Text Blaster:

A Multi-Player

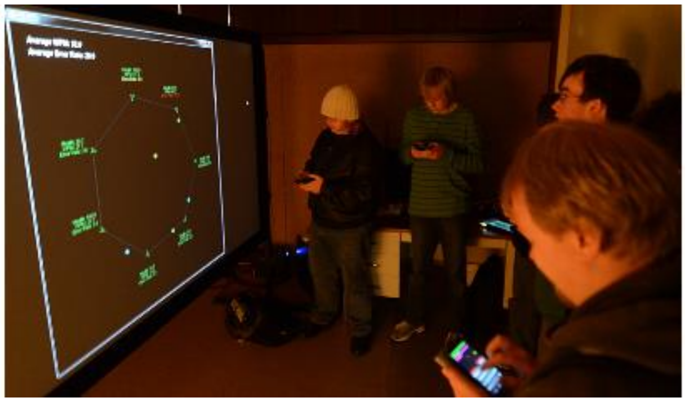
Touchscreen Typing Game

Fahimeh Mohammadi , Masoud Allahyari
2016 WS

Contents

Introduction

In this paper we describe our implementation of a Multi-player game based on (Text Blaster: A Multi-player Touchscreen Typing Game, 2014) , which is created based around users typing sentences on a touchscreen mobile device. Our game depends on players typing sentences both quickly and accurately. A player's shot is influenced by how fast a given sentence is entered. The exact path of a player's shot is influenced by how accurately a sentence is typed. Players attempt to be the last player standing by using the speed, precision, and timing of their typing to destroy competing players. Not only is this game fun, but it can also serve as a research platform for investigating performance and design questions related to touchscreen keyboard text entry. This is similar to past work that has attempted to advance text entry research by developing a game.

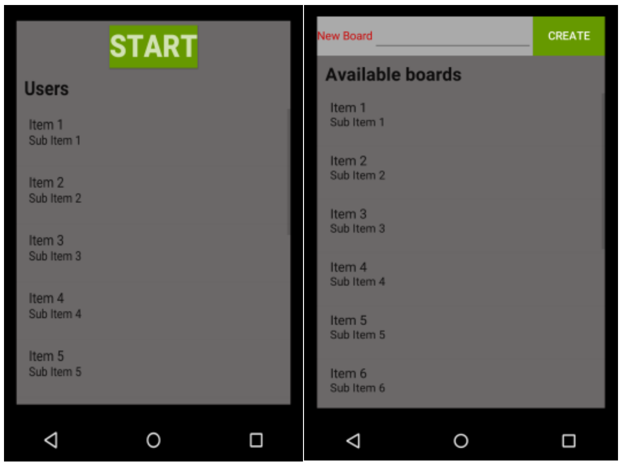


(Text Blaster: A Multi-player Touchscreen Typing Game, 2014)

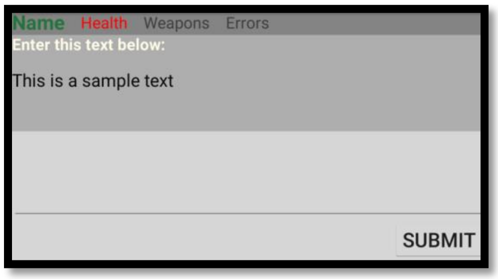
This picture shows a group playing that each player uses an Android mobile device to control their ship. The global state of the game is shown on the large projection wall. This game encourages users to enter text both quickly and accurately.

Game mechanics

The game is played by two or more players. Each player's name is located at the vertex of a polygon. One of users should make a board and then the other users could see board's name and join to the board. After all users join to the board, Admin (the user who made the board) could start the game.



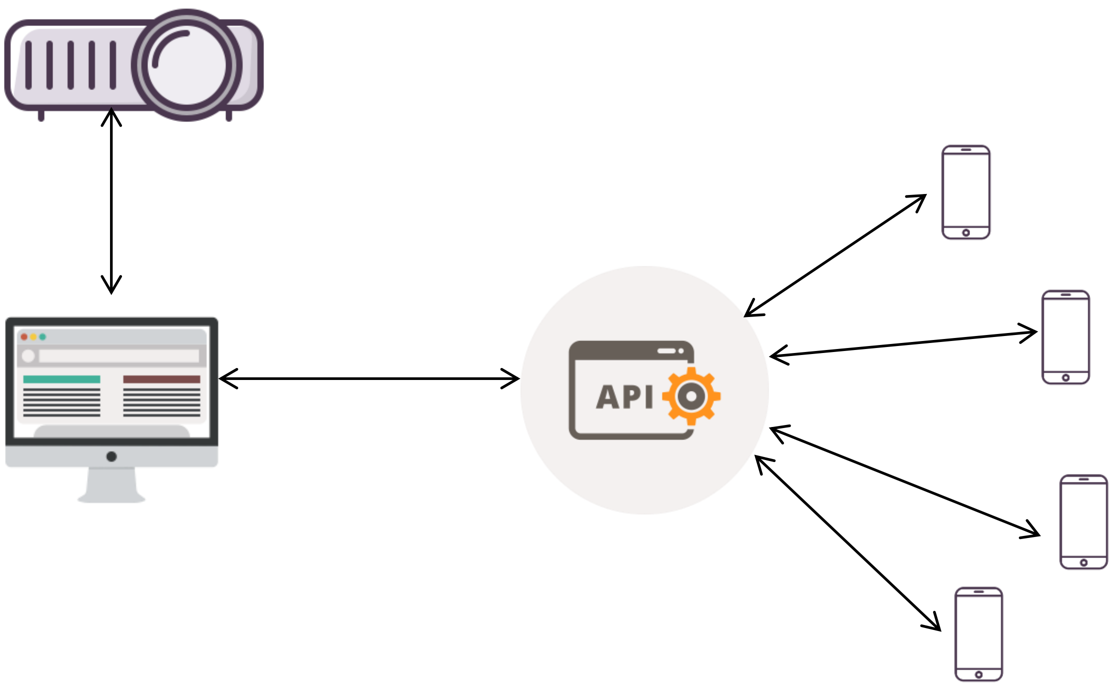
This image shows a screenshot of a player's device during the game. Text at the top is the next sentence to be entered. At the top of screen, the player's name, health, weapons, errors are shown as well.



The last two players are situated on a line and the last player remaining is declared the winner.

Project in detail

There are some ways in multiplayer games to make the communication between them but the most common way is to make an online server and then the other user communicate with it. We are using of this method as you can see in this figure:



Here we have a web client to show the game in a big screen and maybe show it on a big wall with a video projector. All data save in a database which is in the server and by using the API on that server web client and mobile apps can sync their data and communicate with each other.

How different parts communicate

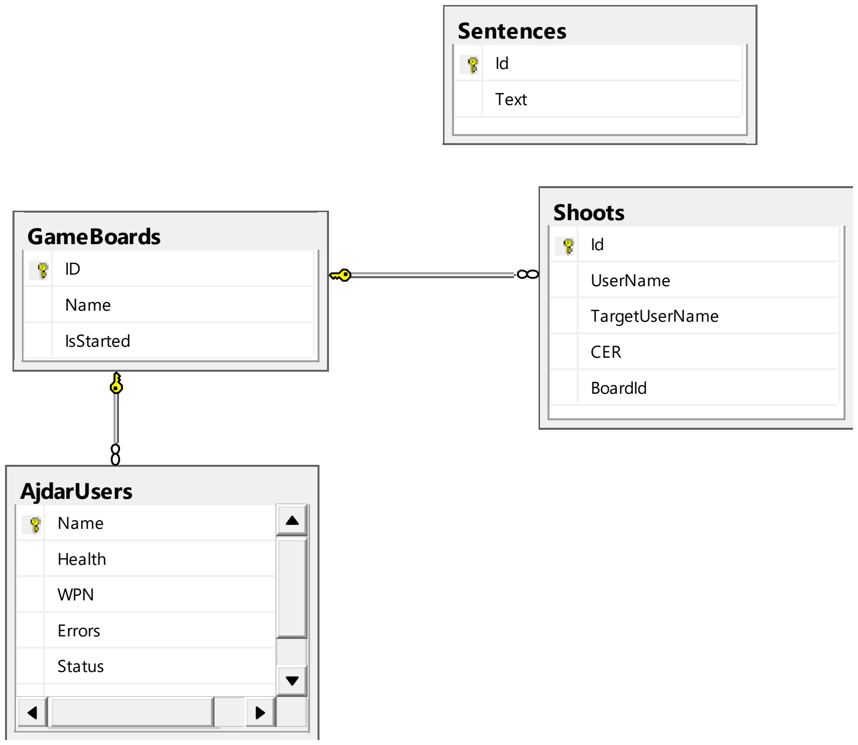
App and web client check the API every second to show the game and send a request to it to make an action like shoot or get new sentence.

We have the whole procedure here to explain how it is happening in detail:

- **Select a user name by one of the users**
- **Load available boards**
  - *App send a request to API to get available boards : boards which are ready waiting for the other users to join*
  - *API get boards name from database and return a list of them*
  - *Load boards in a list which we have in the first activity*
- **Make a board**
  - *App send a request to API to make a board with current user name as admin*
  - *API insert new records in Database for this board and username*
- **Admin App wait for other users to join**
  - *App checks the API every second and update the list of users*
  - *API get users of the selected board from database and return a list of them*
- **Other users can see the board name in their app and they select the board to join**
  - *Other app send a request to API to join the current user name to the selected board*
  - *API insert the username to data base*
- **Admin starts the game**
  - *Admin's App send a request to API to start the game*
  - *API change the status of board in database to ready to start*
- **Someone select the board in web client and start it**
  - *Web client send start request to API*
  - *API change the status of board in database*
- **Web client draws the board and users start the game procedure**
  - *Web client send and check requests every second and animates the objects*
- **Apps update user status and send shoot request during the game**
- **Apps finish the game when the health of user goes less or equal to zero**
  - *App send a request to API to remove the users*
- **Game finishes when just one user remain**

Database

Because we used “DotNet” technology we preferred to use MSSQL Server 2014. The database contains four tables as you can see in the diagram:



Sentences: *All the sentences which we want to ask the users to enter should be saved here.*

GameBoards: *Active and inactive boards are here and Started field contains three value which are null: waiting to start, true: playing and false :game is finished.*

Shoots: *New orders to shoot are here. Username is for shooter, TargetUserName is the target and CER is the angel of shoot.*

AjdarUsers: *User information is here. They have Health, Weapon, Error and status to show user alive or not.*

Web API

We used Asp.net MVC to build this API. Input should be send by GET method and output are JSON. All the processes are happening here which we are describe them below.

<u>Important functions:</u>

GetBoards(): *Return a list of boards which are waiting to start.*

MakeBoard(name,adminName): *Name is the board name and adminName is the user name of* board creator.

GetUsers(boardId): *Return a list of users which belong to the board with this boardId.*

RequestToStartGame(boardId): *Make the status of board to ready for start the first time and for the second time make the status of board is started, because we have two steps for starting the game. First is with admin app when all users join to the game then he should start the game from app. Second, someone in web browser should find the board name in list and start it as well.*

IsGameStarted(boardId): *Return the board status. Using in web browser.*

GetNewSentence(): *Return a new sentence by random.*

Shoot(boardId, userName, cer): *Select a target user by random and then save a new shoot in database.*

GetNewShoots(boardId): *Return all new shoots from database.*

Shooted(ID): *Find a shoot with this Id in database and remove it.*

UpdateUsersStatus(jsonUsers,boardId): *In jsonUsers we have a JSON format of all users with their new status then we update all user status in database with this new status.*

GetUserStatus(name,boardId): *Return an object of this user.*

Web Page

There are two pages for this part based on HTML and JQuery. First page is showing board names which are just created and has not started yet. After selecting the board, we have another simple HTML page which is showing the game using HTML5 canvas objects and communicating with server using JQuery.

<u>Important functions:</u>

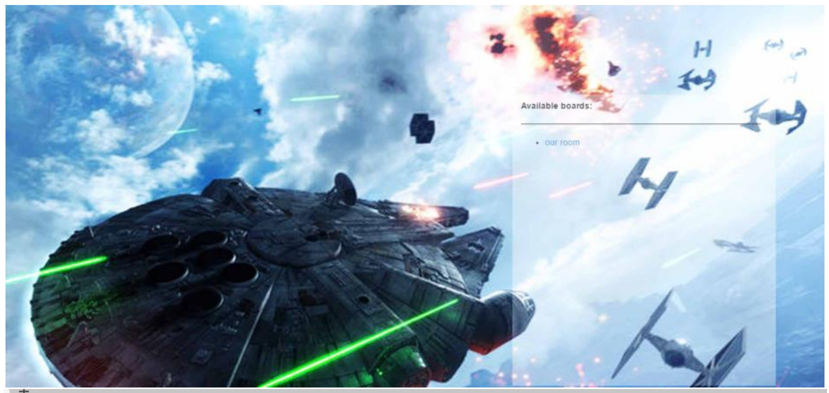
drawBoard: *Get users from server and draw the board.*

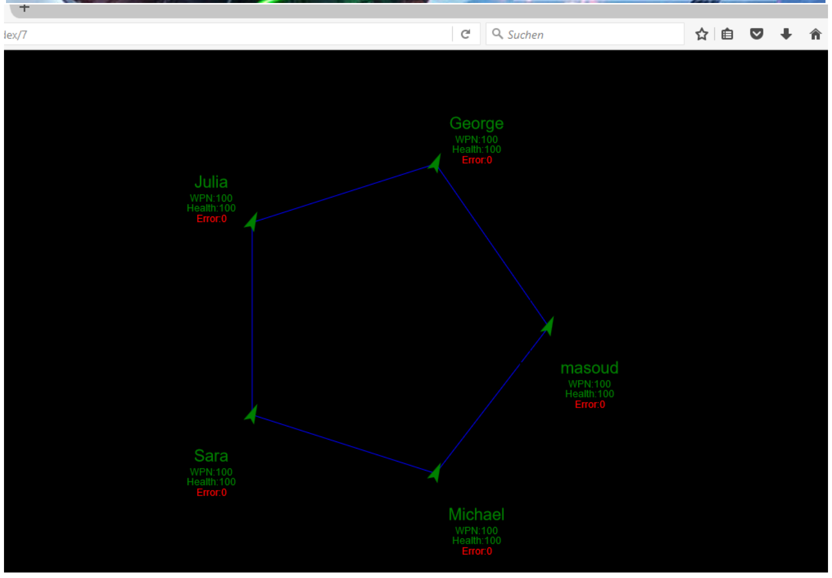
removeUser: *Remove user if it's health comes less than one.*

fire: *Send a shoot request to server*.

getNewShoots: *Get shoots every second from server.*

updateUserStatus: *Get user status every second from server.*





Mobile APP

There are three activities:

MainActivity: *here user can see boards and make a board.*

<u>Important functions:</u>

MakeBoard: *Send a request to API for making a board.*

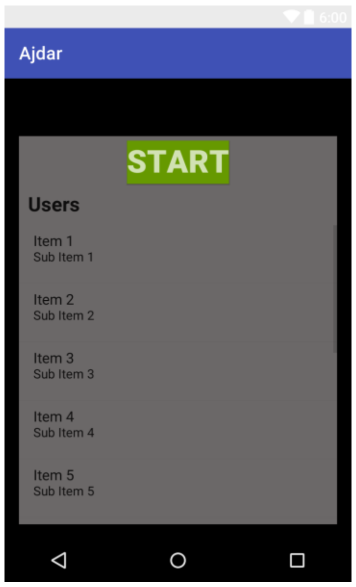
joinToABoard(gameboard): *Send a request to API for joining current user to the selected board.*

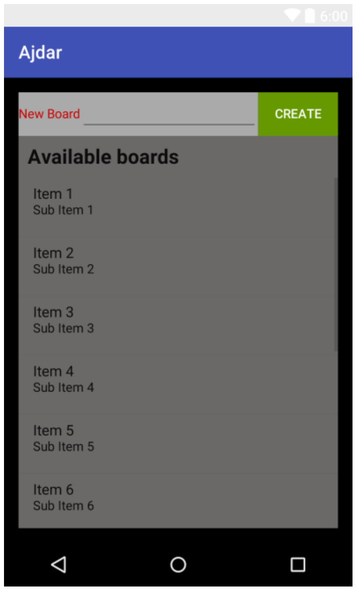
updateUsersList: *This function is called every second to update the list of users.*

checkIsStartedGame: *This function is called every second to check if the game status is stared then calls startGame().*

startGame: *Send a request to API to start the game and show the boardActivity.*

refreshList: *This function is called every second to update the list of boards.*





BoardActivity: *Here user can see a sentence, write it and send a shoot request.*

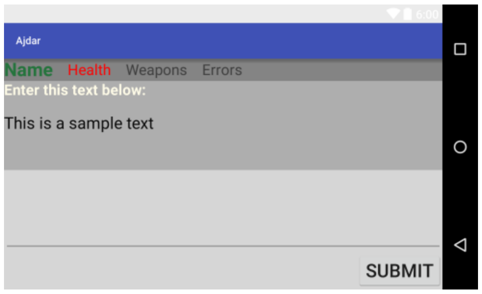
submitSentence: *The similarity between the requested sentence and user given sentence will be calculated and then a float number between zero to One will be generated. Then App sends a request to API for a shoot to a random user with this similarity.*

getNewSentence: *App send a request to ApI to get a new sentence and then shows it to user.*

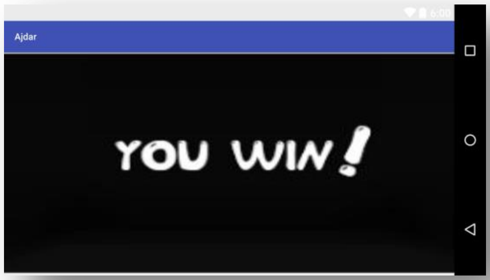
gameOver: *App will lock the user activity by showing "GAME OVER!" screen.*

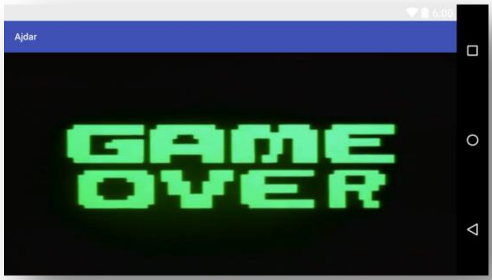
win: *App will lock the user activity by showing "win" screen.*

updateStatus: *App send a request to API to get the status of current user every second and call win or gameOver function if the health of user goes less than 1 or the status changed to 1.*



This is a sample text





Issues we faced during the project

Firstly we had some difficulty in drawing and animating objects in Html5.We had some problem for calculating the vertices of polygon which we have to show the user ships. For example if we have 5 users then we should draw a 5 vertices polygon, it sounds simple but because we used iio.net for drawing canvas we had to follow its rules and for drawing such a polygon we had to give it all of the vertices positions. What we did is find a center point position and then by following formula we calculated each position:

```
angle = 360 / users.length ;

radius=200;

x = Math.round(Math.cos(angle * Math.PI / 180) * radius + x);

y = Math.round(Math.sin(angle * Math.PI / 180) * radius + y);
```

The second issue which we faced was sentence similarity. There are some ways to calculate it but we tried a lot of them and each one has a problem for us then we decide to use a library and we found SimMetric [https://github.com/Simmetrics/simmetrics] Java library which is calculating similarity and distance metrics between two strings.

Differences between our implementation and the original paper

In the original paper ,the method of selecting target user in shoot is different with our method. We select target user randomly but in the paper it is not clear how they select it but they mentioned that the angel of ship will change by considering the similarity of user sentence.

Refrences

Text Blaster: A Multi-player Touchscreen Typing Game. **Vertanen, Keith and Emge, Justin and Memmi, Haythem and Kristensson, Per Ola. 2014.** New York, NY, USA : ACM, 2014. pp. 379--382.