Nomi studenti: Riccardo Risi e Roberto Forner (GRUPPO 20)

Progetto finale
Python

---

introduzione

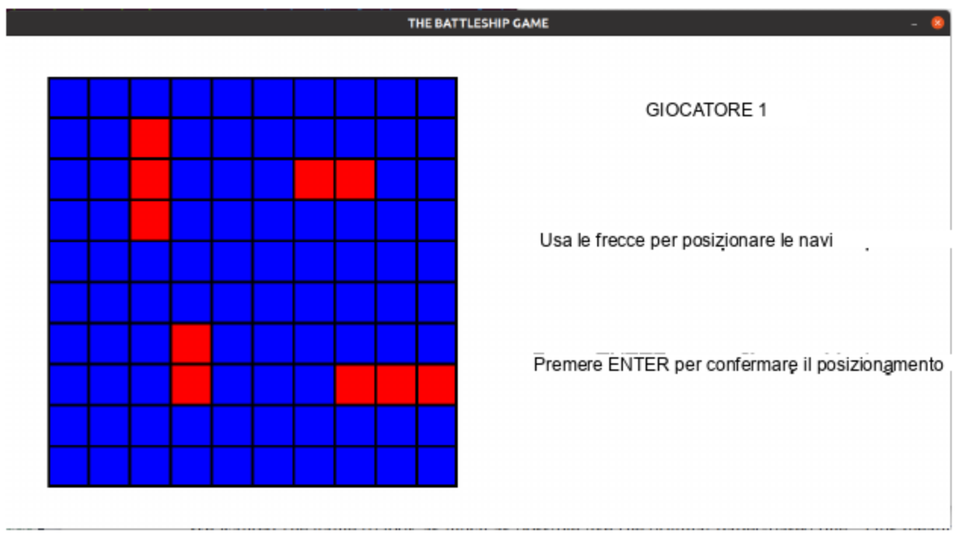
Lo scopo di questo progetto era costruire il gioco Battleship utilizzando il linguaggio di programmazione Python. Battleship è un gioco da tavolo di tipo strategico a cui possono partecipare due persone: ogni giocatore ha una griglia dove può posizionare la sua flotta di navi da guerra, la posizione esatta di queste navi rimane sconosciuta all'altro giocatore. A turno, ogni giocatore spara alla griglia del suo avversario dichiarando una coppia di coordinate (riga e colonna della griglia). Se il giocatore annuncia una coppia di coordinate dove si trova una nave, l'altro giocatore deve dichiarare che una delle sue navi è stata “colpita”, oppure che le sue navi sono state “mancate”. Il gioco termina quando un giocatore perde tutte le sue navi.

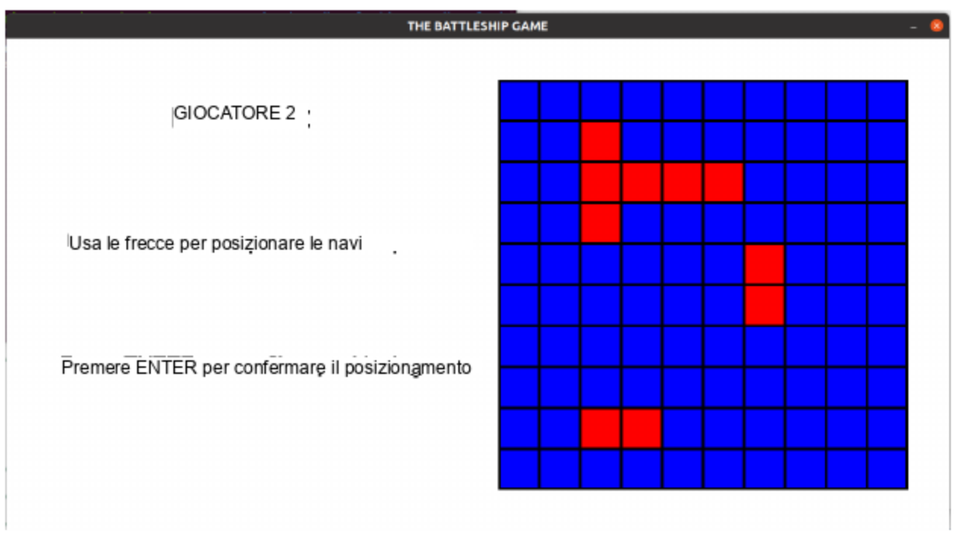
Specifiche

Come affermato in precedenza, il nostro obiettivo era implementare il gioco Battleship utilizzando Python. Per cominciare, abbiamo deciso che avremmo utilizzato il modulo Pygame, che è un insieme di moduli Python progettati per scrivere videogiochi. In effetti questo modulo ci risparmierebbe il peso di scrivere funzioni per gestire la visualizzazione dello schermo del gioco, così come per gestire gli input dell'utente. In questo modo potremmo concentrarci solo sullo sviluppo della logica del gioco, che è il fulcro di questo progetto.

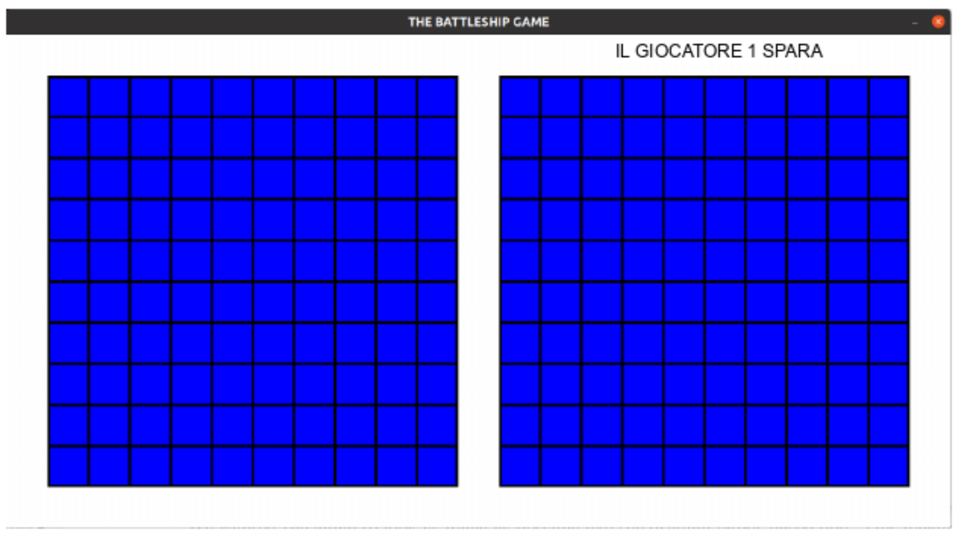
Volevamo che il gioco somigliasse il più possibile all'originale cartaceo. Ciò significava, in primo luogo, che ogni giocatore avrebbe dovuto avere una griglia per posizionare le sue navi, quindi dovevano essere disegnate sullo schermo due griglie 12x12. Pygame è tornato utile a questo scopo, permettendoci di disegnare griglie con poche righe di codice.

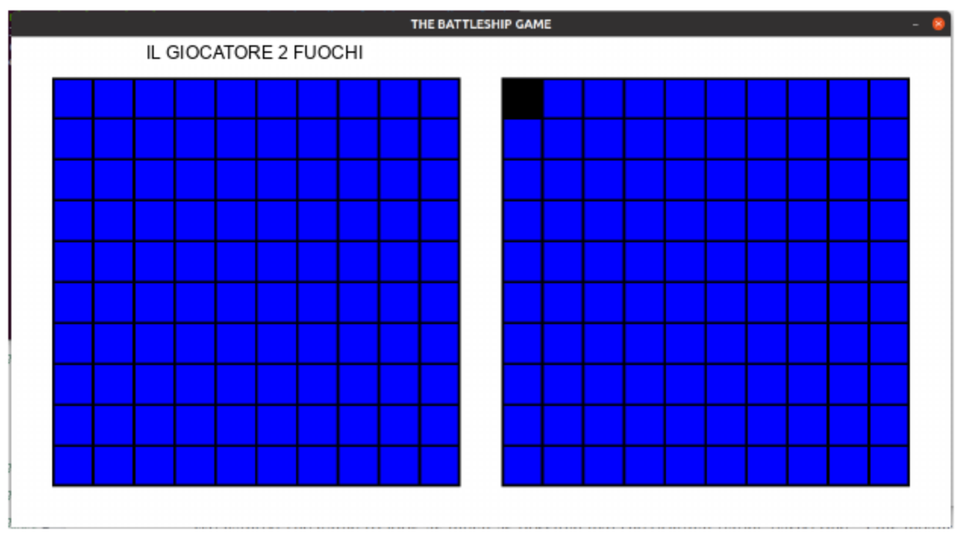
All'inizio, il programma chiede al Giocatore-1 di posizionare le sue navi sul tabellone (mentre il Giocatore-2 guarda lontano dallo schermo) e, una volta terminato, le sue navi scompariranno dallo schermo e al Giocatore-2 verrà chiesto per posizionare le sue navi. Ogni giocatore può posizionare un totale di quattro navi: due 3x1 e due 2x1. Due in verticale e due in orizzontale. Le navi appaiono sullo schermo una per una e al giocatore viene chiesto di utilizzare le frecce per spostarle sul tabellone come preferisce. Una volta premuto Invio, la posizione viene bloccata e la nave non può più essere spostata. Inoltre, il programma non consente all'utente di spostare una nave fuori dal tabellone e nemmeno di posizionare una nave sopra un'altra.



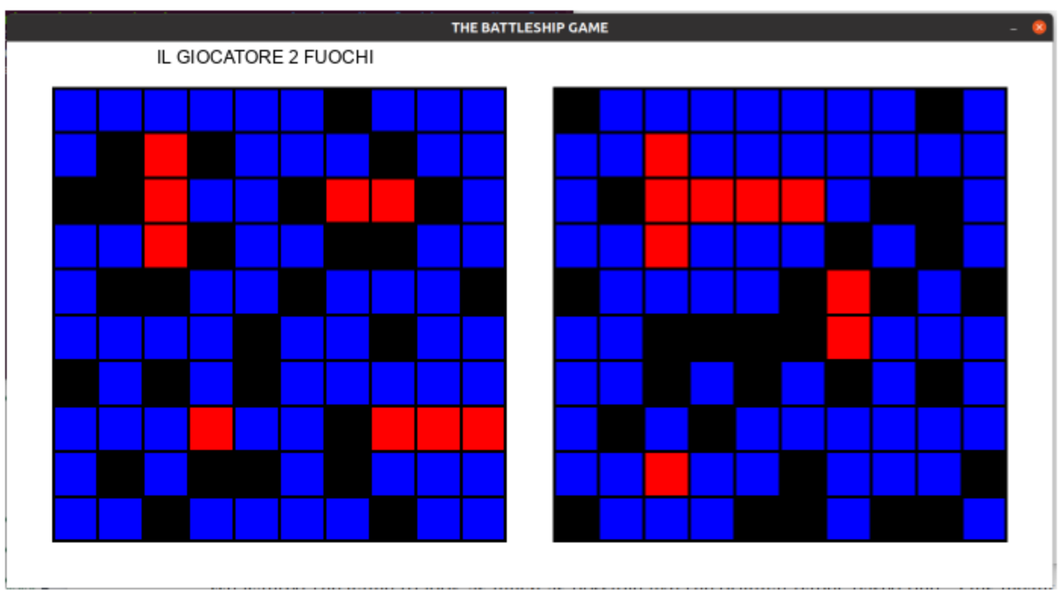


Una volta che entrambi i giocatori hanno terminato il posizionamento, la prima fase del gioco termina ed entrambe le griglie appaiono nuovamente sullo schermo, con tutte le celle che mostrano lo stesso colore (blu). Ora può iniziare la fase di attacco. Volevamo trovare un modo per sostituire la dichiarazione riga-colonna del gioco originale con un meccanismo più veloce. Pertanto, abbiamo deciso di lasciare che l'utente clicchi su una cella della griglia dell'avversario per simulare un attacco su quella cella. Inoltre, avevamo bisogno di un modo per mostrare se l'attacco ha avuto successo (hit) o meno (miss). Abbiamo usato i colori per farlo: la cella rossa indica un "colpo a segno" e la cella nera un "mancato".

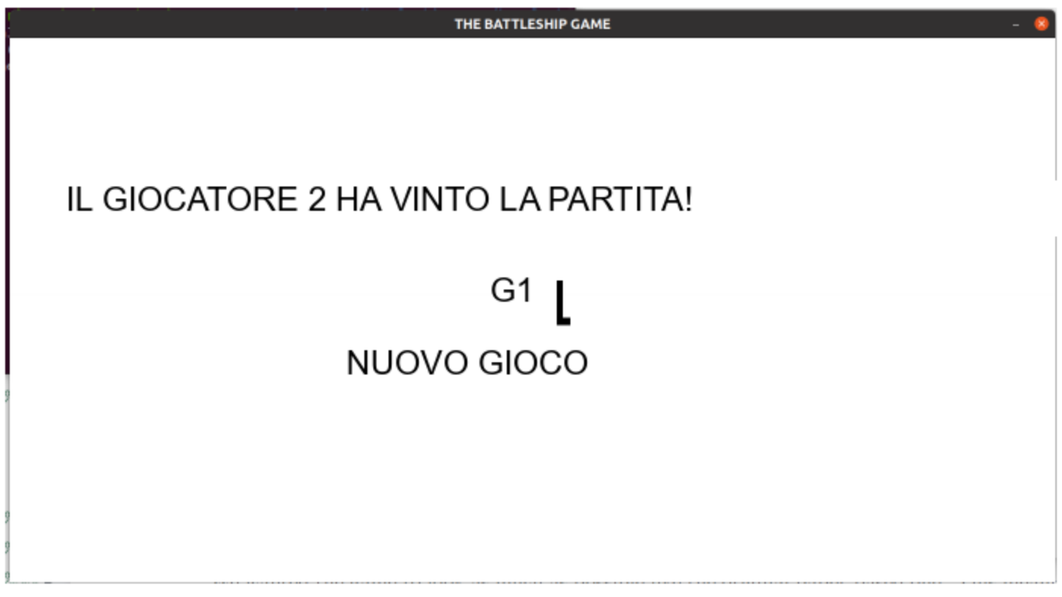




In questa fase, i giocatori dovrebbero sparare a turno l'uno contro l'altro e il programma dovrebbe ricordare (e visualizzare) tutti i successi e gli errori. Inoltre il programma impedisce che un utente clicchi erroneamente su una cella già scattata, su una delle proprie celle o anche al di fuori delle due griglie.



Il programma tiene anche traccia di quante navi ha lasciato un giocatore e termina il gioco ogni volta che un giocatore le perde tutte. A questo punto viene aggiornato il punteggio dei giocatori e sullo schermo viene stampato un messaggio di fine partita. Ai giocatori viene quindi data la possibilità di giocare una nuova partita, senza perdere il conto del punteggio complessivo delle partite precedenti.



Decisioni di progettazione

Nei paragrafi successivi verranno presentate le fasi progettuali comuni a tutte e tre le versioni. Successivamente, tre sottosezioni illustreranno le scelte progettuali specifiche di ciascuna versione.

Abbiamo deciso di creare questo gioco sul modulo Pygame. Pygame ha un oggetto di visualizzazione che può essere utilizzato come "tela" per il gioco. Il primo passo nella realizzazione di questo programma è inizializzarlo. Viene quindi creato un oggetto di visualizzazione, con determinate misure di larghezza e altezza: 1380x720. Questo rapporto è stato scelto appositamente per permetterci di utilizzare celle da 60x60 pixel per le griglie.

indipendentemente dalla CPU. Pertanto, è stato utilizzato il modulo time.Clock in modo che il display possa essere aggiornato a una frequenza fotogrammi specifica (che altrimenti sarebbe la velocità massima che la CPU può gestire). Abbiamo scoperto che un numero adatto era 30 FPS.

Utilizzando l'orologio così definito, il display viene ridisegnato ad ogni ticchettio dell'orologio. Questo ci permette di visualizzare tutti gli eventi del gioco. Ad esempio, quando una nave viene spostata sulla griglia, un nuovo segno di spunta dell'orologio ridisegnerà il display mostrando la nave nella nuova posizione (eliminando la nave precedentemente disegnata). Quando una nave viene colpita, all'iterazione successiva la cella corrispondente verrebbe ridisegnata utilizzando il colore rosso, eccetera.

Abbiamo utilizzato un dizionario per memorizzare comodamente la posizione di ciascuna cella sulla griglia e il suo stato (non toccato, colpito, mancato). La chiave di questo dizionario è la tupla delle coordinate di questa cella, mentre il valore è il suo colore (tupla di valori RGB). Blu per non toccato, rosso per colpito e nero per mancato.

Versione 0

Questa versione aveva l'obiettivo di permetterci di sviluppare e testare le principali funzioni del gioco prima di implementarle nell'attuale versione per 2 giocatori.

Volevamo sviluppare una versione del gioco in cui l'utente gioca contro la macchina. In questo gioco, infatti, il programma posiziona casualmente cinque navi sul tabellone. Quindi, il giocatore ha a disposizione un totale di 50 colpi per affondare tutte le navi. La partita è vinta se tutte le navi vengono affondate (e viene visualizzato un messaggio di vittoria). La prima azione che questo programma esegue una volta eseguito è creare un elenco di tutte le celle contenenti una parte di una nave. Chiamando la funzione getShips(), si ottiene un elenco scegliendo casualmente una flotta di navi da un file json in cui è memorizzato un insieme di flotte diverse. L'elenco delle navi conterrà un elenco di tuple, ognuna delle quali rappresenta le coordinate della griglia di una cella che contiene un pezzo di nave, e dovrebbe quindi essere rappresentata in rosso se colpita.

Il passo successivo è creare un dizionario per la griglia, dove ogni chiave rappresenta una cella (in realtà le sue coordinate) e il valore è il colore della cella. All'inizio del gioco, poiché non è stato ancora sparato alcun colpo, tutte le celle saranno blu per impostazione predefinita. Ciò si ottiene chiamando la funzione initColorDict(). Una volta posizionate le navi e stampata la griglia sul display, il giocatore può iniziare a indovinare. Un ciclo while viene utilizzato per ridisegnare lo schermo in base all'input dell'utente. Nello specifico, ad ogni iterazione, il programma cerca i click sulla griglia tramite la funzione clickCell(), e controlla se il tiro è valido. Se lo è, all'iterazione successiva la cella diventerà rossa o nera, e il contatore (cioè il tiro è su una cella blu). le munizioni disponibili diminuiranno di 1.

Inoltre il programma tiene il conto delle navi e dei proiettili rimanenti, ogni volta che uno dei due diventa 0, viene visualizzato un messaggio di vincita o di perdita utilizzando la funzione displayMessage() e il gioco viene terminato.

Versione 1

Questa versione del gioco è un passo intermedio verso lo sviluppo della versione finale. Qui due giocatori possono giocare uno contro l'altro. Questo programma è costruito sopra il precedente, riciclando la maggior parte delle sue funzioni.

La prima azione che realizza il programma è inizializzare il dizionario delle coordinate e dei colori delle celle. Lo fa chiamando due volte la funzione initColorDict(). Naturalmente, all'inizio, tutte le celle sono impostate come blu (non modificate).

A questo punto è possibile disegnare sullo schermo le due griglie, così come le celle. Questo viene fatto utilizzando il

funzioni drawCells(), drawLeftGrid() e drawRightGrid().

Ora ogni giocatore, a turno, posiziona le sue navi. La funzione placeShips() viene richiamata per Player-1. Questa funzione contiene quattro cicli while, uno per ciascuna delle navi da posizionare. Ogni nave viene generata nell'angolo in alto a sinistra della griglia (o nella cella libera dalla nave più vicina), ovvero alcune celle della griglia diventano rosse. Quindi il ciclo while cerca l'input dell'utente (tastiera). Se viene premuta una freccia, la posizione della nave sulla griglia deve essere aggiornata di conseguenza. Quindi, prima le celle diventano blu per eliminare la nave precedente, quindi viene disegnata la nuova nave facendo diventare rosse le nuove celle. Allo stesso tempo, sono in atto controlli per impedire al giocatore di spostare la nave fuori dalla griglia. Una volta che il giocatore è soddisfatto della posizione di una nave, può premere Invio. Il programma lo rileverà e interromperà il loop. La posizione della nave viene salvata nel set navi. Questo set è necessario per tenere traccia di dove si trovano le navi già posizionate, in modo che il programma non consenta all'utente di posizionare una nave sopra un'altra nave. Alla fine, questa funzione restituisce il dizionario aggiornato contenente tutte le posizioni e i colori delle celle. La variabile colori delle celle del giocatore 1 è assegnata a questo dizionario.

Dopo questa fase, tutte le navi del Giocatore-1 verranno disegnate sullo schermo. Ovviamente devono essere nascosti alla vista del Giocatore-2 prima che possa posizionare le sue navi. Questa è stata una delle decisioni progettuali più difficili del progetto.

Abbiamo deciso di occuparci di questo nel modo seguente: abbiamo inizializzato un nuovo dizionario celle-colori, chiamato indovina celle p1. In questo modo, potremmo utilizzare il vecchio dizionario per memorizzare le coordinate delle navi, mentre il nuovo dizionario verrà utilizzato solo per scopi di visualizzazione, e potrà quindi essere modificato durante il runtime a seconda degli spari dei giocatori.

Una volta fatto questo per il Giocatore-1, il nuovo dizionario potrà essere utilizzato per “affondare” tutte le sue navi. Le funzioni drawCells(indovina celle p1) e drawLeftGrid() vengono utilizzate per coprire la griglia di Player-1 con celle blu.

In questa fase, il Giocatore-2 deve posizionare le sue navi e vengono richiamate le stesse funzioni. Una volta finito,

tutte le navi vengono nuovamente affondate e la fase di attacco può iniziare.

Questa fase viene gestita utilizzando un ciclo while. La prima cosa da fare all'interno di questo ciclo è chiamare la funzione clickCell() con la variabile indovina celle p2 come argomento. Questa funzione è presa dalla versione 0. Attende un clic dell'utente su una cella della griglia dell'avversario, quindi restituisce le coordinate di quella cella. Questa cella viene quindi confrontata con il dizionario dei colori delle celle del giocatore 1, per vedere se il tiro risulta in un successo o in un errore. Il colore della cella viene modificato di conseguenza aggiornando il dizionario p2 delle celle indovinate. Quindi, anche il dizionario dei colori delle celle del giocatore 1 deve essere aggiornato e la cella viene trasformata in "affondata". Il motivo per cui ciò avviene verrà spiegato più avanti. Una volta terminata questa prima parte, lo schermo deve essere ridisegnato, e per farlo si utilizzano le solite funzioni.

È ora che il Giocatore-2 effettui il suo attacco. Questa fase viene gestita utilizzando un codice molto simile a

quello per il Giocatore-1.

Al termine di ogni attacco, è importante verificare se c'è una vittoria. Una vittoria avviene quando un giocatore ha affondato tutte le navi dell'avversario. Pertanto, viene definita una funzione per verificare quante navi ha lasciato un giocatore. Conta quanti globuli rossi ci sono in ogni giocatore. Questa funzione si chiama countShips(). quindi dizionario celle-colori. Questo è il motivo per cui è importante cambiare ogni cella rossa in "affondata" una volta colpita: in modo che non venga conteggiata come rossa. Quando questa funzione rileva che un giocatore non ha più globuli rossi, pubblica un pygame.event. Sono stati definiti due eventi personalizzati: uno per la vittoria del Giocatore-1, uno per la vittoria del

Giocatore-2.

A questo punto il codice all'interno del ciclo while arriva alla fine e quindi viene reiterato, e la prima azione che viene effettuata è verificare se è stato pubblicato qualche evento. Se viene rilevata la vittoria di un giocatore, il gioco deve essere interrotto. Questo viene fatto utilizzando la funzione displayMessage(), che ridisegnerà lo schermo con un messaggio che dice agli utenti quale giocatore ha vinto la partita.

Versione 2

Anche se la versione 1 permetteva già a due giocatori di giocare, non soddisfaceva ancora tutte le specifiche. In particolare, permetteva ai giocatori di giocare solo una partita, mentre volevamo che i giocatori potessero continuare a giocare quante partite volevano (tenendo conto del punteggio complessivo).

In questa versione abbiamo implementato tre aggiornamenti principali, basandoci anche sui feedback di alcuni nostri amici che hanno provato il gioco:

- Innanzitutto, abbiamo risolto il problema che i nuovi utenti potrebbero incontrare quando giocano al loro primo gioco. Nello specifico, abbiamo aggiunto istruzioni sullo schermo per guidare i giocatori nel posizionamento delle navi.
- Successivamente, abbiamo aggiunto del testo sullo schermo per indicare chiaramente a quale utente tocca, cioè chi dovrebbe stiamo sparando adesso.
- Infine, abbiamo aggiunto un contatore del punteggio, che viene aggiornato dopo ogni partita, nonché un "gioca di nuovo" pulsante.

Per risolvere il primo problema abbiamo sviluppato una funzione chiamata drawRules(), che guida l'utente nel posizionamento delle navi sulla sua griglia. Questa funzione viene chiamata all'inizio del ciclo principale. Utilizza i moduli Pygames per disegnare del testo sullo schermo in una posizione specifica e con un determinato carattere. Successivamente il codice è lo stesso della versione-1 per la fase di posizionamento.

Quando inizia la fase di attacco, una nuova funzione chiamata drawTurn() viene utilizzata per stampare un messaggio sullo schermo che dice ai giocatori a chi tocca tirare. Il resto del codice per la fase di attacco è lo stesso della versione precedente.

Avevamo bisogno di un modo per tenere traccia e aggiornare il punteggio complessivo di tutte le partite giocate. Ciò potrebbe essere fatto semplicemente utilizzando un elenco contenente due elementi, a cui è stata assegnata la variabile del punteggio e inizializzata su [0,0] all'esterno della funzione principale. Ogni volta che viene pubblicato un evento vincente, questa variabile viene aggiornata di conseguenza.

Infine, quando viene rilevata una vittoria, ai giocatori dovrebbe essere data la possibilità di giocare una nuova partita. Sullo schermo viene quindi stampato un messaggio di vittoria, insieme al punteggio attuale. Inoltre, è possibile premere un pulsante di riproduzione. esegue una chiamata alla funzione main, in modo che tutte le variabili del gioco (eccetto il punteggio) vengano reinizializzate. Può quindi iniziare una nuova partita. Questo pulsante è stato implementato utilizzando alcune funzioni del modulo Pygame, definite all'interno della funzione displayWinner().

Rapporto '

problemi. Come accennato in precedenza, durante lo sviluppo di questo programma, abbiamo dovuto affrontare diversi problemi, uno dei più difficili da risolvere è stato come gestire la visualizzazione delle navi sullo schermo ricordando la posizione originale in cui le aveva posizionate il giocatore. Riteniamo che la soluzione che abbiamo trovato (utilizzando due dizionari coordinate-colori) sia molto efficace, poiché potremmo utilizzare la stessa struttura dati per questi due scopi diversi.

Inoltre, abbiamo scoperto che era difficile permettere all'utente di muoversi tra le navi sul tabellone durante la prima fase. Ci è voluto del tempo per pensare a tutte le condizioni da verificare in modo che nessuna nave uscisse

della scheda e il codice che abbiamo finito per scrivere è piuttosto lungo e ridondante. Comunque funziona bene e fa quello che deve.

Alla fine abbiamo rispettato tutte le specifiche che ci eravamo prefissati per il nostro progetto e siamo particolarmente soddisfatti dei risultati. Tuttavia, ci sono ancora diverse funzionalità che potremmo aggiungere negli sviluppi futuri. Ad esempio, potremmo sviluppare un nuovo metodo di posizionamento delle navi, rendendolo drag-and-drop. Questo potrebbe probabilmente essere più comodo da usare per l'utente, quindi migliorerebbe l'esperienza dell'utente. Vorremmo anche aggiungere una classifica che verrà visualizzata alla fine di ogni partita. Ciò terrebbe traccia di Potrebbe essere archiviato come dizionario all'interno di un file json. I nomi utente di altri giocatori e tutte le loro vittorie. Una caratteristica interessante su cui potremmo lavorare è la grafica. Potremmo usare disegni di navi invece che di quadrati, e rendere anche il resto del gioco più realistico, magari utilizzando effetti sonori. Crediamo che tutto questo possa essere fatto utilizzando Pygame. Il miglioramento più grande che vorremmo apportare, tuttavia, è consentire agli utenti di giocare a questo gioco su due macchine separate utilizzando una Wi-Fi locale (crediamo che ciò possa essere ottenuto utilizzando, ad esempio, la libreria networkzero).

Conclusione

In conclusione, abbiamo trovato questo progetto piuttosto impegnativo, ma ci ha sicuramente permesso di imparare di più sui meccanismi principali di Python (e in particolare sul modulo Pygame). La nostra conoscenza e comprensione di questa lingua è stata messa alla prova e questo ci ha costretto a mettere in discussione ciò che credevamo di sapere e ad impararlo di nuovo se necessario. Spesso ci siamo ritrovati a studiare nuovamente le funzionalità base di Python. Naturalmente abbiamo avuto anche la possibilità di conoscere argomenti e biblioteche più avanzati. Tutto sommato, questo progetto ci ha dato la possibilità di approfondire le nostre conoscenze in questo campo. Inoltre è stata davvero una soddisfazione costruire qualcosa che funzioni e con cui possiamo divertirci.