INTERFACES TACTILES

Thomas GILLOT

Rémy KALOUSTIAN

Adrian PALUMBO

Arnaud ZAGO

Fiers créateurs de PartyTableInMyEzFridge

Installation du jeu

Pour installer le jeu, suivez les instructions suivantes:

- cloner le repo https://github.com/AdrianPal/PartyTableInMyEzFridge
- Installer MongoDB express (si ce n'est déjà fait)
- Dans client/TUIOManager, client/TUIOSamples et client/TUIOClient, lancer npm install
- Dans client/TUIOManager, lancer npm link
- Dans client/TUIOSamples, lancer npm link tuiomanager
- Dans server/scripts, lancer sur deux consoles différentes

  sh start_db.sh

  sh start_server.sh

- Dans client/TUIOSamples, ouvrir config.js et le modifier de la sorte :

```
module.exports = {
      // API Server
      server: 'http://[votre adresse ip]:4000',
      // Current IP
      ip: '[votre adresse ip]',
      // Current Port
       port: 3000,
};
```

- Dans client/ lancer launch.bat
- Dans votre navigateur, aller sur [votre adresse ip]:3000
- Have fun

Description du sujet

1. Explication du problème d'usage que le projet se propose de résoudre

Aujourd'hui de plus en plus d'applications de jeux de collaboration se démocratisent afin de divertir un plus grand nombre de personnes. Le problème est que ces différentes applications, que ce soit pour smartphone ou tablette ou ordinateur, sont majoritairement conçues sur le fait que chaque joueur dispose de son propre dispositif. De plus chaque application est spécialisée sur un jeu particulier qui ne plait pas forcément à la majorité des personnes réunies.
Ici nous proposons de résoudre ce problème en proposant aux personnes de se retrouver autour d'une table afin de pouvoir jouer tous ensemble à un jeu de plateau tout en composant les multiples jeux de sociétés qu'ils pourraient trouver dans leur placard afin de satisfaire tout le monde.

Notre idée démarre avec un parcours, qui est matérialisé sur la table (un chemin avec des cases). Les joueurs avancent le long du parcours, leur déplacement est déterminé par un lancer de dé. Une fois le dé lancé (depuis la table), et le joueur déplacé, un mini jeu est lancé entre les joueurs sur la table. A la fin du mini jeu, un autre joueur lance le dé, se déplace et lance un mini-jeu, et ainsi de suite. Lorsqu'un joueur arrive sur la dernière case du parcours, la partie se termine et c'est le joueur qui a le plus de points (ramassée en gagnant des mini jeux) qui gagne la partie.

Pour plus de détails sur le jeu, consulter la partie avec les scénarios.

2. le système interactif actuel pour ce projet

Tout d'abord rappelons la définition d'un système interactif, il se compose de dispositifs, de différents type d'interactions et d'utilisateurs. Dans notre cas le système interactif proposé se compose des dispositifs tels qu'une table tactile Microsoft et de smartphones de type Android ou Iphone. Les utilisateurs cibles sont toutes des personnes âgées de 7 à 77 ans qui souhaitent se divertir avec des amis. Enfin les interactions présentes dans ce projet se feront à l'aide du tactile classique, du multitouch, et du tangible. A noter que les interactions multitouch et tangibles seront uniquement présentes sur la table tactile.

3. Un scénario d'usage

Le jeu peut se jouer de 2 à 4 personnes.

Les joueurs, doivent se disposer chacun devant la table, là où un QR code est affiché. Un QR code est présent sur chaque côté de la table pour respecter l'aspect non-orienté. Pour pouvoir s'enregistrer à la partie, tous les joueurs doivent scanner les QR code à l'aide de leur dispositif personnel (i.e. smartphone). Ils sont redirigés vers une url (sur leur téléphone) pour saisir leur nom, le tag de l'objet tangible associé, et leur image de profil. Une fois qu'au moins deux joueurs sont enregistrés, on peut lancer la partie en touchant un bouton "Let's Go !".

Lorsque la partie commence, le jeu décide quel joueur tire le dé en premier. Suite à cette désignation, le plateau de jeu s'affiche donc. Le plateau avec les pions (faits avec les images de profil des joueurs) est au centre de la table. Sur chaque côté de la table, il y a à présent l'image de profil du joueur et son tag tangible. Le dé est alors affiché à côté de l'image de profil du joueur qui doit jouer. Le joueur lance le dé en le touchant sur la table. Son pion se déplace ensuite du même nombre de cases que ce qui s'affiche sur le dé. Après ce déplacement un mini jeu parmi les suivants se lance au hasard :

Pictionary :

Un des joueurs est désigné comme le dessinateur et les autres doivent deviner ce qu'il va dessiner (ces rôles sont indiqués sur le téléphone de chacun. Le dessinateur doit dans un temps imparti (un chrono sous forme de progress bar sur la table) dessiner ce qu'on lui demande (un mot clé sur son téléphone). Il a une zone de dessin réservée sur son téléphone, au fur et à mesure que son dessin est complété, il est aussi affiché sur la table pour que les autres joueurs puissent deviner de quoi il s'agit. Lorsqu'un joueur pense avoir la réponse, il remplit un champ de réponse sur son téléphone et valide en envoyant la réponse sur le téléphone du dessinateur. Le dessinateur reçoit une notification représentant la proposition d'un devineur, il peut alors choisir de la valider ou non.

A la fin du mini-jeu, la table affiche le joueur qui a gagné. On retourne ensuite sur le plateau de jeu.

Labyrinthe:

Un labyrinthe est affiché avec le départ et l'arrivée. Il n'est cependant affiché que pour un temps limité et les joueurs doivent mémoriser le chemin qui les amènerait du départ vers l'arrivée. Une fois le temps écoulé, le labyrinthe est flouté, et les joueurs disposent sur leur téléphone de flèches représentant les directions à emprunter. Ils doivent constituer avec ces flèches le chemin pour

sortir du labyrinthe. Par exemple, si pour sortir du labyrinthe il faut aller à droite puis deux fois en bas, le joueur tapera sur son téléphone la flèche droite, puis deux fois la flèche du bas. Les joueurs peuvent alors soit envoyer leur résultats soit attendre que le temps soit écoulé, ce qui déclenchera un envoi automatique des directions saisies et le labyrinthe se révèle. A ce moment là le système résout les chemins créés par chaque joueurs les uns après les autres. Les chemins sont représentés à l'aide de la couleur propre à chaque joueur. Celui qui est arrivé le plus près de la sortie gagne.

Twister :

Au départ, si il y a plus de deux joueurs, le jeu définit les équipes, sinon c'est du "1 vs 1". Le but du jeu et d'être l'équipe qui est capable de former le plus de combinaisons de couleur en un temps donné.

Au centre de l'écran sont affichées des rangées de pastilles de couleurs différentes, et on retrouve sur le côté gauche et droit de la table des instructions sur les combinaisons à réaliser.

Par exemple, une combinaison de 3 pastilles rouge et 2 pastilles bleu veut dire que le(s) joueur(s) doi(ven)t, en même temps, toucher 3 pastilles rouges et 2 pastilles bleues pour passer à la combinaison suivante. Si une combinaison est trop difficile, il est possible d'utiliser un tangible comme "bonus" pour atteindre une pastille que l'on ne pourrait toucher avec nos doigts.

Une fois que la combinaison est validée, une autre est demandée, et ainsi de suite jusqu'à la fin du temps imparti. Le gagnant est donc déterminé par son nombre de combinaison réussies.

Balls:

Le but du jeu est d'attraper des billes qui apparaissent en continu en les plaçant dans un container. Il y a un container par joueur, placé sur le côté de la table correspondant (si le joueur a pris le QRCode en haut au lancement du jeu général, son container est placé en haut). Le container a les bords de la couleur du joueur (chaque joueur a une couleur attribué dès qu'il s'inscrit dans au lancement du jeu général) et a en fond l'image de profil du joueur. Le container possède également un compteur qui représente les billes capturées.

Les billes apparaissent aléatoirement et régulièrement sur la table. Elles ont chacune une couleur de joueur, choisie au hasard. Un joueur ne peut capturer que les billes qui sont de sa couleur (et celle de son container par conséquent). Si une bille de couleur rouge est placé dans le container correspondant, elle disparait et le score du container est augmenté de 1. Au bout d'un certain temps, la bille disparait.

Moins régulièrement, des billes bonus apparaissent. Elles sont représentées avec une étoile. Elles ne sont pas capturables par les containers, il faut utiliser un objet tangible pour les capturer, en posant l'objet tangible dessus.

Lorsqu'un joueur pose un objet tangible dessus, il gagne un bonus de point et enlève des points aux autres joueurs.

Le jeu dispose de plusieurs retours pour que l'expérience utilisateur soit plus agréable. Premièrement, le jeu utilise des sons pour notifier que quelque chose est arrivé. Nous utilisons les sons lorsqu'une bille a été capturée, lorsqu'un bonus apparait, lorsqu'un bonus est capturé, et lorsque le jeu est terminé. Ces effets sonores permettent au joueur d'avoir un retour immédiat par un autre moyen que le visuel. Ainsi, l'affichage sur la table n'est pas surchargé et nous avons quand même un moyen de notifier à l'utilisateur que quelque chose s'est produit. Il y a également des retours visuels dans le jeu qui aident le joueur lors de son expérience. Premièrement, lorsqu'un bille est touchée, nous lui ajoutons une légère ombre pour la faire ressortir. Le choix de l'ombre est stratégique, il permet de faire ressortir cette bille avec un peu de relief sans toutefois masquer les alentours de ma bille. Ensuite, nous avons un retour visuel quant à l'écoulement du temps. Il y a en effet à côté de chaque container une progress bar qui diminue au fur et à mesure que le temps de jeu diminue. Ainsi le joueur à une idée globale du temps restant, ce qui est plus parlant qu'un nombre de secondes. Enfin, lorsque la partie est terminée, nous affichons le gagnant avec un icône de coupe et les perdants avec un smiley triste. Nous sommes ainsi moins dépendant de la langue et nous ancrons plus dans un côté visuel et graphique.

Le système interactif visé en utilisant les propriétés (5W1H) vues en cours pour le qualifier

5WH1 pour le dessin de Pictionary

WHO:
Il n'y a qu'un seul utilisateur qui dessine, c'est celui qui est arrivé sur la case correspondant à ce mini-jeu. L'objectif est de dessiner quelque chose de précis pour le faire deviner aux autres joueurs. Le joueur est debout ou assis avec son téléphone en main.

WHICH:
C'est une tâche individuelle, une seule personne dessine, l'enjeu est de faire deviner le dessin pour pouvoir gagner le mini-jeu.

WHERE:
La tâche s'effectue sur un smartphone. Le joueur est autour de la table.

WHEN:

Cette tâche est effectuée une fois la partie lancée et quand un mini-jeu de Pictionary a été lancé. Il n'y a pas de tâches simultanées, la tâche précédente voit le joueur qui va dessiner vérifier sur son téléphone ce qu'il doit dessiner, la tâche suivante voit ce même joueur montrer son dessin (via la table) aux autres joueurs pour qu'ils devinent ce qu'il représente.

WHAT:
La tâche manipule uniquement des données publiques, juste un dessin et le mot associé.

HOW:
Le joueur interagit sur son téléphone pour dessiner, et le dessin se complète au fur et à mesure sur la table. C'est donc une interaction tactile, qui est plus pratique que du tangible pour dessiner, surtout sur téléphone. Nous aurons sûrement besoin d'un widget de type InkCanvas, qui permet de dessiner une forme et de l'exporter en tant qu'image (nous permettra de l'envoyer sur la table une fois le dessin fini).

5WH1 pour la complétion d'une combinaison du Twister de doigts

WHO:
Une seule équipe effectue la tâche *(composée soit d'un seul joueur, soit de deux)*, et son objectif est de toucher en même temps et avec ses doigts toutes les pastilles affichées dans les instructions de combinaison à effectuer.

WHICH:
C'est une tâche individuelle *(si joueur seul)* ou collaborative *(si en équipe)*. L'enjeu est de compléter toutes les combinaisons de couleurs les unes après les autres dans le temps imparti pour gagner le mini-jeu.

WHERE:
Le(s) joueur(s) se trouve(nt) autour de la table.

WHEN:
Cette tâche arrive soit après le lancement du mini-jeu "Twister" et l'affichage des équipes *(si équipes il y a)*, soit après la complétion d'une combinaison.
La tâche suivante est soit la complétion d'une autre combinaison *(le but étant d'en effectuer le maximum en un temps donné)*, soit le retour au menu principal avec un nouveau lancé de dé.

WHAT:
La tâche manipule des données publiques, les pastilles touchées par le(s) joueur(s), mais a également accès à l'avatar et au pseudo du/des joueur(s).

HOW:
Nous utilisons l'interaction tactile avec les doigts. Nous utilisons cette méthode pour rendre le jeu plus difficile : en effet, il est difficile de réaliser les combinaisons de couleurs avec nos doigts, car ces pastilles peuvent être éloignées les unes des autres.

Toutefois, un objet tangible est utilisable comme “bonus” pour toucher une pastille qui ne peut être touchée avec la main. Dans les cas où l’utilisateur ne saurait reproduire la combinaison demandée, car la pastille est trop loin par exemple, il pourrait donc poser son tangible sur ladite pastille afin de l’aider dans la réalisation de la combinaison.
Les pastilles sont représentées par des ***ElementWidget***.

5WH1 résoudre le labyrinthe

WHO:
Chaque joueur effectue la tâche à savoir retenir le chemin permettant de se frailler un chemin dans le labyrinthe qui est affiché sur la table. Le but est que le joueur se rapproche le plus possible de la sortie du labyrinthe.

WHICH:
C’est une tâche à effectuer seul, elle est compétitive, car en même temps chaque joueur essaie de composer son propre chemin afin d’arriver le plus près de l’arrivé.

WHERE:
Le joueur est placé autour de la table et interagit avec son smartphone de manière non orienté.

WHEN:
La tâche est effectuée une fois que le mini-jeu labyrinthe est lancé et que tous les utilisateurs ont appuyé sur le bouton “Prêt”. L’utilisateur a alors quelques secondes pour mémoriser le labyrinthe puis il a une minute pour créer le chemin qui le rapproche le plus de la sortie.

WHAT:
La tâche manipule des données privées, à savoir chaque joueur écrit son propre chemin sur son téléphone. La tâche suivante est d’envoyer sa proposition puis d’attendre que les autres joueurs finissent ou que le temps imparti ce termine pour voir s’afficher le chemin.

HOW:
Nous utilisons des interactions tactiles avec le téléphone afin de créer le chemin, nous avons fait ce choix car le chemin de chaque utilisateur doit être privé. Sur ce même dispositif nous affichons donc l'enchaînement des directions sélectionnées. Une fois effectuée l’utilisateur peut envoyer son résultat en appuyant sur le bouton associé. Cette partie permet de faire travailler la mémoire de chaque utilisateur afin de recréer mentalement le chemin dans sa tête et de le transcrire en une liste de directions. Les directions sont orientées en fonction de l’utilisateur. Effectivement si le joueur est sur un la “gauche” de la table sa vision du labyrinthe ne sera pas la même que celui qui est en ”bas” de la table.
Enfin au début de ce projet il était dit qu’une interaction tangible pourrait être disponible, or après avoir passé plusieurs tests utilisateurs il nous a paru raisonnable de ne pas ajouter de complexité dans ce jeu afin de rester dans le

thème des mini jeux. Effectivement la charge cognitive aurait été trop lourde pour rester dans l'aspect "mini jeu".

5WH1 pour ramasser une bille dans le Ramassage de billes

WHO:
Un joueur effectue la tâche. Son objectif est de ramasser les billes qui sont de sa couleur. Il doit en ramasser le plus possible avant le temps imparti en les plaçant dans un container prévu à cet effet.

WHICH:
C'est une tâche à effectuer seul, elle est compétitive, car en même temps, chaque joueur ramasse les billes qui lui correspondent.
L'enjeu est d'agrandir sa collection pour ainsi gagner le mini-jeu.

WHERE:
Le joueur est placé autour de la table et interagit avec la table. Il est préférable que le joueur soit assis autour de la table. Chaque joueur est positionné de son côté de la table.

WHEN :
La tâche est effectuée une fois que le mini-jeu de Ramassage de billes est lancé, étant donné qu'il faut ramasser plusieurs billes, les tâches précédentes et suivantes sont donc ramasser une bille, jusqu'à la fin du temps imparti.

WHAT:
La tâche manipule des données publiques, à savoir les billes ramassées par un joueur, mais a également un accès de l'image de profil du joueur et au pseudo.

HOW:
Nous utilisons des interactions tactiles pour ramasser les billes, il n'est pas nécessaire de passer par un objet tangible qui rendrait le jeu moins intuitif pour ramasser les billes. Le joueur ramasse les billes en les faisant glisser (une ou plusieurs à la fois) avec ses doigts. En revanche, il faut que le joueur utilise l'objet tangible qui lui est attribué pour capturer les billes bonus qui apparaissent de temps en temps. Le joueur pose un objet tangible sur le bonus et ce dernier est pris en compte. Pour représenter les billes, nous utiliserons un widget de type ImageElementWidget, pour le container dans lequel il faut placer les billes nous utiliserons un widget de type LibraryStack, car il nous faut juste un container pour empiler les éléments ramassés. Nous n'avons pas besoin de visualiser les éléments dans le container comme par exemple avec une LibraryBar, de ce fait l'usage d'une LibraryStack est justifié.

La répartition du travail

Tout d'abord nous devons mettre en place un serveur pour communiquer entre les différents dispositifs, pour cette première tâche nous assignerons deux personnes, pendant que les deux autres se focaliseront sur la conception du plateau de jeu.
Une fois ces parties effectuées nous souhaitons réaliser de manière indépendante chacun des 4 jeux. Chaque personne sera donc responsable de son mini jeu et de son intégration.

A noter qu'à la fin de chaque mini-jeu, celui qui gagne peut continuer à lancer les dés.

Thomas Gillot a travaillé dans un premier temps sur l'écran d'accueil de l'application (avec les QR codes). Il a aussi réalisé le jeu Pictionary.
Le jeu est implémenté en Javascript (ES6) à l'aide de la librairie jQuery.
Le code est divisé en deux parties : une partie mobile se trouvant dans le fichier *pictionary.mobile.js* et une partie pour la table. La classe représentant le jeu mobile hérite de la classe *MobileHandler*, classe créée par Adrian ayant pour but de manipuler les parties mobiles des jeux.
En ce qui concerne le fonctionnement du jeu, il est très simple. Un des joueurs est choisi, côté backend, pour être le dessinateur et les autres devront essayer de trouver ce que ce dernier est en train de dessiner. Ces messages sont transmis au travers de WebSocket aux différents protagonistes. Le dessinateur décide du début de la partie, et lorsqu'il clique sur le bouton "Play" un message sous forme de WebSocket est envoyé aux autres dispositifs et apparaissent sur les différents dispositifs les éléments suivants :

- un Canvas HTML sur le mobile du dessinateur et sur la table
- un champ de réponse pour les autres

Chaque point dessiné par le dessinateur, sur son dispositif mobile, est envoyé à la table, toujours à l'aide de Web Socket, et affiché sur le canvas de la table (il peut changer la couleur de son dessin, l'épaisseur du crayon et effacer le dessin).
Si les autres joueurs veulent effectuer une proposition, ils doivent remplir le champ et envoyer la proposition transitée par socket à la table et au dessinateur. Ce dernier peut alors accepter ou refuser la proposition. Une fois le jeu terminé on revient au plateau de jeu principal, par instanciation de la classe Home.

Arnaud Zago a travaillé sur le jeu labyrinthe.

Le jeu est fait en JavaScriptVanilla, à l'aide de la librairie jQuery.

Tout le code se trouve dans le fichier labyrinthe.js. Nous retrouvons à l'intérieur de ce fichier différentes méthodes permettant de générer aléatoirement un labyrinthe. Ces méthodes sont largement inspirées du site rosettaCode que vous pouvez retrouver à l'adresse suivante. Une fois ces méthodes appropriées puis tunées l'affichage du labyrinthe a pu être effectué..
Revenons au fonctionnement global du jeu. Le fichier labyrinthe.js expose une méthode spécifique permettant de lancer ce jeu. Une fois la méthode permettant de lancer le labyrinthe exécutée le code se découpe en deux

parties, qui correspondent aux différents dispositifs utilisés (mobile et table). Dans un premier temps un bout de code se charge de vérifier l'url afin d'identifier si c'est un mobile ou non, puis le code s'enchaine et permet d'afficher les différentes actions disponibles sur ce dispositif.
Dans un deuxième le code vérifie qu'il est le dispositif est bien sur la table et établit des connexions avec les différents mobiles.
Dans ces deux parties différentes fonctions sont appelées afin d'avoir un fonctionnement correct du jeu. Toutes les communications entre la table et les mobiles sont faites par socket.
Par exemple la table va attendre que tous les tableaux comportants des propositions sont arrivés avant de déclencher la méthode resolveMaze().
Une fois que le jeu est terminé la partie de code gérant la table, déclenchera la création de l'objet Home afin de reset la vue de la table.

Au départ ce jeu devait comporter une interaction tangible, permettant de révéler le labyrinthe pendant un court instant. Après les différents retours que nous avons pu avoir sur ce jeu, il nous a paru judicieux de ne pas implémenter cette fonctionnalité. La justification de cette suppression de fonctionnalité est assez simple, actuellement les utilisateurs n'arrivent pas à comprendre rapidement comment il faut jouer. Donc pour éviter un ajout de complexité nous avons décidé de ne pas rajouter cette fonction.

Adrian Palumbo a travaillé sur son jeu Twister, sur le plateau de jeu, l'écran d'accueil ainsi que les vues mobile.

Tout le travail a été réalisé en JavaScript (ES6) couplé à jQuery, avec HTML et CSS.

- **L'écran d'accueil**

Le code est réparti dans le dossier **/src/home**. On retrouve une classe ***Home*** permettant de lancer l'écran d'accueil avec les différents QRCode pour ajouter les nouveaux utilisateurs.

Un bouton est présent au centre de l'écran et permet de lancer le jeu. Il s'agit de la classe ***StartButton*** *(toujours dans le même dossier)*, qui hérite de ***ElementWidget*** afin de récupérer l'événement tactile sur ce bouton. Cette dernière va créer un élément HTML * * avec certaines classes pour l'afficher en vert par exemple.

Aucun autre objet n'étant "touchable" sur cet écran, les autres éléments sont en HTML / CSS.

- **Le plateau de jeu**

Sur le plateau de jeu, dans l'optique de proposer des composants réutilisables par tous, on distingue deux parties : la partie utilisateurs et la partie plateau.

  - **Utilisateurs**

Les utilisateurs sont gérés par la classe ***User*** dans le dossier **/src/user**. Cette dernière se charge d'afficher les avatars des utilisateurs à la bonne position, ainsi que trois éléments autour.

Le premier élément est un bouton en forme de téléphone, qui, une fois touché par l'utilisateur, affichera un QRCode pour lui permettre de relancer la vue mobile sur son téléphone. Ce bouton est associé à la classe ***QrcodeHelper*** *(dans le même dossier)* et hérite de ***ElementWidget***. La classe créer alors un * * avec les bonnes classes et les bonnes propriétés d'affichage, et gère l'évènement "toucher".

Les deux autres éléments entourants l'utilisateur ne sont pas "touchables" : ce sont donc simplement des éléments HTML / CSS. Il s'agit de l'affichage de l'identifiant du tangible de l'utilisateur, et de son nombre de points.

  - **Plateau**

Le plateau en lui-même n'est pas "touchable", et il s'agit donc que d'éléments HTML / CSS, avec des animations à l'aide de jQuery. La classe associée au plateau se trouve dans **/src/board** et est ***Board***.

En revanche, un dé est positionné devant le joueur dont c'est au tour de jouer, et on lance ce dé en le touchant. Il s'agit de la classe ***Dice*** héritant de

ImageElementWidget. Ici, on se charge d'afficher la bonne image du dé au bon moment, en aléatoire.

A la fin d'une partie, la vue "*Winner is*" est aussi gérée par le plateau. Cette dernière est composée de l'avatar du gagnant et de son nombre de points qui ne peuvent être touchés, ainsi que de quatre boutons permettant de *"relancer la partie"* ou de *"relancer la partie avec les mêmes joueurs"*.
Ces boutons sont des ***PlayButton*** qui héritent de ***ElementWidget***. Ici encore, la classe créer des * * avec les bonnes propriétés et gère l'événement tactile.

- **Le jeu Twister**

Le jeu twister est associé à la classe ***Twister*** dans **/src/games/twister**. On distingue plusieurs vues dans ce jeu : la partie en amont, qui va constituer les équipes, la partie jeu en elle-même, et la partie "Winner is".

- **"Cliquez n'importe où pour..."**

Avant de parler des vues, je vais parler d'un événement tactile associé à la classe ***Anywhere*** dans **/src/tools** et héritant de ***ElementWidget***. Le besoin d'avoir un widget permettant de récupérer le toucher de l'utilisateur n'importe où sur la table s'est fait ressentir par mon jeu, et c'est pourquoi j'ai développé un tel widget. Ce dernier va créer un * * sur toute la table, et récupérer les événements tactiles associés.

Très rapidement, je me suis aperçu qu'un tel widget pouvait être intéressant pour tout le monde, et je l'ai alors rendu générique, en utilisant alors un système de ***callback*** une fois que l'événement tactile est récupéré.

- **Constitution des équipes**

Dans la classe ***Twister***, on appelle la méthode ***fetchPlayersAndBuildTeam*** qui va récupérer les utilisateurs et constituer les équipes. Ces dernières sont formées aléatoirement, et affichées de façon non-orientées : les avatars des utilisateurs sont utilisés pour les identifier rapidement, et ceux-ci tournent sur eux-mêmes constamment, ce qui permet à tout un chacun d'identifier les équipes rapidement.

Les utilisateurs peuvent alors, à l'aide du widget ***Anywhere***, toucher n'importe où pour passer à la vue suivante.

Avant de lancer le jeu réellement, une vue *"It's your turn"* est présente, avec le(s) avatar(s) des joueurs courants au centre. Quand ceux-ci sont prêts, ils peuvent toucher la table et lancer la partie *(grâce au widget **Anywhere**)*.

- **Le jeu en lui-même**

Le jeu est coupé en trois parties : à gauche on trouve les instructions orientées de bas en haut, au centre on trouve les pastilles et à droite on trouve les instructions de haut en bas.

Après divers essais, j'ai finalement choisi de garder les instructions sous la forme de "mini-pastilles". Lorsqu'un utilisateur touche une pastille, une mini-pastille de la couleur associée est alors colorisée pour qu'il puisse distinguer rapidement "combien de pastilles il doit encore toucher".

Les pastilles sont de la classe ***Pastille*** et héritent de ***ElementWidget***. La classe va alors créer un pour afficher les pastilles de la bonne couleur, gérer l'événement tactile et enfin gérer l'événement tangible.

- **"Winner is"**

Cette vue affiche le(s) avatar(s) de l'équipe gagnante, qui vont alors tourner sur eux-mêmes, et l'utilisateur peut toucher n'importe où pour revenir au plateau de jeu *(grâce au widget **Anywhere**)*.

- **La vue mobile**

Dans l'optique d'uniformiser nos vues, j'ai développé un gestionnaire de vues mobile. Ce dernier est dans la classe ***MobileHandler*** dans **/src/mobile**. Il va permettre à tous de mettre à jour le titre de la vue mobile, et le contenu, sans casser le style de cette dernière.

De plus, l'affichage de la vue mobile est géré dans le fichier ***index.js*** dans **/src**. Ce dernier va vérifier si certains paramètres dans l'URL sont présents et va alors afficher la vue mobile adéquate : par défaut, la vue ***MobileUnused*** est affichée, en attendant qu'un jeu utilise sa propre vue mobile *(grâce aux WebSocket)*.

Rémy Kaloustian a travaillé sur le jeu Balls et sur le plateau de jeu.

Le jeu est fait en JavaScript(ES6) avec jQuery, HTML et CSS.

Les containers sont des widgets qui héritent de LibraryStack. Ils ne prennent en compte que des éléments de la classe Ball, utilisée pour les billes. Leur html contient un paragraphe pour afficher le nombre de billes engrangées. Ils ne sont pas déplaçables, ni redimensionnables, grâce à l'utilisation des fonctions canMove() et canZoom().
Les billes sont des widgets qui héritent de ElementWidget. Elles ne répondent qu'aux événements onTouchCreation, onTouchUpdate et onTouchDeletion. Cela permet de couvrir le déplacement des billes et l'ajout d'ombre lorsqu'une est touchée.
Les bonus sont des widgets qui héritent de ElementWidget, ils ne répondent qu'à onTagCreation et disparaissent aussitôt qu'ils sont en contact avec un tag.

Le jeu est dans un fichier balls.js.

Au début du fichier se trouvent les constantes de jeu et les variables.
S'ensuit une fonction launchBalls() appelée par le plateau du jeu pour lancer le jeu Balls. Dans l'ordre, on enlève la vue avec les utilisateurs et le plateau, on récupère les joueurs avec une requête Ajax (pour voir leur noms, tangibles associés, position sur la table...), on se concentre ensuite sur l'affichage (containers des joueurs, barres de timer, puis on lance les fonctions d'apparition des billes et des billes bonus. Grâce à un window.setInterval() on peut faire apparaître des billes à intervalle régulier. On lance ensuite le timer avec un setTimeout() qui permet, une fois que le temps est écoulé, d'appeler la fonction de détermination du gagnant. Une fois le timer fini, on affiche le gagnant, on met à jour les scores avec une requête Ajax, puis on revient au plateau de jeu.