

## 1. Introduzione

- Informazioni sul progetto
- Abstract
- Scopo

## 2. Analisi

- Analisi del dominio
- Analisi dei mezzi
- Analisi e specifica dei requisiti
- Use case
- Pianificazione

## 3. Progettazione

- Design dell'architettura del sistema
- Design dei dati e database
- Design delle interfacce
- Design procedurale

## 4. Implementazione

## 5. Test

- Protocollo di test
- Risultati test
- Mancanze/limitazioni conosciute

## 6. Consuntivo

## 7. Conclusioni

- Sviluppi futuri
- Considerazioni personali

## 8. Bibliografia

- Sitografia

## 9. Allegati

# Introduzione

## Informazioni sul progetto

- Allievi coinvolti nel progetto: Nathan Ferrari, Andrea Masciocchi, Xavier Horisberger
- Classe: I3AA Scuola Arti e Mestieri Trevano, sezione Informatica.
- Docenti responsabili: Luca Muggiasca.
- Data inizio: 27 gennaio 2022
- Data di fine: 05 maggio 2022.

## Abstract

*Have you ever wanted to play a simple game with your friends to have some fun, but didn't know what game to play? LANIceHockey enables you to play a simplified version of ice hockey with your friends, and there are no limits to how many players can enter the match, everyone can play. You all connect yourselves to the same network, a main computer with a sufficiently big monitor will host the game, on it there will be the ice hockey field, meanwhile the controller to interact with your avatar will be on your phone. Now you and your friends can easily play together and have a good time.*

## Scopo

Lo scopo di questo progetto è di creare un videogioco di hockey su ghiaccio, il gioco è ospitabile in una LAN, per giocare bisogna quindi collegarsi alla stessa rete della macchina server, il campo sarà sul monitor della macchina che ospita la partita, e il controller sarà sui telefoni delle persone che si uniscono alla partita. Non ci sono limiti su quanti giocatori possono entrare in partita, non ci sono presenti neanche le regole che sono normalmente presenti in una partita di hockey. Il controller permette all'utente di muoversi, tirare, scattare e cambiare squadra. È anche presente una classifica dove si vedono i 10 giocatori che hanno segnato più goal.

## Analisi

### Analisi del dominio

Il prodotto può essere utilizzato su qualsiasi piattaforma, ma bisogna essere collegati ad un router appositamente usato per questo progetto. Una volta connessi al router si cerca il sito apposito, una volta sul sito del prodotto si può scegliere se essere l'host o se essere un giocatore. Può esserci un solo host e i giocatori possono connettersi alla partita solo se è già presente un host. Per quanto riguarda il client è consigliabile utilizzare il browser firefox, che dopo vari test abbiamo notato avere una velocità di reazione nettamente maggiore rispetto ad altri browsers. Come abbiamo specificato nelle limitazioni, siamo costretti a rimuovere il dito dal joystick per tirare, ed è ulteriormente limitante una velocità di reazione bassa da parte del browser.

### Analisi e specifica dei requisiti

<b>ID:</b>	<b>REQ-001</b>
<b>Nome</b>	Rappresentazione semi-reale di una partita di Hockey
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-002</b>

<b>ID:</b>	<b>REQ-002</b>
<b>Nome</b>	La partita non ha un limite di giocatori
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-003</b>
<b>Nome</b>	Gli utenti devono potersi collegare alla partita tramite telefono in LAN
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-004</b>
<b>Nome</b>	Gli utenti devono poter muovere il loro giocatore in LAN
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-005</b>
<b>Nome</b>	Gli utenti devono poter tirare in porta
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-006</b>
<b>Nome</b>	Devono esserci due squadre (verdi e gialli)
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-007</b>
<b>Nome</b>	Gli utenti devono poter dare un nome al loro giocatore
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-008</b>
<b>Nome</b>	Una volta assegnato il nome non può più essere cambiato
<b>Priorità</b>	3
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-009</b>
<b>Nome</b>	In alto a destra ci sarà una classifica con i 10 giocatori che hanno fatto più goal
<b>Priorità</b>	2

<b>ID:</b>	<b>REQ-009</b>
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-010</b>
<b>Nome</b>	I telefoni degli utenti agiranno solo da controller
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-011</b>
<b>Nome</b>	Il campo di gioco viene visualizzato su un monitor a parte (non sugli schermi dei telefoni)
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-012</b>
<b>Nome</b>	Il campo da gioco è visto dall'alto
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-013</b>
<b>Nome</b>	La partita è infinita
<b>Priorità</b>	3
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-014</b>
<b>Nome</b>	Gli utenti devono potersi rubare il disco
<b>Priorità</b>	2
<b>Versione</b>	1.0
<b>ID:</b>	<b>REQ-015</b>
<b>Nome</b>	Il disco rimbalza contro le pareti
<b>Priorità</b>	2
<b>Versione</b>	1.0

### Spiegazione elementi tabella dei requisiti:

**ID:** identificativo univoco del requisito

**Nome:** breve descrizione del requisito

**Priorità:** indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente. Ad esempio poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un

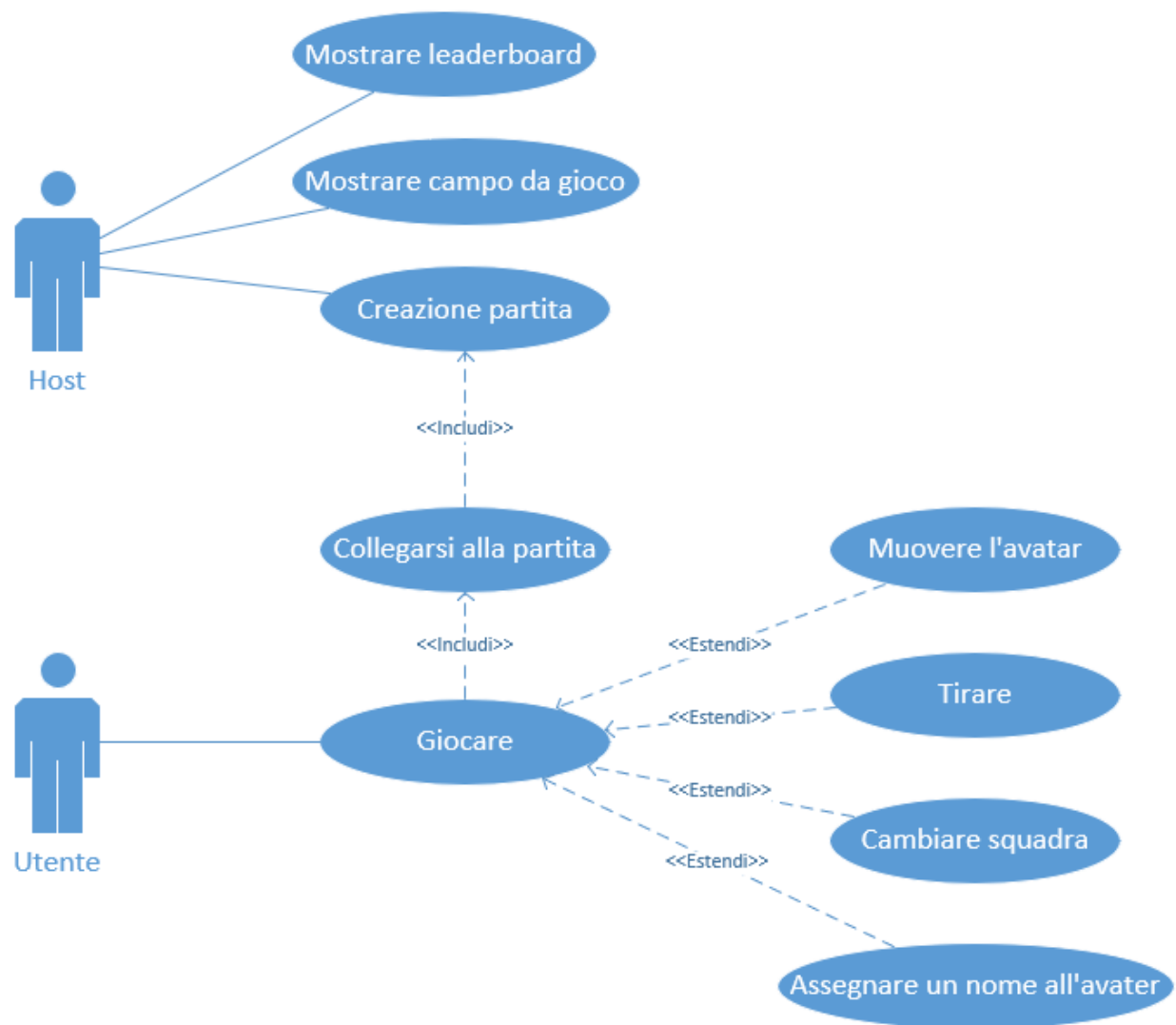
database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione:** indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l'ultima versione, mentre le vecchie dovranno essere inserite nei diari.

## Use case

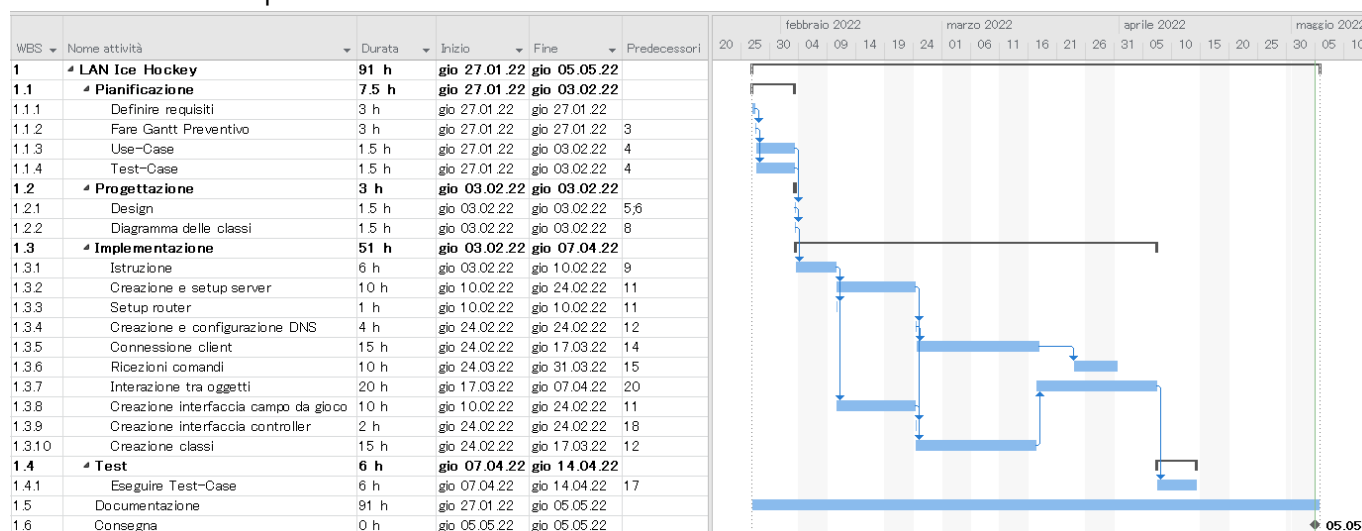
Ecco il nostro Use Case:



### Use case

## Pianificazione

Abbiamo scelto una pianificazione waterfall:



## Gantt preventivo

## Analisi dei mezzi

### Hardware

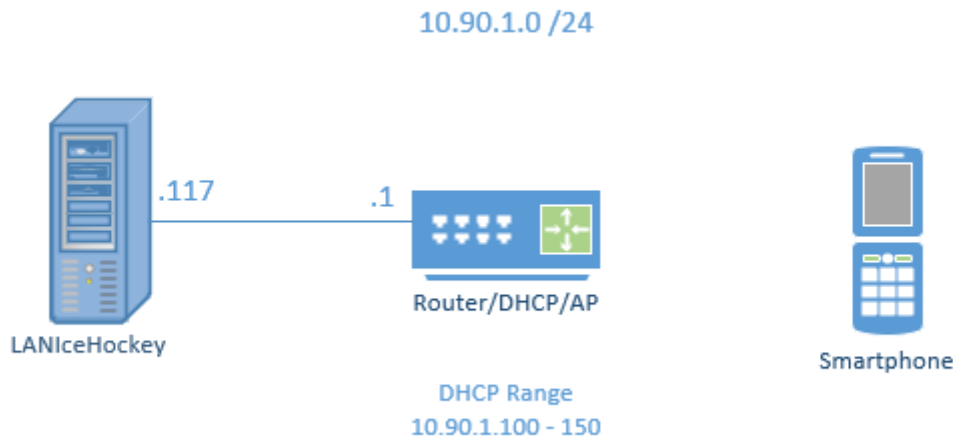
- Computer1:
  - Processore: Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz
  - RAM: 16 GB
- Computer2:
  - Processore: Intel Xeon (R) CPU E3-1240 V2 @ 3.40GHz x 8
  - RAM: 8 GB
- Router:
  - Model: Linksys EA6350
  - Wifi name: Linksys10206
  - Wifi password: LANIceHockey
  - Ip address: 10.90.1.1
- Switch:
  - Model: Allied Telesis AT-FS708

### Software

- OS Computer1: Windows 10 Enterprise versione 20H2 (build SO 19042.1586)
- OS Computer2: Ubuntu 21.10
- Visual Studio Code versione 1.58
- Notepad++ v8.1.4
- Nipplejs v0.9.0
- Nodejs v12.22.5
- Phaser v3.55.2

## Progettazione

### Design dell'architettura del sistema



Rete:

Diagramma di rete

Diagramma delle classi che compongono il nostro progetto:

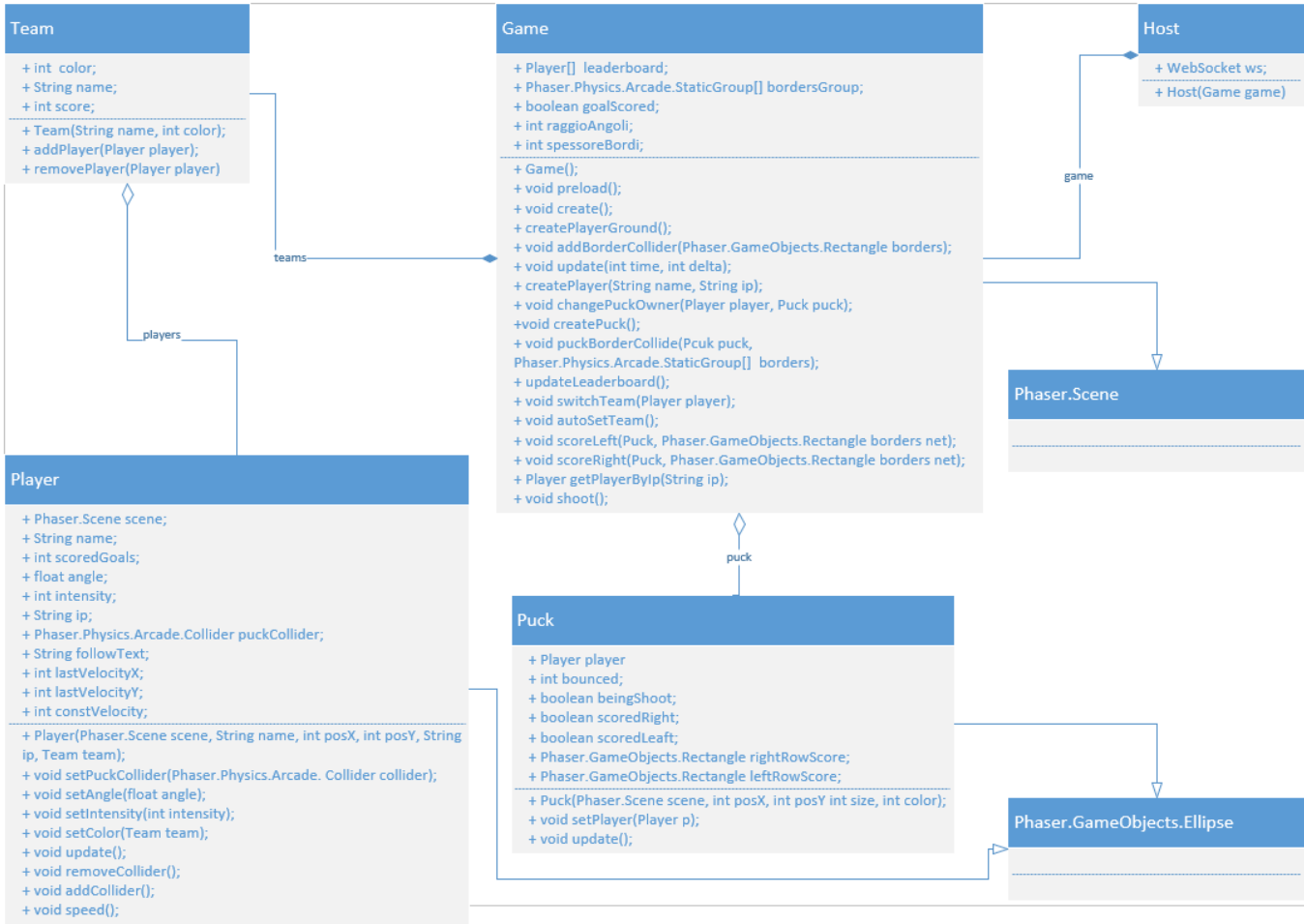
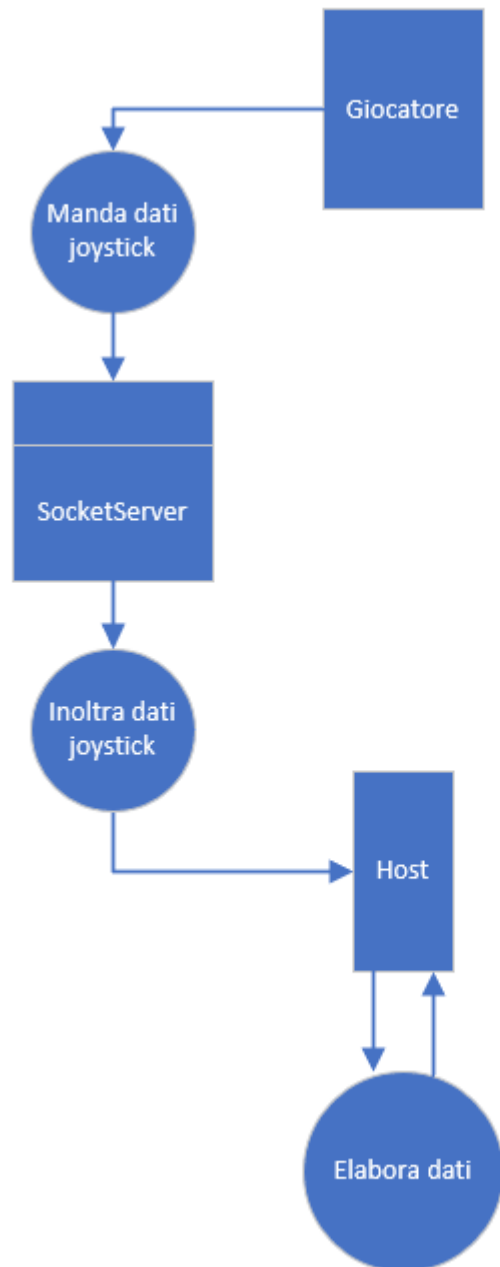


Diagramma delle classi

Diagramma di flusso dei dati:



## DFD

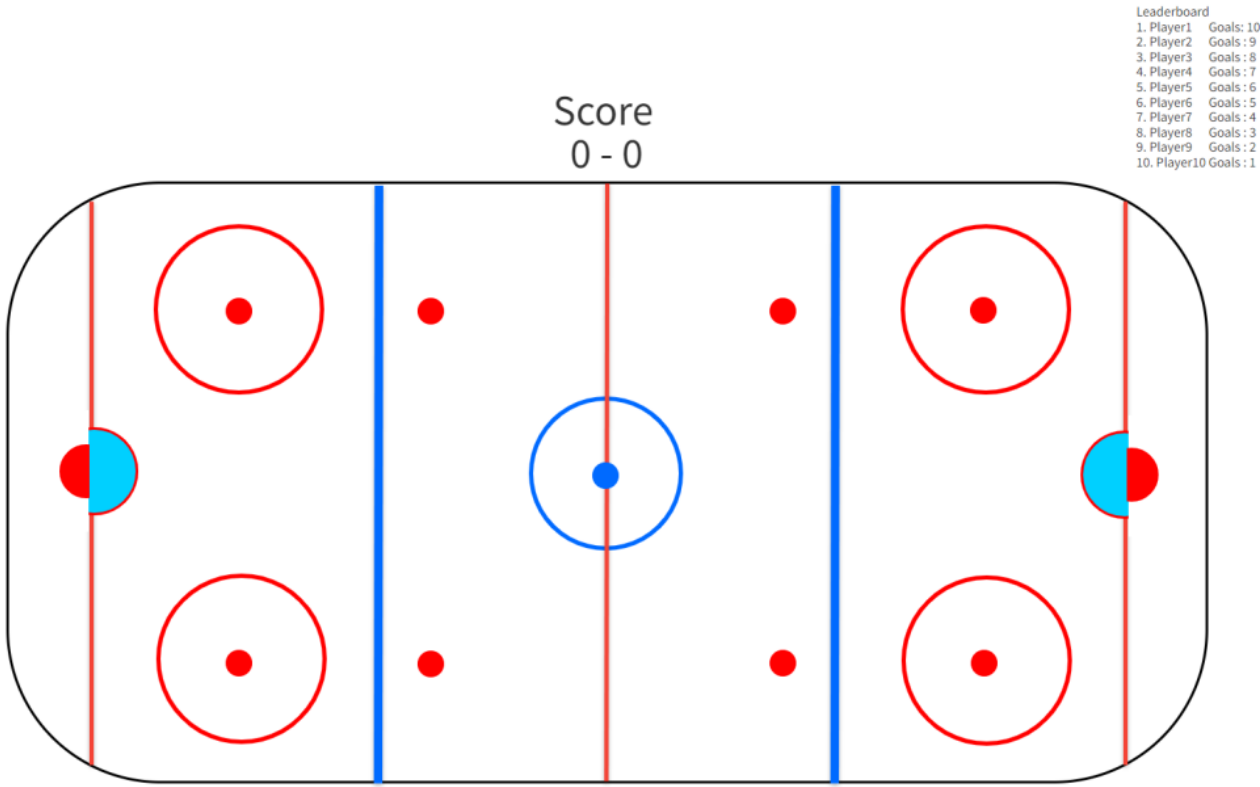
### Design dei dati e database

Descrizione delle strutture di dati utilizzate dal programma in base agli attributi e le relazioni degli oggetti in uso.

### Design delle interfacce

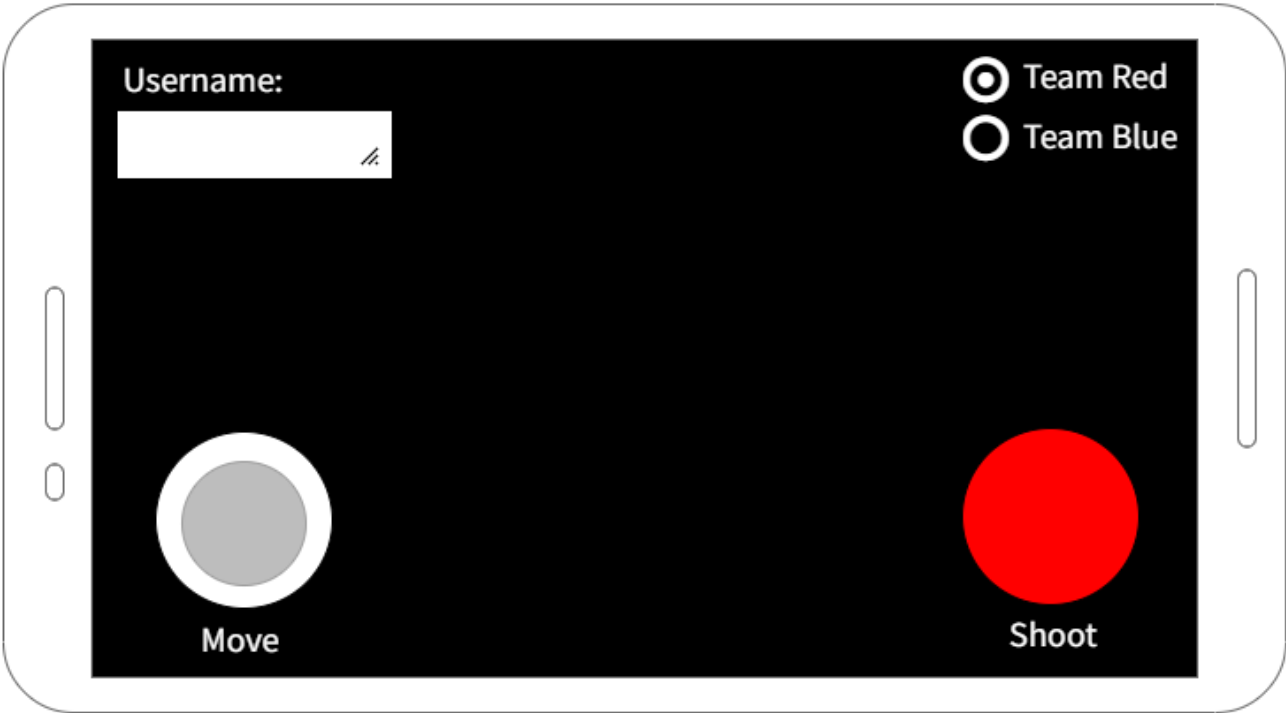


Design dell'interfaccia del campo da gioco:



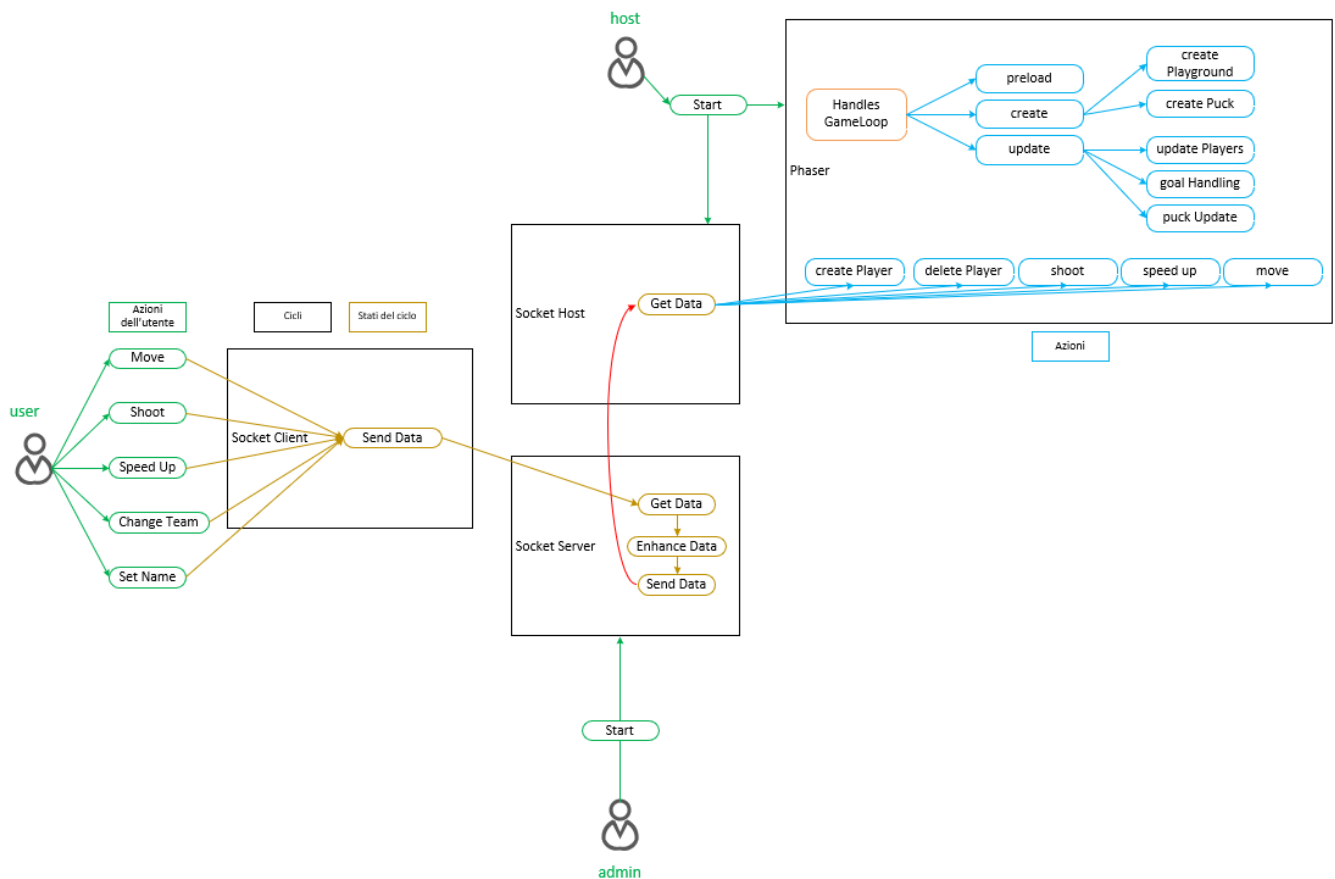
Campo da gioco

Design dell'interfaccia del controller:



Controller

Design procedurale



## Design Procedurale

# Implementazione

## Requisiti

Avere installato nodejs ed avere il comando 'node' nelle variabili d'ambiente

## Classi

### Player

Questa classe rappresenta un giocatore, estende Phaser.GameObjects.Ellipse, che gli permette di essere un GameObject con forma di ellisse.

#### Attributi:

- scene: è la scena a cui appartiene, serve al player per auto-disegnarsi nel campo da gioco.
- name: è il nome con cui il giocatore decide di giocare.
- scoredGoals: rappresenta il numero di reti segnate dal giocatore.
- angle: è la direzione in cui il giocatore punta il joystick, serve a muovere il player nella direzione desiderata.
- intensity: rappresenta la forza con cui il player si sposta, ovvero la velocità. Questa viene calcolata rispetto a quanto il joystick è spostato dalla posizione di riposo (il centro).
- ip: è l'indirizzo di rete del dispositivo corrispondente al giocatore.
- followText: contiene l'oggetto di phaser che serve per rendere visibile a schermo il nome.

- lastVelocityX: si salva l'ultima componente orizzontale della velocità e a calcolare la direzione del puck una volta tirato.
- puckCollider: è l'attributo che contiene il collider con il puck, ovvero ciò che permette al player di raccogliere il disco.
- lastVelocityY: si salva l'ultima componente verticale della velocità e a calcolare la direzione del puck una volta tirato.
- constVelocity: è moltiplicatore della forza ricevuta dal joystick, settata di default a 4.

**Metodi:**

- constructor(scene, name, posX, posY, ip, team): questo metodo è il costruttore, invoca il supercostruttore della classe ellipse, setta la posizione di partenza, nome, indirizzo ip e crea il testo con scritto il nome che segue il giocatore. Abilita la fisica e auto-disegna all'interno del campo sia il player, che il testo che lo segue.
- setAngle(angle): questo metodo riceve l'angolo passato come argomento e lo assegna all'attributo angle.
- setIntensity(intensity): questo metodo riceve l'intensità e la assegna all'attributo intensity.
- setColor(team): questo metodo riceve un team e colora il player del colore del team.
- addCollider(): questo metodo serve ad attivare un collider tra il puck e il player.
- setPuckCollider(collider): questo assegna il collider ricevuto all'attributo puckCollider.
- removeCollider(): questo metodo serve per disattivare il collider e dunque fare in modo che esso non interferisca con il lancio del disco, se questo metodo non venisse eseguito quando necessario, il player proverebbe a tirare ma riacquisirebbe immediatamente il puck.
- update(): questo metodo viene invocato da noi tramite il metodo eseguito in automatico di Phaser, omonimo e presente in game, e permettere di fare delle azioni ad ogni ciclo di gioco. Viene utilizzato per calcolare il moto del player e aggiorniamo la posizione per permettere al testo rappresentante il nome di seguire il player. Aggiorniamo anche lastVelocityX e lastVelocityY.
- speed(): Questo metodo serve a duplicare la velocità di movimento per un tempo limitato, è invocato dal giocatore tramite il tasto "speed up".

**Team**

Questa classe rappresenta un team

**Attributi:**

- players: questo attributo contiene una lista di tutti i player fanno parte del team.
- name: contiene il nome del team.
- color: contiene il colore del team in esadecimale, serve per colorare i player.
- score: serve per sapere quanti goal ha fatto il team.

**Metodi:**

- constructor(name, color): una volta passato il nome e il colore del team questo viene generato, ovviamene senza players.
- addPlayer(player): questo metodo serve ad aggiungere un player alla lista di player.
- removePlayer(player): questo metodo serve a rimuovere un player dalla lista di player.

## Puck

La classe puck rappresenta il disco della partita. Estende Phaser.GameObjects.Ellipse in modo da disegnarlo con ellisse.

### Attributi:

- player: contiene il giocatore che ha il possesso del disco.
- beingShoot: se è stato tirato contiene true, altrimenti false.
- bounced: utilizzato per contare su quante pareti è rimbalzato da quando viene lanciato.
- scoredRight && scoredLeft: sono gli attributi che servono a sapere in che porta si ha segnato, e servono a triggerare l'evento di riposizionamento degli elementi nel campo, oltre all'aggiornamento del punteggio.
- rightRowScore && leftRowScore: contengono le istanze delle linee di porta.

### Metodi:

- constructor(scene, posX, posY, size = 35, color = 0x202020): il costruttore serve ad istanziare un nuovo puck, viene richiamato il supercostruttore di ellipse. Si abilita la fisica dell'oggetto e si auto-genera nella scena.
- setPlayer(player): questo metodo serve a riassegnare il player che possiede il puck, player è il nuovo player.
- update(): questo metodo è richiamato da noi tramite il metodo automatico di Phaser in automatico ad ogni ciclo di gioco, serve per posizionare il puck al centro del player che lo possiede, ovviamente se è posseduto da un player.

## Game

La classe Game rappresenta e gestisce la partita, estende la classe Phaser.Scene, infatti è ciò che viene raffigurato a schermo. Vengono importate le classi che abbiamo scritto: Player, Team e Puck, la classe Host da hostconnection e delle costanti da main.js che servono per conoscere altezza e larghezza del campo.

### Attributi:

- teams: questo attributo contiene un array con i team della partita, che sono 2.
- bordersGroup: contiene il gruppo di bordi di tipo Phaser.Physics.Arcade.StaticBorders, che serve a delimitare il campo e creare i collider con le porte.
- puck: contiene il puck della partita.
- raggioAngoli: serve a disegnare il campo, è quanto è lungo il raggio delle curve degli angoli.(viene anche utilizzato per altre formule nei disegni).
- spessoreBordi: serve a disegnare il campo, è quanto è spesso il bordo.(viene anche utilizzato per altre formule nei disegni).

### Metodi:

- constructor(): serve ad istanziare una nuova partita, richiama il costruttore della superclasse, crea due teams di default (in futuro si potrebbero rendere customizzabili) e infine crea l'Host della partita per la connessione.

- `preload()`: viene richiamato da Phaser appena prima del `create`, serve a preimpostare delle cose, noi preimpostiamo il canvas in una posizione più facile da raggiungere, non è necessario ma è più comodo da usare.
- `create()`: viene richiamato da Phaser alla creazione dell'oggetto, lo utilizziamo per disegnare il campo e creare il puck invocando i metodi appositi.
- `createPlayGround()`: viene incocato da `create()`, e serve a disegnare gli elementi del campo, al suo interno richiamiamo anche il metodo che permette di aggiungere i collider ai bordi e alle porte.
- `addBorderCollider(borders)`: una volta passato l'array di bordi attiva la fisica e li aggiunge alla lista `bordersGroup` e assegna a questo gruppo il collider a tutti i player a quel momento generati, teoricamente nessuno.
- `update(time, delta)`: questo metodo viene richiamato da Phaser in automatico ad ogni ciclo di gioco, dentro questo metodo richiamiamo l'`update` di tutti i player e del puck. inoltre controlla i flag del puck per assegnare eventuali goal e se ciò succede bisogna riportare la partita alla situazione iniziale ovvero puck al centro e giocatori nella loro metà campo.
- `createPlayer(name, ip)`: dato nome e ip, questo metodo crea un nuovo player e lo assegna autonomamente al team con meno giocatori. Inoltre assegna il collider con i bordi.
- `changePuckOwner(player, puck)`: dato il puck e il player riassegna il player del puck e setta i valori all'interno del puck in maniera che risulti che il nuovo player posseda il disco. Viene riaggiunto il collider del disco al player che possedeva precedentemente il puck dopo pochi millisecondi, per permettere una dinamica di scambio del puck più pulita.
- `createPuck()`: permette di creare il puck all'interno del game, genera anche le righe delle porte ed assegna gli overlap che richiamano la funzione di goal. Aggiunge i collider dei bordi e ogni volta che ci rimbalza addosso la velocità dimezza.
- `puckBorderCollide(puck, borders)`: questo metodo permette di contare i rimbalzi del puck cambiandone l'attributo. Questa funzione viene invocata dall'evento `collide` tra puck e bordi.
- `updateLeaderboard()`: serve ad aggiornare la leaderboard e il risultato. Per farlo va a cercare nell'HTML gli elementi predisposti, poi li riempie.
- `switchTeam(player)`: dato un player, cambia il team di esso, si controlla a che team appartiene, da quello lo rimuove e poi lo aggiunge nell'altro.
- `autoSetTeam()`: questo metodo di aiuto serve a scoprire quale team ha meno player, serve alla creazione dei nuovi player.
- `scoreLeft(puck, net)`: invocata dall'evento di quando si segna nella rete di sinistra. Serve a settare il flag `scoredLeft` a true, ciò permette al metodo `update` di far segnare il team di destra.
- `scoreRight(puck, net)`: invocata dall'evento di quando si segna nella rete di destra. Serve a settare il flag `scoredRight` a true, ciò permette al metodo `update` di far segnare il team di sinistra.
- `getPlayerByIp(ip)`: dato l'indirizzo ip di un player viene cercato all'interno di tutti i team e lo ritorna.
- `shoot()`: permette di tirare il puck solo se non è già stato tirato, gli viene impostata la velocità del player moltiplicata per due.

## Scripts

### Main

Questo script si occupa di creare il `Phaser.Game`.

### Variabili

- config: la variabile contiene le impostazioni di phaser, con le quali viene istanziato il Phaser.Game.
- SET\_WIDTH: costante che determina la larghezza del campo da gioco.
- SET\_HEIGHT: costante che determina l'altezza del campo da gioco.
- game: contiene la nuova istanza di Phaser.Game, principale motivo dell'esistenza di questo script.

## Connection

### Socket

Questa classe è il socket server, lo si avvia all'inizio per ricevere e mandare i dati necessari.

#### Variabili

- wss: è un'istanza di WebSocket.Server
- clients: è un array che contiene tutti i client connessi
- host: contiene il WebSocket di host, serve per avere il riferimento a cui mandare i dati

#### Eventi

- wss.on('connection', ...): ogni volta che un client si connette attiva gli eventListener per quel client
  - ws.on('message', ...): ogni volta che il socket server riceve un messaggio, se è un segnale di connessione inoltra all'host il segnale e lo mette nell'array di client, nel caso richiedesse la connessione host salviamo il websocket di host, nel caso mandasse un messaggio generico lo inoltra direttamente all'host allegandoci anche l'ip del client.
  - ws.on('close', ...): manda all'host il segnale di disconnessione.

#### Metodi

- removeWithWSFromClients(ws): rimuove dall'array clients il client corrispondente all'WebSocket passato tramite argomento.

### ClientConnection

Questa classe è importante per la comunicazione tra client e host, manda le informazioni al socket server.

#### Variabili

- manager: crea il joystick in base alle impostazioni definite nella variabile "options"

#### Eventi

- manager.on('move', ...): ascolta il movimento del joystick e manda i dettagli importanti per il movimento del player al socket server.
- manager.on('end', ...): ascolta quando il player smette di muovere il joystick, e manda il segnale di stop al socket server.
- ws.addEventListener('open', ...): quando il client si connette manda al socket server la richiesta di connessione, se il nome non è valido lo rimanda alla pagina index.

## Metodi

- `teamChanged()`: quando il giocatore vuole cambiare team, mandiamo al socket server il segnale.
- `shoot()` : quando il giocatore clicca il tasto per lanciare il disco, mandiamo al socket server il segnale.
- `speed()` : quando il giocatore clicca il tasto per scattare, mandiamo il segnale al socket server, disabilitiamo il tasto per scattare e usiamo una funzione asincrona che riattiva il tasto dopo un tempo predefinito.

## HostConnection

Questa classe riceve i dati dal socket server e si occupa di renderizzare il gioco.

## Variabil

- `ws`: è un'istanza di `WebSocket`, serve per connettersi al socket server e comunicare con i client e il server stesso.

## Eventi

- `ws.addEventListener("open", ...)`: alla connessione dell'host manda un messaggio al socket server per farsi riconoscere come host e crea il listener per la ricezione di messaggi.
  - `ws.onmessage`: alla ricezione dei messaggi da parte del socket (che ha ricevuto i segnali da `clientConnection`) controlla il tipo di segnale e agisce in base all'azione desiderata da parte del player.

## Interfacce

### Client

Contiene un joystick importato dalla libreria `NippleJS`, due bottoni per lanciare il puck e per attivare lo scatto e un bottone per cambiare team.

### Host

Contiene il campo da gioco aggiornato in tempo reale da `Phaser`, una leaderboard contenente i giocatori ordinati per goal fatti, il punteggio delle due squadre e un tasto per resettare la posizione del puck in caso di problemi.

## Test

Protocollo di test

Test Case	TC-001
Nome	Tutti gli elementi sommati creano una rappresentazione simile alla realtà
Riferimento	REQ-001
Descrizione	Provare il gioco e controllare che tutti gli elementi assieme generino una specie di partita di hockey.

<b>Test Case</b>	<b>TC-001</b>
<b>Prerequisiti</b>	<ul style="list-style-type: none"> <li>• TC-003</li> <li>• TC-004</li> <li>• TC-005</li> <li>• TC-006</li> <li>• TC-009</li> </ul>
<b>Procedura</b>	Accedere al gioco in 4 o più e provare a fare una partita.
<b>Risultati attesi</b>	Il gioco ricorda una partita di hockey.
<b>Test Case</b>	<b>TC-002</b>
<b>Nome</b>	Nessun limite di giocatori
<b>Riferimento</b>	REQ-002
<b>Descrizione</b>	Bisogna provare a giocare con diversi dispositivi in maniera da controllare che non ci siano limiti nelle persone che giocano.
<b>Prerequisiti</b>	TC-001
<b>Procedura</b>	Collegheremo più dispositivi alla partita.
<b>Risultati attesi</b>	Riusciremo a collegare tutti i dispositivi.
<b>Test Case</b>	<b>TC-003</b>
<b>Nome</b>	Controllo collegamento in LAN
<b>Riferimento</b>	REQ-003
<b>Descrizione</b>	Bisogna controllare che i telefoni collegati alla LAN dove si hosta una partita riescano a collegarsi e a giocare.
<b>Prerequisiti</b>	TC-001
<b>Procedura</b>	Innanzitutto, bisogna collegarsi alla stessa rete, in seguito bisognerà accedere all'applicazione, bisognerà vedere sul PC host che si è aggiunto un nuovo giocatore e sul telefono saranno comparsi i controlli per muovere il giocatore.
<b>Risultati attesi</b>	Sullo schermo sarà effettivamente comparso un giocatore nuovo.
<b>Test Case</b>	<b>TC-004</b>
<b>Nome</b>	L'avatar risponde ai comandi del giocatore
<b>Riferimento</b>	REQ-004
<b>Descrizione</b>	Bisogna essere sicuri che l'avatar segua le istruzioni dell'utente e non si muova casualmente all'interno del campo o che non stia fermo.



<b>Test Case</b>	<b>TC-004</b>
<b>Prerequisiti</b>	TC-003
<b>Procedura</b>	Una volta collegati alla partita si proverà a fare un giro del campo magari facendo qualche zig zag per verificare che si muova correttamente.
<b>Risultati attesi</b>	L'avatar segue i comandi del giocatore.
<b>Test Case</b>	<b>TC-005</b>
<b>Nome</b>	Controllare che l'avatar possa tirare
<b>Riferimento</b>	REQ-005
<b>Descrizione</b>	Bisogna essere sicuri che l'avatar segua le istruzioni dell'utente e non si muova casualmente all'interno del campo o che non stia fermo.
<b>Prerequisiti</b>	TC-004
<b>Procedura</b>	Una volta fatto un giro di prova per controllare i comandi si recupera il disco e si preme il pulsante e il disco dovrebbe essere lanciato in avanti.
<b>Risultati attesi</b>	Il disco parte effettivamente in avanti.
<b>Test Case</b>	<b>TC-006</b>
<b>Nome</b>	Controllare che ci siano 2 squadre
<b>Riferimento</b>	REQ-006
<b>Descrizione</b>	Bisogna essere sicuri che l'avatar segua le istruzioni dell'utente e non si muova casualmente all'interno del campo o che non stia fermo.
<b>Prerequisiti</b>	TC-004
<b>Procedura</b>	Una volta fatto un giro di prova per controllare i comandi si recupera il disco e si preme il pulsante e il disco dovrebbe essere lanciato in avanti.
<b>Risultati attesi</b>	Il disco parte effettivamente in avanti.
<b>Test Case</b>	<b>TC-007</b>
<b>Nome</b>	Controllare che il nome sia scelto dall'utente ma che non lo possa cambiare a partita in corso.
<b>Riferimento</b>	<ul style="list-style-type: none"> <li>• REQ-007</li> <li>• REQ-008</li> </ul>
<b>Descrizione</b>	Una volta collegati prima di essere creati si dovrà scegliere un nome, questo nome non potrà essere cambiato.
<b>Prerequisiti</b>	TC-003

<b>Test Case</b>	<b>TC-007</b>
<b>Procedura</b>	Collegarsi, dovremmo vedere una schermata che ci chiede il nome. In seguito, non sarà presente nessuna opzione per farlo.
<b>Risultati attesi</b>	Il nome inserito verrà memorizzato e verrà mostrato sopra l'avatar corrispondente.
<b>Test Case</b>	<b>TC-008</b>
<b>Nome</b>	Controllare che nello schermo dell'host ci sia una classifica.
<b>Riferimento</b>	REQ-009
<b>Descrizione</b>	Bisognerà guardare in alto a destra nello schermo dell'host per vedere che è presente la classifica dei top-scorer.
<b>Prerequisiti</b>	<ul style="list-style-type: none"> <li>• TC-005</li> <li>• TC-007</li> </ul>
<b>Procedura</b>	Entrando nella partita dovremo segnare per vedere comparire il nostro nome in classifica, se ciò accadrà vorrà dire che funziona correttamente.
<b>Risultati attesi</b>	Il nome inserito inizialmente verrà inserito nella classifica con a fianco quanti gol ho fatto.
<b>Test Case</b>	<b>TC-009</b>
<b>Nome</b>	Collegamenti alle pagine funzionanti
<b>Riferimento</b>	<ul style="list-style-type: none"> <li>• REQ-010</li> <li>• REQ-011</li> <li>• REQ-012</li> </ul>
<b>Descrizione</b>	Una volta collegati guarderemo lo schermo del telefono e vedremo i controlli, una volta guardato il monitor dell'host vedremo invece il campo visto dall'alto con i giocatori che si muovono.
<b>Prerequisiti</b>	TC-003
<b>Procedura</b>	Collegarsi, dovremmo vedere una schermata che ci chiede il nome. In seguito, non sarà presente nessuna opzione per farlo.
<b>Risultati attesi</b>	Il nome inserito verrà memorizzato e verrà mostrato sopra l'avatar corrispondente.
<b>Test Case</b>	<b>TC-010</b>
<b>Nome</b>	Partita infinita
<b>Riferimento</b>	REQ-013
<b>Descrizione</b>	La partita terminerà solo quando l'host smetterà la sua attività.

<b>Test Case</b>	<b>TC-010</b>
<b>Prerequisiti</b>	TC-003
<b>Procedura</b>	Non è semplice da testare ma si può lasciare andare la partita per un po' di tempo e vedere che non si ferma.
<b>Risultati attesi</b>	Non sarà presente nessun timer, e la partita non si fermerà se non interrotta dall'host.
<b>Test Case</b>	<b>TC-011</b>
<b>Nome</b>	Rubare disco
<b>Riferimento</b>	REQ-014
<b>Descrizione</b>	Se un giocatore senza disco va addosso a uno con il disco, lo ottiene
<b>Prerequisiti</b>	<ul style="list-style-type: none"> <li>• TC-003</li> <li>• TC-004</li> </ul>
<b>Procedura</b>	Collegarsi alla partita con due utenti uno prende il disco, l'altro gli va addosso.
<b>Risultati attesi</b>	Il giocatore senza disco dovrebbe ottonere il possesso del disco, mentre l'altro lo perde.
<b>Test Case</b>	<b>TC-012</b>
<b>Nome</b>	Rimbazo disco
<b>Riferimento</b>	REQ-014
<b>Descrizione</b>	Il disco deve rimbalzare contro le pareti
<b>Prerequisiti</b>	<ul style="list-style-type: none"> <li>• TC-003</li> <li>• TC-004</li> </ul>
<b>Procedura</b>	Collegarsi alla partita con un utente, prendere il disco andandogli addosso e tirarlo contro una parete.
<b>Risultati attesi</b>	Il disco rimbalza contro la parete.

## Risultati test

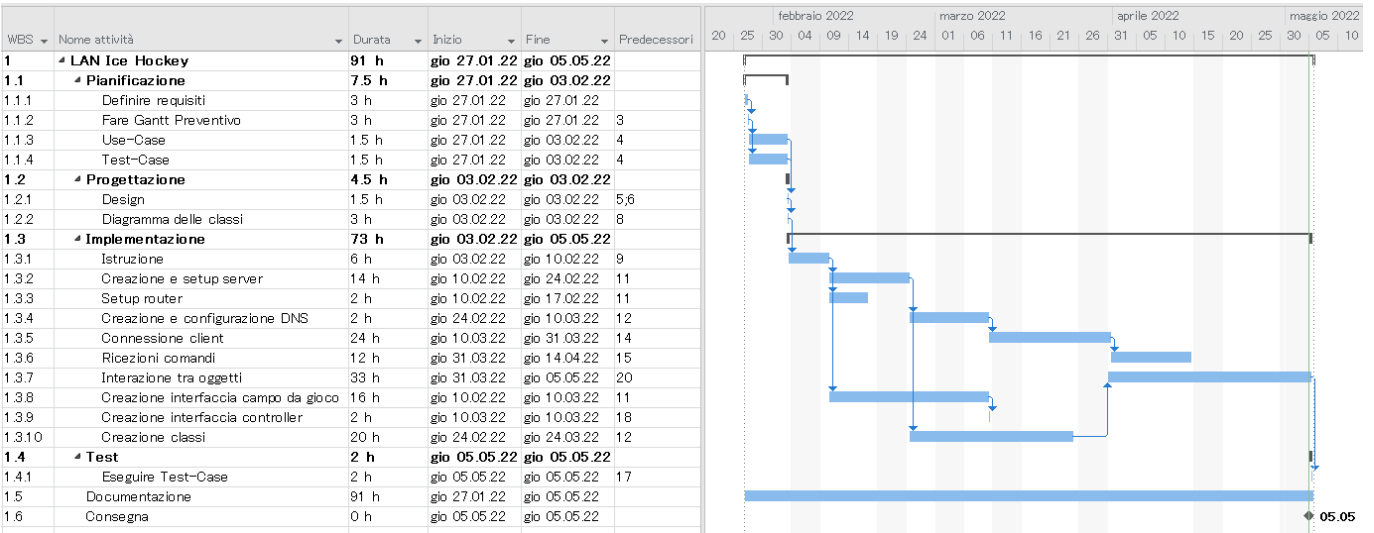
Test	Risultato
<b>TC-001</b>	Esito positivo.
<b>TC-002</b>	Esito positivo.
<b>TC-003</b>	Esito positivo.
<b>TC-004</b>	Esito positivo.

Test	Risultato
TC-005	Esito positivo.
TC-006	Esito positivo.
TC-007	Esito positivo.
TC-008	Esito positivo.
TC-009	Esito positivo.
TC-010	Esito positivo.
TC-011	Esito positivo.
TC-012	Esito positivo.

Mancanze/limitazioni conosciute

- È necessario connettersi inserendo l'ip della macchina invece di un url, questo perchè non è stato possibile utilizzare un DNS per tradurre i nomi.
- Se si cambia ambiente di rete è necessario cambiare gli ip a livello di codice in HostConnection, ClientConnection e in settings.json perchè nei primi due file non è consentito fare il require di settings.json.
- Non è possibile tirare o scattare mentre ci si sta muovendo, quindi bisogna fermarsi per ogni altra azione che si vuole eseguire, questo perchè NippleJS (la classe implementata per il joystick) non permette di utilizzare il multitouch.

Consuntivo



Gantt Consuntivo

Conclusioni

Sviluppi futuri

Rendere il progetto accessibile dalla rete scolastica, e che sia sempre accessibile. Attualmente si può avere una sola partita, si potrebbe far sì che possano esistere più partite in contemporanea, e gli utenti possono scegliere a quale partita unirsi. Rendere il gioco più realistico, non avendo implementato nessuna delle regole presenti nell'hockey su ghiaccio, questo renderebbe il gioco più competitivo e gradibile agli utenti.

## Considerazioni personali

### **Xavier Horisberger:**

- Devo dire che questo progetto mi è piaciuto, finalmente ho imparato qualcosa di nuovo, diversamente dal mio progetto scorso. Abbiamo imparato a usare i websocket, e quindi fare un sito client-server e a fare un gioco usando phaser. Il tempo non l'abbiamo gestito veramente bene, distraendoci anche spesso, e il gantt preventivo non è risultato accurato, per nulla, probabilmente ancora dovuto alla mancanza di esperienza, ma nonostante tutto, abbiamo portato a termine il progetto, ottenendo anche un risultato più che soddisfacente.

### **Nathan Ferrari:**

- Questo progetto mi è piaciuto, purtroppo abbiamo gestito male il tempo e siamo arrivati in ritardo rispetto alla nostra progettazione, all'inizio non avevamo bene le idee in chiaro su come fare il game loop e ci abbiamo messo un po' di tempo a capire come utilizzare i socket. Devo dire che comunque ho più conoscenze rispetto all'inizio e che il risultato finale tutto sommato non è male, di sicuro è migliorabile ma come tutto.

### **Andrea Masciocchi:**

- Mi è complessivamente piaciuto, già dall'inizio mi aspettavo che la produzione del risultato sarebbe stata così appagante, probabilmente per via della natura ludica del progetto, ma alla conclusione mi sono reso conto che oltre al divertimento c'è da tenere in considerazione anche la parte didattica di questo percorso. Ho acquisito molte conoscenze che mi serviranno sicuramente in futuro, è stato difficile ma è stato anche molto interessante imparare ad usare i WebSockets, Phaser eccetera. Sicuramente ci sono state anche cose negative, come la maldestra gestione del tempo a nostra disposizione e dei compiti all'interno del gruppo, o la pianificazione veramente poco accurata rispetto alla realtà che si è poi sviluppata. Questi errori sono dati probabilmente dalla poca esperienza nel gestire un progetto, ma ora che ho imparato dagli errori so per certo che i prossimi lavori saranno migliori. In generale, sono soddisfatto del risultato, nelle nozioni apprese e dei miei colleghi (nonostante ci siamo distratti fin troppo).

## Bibliografia

### Sitografia

- <https://www.youtube.com/watch?v=V1kNrsvQWs>, 03-02-2022
- <https://developer.mozilla.org/en-US/>, *MDN Web Docs*, 10-02-2022
- <http://www.referencedesigner.com/>, *Reference Designer Inc. - Engineering and Design Services*, 10.02.2022
- <https://favicon.io/emoji-favicons/ice-hockey/>, *favicon.io - favicon.io*, 17-02-2022

- [https://media.istockphoto.com/vectors/ice-hockey-crossed-sticks-and-puck-icon-black-silhouette-isolated-on-vector-id1207437659?k=20&m=1207437659&s=170667a&w=0&h=EOxTs6V\\_YZeujFtrG8GkFv9zSVpUk5jwYu5Ubh9Uf4U=,17-02-2022](https://media.istockphoto.com/vectors/ice-hockey-crossed-sticks-and-puck-icon-black-silhouette-isolated-on-vector-id1207437659?k=20&m=1207437659&s=170667a&w=0&h=EOxTs6V_YZeujFtrG8GkFv9zSVpUk5jwYu5Ubh9Uf4U=,17-02-2022)
- <https://getcssscan.com/css-buttons-examples>, *84 Beautiful CSS buttons examples - CSS Scan*, 24-02-2022
- <https://github.com/geo-petrini/phaserasteroids/blob/main/js/scenes/gamescene.js>, 24-03-2022
- <https://phaser.discourse.group/t/is-it-possible-to-have-game-add-text-follow-a-sprite/6027>, 07.04.2022
- <https://www.cloudsavvyit.com/12277/how-to-use-multi-threaded-processing-in-bash-scripts/>, 07.04.2022
- <https://phaser.io/examples/v3/view/physics/arcade/remove-collider>, 14.04.2022
- <https://phaser.io/examples/v3/view/physics/arcade/sprite-overlap-group#>, 14-04-2022
- <https://phaser.io/examples/v3/view/physics/arcade/collider-1#>, 14-04-2022

## Allegati

Elenco degli allegati, esempio:

- Diari di lavoro
- Codici sorgente
- Documentazione di [NippleJS](#), [NodeJS](#) e [Phaser](#)
- Qdc