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 simplefied 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 yourselfs to the same network, a main computer with a sufficiantly 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 sull'hockey su ghiaccio, il gioco è in una LAN a se stante, per giocare bisogna quindi collegarsi al router apposito, 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 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.

Analisi e specifica dei requisiti

ID:	**REQ-001**
Nome	Rappresentazione semi-reale di una partita di Hockey
Priorità	1
Versione	1.0

ID:	**REQ-002**
Nome	La partita non ha un limite di giocatori
Priorità	2
Versione	1.0

ID:	**REQ-003**
Nome	Gli utenti devono potersi collegare alla partita tramite telefono in LAN
Priorità	1
Versione	1.0

ID:	**REQ-004**
Nome	Gli utenti devono poter muovere il loro giocatore in LAN
Priorità	1
Versione	1.0

ID:	**REQ-005**
Nome	Gli utenti devono poter tirare in porta
Priorità	2
Versione	1.0

ID:	**REQ-006**
Nome	Devono esserci due squadre (verdi e gialli)
Priorità	2
Versione	1.0

ID:	**REQ-007**
Nome	Gli utenti devono poter dare un nome al loro giocatore
Priorità	2
Versione	1.0

ID:	**REQ-008**
Nome	Una volta assegnato il nome non può più essere cambiato
Priorità	3
Versione	1.0

ID:	**REQ-009**
Nome	In alto a destra ci sarà una classifica con i 10 giocatori che hanno fatto più goal
Priorità	2
Versione	1.0

ID:	**REQ-010**
Nome	I telefoni degli utenti agiranno solo da controller
Priorità	1

ID:	**REQ-010**
Versione	1.0

ID:	**REQ-011**
Nome	Il campo di gioco viene visualizzato su un monitor a parte (non sugli schermi dei telefoni)
Priorità	1
Versione	1.0

ID:	**REQ-012**
Nome	Il campo da gioco è visto dall'alto
Priorità	2
Versione	1.0

ID:	**REQ-013**
Nome	La partita è infinita
Priorità	3
Versione	1.0

ID:	**REQ-014**
Nome	Gli utenti devono potersi rubare il disco
Priorità	2
Versione	1.0

ID:	**REQ-015**
Nome	Il disco rimbalza contro le pareti
Priorità	2
Versione	1.0

Spiegazione elementi tabella dei requisiti:

ID: identificativo univoco del requisito

Nome: breve descrizione del requisito

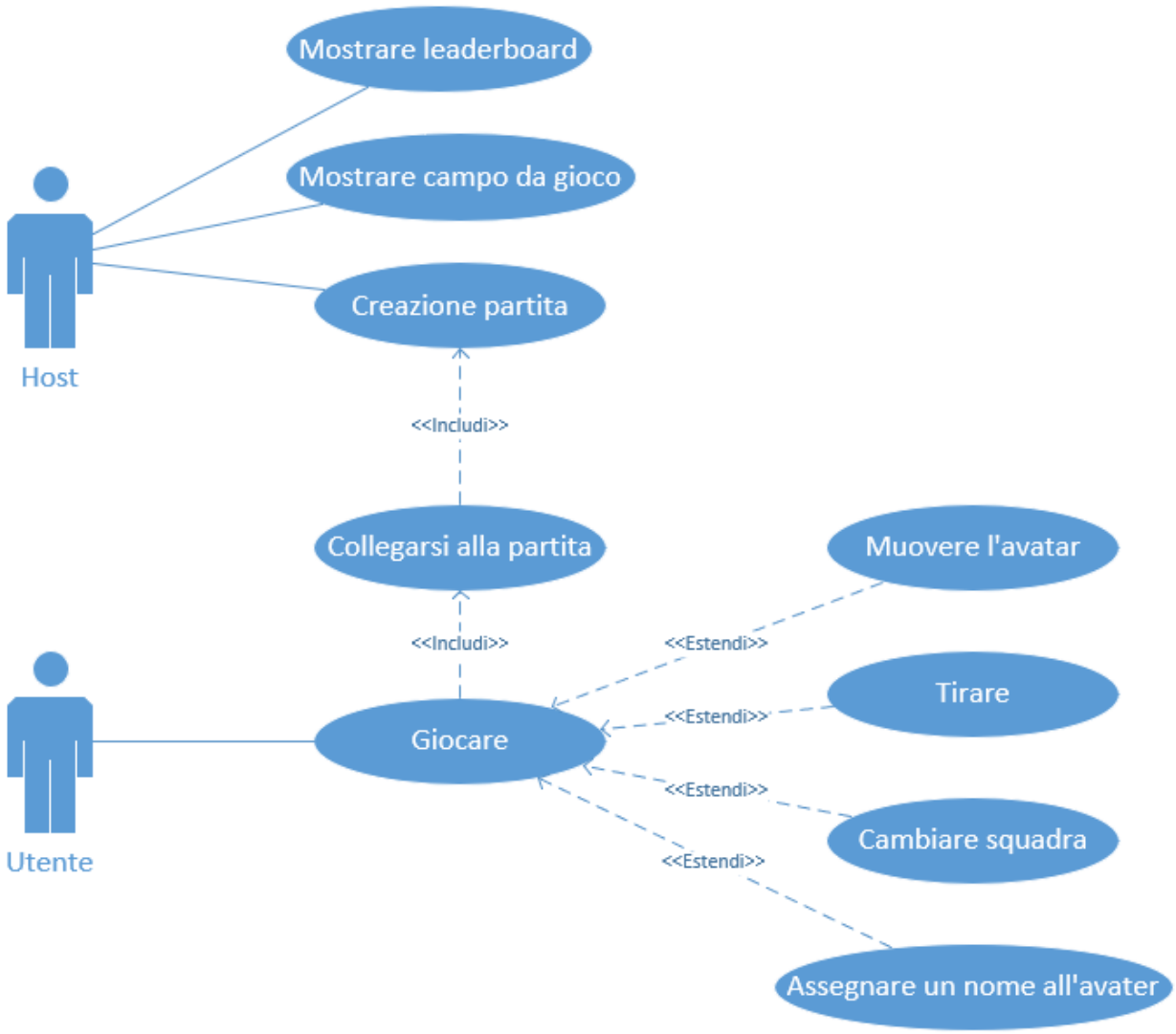
Priorità: indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente. Ad esempio poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo 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:

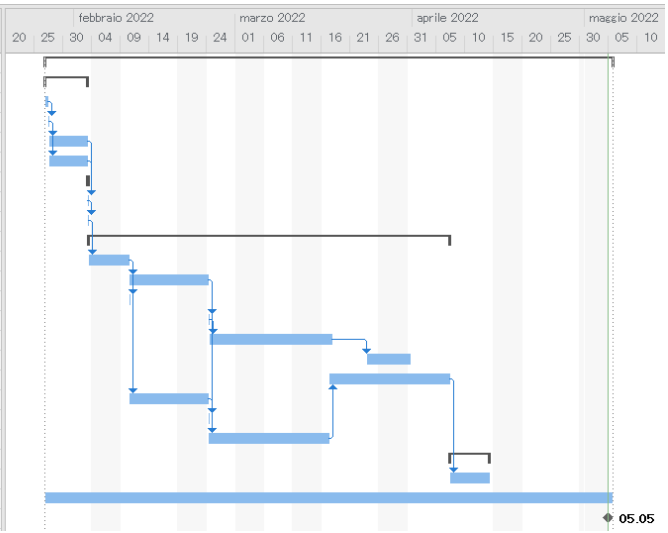


Use case

Pianificazione

Abbiamo scelto una pianificazione waterfall:

WBS	Nome attività	Durata	Inizio	Fine	Predecessori
1	**LAN Ice Hockey**	**91 h**	**gio 27.01.22**	**gio 05.05.22**	
1.1	**Pianificazione**	**7.5 h**	**gio 27.01.22**	**gio 03.02.22**	
1.1.1	Definire requisiti	3 h	gio 27.01.22	gio 27.01.22	
1.1.2	Fare Gantt Preventivo	3 h	gio 27.01.22	gio 27.01.22	3
1.1.3	Use-Case	1.5 h	gio 27.01.22	gio 03.02.22	4
1.1.4	Test-Case	1.5 h	gio 27.01.22	gio 03.02.22	4
1.2	**Progettazione**	**3 h**	**gio 03.02.22**	**gio 03.02.22**	
1.2.1	Design	1.5 h	gio 03.02.22	gio 03.02.22	5;6
1.2.2	Diagramma delle classi	1.5 h	gio 03.02.22	gio 03.02.22	8
1.3	**Implementazione**	**51 h**	**gio 03.02.22**	**gio 07.04.22**	
1.3.1	Istruzione	6 h	gio 03.02.22	gio 10.02.22	9
1.3.2	Creazione e setup server	10 h	gio 10.02.22	gio 24.02.22	11
1.3.3	Setup router	1 h	gio 10.02.22	gio 10.02.22	11
1.3.4	Creazione e configurazione DNS	4 h	gio 24.02.22	gio 24.02.22	12
1.3.5	Connessione client	15 h	gio 24.02.22	gio 17.03.22	14
1.3.6	Ricezioni comandi	10 h	gio 24.03.22	gio 31.03.22	15
1.3.7	Interazione tra oggetti	20 h	gio 17.03.22	gio 07.04.22	20
1.3.8	Creazione interfaccia campo da gioco	10 h	gio 10.02.22	gio 24.02.22	11
1.3.9	Creazione interfaccia controller	2 h	gio 24.02.22	gio 24.02.22	18
1.3.10	Creazione classi	15 h	gio 24.02.22	gio 17.03.22	12
1.4	**Test**	**6 h**	**gio 07.04.22**	**gio 14.04.22**	
1.4.1	Eseguire Test-Case	6 h	gio 07.04.22	gio 14.04.22	17
1.5	Documentazione	91 h	gio 27.01.22	gio 05.05.22	
1.6	Consegna	0 h	gio 05.05.22	gio 05.05.22	



> Gantt preventivo

Analisi dei mezzi

Hardware

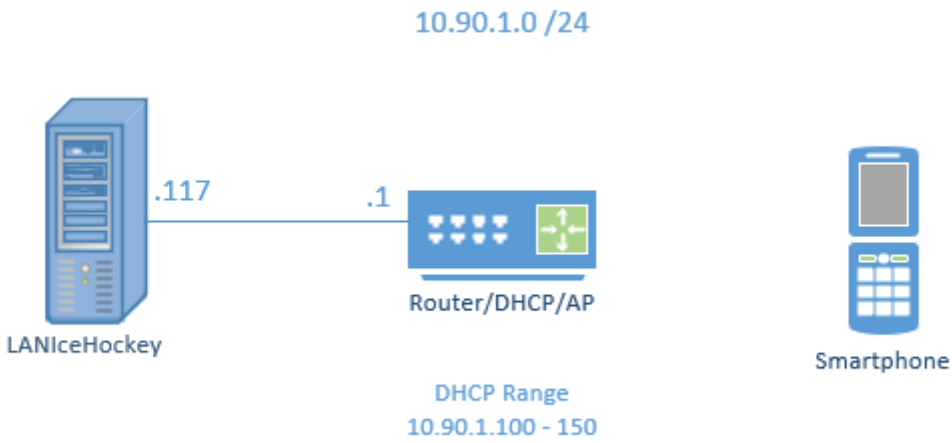
- Computer1:
  - Processore: Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz
  - RAM: 16 GB
- Computer2:
  - Processore: Intel Xeon (R) CPU E3-1240 V2 @ 3.40GHz x 8
  - RAM: 8 GB
- Router:
  - Model: Linksys EA6350
  - Wifi name: Linksys10206
  - Wifi password: LANIceHockey
  - Ip address: 10.90.1.1
- Switch: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

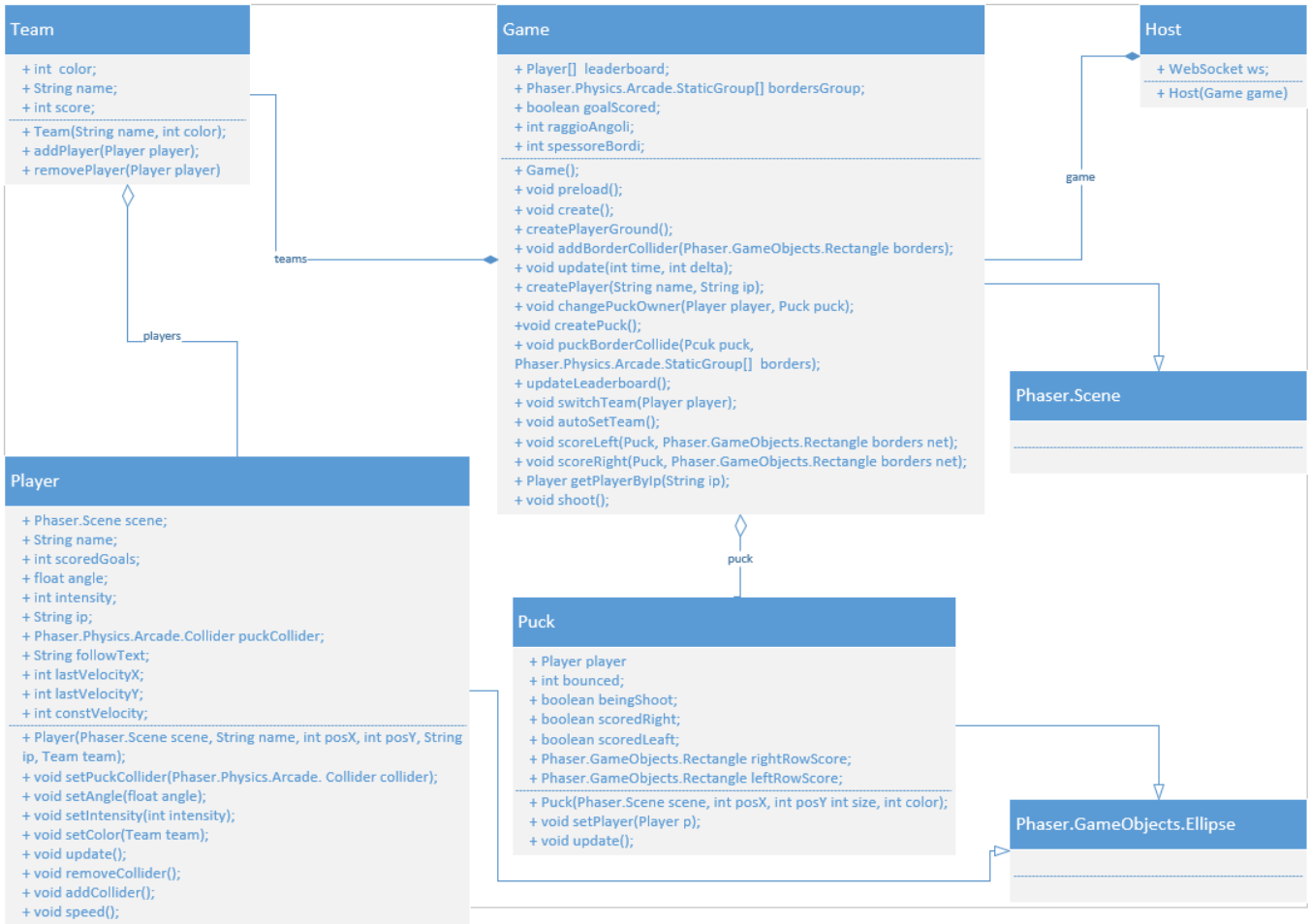
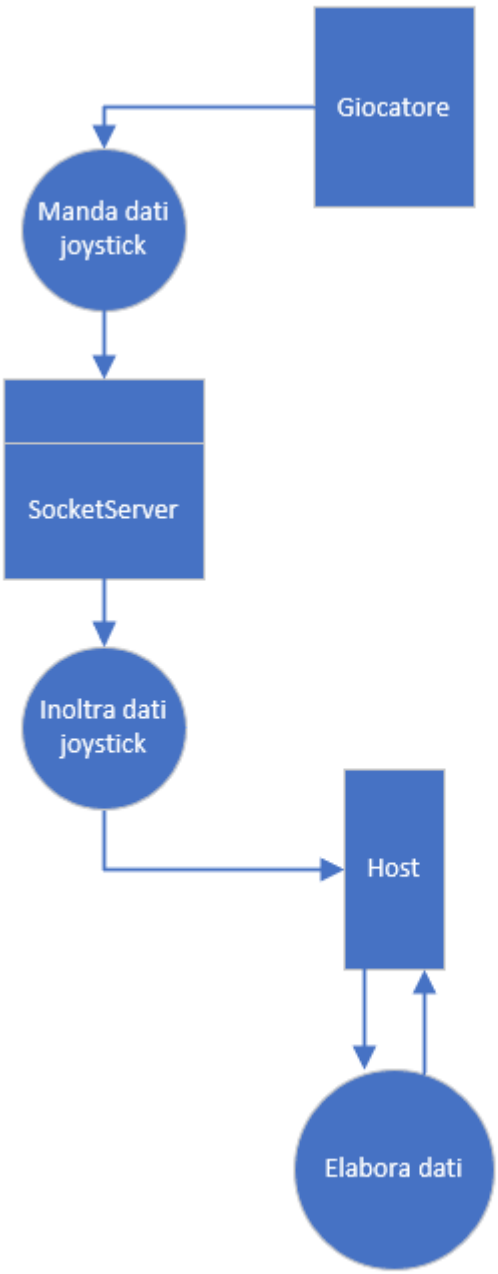


Diagramma delle classi

Diagramma di flusso dei dati:



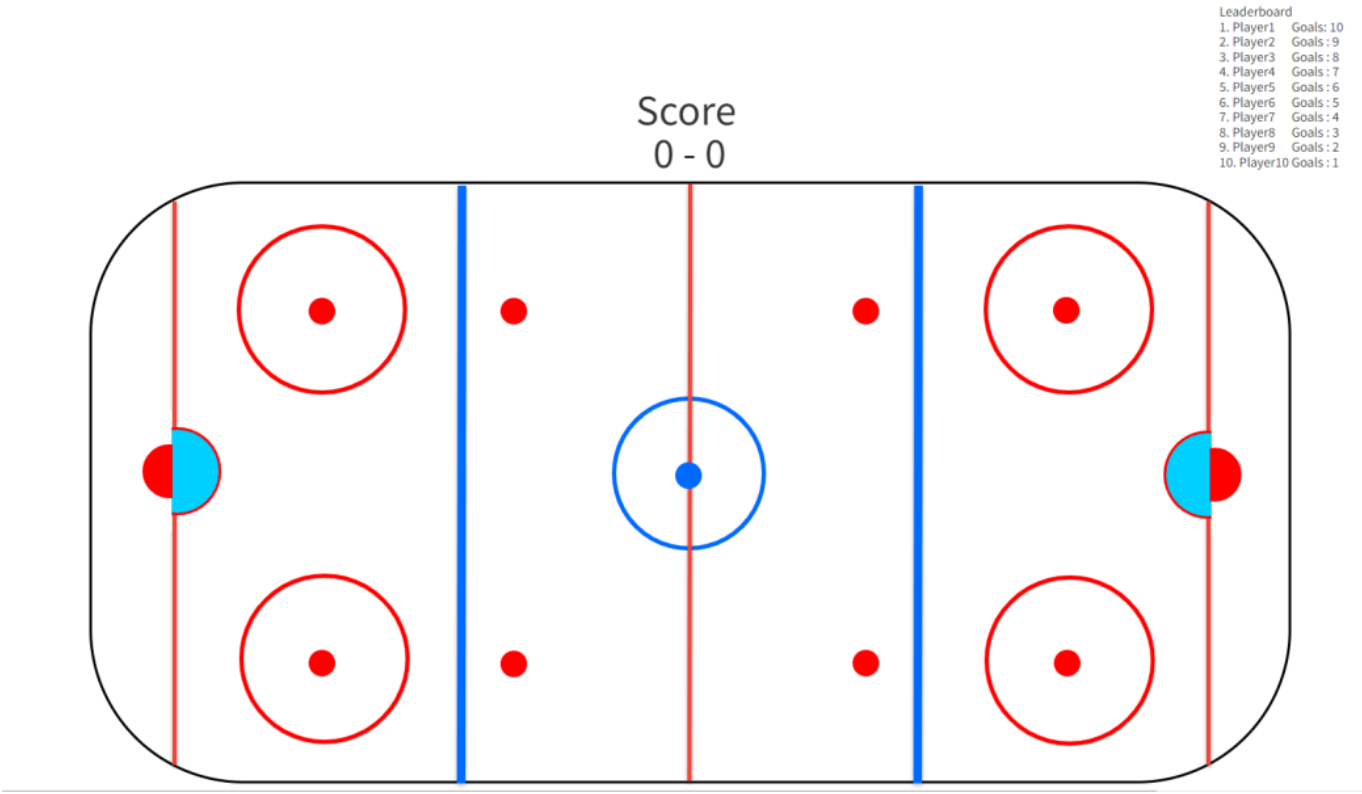
> DFD

Design dei dati e database

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

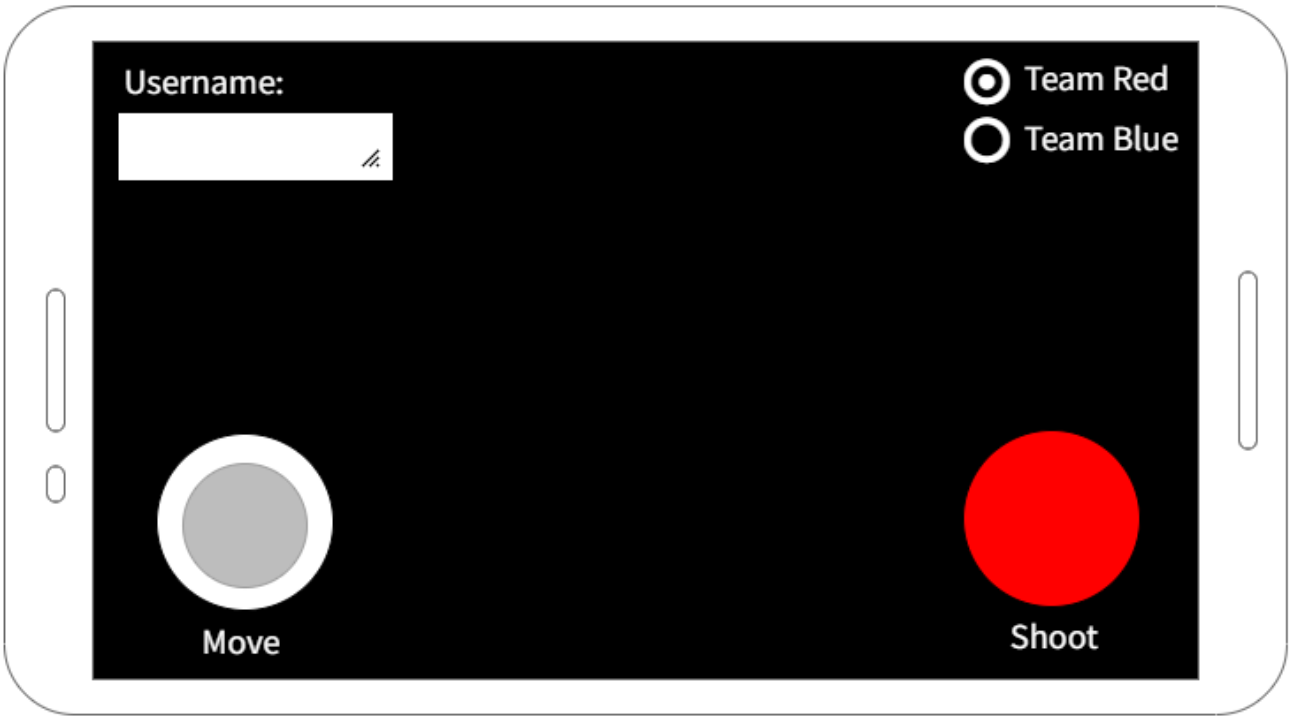
Design delle interfacce

Design dell'interfaccia del campo da gioco:



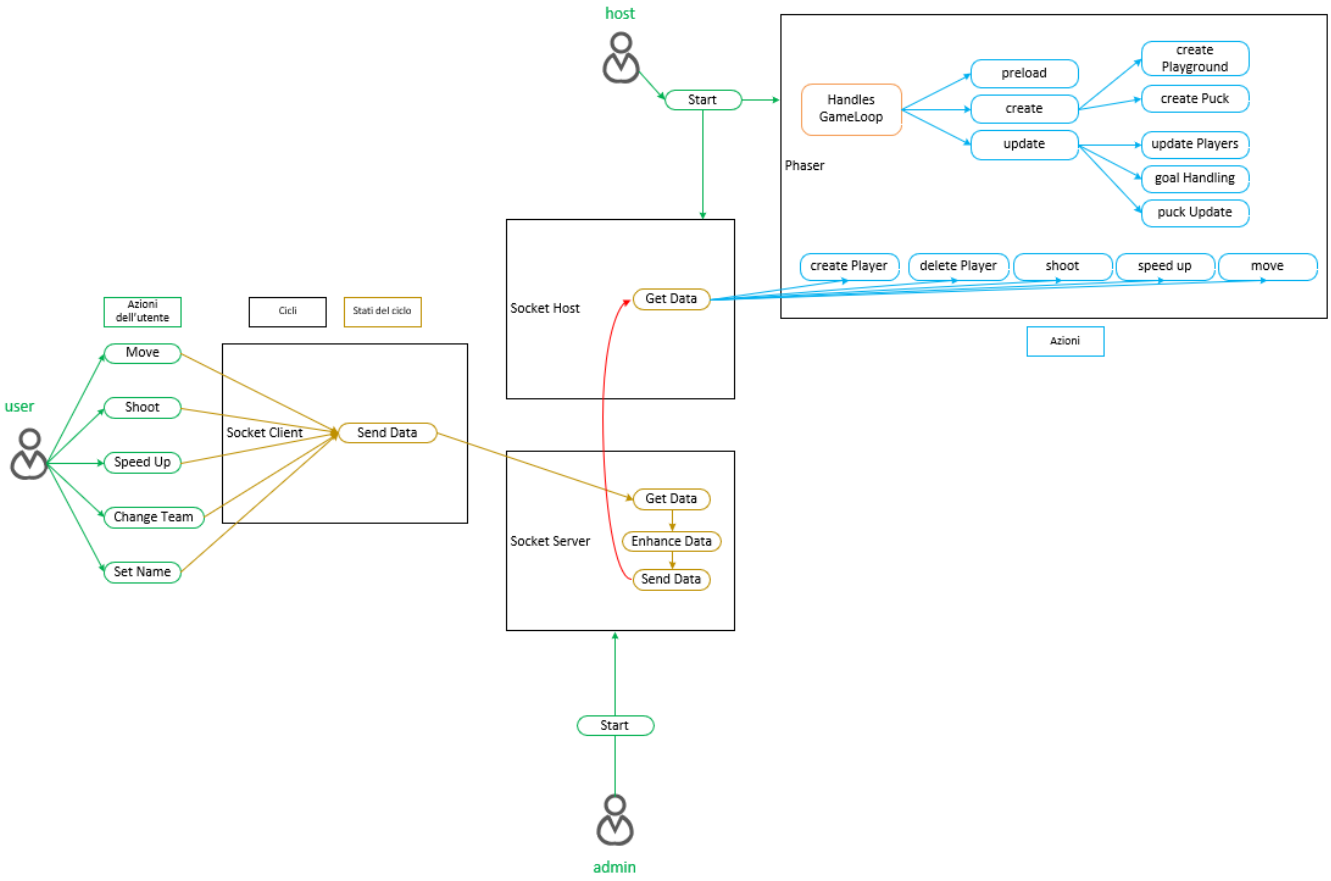
Campo da gioco

Design dell'interfaccia del controller:



Controller

Design procedurale



> Design Procedurale

Implementazione

Installare nodejs.

Classi

Player

Questa classe rappresenta i singoli giocatori che partecipano al gioco, ed estende Phaser.GameObjects.Ellipse perché nel campo vengono rappresentati appunto come ellipse ovvero degli ellissi.

Attributi:

- scene: è la scena a cui appartiene ovvero serve a far sapere a Phaser dove farlo rappresentare nella scena di gioco.
- name: è il nome con cui il giocatore decide di giocare.
- scoredGoals: che rappresenta il numero di reti segnate dal giocatore.
- angle: è la direzione in cui il giocatore punta il joystick, rappresenta l'angolo in cui l'ellipse punta per muoversi.
- intensity: rappresenta la forza con cui il player si sposta, ovvero la velocità. Questa viene calcolata rispetto a quanto il joystick è spostato rispetto al centro.
- ip: serve a salvare l'indirizzo di rete degi dispositivi che si connettono,appunto un player è mosso dal proprio dispositivo.

- followText: serve a creare e salvare l'oggetto di phaser che serve per rendere visibile a schermo il nome.
- lastVelocityX: si salva l'ultima componente orizzontale della velocità, serve 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, serve a calcolare la direzione del puck una volta tirato.
- constVelocity: serve come moltiplicatore della forza ricevuta dal joystick.

Metodi:

- constructor(scene, name, posX, posY, ip, team): questo metodo è il costruttore, invoca il costruttore della classe ellipse, setta la posizione di partenza, nome, indirizzo ip e creiamo il testo con scritto il nome che segue il giocatore. Abilitiamo anche la fisica e aggiungiamo sia player che scritta alla scena.
- setAngle(angle): questo metodo prende l'angolo passato come argomento e lo assegna all'attributo angle.
- setIntensity(intensity): questo metodo una volta passata un'intensità la assegnerà all'attributo intensity.
- setColor(team): questo metodo una volta passato un team colora il player del colore del team.
- addCollider(): questo metodo serve ad attivare un collider tra il puck e il player.
- setPuckCollider(collider): questo metodo una volta passato collider lo assegna all'attributo puckCollider.
- removeCollider(): questo metodo serve per disattivare il collider e dunque fare in modo che il disco venga tirato dal player, se ciò non succedesse il player appena lo tira lo riprende dunque non sarebbe giocabile.
- update(): questo metodo viene invocato da noi tramite il metodo automatico di Phaser e permettere di fare delle azioni ad ogni ciclo di gioco. Noi lo utiliziamo per calcolare la velocità angolare del player in maniera da farlo muovere seguendo il joystick e assegnamo anche la nuova posizione del nome del giocatore in maniera da seguire il player. Assegnamo anche lastVelocityX e lastVelocityY.
- speed(): Questo metodo serve a duplicare la velocità di movimento per un tempo predefinito.

Team

Questa classe rappresenta i team come lista di player e ulteriori attributi assegnati per rappresentare una squadra.

Attributi:

- players: qesto attributo contiene una lista di tutti i player che giocano per il team.
- name: questo attributo contiene il nome del team.
- color: questo attributo 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 com ellisse nel gioco.

Attributi:

- player: si salva il giocatore che ha il disco momentaneamente.
- beingShoot: rappresenta il fatto che se è stato tirato oppure no.
- bounced: questo attributo conta 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 riazzerare il campo.
- rightRowScore && leftRowScore: servono per salvarsi le istanze di Phaser delle linee di porta.

Metodi:

- constructor(scene, posX, posY, size = 35, color = 0x202020): il costruttore serve ad istanziare un nuovo puck, viene richiamato il costruttore di ellipse in maniera da effettivamente farlo diventare tale. Si abilita la fisica dell'oggetto, si genera nella scena e viene loggata la creazione.
- 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 non è stato tirato.

Game

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

Attributi:

- teams: questo attributo contiene 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: questo attributo serve a salvarsi il puck della partita.
- raggioAngoli: serve a disegnare il campo, è quanto è lungo il raggio delle curve degli angoli.(viene anche utilizato per altre formule nei disegni).
- spessoreBordi: serve a disegnare il campo, è quanto e spesso il bordo.(viene anche utilizato per altre formule nei disegni).

Metodi:

- constructor(): questo metodo serve ad istanziare una nuova partita, richiama il costruttore della superclasse, crea due player di default (in futuro si potrebbero rendere customizabili) e infine creo l'Host della partita.

- preload(): questo metodo 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(): questo metodo viene richiamato da Phaser alla creazione dell'oggetto, lo utiliziamo per disegnare il campo e creare il puck invocando i metodi appositi.
- createPlayGround(): questo metodo viene incocato da create(), e serve a disegnare gli elementi del campo, al suo intermo richiamiamo anche il metodo che permette di aggiungere i collider ai bordi e alle porte.
- addBorderCollider(borders): questo metodo una volta passato l'array di bordi gli 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 a 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 player abbia il disco. Viene anche riaggiunto il collider del disco al player che possedeva precedentemento il puck.
- createPuck(): questa funzione permette di creare il puck all'interno del game, genera anche le righe delle porte e gli assegna gli overlap in maniera da poi assegnare i goal. Aggiunge i collider dei bordi e assegna il bounce a 0.5, dunque ogni volta che rimbalza il puck esso dimezza la propria velocità.
- puckBorderCollide(puck, borders): questo metodo permette di contare i rimbalzi del puck cambiandone l'attributo. Questa funzione viene invocata dai collider con i bordi.
- updateLeaderboard(): questo metodo 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 gli cambia il team, si controlla a che team appartine, 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): questa è la funzione 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): questa è la funzione 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(): questo metodo permette di far tirare il puck, se non è già stato tirato gli viene impostata la velocità del player per 2, serve per non poter riprendere subito il disco.

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.

Test Case	**TC-001**
Prerequisiti	• TC-003 • TC-004 • TC-005 • TC-006 • TC-009
Procedura	Accedere al gioco in 4 o più e provare a fare una partita.
Risultati attesi	Il gioco ricorda una partita di hockey.

Test Case	**TC-002**
Nome	Nessun limite di giocatori
Riferimento	REQ-002
Descrizione	Bisogna provare a giocare con diversi dispositivi in maniera da controllare che non ci siano limiti nelle persone che giocano.
Prerequisiti	TC-001
Procedura	Collegheremo più dispostivi alla partita.
Risultati attesi	Riusciremo a collegare tutti i dispositivi.

Test Case	**TC-003**
Nome	Controllo collegamento in LAN
Riferimento	REQ-003
Descrizione	Bisogna controllare che i telefoni collegati alla LAN dove si hosta una partita riescano a collegarsi e a giocare.
Prerequisiti	TC-001
Procedura	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.
Risultati attesi	Sullo schermo sarà effettivamente comparso un giocatore nuovo.

Test Case	**TC-004**
Nome	L'avatar risponde ai comandi del giocatore
Riferimento	REQ-004
Descrizione	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.

Test Case	**TC-004**
Prerequisiti	TC-003
Procedura	Una volta collegati alla partita si proverà a fare un giro del campo magari facendo qualche zig zag per verificare che si muova correttamente.
Risultati attesi	L'avatar segue i comandi del giocatore.

Test Case	**TC-005**
Nome	Controllare che l'avatar possa tirare
Riferimento	REQ-005
Descrizione	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.
Prerequisiti	TC-004
Procedura	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.
Risultati attesi	Il disco parte effettivamente in avanti.

Test Case	**TC-006**
Nome	Controllare che ci siano 2 squadre
Riferimento	REQ-006
Descrizione	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.
Prerequisiti	TC-004
Procedura	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.
Risultati attesi	Il disco parte effettivamente in avanti.

Test Case	**TC-007**
Nome	Controllare che il nome sia scelto dall'utente ma che non lo possa cambiare a partita in corso.
Riferimento	• REQ-007 • REQ-008
Descrizione	Una volta collegati prima di essere creati si dovrà scegliere un nome, questo nome non potrà essere cambiato.
Prerequisiti	TC-003

Test Case	**TC-007**
Procedura	Collegarsi, dovremmo vedere una schermata che ci chiede il nome. In seguito, non sarà presente nessuna opzione per farlo.
Risultati attesi	Il nome inserito verrà memorizzato e verrà mostrato sopra l'avatar corrispondente.
Test Case	**TC-008**
Nome	Controllare che nello schermo dell'host ci sia una classifica.
Riferimento	REQ-009
Descrizione	Bisognerà guardare in alto a destra nello schermo dell'host per vedere che è presente la classifica dei top-scorer.
Prerequisiti	• TC-005 • TC-007
Procedura	Entrando nella partita dovremo segnare per vedere comparire il nostro nome in classifica, se ciò accadrà vorrà dire che funziona correttamente.
Risultati attesi	Il nome inserito inizialmente verrà inserito nella classifica con a fianco quanti gol ho fatto.
Test Case	**TC-009**
Nome	Collegamenti alle pagine funzionanti
Riferimento	• REQ-010 • REQ-011 • REQ-012
Descrizione	Una volta collegati guarderemo lo schermino 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.
Prerequisiti	TC-003
Procedura	Collegarsi, dovremmo vedere una schermata che ci chiede il nome. In seguito, non sarà presente nessuna opzione per farlo.
Risultati attesi	Il nome inserito verrà memorizzato e verrà mostrato sopra l'avatar corrispondente.
Test Case	**TC-010**
Nome	Partita infinita
Riferimento	REQ-013
Descrizione	La partita terminerà solo quando l'host smetterà la sua attività.

Test Case	**TC-010**
Prerequisiti	TC-003
Procedura	Non è semplice da testare ma si può lasciare andare la partita per un po’ di tempo e vedere che non si ferma.
Risultati attesi	Non sarà presente nessun timer, e la partita non si fermerà se non interrotta dall’host.

Test Case	**TC-011**
Nome	Rubare disco
Riferimento	REQ-014
Descrizione	Se un giocatore senza disco va addosso a uno con il disco, lo ottiene
Prerequisiti	• TC-003 • TC-004
Procedura	Collegarsi alla partita con due utenti uno prende il disco, l'altro gli va addosso.
Risultati attesi	Il giocatore senza disco dovrebbe ottonere il possesso del disco, mentre l'altro lo perde.

Test Case	**TC-012**
Nome	Rimbazo disco
Riferimento	REQ-014
Descrizione	Il disco deve rimbalzare contro le pareti
Prerequisiti	• TC-003 • TC-004
Procedura	Collegarsi alla partita con un utente, prendere il disco andandogli addosso e tirarlo contro una parete.
Risultati attesi	Il disco rimbalza contro la parete.

Risultati test

Test	**Risultato**
TC-001	Esito positivo.
TC-002	Esito positivo.
TC-003	Esito positivo.
TC-004	Esito positivo.

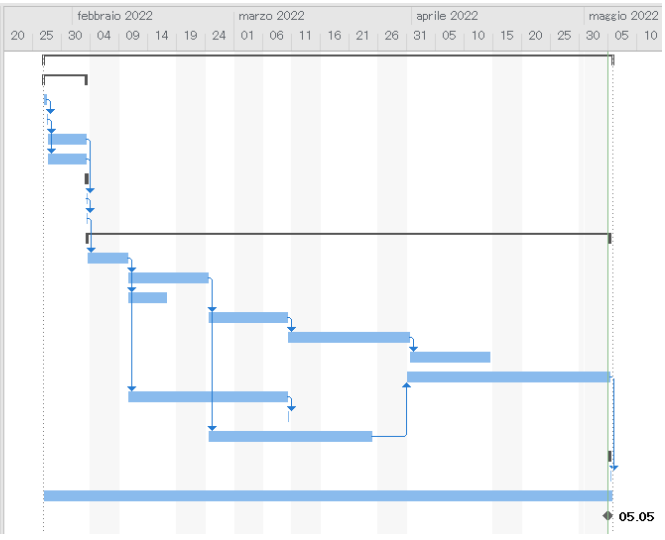
Test	Risultato
TC-005	Esito positivo.
TC-006	Esito positivo.
TC-007	Esito positivo.
TC-008	Esito positivo.
TC-009	Esito positivo.
TC-010	Esito positivo.
TC-011	Esito positivo.
TC-012	Esito positivo.

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

WBS	Nome attività	Durata	Inizio	Fine	Predecessori
1	**LAN Ice Hockey**	**91 h**	**gio 27.01.22**	**gio 05.05.22**	
1.1	**Pianificazione**	**7.5 h**	**gio 27.01.22**	**gio 03.02.22**	
1.1.1	Definire requisiti	3 h	gio 27.01.22	gio 27.01.22	
1.1.2	Fare Gantt Preventivo	3 h	gio 27.01.22	gio 27.01.22	3
1.1.3	Use-Case	1.5 h	gio 27.01.22	gio 03.02.22	4
1.1.4	Test-Case	1.5 h	gio 27.01.22	gio 03.02.22	4
1.2	**Progettazione**	**4.5 h**	**gio 03.02.22**	**gio 03.02.22**	
1.2.1	Design	1.5 h	gio 03.02.22	gio 03.02.22	5,6
1.2.2	Diagramma delle classi	3 h	gio 03.02.22	gio 03.02.22	8
1.3	**Implementazione**	**73 h**	**gio 03.02.22**	**gio 05.05.22**	
1.3.1	Istruzione	6 h	gio 03.02.22	gio 10.02.22	9
1.3.2	Creazione e setup server	14 h	gio 10.02.22	gio 24.02.22	11
1.3.3	Setup router	2 h	gio 10.02.22	gio 17.02.22	11
1.3.4	Creazione e configurazione DNS	2 h	gio 24.02.22	gio 10.03.22	12
1.3.5	Connessione client	24 h	gio 10.03.22	gio 31.03.22	14
1.3.6	Ricezioni comandi	12 h	gio 31.03.22	gio 14.04.22	15
1.3.7	Interazione tra oggetti	33 h	gio 31.03.22	gio 05.05.22	20
1.3.8	Creazione interfaccia campo da gioco	16 h	gio 10.02.22	gio 10.03.22	11
1.3.9	Creazione interfaccia controller	2 h	gio 10.03.22	gio 10.03.22	18
1.3.10	Creazione classi	20 h	gio 24.02.22	gio 24.03.22	12
1.4	**Test**	**2 h**	**gio 05.05.22**	**gio 05.05.22**	
1.4.1	Eseguire Test-Case	2 h	gio 05.05.22	gio 05.05.22	17
1.5	Documentazione	91 h	gio 27.01.22	gio 05.05.22	
1.6	Consegna	0 h	gio 05.05.22	gio 05.05.22	



Gantt Consuntivo

Conclusioni

Sviluppi futuri

Rendere il progetto accessibile dalla rete scolastica, e che sia sempre accessibile. Attualmente si può avere una solo 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:

-

Nathan Ferrari:

-

Andrea Masciocchi:

-

Bibliografia

Sitografia

- https://www.youtube.com/watch?v=V1kNrsdVqWs, 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://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