambiare il valore *is\_touchingDown* su *Firebase*. Questo serve a prevenire il problema spiegato nel capitolo 4.7.15.

Essenzialmente questo metodo scorre tutti i nicknames degli utenti nel gioco, aggiornando il valore di ognuno di essi con *true* o *false*.

Questo è possibile grazie al fatto che ogni dino, essendo anche un oggetto di *Firebase*, possiede una serie di “attributi” ottenibili con una semplice stringa.

```
...
db.ref('session/' + localStorage.getItem("sessionId") + "/" +
    diniNicknames[i]).update({ 'is_touchingDown': dini[i].body.touching.down });
...
```

#### 2. checkJump

Questa funzione scorre tutti i dinosauri, verificando se stanno saltando o meno. Verifica anche se i dinosauri stanno toccando terra.

Se entrambi i controlli vengono passati allora la funzione imposta a *false* il valore “is\_jumping” su *Firebase*.

Questo è fondamentale al meccanismo di salto che abbiamo spiegato all’inizio.

### 4.8.4 La Game scene

La seconda scena del gioco è la *Game scene*. Anche lei è composta da 3 parti, ma per il **preload** e per il **create** vengono usate le stesse scene della lobby. Per questo parleremo solo della parte di **update**.

#### 4.8.4.1 updateGame

Update game utilizza, come la sua controparte nella lobby, una serie di metodi diversi.

##### 1. setTouchingDown

Già documentato.

##### 2. updateTerreni

Come già brevemente accennato nel capitolo precedente sui terreni, la logica è quella di spostare i terreni ogni volta che è necessario. Per questo il metodo contiene un ciclo for che controlla la posizione dei terreni, e se essa scende sotto una certa soglia allora il terreno viene spostato in avanti.

##### 3. updateMontagne

Per le montagne vale lo stesso discorso dei terreni, stessa logica.

##### 4. updateCactus

Anche i cactus vengono spostati, ogni volta che escono dallo schermo vengono automaticamente portati al capo opposto.

##### 5. updateNuvola

La nuvola viene spostata lungo la x di un valore randomico ad ogni volta che si trova fuori dalla schermata.

##### 6. checkJump

Già documentato.

#### 7. `setScore`

Il metodo controlla che un dinosauro abbia effettivamente superato un cactus e, se è il caso, aumenta di uno il punteggio.

Inoltre invoca la funzione `setDifficulty`.

#### 8. `checkEndOfGame`

Questa funzione verifica che il gioco non sia finito: per farlo controlla che la posizione dei dini sia superiore a 50. Infatti quando i dini muoiono vengono fatti precipitare nel voto, anche per non intralciare eventuali altri giocatori ancora in vita. Questo meccanismo è spiegato anche nel capitolo 4.8.2.2 al punto 9.

Infine viene invocata la funzione `endOfGame`, esplorata nel prossimo capitolo.

#### 9. `setDifficulty`

Questa funzione permette di aumentare man mano la velocità di scorrimento e quindi la difficoltà. Il valore dell'incremento è frutto di molti test e prove. Tuttavia la velocità incrementa sempre di 0.2/Il numero di dini. Con l'incrementare la velocità aumenta anche la distanza minima di posizionamento dei cactus.

### 4.8.5 La leaderboard

Quest'ultima parte del gioco non è una scena vera e propria; infatti questa funzione blocca e nasconde tutto il *div* contenente il gioco, sostituendo il contenuto della pagina con la leaderboard. Il processo di creazione della leaderboard ha 3 parti principali:

1. La creazione della medaglia
2. La creazione della classifica
3. Il salvataggio della medaglia
4. Il salvataggio del punteggio

Della prima parte parleremo nel prossimo capitolo, dedicato alle medaglie.

#### 4.8.5.1 leaderboard

La prima funzione per la leaderboard è, chiaramente, la creazione della leaderboard stessa. Per farlo abbiamo deciso di creare una tabella contenente tutti i risultati degli utenti in partita. Per la prima riga della tabella, quella del vincitore, viene anche creata una medaglia.

Una volta che la medaglia è stata creata viene salvata con la funzione `saveMedal` e tutti i punteggi vengono salvati con la funzione `saveScore`.

#### 4.8.5.2 `saveMedal`

Questa funzione prende la medaglia generata e la salva sull'db Firebase. Ciò viene chiaramente fatto solo per gli utenti registrati, che sono salvati in una istanza a parte.

#### 4.8.5.3 `saveScore`

Per il salvataggio dei punti invece vengono passati tutti gli utenti in partita e, per quelli registrati, viene fatto un controllo tramite Firebase sul loro punteggio massimo. Se quello appena relizzato è più alto di quello archiviato, quest'ultimo viene sostituito.

La leaderboard però presenta anche 2 pulsanti, uno nell'angolo destro e l'altro nell'angolo sinistro.

Il primo, il pulsante "Home" riporta l'host alla home page, eliminando per altro la sessione da Firebase.

Il secondo invece, quello “Restart” permette all’host di far ripartire il gioco da 0. Ecco un esempio di leaderboard.


HOME				RESTART
#	Nickname	Score	Medal	
1	user	9		
2	guest_141324	1		
3	guest_397858	1		
4	guest_412564	1		
5	guest_897535	1		

Figura 17 Esempio leaderboard

## 4.9 docs/Game/js/medaglie.js

Questo file serve ad un unico scopo, generare medaglie casuali.

Le medaglie sono composte di base da un cerchio giallo, che funge da sfondo. In seguito la funzione *createMedal* inizia a disegnare delle forme casuali riempiendola di forme, per ora vuote. Il risultato di questa funzione viene passato alla funzione *fillMedal*, che sceglie in modo casuale quali elementi della medaglia disegnare e di quali colori (anch’essi scelti in modo casuale tramite il metodo *getRandomColor*) colorarli. L’array finale che ne risulta viene ordinato dalla forma più grande a quella piccola per evitare sovrapposizioni. Infine tutto questo array di informazioni viene passato al metodo *drawMedal*, che ne crea un svg che sarà ritornato e/o salvato.

## 4.10 docs/Game/js/phaser.js e phaser-arcade-physics.js

Questi due file sono il cuore di Phaser. Grazie ad essi infatti riusciamo ad utilizzare tutte le funzioni di Phaser all’interno del nostro file.

## 4.11 docs/Game/index.html

Questa è la pagina principale del gioco; è qui che i dini appaiono e giocano. A livello di html in realtà non c’è pressoché nulla. C’è solo il pulsante di start e la chiamata alla funzione di Phaser che crea tutto. Ecco come appare una partita in attesa di essere avviata:

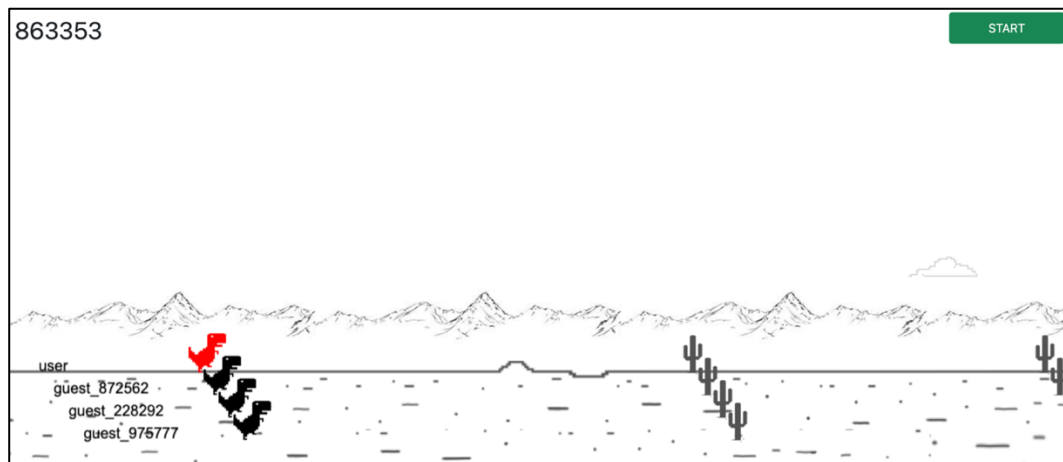


Figura 18 Esempio di partita

#### 4.12 docs/GUI/login.html



Figura 19 Home page con un utente loggato

È la home page del gioco, da questa pagina è possibile registrarsi, effettuare il login, creare una partita e giocare una partita. Inoltre una volta loggati è possibile accedere alla propria pagina personale.

#### 4.13 docs/GUI/collegamentoPartita.html



Da questa pagina, raggiungibile dal pulsante “Connessione alla partita”, è possibile inserire un codice per poi giocare una partita. Se l'utente non ha effettuato il login verrà prima portato alla pagina di personalizzazione del dino, di cui parleremo dopo.

Figura 20 Pagina di collegamento ad una partita

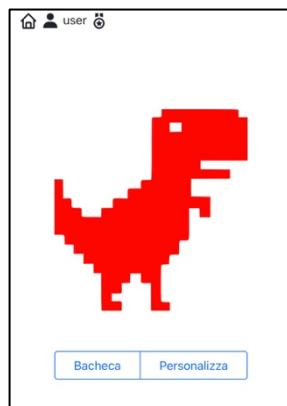
#### 4.14 docs/GUI/game.html



È la pagina principale di gioco: da qui è possibile premere il bottone per saltare, se si hanno problemi di mobilità o quando il sito non ha accesso alle informazioni sui movimenti del dispositivo. Inoltre questa pagina cattura i movimenti dell'utente e li trasmette al database Firebase. Se si possiede un telefono IOS è necessario prima premere il pulsante blu per consentire l'accesso ai sensori del telefono.

Figura 21 Pagina di gioco

#### 4.15 docs/GUI/paginaUtente.html



La pagina mostra le varie informazioni dell'utente, a patto che si sia precedentemente autenticato. Inoltre sono presenti i collegamenti alla pagina di personalizzazione e alla bacheca.

Figura 22 Pagina utente



#### 4.16 docs/GUI/personalizzaDino.html



Figura 23 Pagina personalizzazione dinosauro

Questa pagina permette all'utente registrato di cambiare il colore del proprio dinosauro; una volta che si è impostato il colore prescelto si preme su salva e si torna alla pagina utente. Al contempo permette anche agli utenti guest di personalizzare temporaneamente il dinosauro: nel loro caso una volta che viene premuto il pulsante "salva" il sito reindirizza il browser alla pagina di gioco.

#### 4.17 docs/GUI/bacheca.html

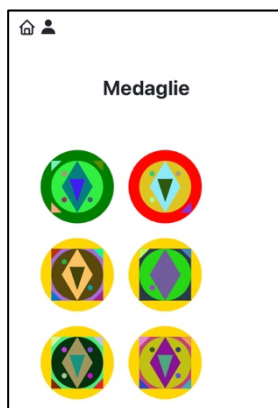


Figura 24 Bacheca

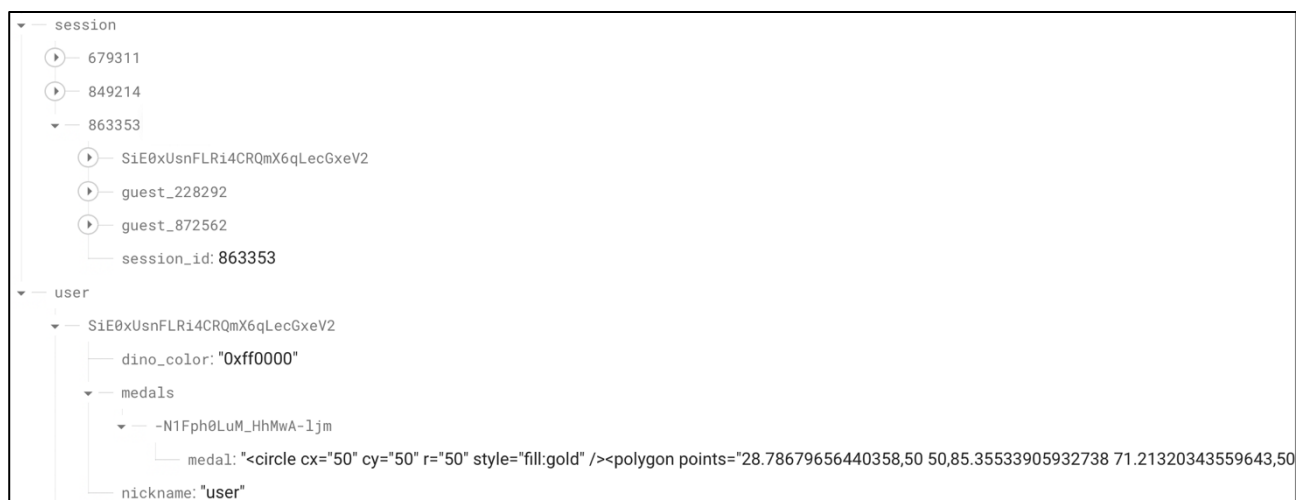
La pagina mostra agli utenti con un account le medaglie ottenute nel corso del tempo.

## 4.18 La configurazione di Firebase

Per quanto riguarda Firebase non abbiamo molto da dire. All'inizio ci siamo attenuti all'architettura pensata ma abbiamo dovuto espanderla molto per adattarsi alle nostre esigenze. Per comunicare con i server è necessario, all'inizio di ogni file js, inserire tutte le variabili, accessi e dati utili al funzionamento. Alleghiamo di seguito il codice:

```
var firebaseConfig = {
  apiKey: "AIzaSyA4fyvc6p7bnP6TipjCpcc4V-dhysnRdx0",
  authDomain: "dino-run-and-jump-d2cdb.firebaseio.com",
  databaseURL: "https://dino-run-and-jump-d2cdb-default-
    rtdb.europe-west1.firebaseio.com",
  projectId: "dino-run-and-jump-d2cdb",
  storageBucket: "dino-run-and-jump-d2cdb.appspot.com",
  messagingSenderId: "200118050769",
  appId: "1:200118050769:web:71d81d4e42b99e56110af0",
  measurementId: "G-S7CQ1S72TK"
};
```

Ecco una foto del nostro DB Firebase finale



## 5 Test

### 5.1 Protocollo di test

Test Case:	TC-01	Nome:	Creazione di una sessione
Riferimento:	REQ-01		
Descrizione:	L'utente raggiunge il sito e crea una partita		
Prerequisiti:	-		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere il pulsante "Crea una partita"</li> </ol>		
Risultati attesi:	<ol style="list-style-type: none"> <li>1. La pagina deve mostrare la schermata di gioco</li> <li>2. La console di Firebase deve mostrare la sessione creata</li> </ol>		

Test Case:	TC-02	Nome:	Registrazione utente
Riferimento:	REQ-02		
Descrizione:	L'utente deve potersi registrare nel sito		
Prerequisiti:	-		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere sul pulsante "Sign Up"</li> <li>3. Immettere username e password</li> <li>4. Confermare</li> </ol>		
Risultati attesi:	<ol style="list-style-type: none"> <li>1. L'utente deve essere loggato</li> <li>2. Nella console di Firebase deve essere presente un nuovo utente con l'username corretto (la password è cifrata)</li> </ol>		

Test Case:	TC-03	Nome:	Login
Riferimento:	REQ-03		
Descrizione:	L'utente deve potersi autenticare		
Prerequisiti:	Un utente creato		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere sul pulsante "Sign In"</li> <li>3. Immettere username e password</li> <li>4. Confermare</li> </ol>		
Risultati attesi:	L'utente deve essere loggato		

Test Case:	TC-04	Nome:	Unione ad una partita
Riferimento:	REQ-04		
Descrizione:	Un utente deve poter giocare una partita		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Un utente creato</li> <li>2. Una partita creata ed aperta su uno schermo</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Effettuare il login</li> <li>3. Premere sul pulsante "Connessione alla partita"</li> <li>4. Immettere il codice della partita</li> <li>5. Confermare</li> </ol>		
Risultati attesi:	L'utente deve apparire nella partita nella pagina dell'host		

Test Case:	TC-05	Nome:	Funzione aggiuntive login 1
Riferimento:	REQ-05		
Descrizione:	L'utente deve poter personalizzare il suo dinosauro		
Prerequisiti:	Un utente creato		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Effettuare il login</li> <li>3. Recarsi nella pagina utente premendo sull'icona dello stesso</li> <li>4. Premere su "Personalizza"</li> <li>5. Scegliere un colore per il dinosauro</li> <li>6. Premere "Salva"</li> </ol>		
Risultati attesi:	Il colore del dinosauro deve essere cambiato nella pagina utente		

Test Case:	TC-06	Nome:	Funzione aggiuntive login 2
Riferimento:	REQ-05		
Descrizione:	Il punteggio dell'utente deve essere salvato		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Un utente creato che non ha mai giocato una partita</li> <li>2. Una partita creata ed aperta su uno schermo</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Effettuare il login</li> <li>3. Premere sul pulsante "Connessione alla partita"</li> <li>4. Immettere il codice della partita</li> <li>5. Confermare</li> <li>6. Avviare la partita dall'host</li> <li>7. Giocare</li> </ol>		

	8. Quando la partita finisce recarsi nella pagina dell'utente
Risultati attesi:	Il punteggio fatto deve essere visibile

Test Case:	TC-07	Nome:	Funzione aggiuntive login 3
Riferimento:	REQ-05		
Descrizione:	Le medaglie devono essere presenti nella bacheca		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Un utente creato con almeno una medaglia</li> <li>2. Una partita creata ed aperta su uno schermo</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Aprire la pagina utente</li> <li>3. Premere su "Bacheca"</li> </ol>		
Risultati attesi:	Le medaglie dell'utente devono visibili		

Test Case:	TC-08	Nome:	GUI responsive
Riferimento:	REQ-06		
Descrizione:	La GUI del sito deve essere responsive		
Prerequisiti:	Un utente loggato		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Effettuare il login</li> <li>3. Provare a ridimensionare la pagina e verificare che nessun elemento venga nascosto</li> <li>4. Ripetere il test per tutte le pagine eccetto quella di gioco</li> </ol>		
Risultati attesi:	Tutti gli elementi nella pagina si spostano/ridimensionano correttamente		

Test Case:	TC-09	Nome:	Pagina di gioco delle dimensioni massime
Riferimento:	REQ-06		
Descrizione:	La pagina di gioco deve essere sempre delle dimensioni massime della pagina		
Prerequisiti:	-		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Creare una nuova partita</li> <li>3. Verificare che la pagina occupi tutto lo spazio disponibile</li> <li>4. Provare a ridimensionare la pagina e poi refreshare</li> </ol>		
Risultati attesi:	La pagina assume sempre le dimensioni massime consentite		

Test Case:	TC-10	Nome:	Gioco come guest
Riferimento:	REQ-07		
Descrizione:	Si deve poter giocare anche senza aver effettuato il login		
Prerequisiti:	Una partita creata ed aperta su uno schermo		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere sul pulsante "Connessione alla partita"</li> <li>3. Immettere il codice della partita</li> <li>4. Confermare</li> <li>5. Avviare la partita dall'host</li> <li>6. Giocare</li> </ol>		
Risultati attesi:	L'utente deve poter giocare normalmente		

Test Case:	TC-11	Nome:	Personalizzazione dinosauro guest
Riferimento:	REQ-08		
Descrizione:	Si deve poter personalizzare il proprio dinosauro anche senza aver effettuato il login, appena prima di unirsi alla partita		
Prerequisiti:	Una partita creata ed aperta su uno schermo		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere sul pulsante "Connessione alla partita"</li> <li>3. Immettere il codice della partita</li> <li>4. Selezionare il colore del dinosauro</li> <li>5. Confermare</li> </ol>		
Risultati attesi:	Il dinosauro deve apparire nella pagina dell'host con il colore prescelto.		

Test Case:	TC-12	Nome:	Giocare con i sensori del telefono
Riferimento:	REQ-09		
Descrizione:	Si deve poter giocare sfruttando i sensori del telefono		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Una partita creata ed aperta su uno schermo</li> <li>2. Un telefono con i sensori di movimento appropriati (<b>NO</b> IOS)</li> <li>3. Un utente creato</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere sul pulsante "Connessione alla partita"</li> <li>3. Immettere il codice della partita</li> <li>4. Confermare</li> <li>5. Provare a saltare con il telefono in mano o in tasca</li> <li>6. Ripetere il test con un utente loggato</li> </ol>		

Risultati attesi:	Il dinosauro deve saltare nella pagina dell'host.
-------------------	---

Test Case:	TC-13	Nome:	Giocare con il pulsante
Riferimento:	REQ-09		
Descrizione:	Si deve poter giocare sfruttando i sensori del telefono.		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Una partita creata ed aperta su uno schermo</li> <li>2. Un utente creato</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere sul pulsante "Unisciti ad una partita"</li> <li>3. Immettere il codice della partita</li> <li>4. Confermare</li> <li>5. Provare a premere il pulsante "Jump"</li> <li>6. Ripetere il test con un utente loggato</li> </ol>		
Risultati attesi:	Il dinosauro deve saltare nella pagina dell'host.		

Test Case:	TC-14	Nome:	Numero di giocatori dinamico
Riferimento:	REQ-10		
Descrizione:	Si deve poter giocare in più persone.		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Una partita creata ed aperta su uno schermo</li> <li>2. Almeno un utente creato</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo</li> <li>2. Premere sul pulsante "Connessione alla partita"</li> <li>3. Immettere il codice della partita</li> <li>4. Confermare</li> <li>5. Ripetere più volte, fino ad un massimo di 11</li> <li>6. Utilizzare anche almeno un utente loggato</li> <li>7. Provare a saltare/premere jump su i vari dispositivi</li> <li>8. Avviare la partita</li> </ol>		
Risultati attesi:	<ol style="list-style-type: none"> <li>1. Ad ogni connessione deve apparire un nuovo dinosauro</li> <li>2. Ogni dinosauro deve saltare quando il dispositivo che lo controlla ordina un salto</li> <li>3. Ci possono essere un massimo di 10 giocatori</li> <li>4. Quando la partita viene avviata tutti i dinosauri devono ancora poter saltare</li> </ol>		

Test Case:	TC-15	Nome:	Nome dei giocatori a schermo
Riferimento:	REQ-11		
Descrizione:	Quando ci si unisce ad una partita, oltre al dinosauro, appare anche il nickname		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Una partita creata ed aperta su uno schermo</li> <li>2. Almeno un utente creato</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo con 2 dispositivi</li> <li>2. Effettuare il login</li> <li>3. Premere sul pulsante "Connessione alla partita"</li> <li>4. Immettere il codice della partita</li> <li>5. Confermare</li> <li>6. Ripetere il test con un utente loggato e con più utenti contemporaneamente</li> </ol>		
Risultati attesi:	I nickname degli utenti devono apparire correttamente		

Test Case:	TC-16	Nome:	Classifica finale
Riferimento:	REQ-12		
Descrizione:	Quando una partita finisce deve apparire una classifica		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Una partita creata ed aperta su uno schermo</li> <li>2. Almeno un utente creato</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo con 2 dispositivi</li> <li>2. Effettuare il login</li> <li>3. Premere sul pulsante "Connessione alla partita"</li> <li>4. Immettere il codice della partita</li> <li>5. Confermare</li> <li>6. Giocare</li> </ol>		
Risultati attesi:	La classifica appare alla fine della partita		

Test Case:	TC-17	Nome:	Classifica finale ordinata
Riferimento:	REQ-12		
Descrizione:	Quando una partita finisce deve apparire una classifica ordinata per punteggio		
Prerequisiti:	<ol style="list-style-type: none"> <li>1. Una partita creata ed aperta su uno schermo</li> <li>2. Almeno un utente creato</li> </ol>		
Procedura:	<ol style="list-style-type: none"> <li>1. Raggiungere il sito dell'applicativo con 2 dispositivi</li> <li>2. Effettuare il login</li> <li>3. Premere sul pulsante "Connessione alla partita"</li> <li>4. Immettere il codice della partita</li> <li>5. Confermare</li> </ol>		



	6. Unirsi con un secondo utente (loggato o normale) 7. Giocare
Risultati attesi:	La classifica appare alla fine della partita ed è ordinata

Test Case:	TC-18	Nome:	Medaglie nella classifica
Riferimento:	REQ-12		
Descrizione:	Quando una partita finisce deve apparire una classifica con una medaglia per il primo classificato		
Prerequisiti:	1. Una partita creata ed aperta su uno schermo		
Procedura:	1. Raggiungere il sito dell'applicativo con 2 dispositivi 2. Premere sul pulsante "Connessione alla partita" 3. Immettere il codice della partita 4. Confermare 5. Unirsi con un secondo utente (loggato o normale) 6. Giocare		
Risultati attesi:	La classifica appare alla fine e solo il primo classificato ha una medaglia		

Test Case:	TC-19	Nome:	Medaglie assegnate al vincitore
Riferimento:	REQ-12		
Descrizione:	Quando una partita finisce la medaglia viene assegnata al vincitore		
Prerequisiti:	1. Un utente creato con almeno una medaglia 2. Una partita creata ed aperta su uno schermo		
Procedura:	1. Raggiungere il sito dell'applicativo 2. Premere sul pulsante "Connessione alla partita" 3. Immettere il codice della partita 4. Confermare 5. Avviare la partita dall'host 6. Giocare 7. Quando la partita finisce verificare la presenza della medaglia nella classifica e recarsi nella pagina dell'utente 8. Premere su "Bacheca"		
Risultati attesi:	La medaglia appena vinta deve essere nella bacheca		

## 5.2 Risultati test

Test Case	Risultato	Commenti
TC-01	Passato	
TC-02	Passato	
TC-03	Passato	
TC-04	Passato	
TC-05	Passato	
TC-06	Passato	
TC-07	Passato	
TC-08	Passato	
TC-09	Passato	
TC-10	Passato	
TC-11	Passato	
TC-12	Passato	
TC-13	Passato	
TC-14	Passato	
TC-15	Passato	
TC-16	Passato	
TC-17	Passato	
TC-18	Passato	
TC-19	Passato	

## 5.3 Mancanze/limitazioni conosciute

Pensiamo che la mancanza principale del nostro progetto sia la struttura del codice. Pur essendo stato compiuto un buon lavoro di refactoring, che ci ha permesso di portare a termine questo progetto con successo, abbiamo purtroppo individuato, cammin facendo, una mancanza riguardante non tanto il refactor in sé, quanto nella modalità dello stesso.

Un approccio a classi, infatti, dove ogni parte del gioco sarebbe stata rappresentata da un oggetto e non da un array, avrebbe semplificato e abbellito ulteriormente il codice.

Inoltre, un simile approccio avrebbe forse risolto anche il problema del ghost. Problema che abbiamo peraltro già esplicitato in maniera esauriente nel capitolo 4. Ciò, pur non essendo a conti fatti, una vera e propria limitazione, è sicuramente un punto a sfavore di questo progetto.

A livello di organizzazione procedurale, con il senno di poi, ci siamo resi conto che la stima del tempo occorrente per implementare alcune parti era sbagliata, per eccesso o per difetto. Quando è capitato, il buon clima collaborativo che si è creato nel gruppo, ci ha permesso di aiutarci a vicenda.

## 6 Consuntivo

Per questo capitolo alleghiamo il nostro Gantt consuntivo.

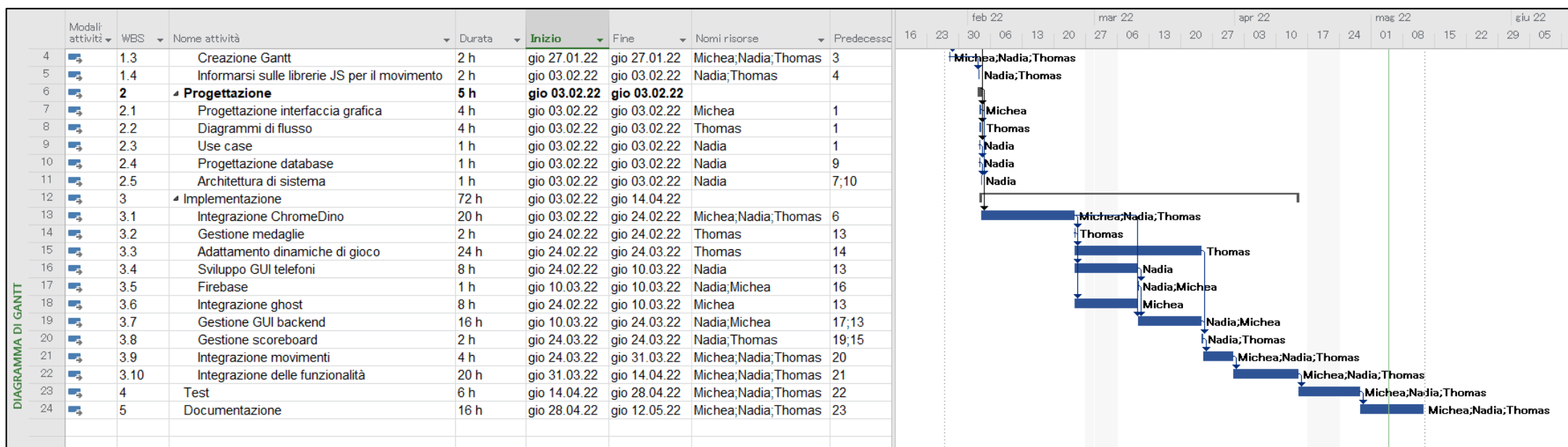


Figura 25 Gantt consuntivo

## 7 Conclusioni

A bocce ferme, possiamo dirci moderatamente soddisfatti del nostro progetto. Non solo perché ci siamo impegnati a rispettare il più fedelmente possibile le specifiche, ma anche perché pensiamo di aver realizzato, nel complesso, un buon gioco. Per due di noi era la prima esperienza di lavoro in gruppo su un progetto semestrale: si trattava di evolvere da piccoli progetti svolti a coppie o in gruppo in passato, ad un progetto maggiormente complesso ed articolato, che avesse uno sviluppo puntuale sotto più aspetti, come progettazione, codice, documentazione, ecc.

Per quanto riguarda la modalità e il clima di lavoro nel gruppo, riteniamo che sia stato molto positivo. Non abbiamo avuto momenti di tensione e gli scambi di idee sono stati costruttivi e orientati al raggiungimento dell'obiettivo comune. Cosa non scontata perché non avevamo mai lavorato insieme. Il progetto ci ha dunque permesso di affinare le nostre abilità di collaborazione in modo piuttosto significativo.

Il progetto ci ha sicuramente permesso di apprendere e mettere in pratica nuove importanti nozioni. Questo non solo per quanto riguarda JavaScript. Abbiamo imparato anche ad usare il framework Phaser e due di noi hanno avuto anche la possibilità di conoscere Firebase. L'utilizzo di strumenti come il Gantt e GitHub ci ha aiutato a dividerci i compiti e a condividere il nostro lavoro cammin facendo.

Un altro aspetto che riteniamo sia stato importante per la buona riuscita del progetto è l'aver messo in comune le nostre conoscenze, in modo da aiutarci l'un l'altro e sopperire così a eventuali lacune che potevano sorgere. Per esempio, Nadia nel progetto ha aiutato per la parte di comunicazione con il database Firebase: avendo lei già sviluppato un progetto con questo sistema è stato più semplice implementare il codice e apprendere nuove nozioni. Thomas ha portato un'ottima conoscenza di JavaScript che, unita a molta pazienza, ci ha permesso di migliorare e convertire il codice prodotto in precedenza da Manuel Grosso. Michea, per contro, ha sicuramente aiutato a rendere tutta la parte di documentazione più chiara ed efficace, grazie alla maggiore esperienza accumulata lo scorso anno.

Un aspetto che ha molto interessato tutti è il fatto di non essersi dovuti confrontare con uno sviluppo di progetto dall'inizio, bensì di aver potuto usare come base il progetto di Manuel, anche se prima di svilupparlo si è comprensibilmente presentata una fase di comprensione e di ristrutturazione del codice. Questa fase è stata una delle più complesse, in quanto comprendere del codice scritto da un'altra persona, con una documentazione poco consistente, è stato molto difficile. In particolare, perché l'interezza del codice era stata sviluppata con un approccio *Hard coded*. Questo ha aumentato la difficoltà.

Come già accennato, sarebbe stato bello e utile essere più performanti nella fase di progettazione, in modo da non incappare nel problema del ghost. Ma lo riteniamo comunque un errore "utile", nel senso che eviteremo di compierlo ancora in futuro.

Infine, esprimiamo il nostro sincero ringraziamento al prof. Petrini, che lungo tutto l'arco del progetto ha vegliato su di noi, fornendoci utili spunti di riflessione quando incontravamo dei problemi e spingendoci a cercare delle soluzioni efficaci. Senza la sua paziente supervisione il progetto non avrebbe avuto uno sbocco così positivo. Grazie di cuore.

### 7.1 Sviluppi futuri

Per quanto riguarda gli sviluppi futuri di questo progetto, abbiamo identificato alcune opzioni:

La prima non riguarda una funzione o aggiunta al progetto in sé, bensì il codice di base. Come detto, nel refactor abbiamo deciso di utilizzare degli array per rendere l'applicazione multiplayer, ma un migliore approccio sarebbe stato quello di utilizzare le classi. È quindi questa la prima miglioria che può essere fatta al progetto; ciò permetterebbe non solo di avere un codice molto più comprensibile, ma semplificherebbe anche future modifiche.

Un altro sviluppo è da ricercare nell'implementazione della funzione *ghost*. Ci è dispiaciuto molto infatti non poterla realizzare, in quanto ritenevamo molto interessante la possibilità di assistere da remoto ad una partita come spettatore. Tuttavia, se ciò non risultasse comunque possibile tramite un'implementazione ad oggetti o tramite migliorie al codice, sarebbe necessario cambiare la piattaforma di base del database, ovvero Firebase. Nel caso si prendesse questa strada bisognerebbe ripensare l'applicazione quasi da zero, modificando tutte le logiche per il passaggio e l'ottenimento dei dati.

Ulteriori sviluppi futuri ipotizzabili, e che riguardano maggiormente il progetto nel suo aspetto principale, sono la possibilità di personalizzare il proprio dinosauro, implementando l'inserimento di *skin* o la possibilità di utilizzare più colori per il proprio personaggio. Inoltre si potrebbe pensare alla creazione di oggetti esterni da aggiungere, come cappelli o occhiali, fruibili solo da utenti registrati che hanno raggiunto un determinato punteggio.

Ispirandoci invece al vero Chrome Dino, sono molteplici le modifiche/aggiunte che si potrebbero applicare: ad esempio diverse tipologie di cactus, diversi in altezza e che compaiono a coppie. Si potrebbero inserire anche gli uccelli che sono presenti nel gioco originale, sviluppando la possibilità di potersi abbassare, in modo da variare i movimenti che l'utente deve fare. Infine, dal punto di vista temporale, si potrebbe implementare lo scorrimento del tempo, in modo che ogni tanto il gioco passi dal giorno alla notte.

## **7.2 Considerazioni personali**

### **7.2.1 Michea**

Personalmente la cosa che più ho apprezzato nel corso del progetto è stata la sfida, per me praticamente nuova, di creare un gioco. Pur avendo fatto ormai -nel bene e nel male- ormai 3 progetti, non mi era mai capitato di dover sviluppare un gioco. Sono sempre stato convinto che lo sviluppo dei giochi fosse una branca piuttosto complessa e a me non vicina, dato che non sono mai stato un videogiocatore, ma sviluppare il progetto mi ha molto appassionato. Sono abbastanza soddisfatto del risultato finale e anche del percorso fatto con i miei compagni. La buona armonia presente nel gruppo ci ha aiutato a pianificare le varie fasi senza troppe difficoltà, così come a dividerci il lavoro da fare in modo piuttosto efficace, rispettando al contempo le varie individualità e punti di forza all'interno del gruppo.

Spero che questo progetto, prima o poi, venga riproposto ad altri allievi, sia in ottica di un possibile miglioramento, sia come progetto da sviluppare da zero.

Un ulteriore motivo per cui ho apprezzato molto questo progetto, è stato il fatto che dopo un anno e mezzo sono tornato ad occuparmi di un progetto in JavaScript, linguaggio che mi è sempre piaciuto e che nell'ultimo periodo avevo purtroppo dovuto abbandonare. Spero quindi che questo possa essere, oltre che un arricchimento per le mie conoscenze, un punto di ripartenza per tornare a masticare JavaScript e dintorni.

### **7.2.2 Nadia**


### **7.2.3 Thomas**

A mio parere, Il progetto a noi assegnato, è stato molto interessante. I mezzi che abbiamo utilizzato per la realizzazione del progetto come Phaser, Firebase, non li conoscevo e questo progetto mi ha aiutato a comprenderne i vantaggi e le funzionalità.

Durante il progetto personalmente mi sono occupato principalmente della parte legata alle dinamiche di gioco e al refactoring del codice di Manuel Grosso. Questo compito assieme alla gestione della leaderboard e delle medaglie, mi ha fatto scoprire e comprendere Phaser e aumentare le mie conoscenze di JavaScript. Inoltre mi ha anche fatto ragionare molto sulla struttura del codice e dell'importanza del costruire un codice ordinato.

Anche se non è stato il mio compito principale ho avuto l'occasione di lavorare con un database ad oggetti, Firebase, e utilizzarlo interfacciandomi con esso. Questo mi ha aiutato a capirne le potenzialità e l'utilità, anche se sarebbe stato ancora più comodo utilizzando gli oggetti al posto degli array nel codice di gioco.

In conclusione sono contento del risultato finale che siamo riusciti ad ottenere, abbiamo rispettato praticamente tutte le richieste e siamo riusciti senza troppi aiuti esterni a risolvere i problemi più complessi, tutto grazie, a mio parere, ad un ottimo team working.

	<b>SAMT – Sezione Informatica</b>	Pagina 46 di 46
	<b>Esempio di documentazione</b>	

## 8 Sitografia

---

- <http://standards.ieee.org/guides/style/section7.html>, *IEEE Standards Style Manual*, 07-06-2008.

## 9 Allegati

---

- Diari di lavoro
- QdC
- Prodotto