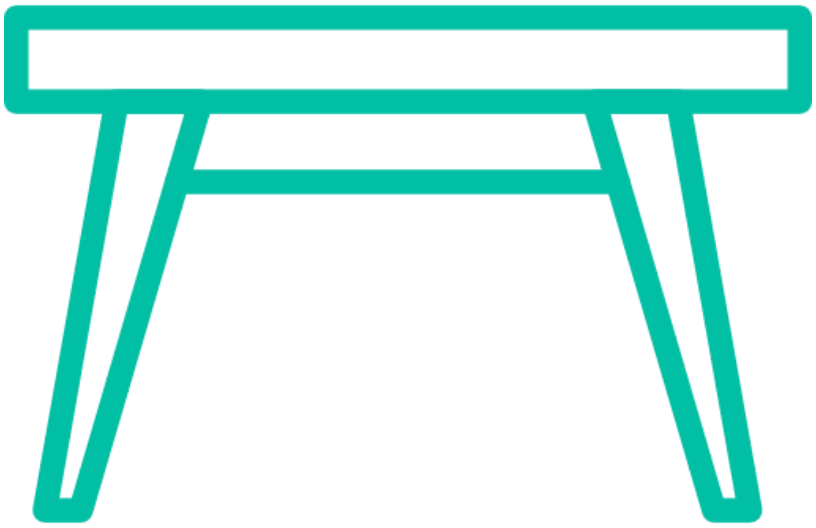
Projet de Fin d'Études



Kevin DUGLUÉ
Rémy KALOUSTIAN

Enseignants responsables : Anne-Marie DERY & Christian BREL

I - Présentation du contexte

1) Le matériel (table & objets tangibles) , interactions possibles (tactile & tangible)
2) Esprit de collaboration de la table
3) Ce qui existait déjà (Wrapper TUIO & les classes de base & les 3 repos)
4) Objectif du PFE

II - Travail réalisé

1) Planning prévisionnel & ce qui a finalement été fait
2) Les widgets de base
3) Le menu circulaire
4) La LibraryStack
5) Création d'exemples
6) Tests utilisateurs

III - Prise de recul

1) Si nous devions recommencer
2) Si nous devions continuer
3) Ce que nous laissons aux développeur
4) Qu'avons nous appris

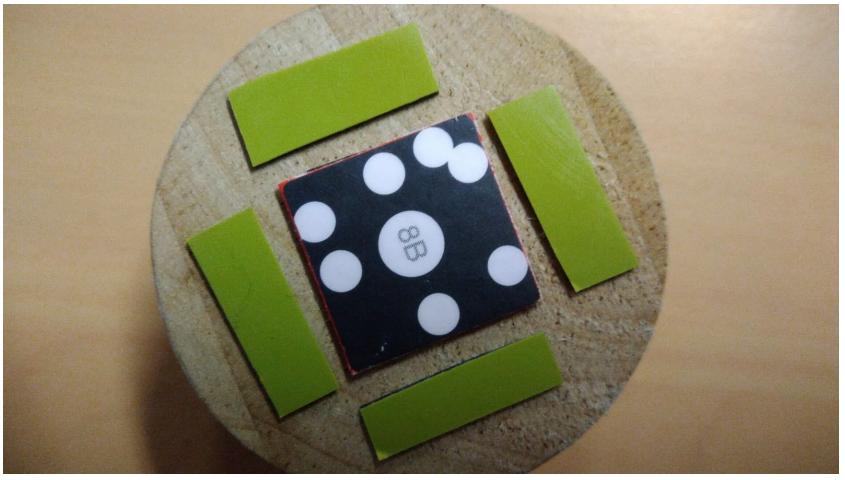
I - Présentation du contexte

1) Objectif du PFE

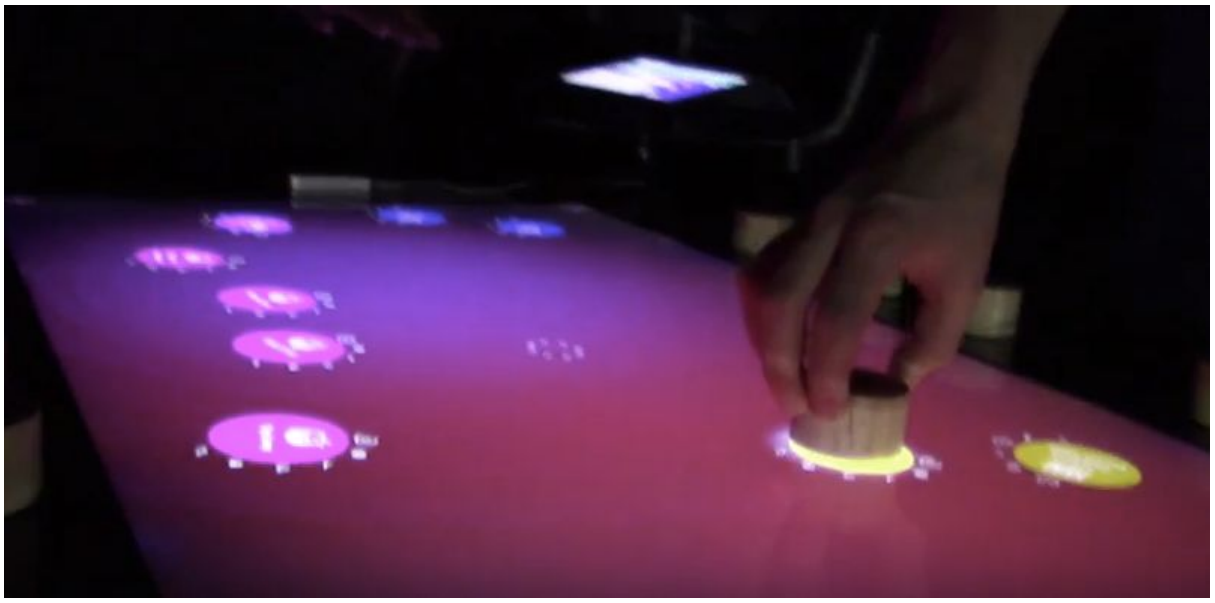
Notre objectif était de créer des widgets génériques et réutilisables dans d'autres applications. Ces widgets sont destinés à être utilisés sur une table tactile. De plus, ils doivent correspondre à l'esprit de collaboration de la table (cf. 3)). Par génériques, nous voulons dire qu'à partir d'une même classe de widget, nous pouvons obtenir, via les paramètres à la construction ou des fonctions, un comportement différent. Par réutilisables, nous voulons dire que les widgets peuvent être utilisés dans plusieurs types d'application. Créer des widgets génériques et réutilisables était d'autant plus crucial qu'ils seront utilisés en deuxième partie de semestre par nos camarades de classe.

2) Matériel et interactions

Nous avons à notre disposition une table tactile Samsung SUR40. Elle permet de détecter le tactile mais surtout elle permet des interactions avec des objets tangibles. Un objet tangible peut-être n'importe quel objet (pion, objet du quotidien) sur lequel on colle une petite carte qui sert d'identifiant, appelée tag (voir image ci-dessous).



Le tag est lisible par la table, qui nous retourne son identifiant. Ce type d'interaction est utile si on veut faire manipuler des objets au public visé, par exemple des capsules de café comme dans la photo ci-dessous.

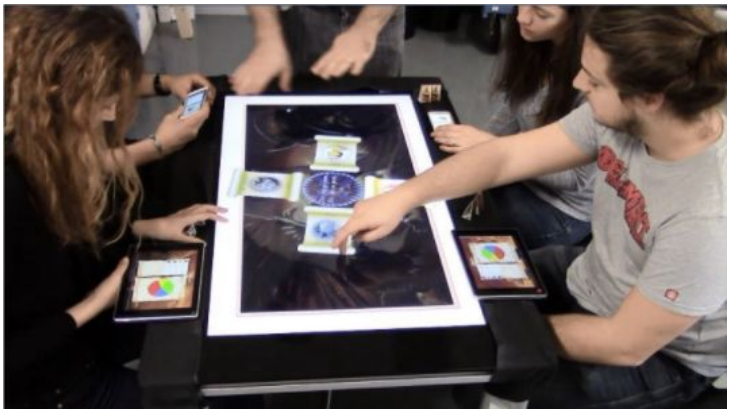


3) Esprit de collaboration

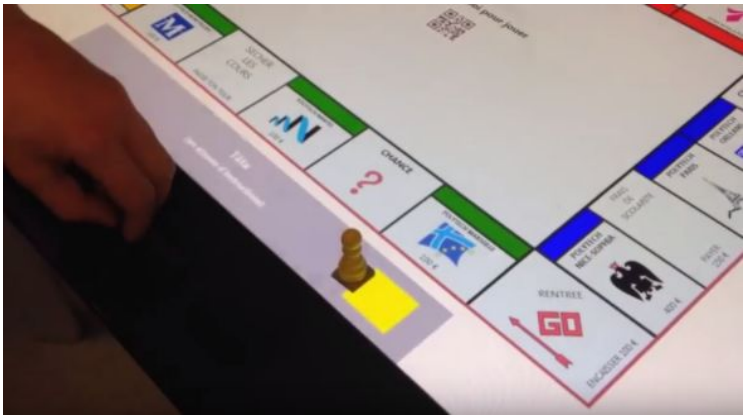
Nous pourrions penser qu'au final, la table est juste un ordinateur tactile. Nous l'avons pensé au départ de notre PFE, et ce fut une erreur. En effet, avec une table tactile, il faut prôner un esprit de collaboration. Cela veut dire que les applications doivent être utilisables par plusieurs personnes en même temps tout autour de la table. Elles doivent donc utiliser des widgets non-orientés, c'est à dire visibles quelle que soit la position de l'utilisateur autour de la table, ou alors orientable (les faire pivoter à la main par exemple).

En IHM, des applications ont déjà été réalisées sur la table dans cet esprit:

Conquistador
Un jeu de stratégie non orienté et jouable à quatre.



Monopolytech
Un jeu de monopoly sur le thème de Polytech jouable à quatre au plus



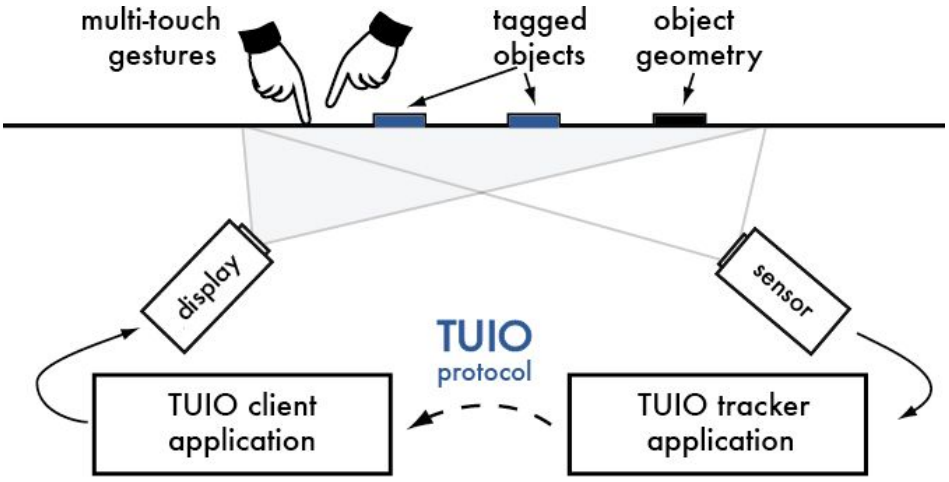
Cluedo
Un jeu de cluedo sur table jouable à 6 au plus

Pour respecter l'aspect collaboratif et non-orientation, les widgets doivent respecter à certain nombre de comportement. Il faut dans un premier temps qu'ils soient **déplaçables** facilement, que ce soit tactilement ou via un objet posé sur le widget. Il faut également avoir la possibilité de **pivoter** le widget. En effet, dans un contexte de non-orientation et où chacun autour de la table puisse manipuler le widget, la rotation d'un widget est nécessaire.
Dans un contexte de collaboration, le **redimensionnement** d'un widget peut également être intéressant permettant de demander par exemple un avis sur une image,...

4) Ce qui existait déjà

Jusqu'à présent, pour développer des applications destinées à un usage sur table, les développeurs utilisaient une librairie fournie par Microsoft. Elle était très complète et fournissait un ensemble de widgets adaptés pour la table avec la prise en charge des interactions tactiles et tangibles. Cependant, cette bibliothèque n'est actuellement plus maintenue. Elle permettait d'utiliser des widgets tels que le InkCanvas (zone de dessin), la LibraryBar, la LibraryStack (toutes les deux décrites plus loin dans le rapport) et bien d'autres.

Une alternative à cette librairie est le framework open-source TUIO. Il definit un protocol commun et une API pour utiliser les différentes interactions qu'offre la table.



Le première partie de TUIO concerne le tracker, qui permet de détecter les évènements de la table (tactiles et tangibles) et de les traduire au format TUIO. Nous ne sommes pas intervenu sur cette partie étant donné qu'elle est effectué par le logiciel **SurfaceToTUIO**.
De l'autre côté, nous avons la partie orienté client. C'est ici que l'on récupère les différents évènements fournit par le tracker et qu'on les interprète. Nous sommes donc intervenus sur cette deuxième partie. Les langages utilisés ont été Javascript pour le code métier, et HTML/CSS pour l'aspect graphique des widgets.

Avant le début du PFE, un travail préliminaire avait déjà été effectué, notamment au niveau de l'architecture que doit avoir la librairie. Ainsi, le coeur de la librairie avait déjà été implémenté notamment au niveau de la récupérations des évènements du tracker. Un certain nombre de classe ont également été déjà créé comme par exemple TUIOTouch définissant un touch sur la table, TUIOTag définissant un objet tangible ou encore la classe la plus importante TUIOWidget définissant une widget ainsi que toutes les fonctions obligatoires d'un widget, afin qu'on puisse pendant les 4 semaines du PFE nous concentrer uniquement sur la création de widgets.
Comme dit précédemment, un widget doit obligatoirement hérité de la classe TUIOWidget. Celle-ci définit un certain nombre de fonctions à surcharger lié aux interactions tactiles et tangibles :

- onTouchCreation(TUIOTouch) / onTagCreation(TUIOTag)
- onTouchUpdate(TUIOTouch) / onTagUpdate(TUIOTag)
- onTouchDeletion(TUIOTouch) / onTagDeletion(TUIOTag)

Lorsqu'un évènement sur la table à lieu, ces fonctions de toutes les widgets instancié sont appelés (par exemple un objet est posé sur la table, la fonction onTagCreation() de tous les widgets sera appelée).
Le réel enjeu de ce PFE aura donc été de définir à quel widget appartient cet évènement sur la table, implémenter les différentes interactions de bases comme le déplacement, la rotation et le zoom/dezoom, uniquement à partir des coordonnées des évènements reçus à travers les différentes fonctions décrites justes au dessus.

II - Travail réalisé

1) Notre planning

Durant la première phase du PFE,, nous avons établi un premier planning pour la phase des 4 semaines de développement. Celle-ci prenait uniquement en compte le développement de widgets, avec un peu de temps vers la fin réservé à la création d'exemples :

> **Semaine 1:** ElementWidget(classe abstraite qui sert de base), ImageElementWidget (une image déplaçable), VideoElementWidget(une vidéo déplaçable)
> **Semaine 2:** CircularMenu (un menu circulaire qui apparaît quand on pose un objet sur la table)
> **Semaine 3:** LibraryBar(un container qui affiche les éléments dans plusieurs sections), LibraryStack (un container qui empile les éléments les un au-dessus des autres)
> **Semaine 4:** InkCanvas, Exemples

Au final, nous n'avons pas réalisé tous les widgets mais avons intégré des initiatives nous permettant de juger de la pertinence de notre travail (plus d'exemples, les exemples sont réalisés au fur et à mesure, mise en place de tests utilisateurs…)

> **Semaine 1:** ElementWidget, ImageElementWidget, VideoElementWidget, Exemples, documentation en ligne
> **Semaine 2:** CircularMenu, Exemples, documentation en ligne
> **Semaine 3:** LibraryStack, Exemples, Mise en place des tests utilisateurs, documentation en ligne
> **Semaine 4:** Tests utilisateurs, LibraryBar

Le changement de planning par rapport au prévisionnel nous a permis de prendre du recul sur notre travail et de voir si notre travail était efficace à travers les exemples et les tests utilisateurs.

2) Choix sur les comportements

Nous avons vu qu'un widget, pour être utilisé sur table, peut ou doit comporter un certain de nombre de comportements essentiels comme le déplacement, la rotation ou encore le redimensionnement. Dans certain cas, un moyen de suppression ou de duplication peut s'avérer utile comme pour les widgets de base présentés juste après.
Nous avons décidé, pour presque chacuns de ces comportements, d'utiliser soit une interaction tactile ou tangible.

En effet, pour certains comportements l'interaction tactile est beaucoup plus intuitives (**déplacement** et **rotation**) tandis que pour certains le comportement peut facilement être caractérisé par un objet comme par exemple une gomme pour la **suppression**, ou une loupe pour le **redimensionnement**.
Concernant le tactile, les différents comportements se font de la manière suivante : **un** doigts pour le déplacement d'un widget, rotation de **deux** doigts sur le widget pour le pivoter et éloignement/rapprochement de **deux** doigts sur le widget pour le zoomer/dézoomer.
En ce qui concerne la suppression et la duplication, nous avions pensé à utiliser respectivement 3 et 4 doigts. Cependant, nous jugions premièrement que cela n'était pas du tout intuitifs mais surtout inapproprié dans un contexte où plusieurs personnes peuvent manipuler une même donnée en même temps. Cela aurait amené trop fréquemment des suppressions/duplications de widget involontaires.

3) Les widgets de base

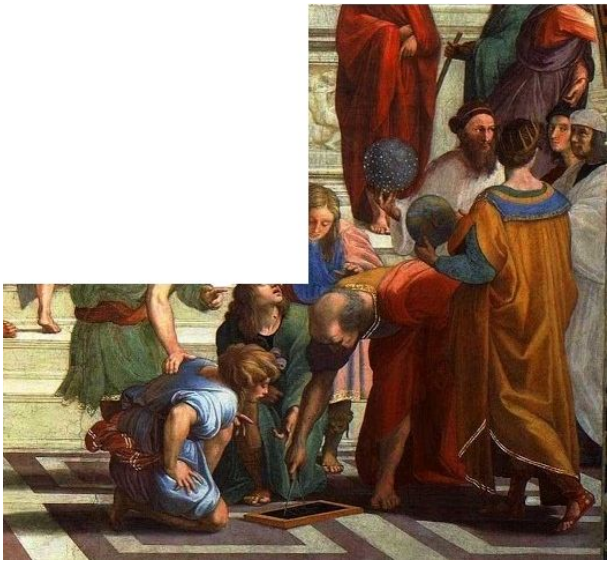
ElementWidget n'est pas un widget à proprement parler. C'est en effet une classe abstraite qui regroupe un ensemble de comportements basiques et communs. Il n'y donc aucun aspect graphique lié à un ElementWidget. Cet aspect là est géré au niveau des différentes classe filles.

Dans notre cas, nous avons implémenté jusqu'à présent deux classes filles dont nous jugions les plus intéressantes : ImageElementWidget représentant une Image et VideoElementWidget représentant une vidéo. En plus de l'aspect graphique des widgets fait en HTML et CSS, les classes filles permettent d'ajouter de nouveaux comportement tel que play/pause pour VideoElementWidget

L'intérêt ici est donc principalement de ne pas avoir de duplication de code entre les différentes classes et de permettre, dans le futur, d'ajouter d'autre type d'ElementWidget en se souciant uniquement de leur comportement spécifique.

Afin de rendre le widget le plus générique possible, nous avons fourni un certain nombre de fonction permettant d'adapter le widget en fonction du contexte d'utilisation. On peut donc permettre de choisir les différents tags associés aux comportements du widget, mais on peut également activer/désactiver ces mêmes comportements.

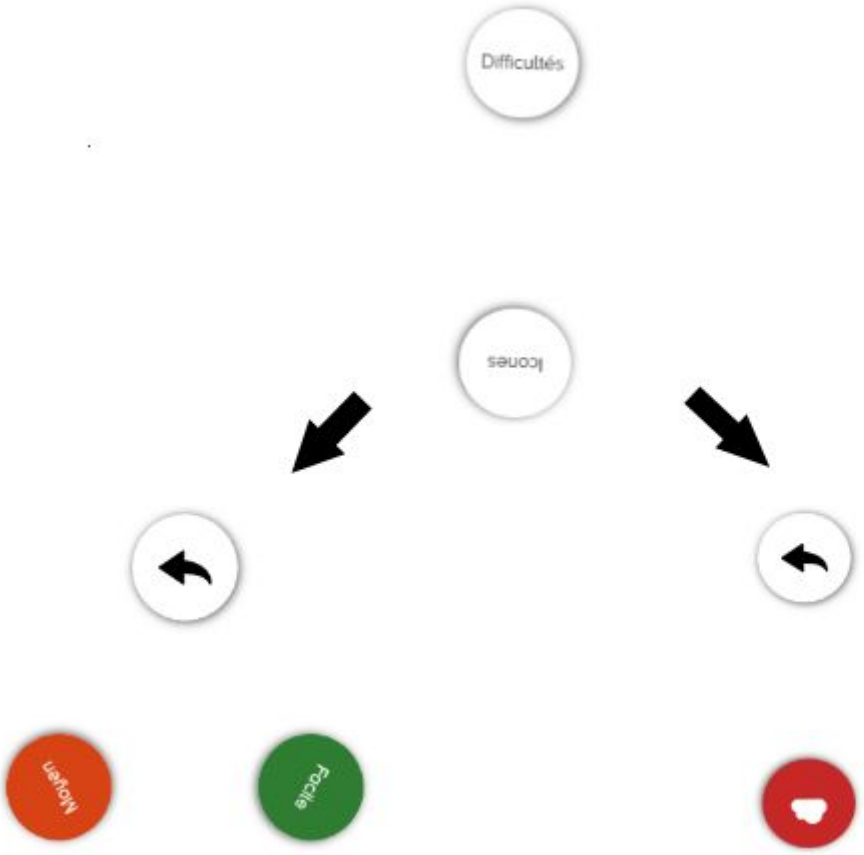
Mais ElementWidget possède quelques limites. La principale se situe au niveau de la détection de si un évènement appartient ou non à un ElementWidget. Etant donné qu'un ElementWidget est rectangulaire, jusqu'à présent nous detections sa position et en fonction de sa hauteur/largeur nous vérifions si un événement (tactile ou tangible) appartient au widget ou non. Si le widget a subit une rotation, il faut au préalable le pivoter pour le faire revenir à sa position initial (sans rotation) et faire subir cette même rotation au point d'événement (correspondant aux coordonnées X et Y du touch/objet).
Cependant, cela est intuitif pour l'utilisateur uniquement si toute la surface d'un ElementWidget est remplie par l'image ou la vidéo.



En effet, si l'image n'est pas rectangulaire(voir figure ci-dessus), cela peut gêner l'utilisateur. Une première solution à laquelle nous avons pensé est d'effectuer une analyse du contenu lors de l'instanciation d'un ElementWidget.
Par exemple pour une image, il est possible d'identifier et différencier les pixels contenant une couleur et les pixels transparents. A partir de là, si l'évènement se situe dans la zone transparentes, on n'accepte pas l'interaction avec le widget.

3) Le menu circulaire

La plupart des applications, que ce soit celles présentes sur atelierIHM (site vitrine du parcours IHM) ou de manière général, nécessitent un menu. En effet, c'est un élément central de l'application qui permet d'effectuer des actions spécifiques ou de naviguer dans l'application sans pour autant que les boutons nécessaire soient constamment visibles.
Mais, pour pouvoir être adapté à un usage sur table, il a fallu respecter une des contraintes principale : la non-orientation.
Pour cela, nous avons choisi d'associer le menu à un objet tangible, et de rendre le menu circulaire. Ainsi, cela permet à toute personne de la table de pouvoir lire le menu (ou au moins un item de celui-ci), et d'avoir la possibilité de le pivoter en tournant l'objet sur la table.

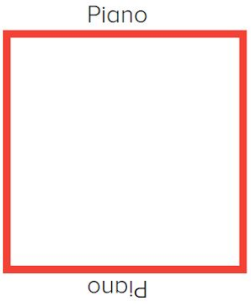


Le CircularMenu peut évidemment comporter des sous-menus. Au niveau de la structure interne, nous avons choisi de représenter le menu sous la forme d'un arbre.
Nous affichons à chaque fois les fils du noeud sur lequel on se trouve, et par défaut à la pose de l'objet sur la table nous affichons la racine. Lorsque l'utilisateur appui sur un item du menu, soit on entre dans un sous-menus si celui-ci possédait des fils, soit on exécute une fonction préalablement défini.

Lorsque l'on créé un item du menu, nous avons laissé aux développeurs deux choix : soit il choisit d'y mettre du texte, soit une icône. Dans tous les cas, il peut en plus choisir la couleur de fond et du texte/de l'icône. De plus, la taille de la police du texte à l'intérieur de chaque item s'adapte en fonction de sa longueur. Enfin, en fonction du nombre d'item présent dans le menu, celui-ci s'adapte pour toujours avoir un affichage homogène c'est à dire que chaque item sont espacés du même angle.

4) La LibraryStack

Le dernier widget développé est une LibraryStack. A l'inverse des deux widgets précédent qui ne possédaient pas d'interactions directes avec d'autres widgets, ici la LibraryStack agit comme un container. Il s'agit plus précisément d'une pile d'ElementWidget avec différentes interactions possibles dessus.



Le but ici est donc de pouvoir empiler différents ElementWidget à un même endroit, en pouvant choisir quel type d'ElementWidget est accepté (soit Image, soit Video ou soit les deux). L'intérêt dans une application est de pouvoir rassembler différents éléments ayant un sens commun, comme par exemple un tri photo de manière collaborative. Cet exemple a été développé pour le test utilisateur qui est présenté un peu plus loin dans le rapport.
Il existe deux type de LibraryStack : une qui une fois instancié apparaît constamment sur l'application, et une autre qui, associé à un objet, apparaît uniquement lorsque celui-ci est posé sur la table. Cela permet d'élargir le contexte d'utilisation de ce widget et d'ajouter un sens physique à la LibraryStack (par exemple un appareil photo où lorsqu'il est posé, les photos apparaissent sur la table dans une LibraryStack).
Outre les interactions essentiel de tout widget tel que le déplacement, la rotation et le redimensionnement, l'ajout et la suppression d'élément à la stack se fait via drag'n drop.

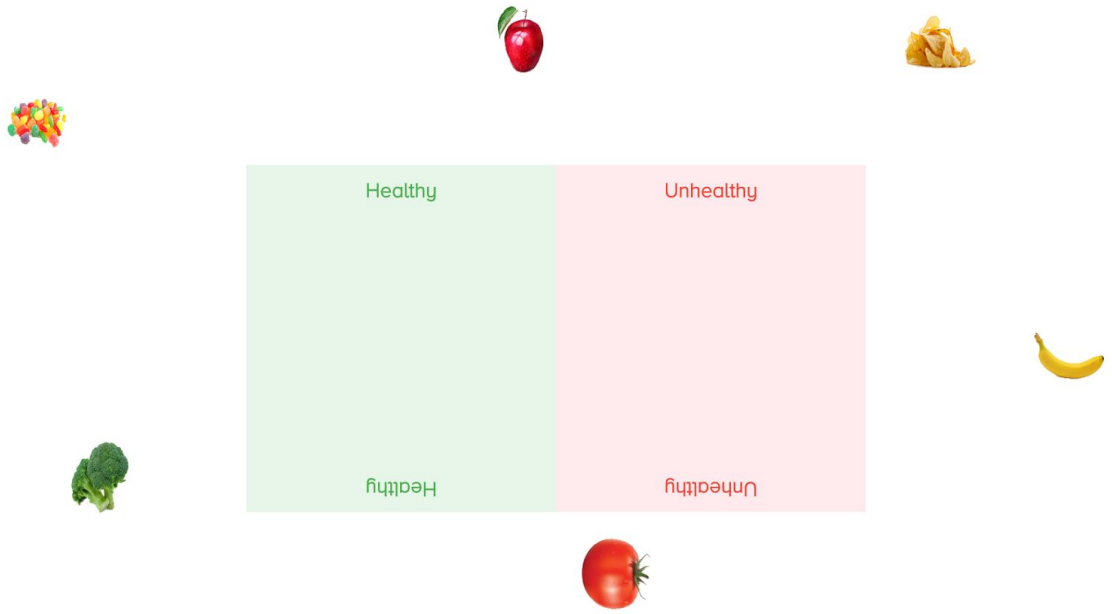
5) Création d'exemples

Pour illustrer un exemple d'utilisation des widgets, nous avons créé chaque semaine des exemples, c'est à dire des mini applications. Le but de ces applications n'est pas de proposer une expérience ou des fonctionnalités à l'utilisateur.

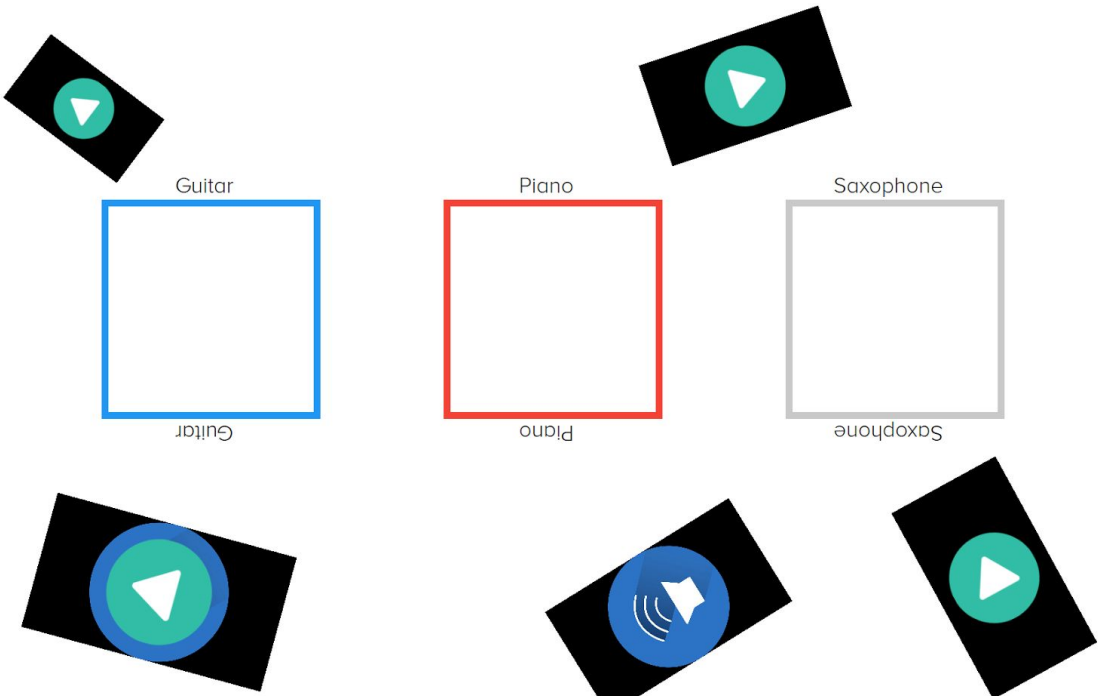
Pourquoi des exemples ?
D'une part, les exemples permettent de vérifier que nos widgets ne sont pas orientés et sont utilisables où que l'on se trouve autour de la table. Ensuite, ils permettent de tester l'utilisation de nos widgets dans un cadre d'application réel et concret. Cet aspect nous permet non seulement de voir si le widget fonctionne, mais également de tester les interactions tangibles et tactiles en quête d'amélioration. Ces exemples permettent aussi d'utiliser notre propre code, comme si nous étions des utilisateurs ayant accès à la bibliothèque.

Nous pouvons alors constater si il faut améliorer, corriger ou rendre plus pratique certains aspects de notre bibliothèque. Enfin, ces exemples nous ont servi de démonstration pour nos enseignants tuteur. Il était ainsi fort pratique de démontrer la généricité et la pertinence d'un widget à travers un véritable exemple.

Description des différents exemples



Dans cet exemple, l'utilisateur dispose de plusieurs aliments(construits avec la classe ImageElementWidget) et doit les ranger dans les cases "Healthy" et "Unhealthy" (Bon pour la santé et Mauvais pour la santé). Cet exemple servait à tester et démontrer l'utilisation possible des ImageElementWidget, dans le déplacement, la rotation, et le redimensionnement.



Dans cet exemple, l'utilisateur dispose de plusieurs sons d'instruments (construits avec la classe VideoElementWidget). Il doit les faire jouer et déplacer le son sur la pile de l'instrument qui lui correspond (les piles sont faites avec la classe LibraryStack). Cet exemple permet de tester et démontrer l'utilisation de VideoElementWidget (déplacement et lecture de vidéo) et LibraryStack (ajout d'un élément et suppression d'un élément).

Ce dernier exemple est sûrement celui qui se rapproche le plus d'une application réelle. Le but est ici de recréer un puzzle à partir des différentes pièces proposées (construites avec la classe ImageElementWidget). Nous utilisons un menu (avec la classe Circular Menu) pour choisir la difficulté et le puzzle à reconstruire.

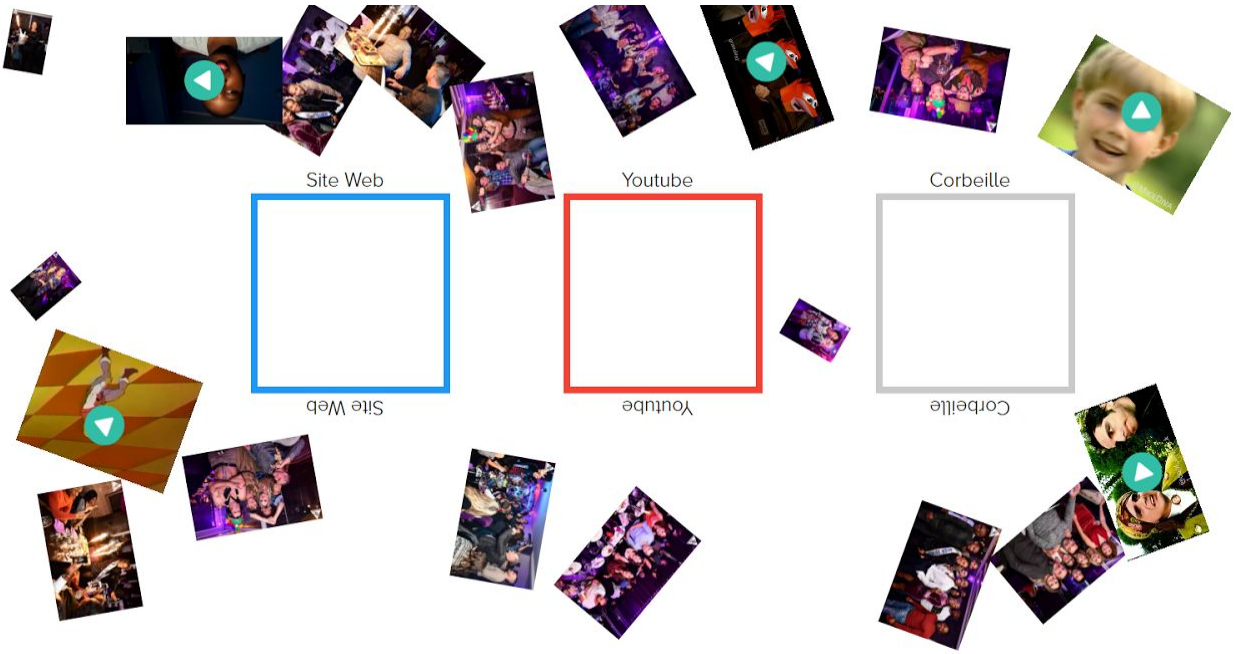
Cet exemple nous permet d'une part de tester le menu au niveau de l'interaction tangible (nous utilisons un objet tangible pour faire apparaître le menu et le faire pivoter) et de ses sous-menus. Il nous permet aussi de tester ImageWidgetElement, au niveau de la position, la rotation et le redimensionnement. Chaque niveau de difficulté joue sur des propriétés différentes. En facile, seule la position des pièces est aléatoire. En moyen, la position et la rotation est aléatoire. En difficile, la position, la rotation et les dimensions sont aléatoires. Cela a donc permis de montrer la généricité de ImageElementWidget, où le widget peut être utilisé de manière différente et dans des contextes différents.

6) Tests utilisateurs

Nous avons consacré une partie de notre temps à l'élaboration de tests utilisateurs. Nous avons demandé à quatre des nos camarades de promotion (Thibaut Terris, Alicia Marin, Gunther Jungbluth et Thomas Monzein) de participer à deux tests utilisateurs.

Le premier est un test de développement. Les testeurs, chacun sur un ordinateur, avaient accès à notre documentation(https://github.com/AtelierIHMTable/TUIOManager/blob/master/docs/Widgets.md), à un fichier de code, et devait créer les widgets demandés, puis vérifier sur leur navigateur si le résultat était celui attendu.

Le deuxième est un test d'utilisation. Nous avons créé une application de test pour l'occasion. Les testeurs devaient par groupe de deux autour de la table, trier des photos et vidéos en les rangeant soit dans la chaîne Youtube, soit sur un site web, soit dans la Corbeille. Ces trois éléments étaient matérialisés par des LibraryStack, une LibraryStack uniquement pour les vidéos (Youtube), une LibraryStack uniquement pour les images (site web) et une LibraryStack pour images et vidéos (Corbeille).



Pourquoi des tests utilisateurs ?
Les tests utilisateurs nous ont permis de nous assurer de la pertinence de notre travail et du respect de notre objectif de PFE.

Concernant le test de développement, il a été élaboré pour mettre des développeurs en situation réelle avec notre bibliothèque. Notre but était de recueillir des informations quant à la facilité d'utilisation de nos widgets ainsi que des pistes d'amélioration.

Pour le test d'usage avec la table, notre but était de vérifier qu'une application utilisant nos widgets correspondait à l'idée de collaboration de la table.

Critères d'observation

Pour le test de développement, nous avons utilisé trois critères:

- La difficulté, évaluée à travers le nombre de questions et remarques des testeurs, notamment grâce à la méthode Think aloud (le testeur dit tout ce qui lui passe par la tête).
- L'efficacité, évaluée à travers le nombre de vérifications échouées sur le navigateur une fois le code compilé.
- La généricité, évaluée à travers des questions aux testeurs (Y a-t-il trop de paramètres ? Quelles fonctionnalités ajouteriez-vous aux widgets ? Seraient-elles faciles à implémenter ? Manque-t-il des paramètres ?)

Pour le test d'usage avec la table, nous avons utilisé deux critères:

- La difficulté, évaluée à travers la méthode Think aloud et nos observations (les interactions sont-elles simples ? fluides ? Le testeur arrive-t-il à faire ce qu'il veut ?)

- L'interaction préférée, qui n'est pas vraiment un critère en soi mais plus un aspect intéressant à observer

Résultats

Le test de développement fut très positif. Les utilisateurs ont trouvé la documentation très claire et nous ont posé très peu de questions. Ils arrivaient la plupart du temps à créer les bons widgets au bon endroit avec les bon paramètres du premier coup. Au niveau de la généricité, ils étaient également satisfaits, les seules pistes d'amélioration seraient de peut-être alléger le nombre de paramètres et d'utiliser plutôt des fonctions, mais cela ne fut pas du tout une grosse limite.

Le test d'usage a lui aussi été extrêmement positif. Les utilisateurs ne nous ont jamais posé de questions sur les interactions, ils ont trouvé ces dernières fluides, faciles et intuitives. Un utilisateur a même préféré utiliser un objet tangible pour la rotation et la redimension. De plus, deux testeurs ont vraiment collaboré en choisissant les images et vidéos à garder. Ils se parlaient constamment et se demandaient leur avis sur les photos et vidéos à garder.

En conclusion, faire ces tests fut une initiative très enrichissante. Le test de développement nous a permis de voir que notre documentation était claire, que notre bibliothèque était facile à utiliser et que la généricité était mise en place de manière pratique. Le test d'usage nous a permis de voir que nos interactions étaient intuitives et nos widgets pouvaient servir à une application qui s'ancrait dans l'esprit de collaboration de la table.

En outre, ces tests ont mis en évidence notre respect de l'objectif du PFE.

Feedback récent

Pour une matière IHM, certains élèves ont commencé à développer des applications en utilisant nos widgets. Nous avons eu un retour en particulier qui nous a réconforté dans la capacité d'extension de notre bibliothèque. En effet un étudiant voulait donner la possibilité d'exécuter une fonction au toucher d'une ImageElementWidget. Pour ce faire, il a étendu notre classe ImageElementWidget et a rajouté la possibilité d'appeler une fonction lors d'une touche de l'élément. Cela montre bien que nos widgets peuvent servir de bonne base et accueillir de nouvelles fonctionnalités sans trop poser de contraintes aux développeurs.

III - Prise de recul

1) Si nous devions recommencer

S'il fallait recommencer ce PFE, nous solliciterions plus nos enseignants responsables dès le début. Cela nous aurait permis de mieux cerner l'aspect collaboratif de la table et nous aurions peut-être pu démarrer plus vite. De plus, il aurait été intéressant de réaliser d'autres tests utilisateurs vers le milieu de la phase de développement pour nous assurer que nous allions dans la bonne direction.

2) Si nous devions continuer

Comme énoncé auparavant, nous aurions continué à travailler sur la LibraryBar. Ce container aurait permis de remplir des besoins que la LibraryStack ne remplit pas et offrirait plus de richesse au niveau des containers disponibles. Nous aurions ensuite travaillé sur un LibraryContainer qui permet de switcher au runtime entre une LibraryStack et une LibraryBar. Enfin, nous aurions travaillé sur le InkCanvas, qui permet de dessiner des figures et de les exporter dans des images (une sorte de paint portatif).

3) Ce que nous laissons au développeur

Nous avons réalisé une bibliothèque composée de plusieurs widgets présentés précédemment. Les développeurs peuvent à présent réaliser des applications pour la table tactile tout en utilisant des widgets que nous avons créés. Ils ont également à leur disposition une documentation qui explique comment utiliser les widgets et modifier leurs propriétés.

Pour commencer un projet qui utilise notre API, il suffit de prendre la structure expliqué sur les repos github de la table. Il est à présent possible de créer des projets qui utilisent nos widgets, et même d'ajouter des widgets qui étendent ceux existant.

4) Qu'avons nous appris

Nous avons appris deux choses fondamentales dans ce PFE. La première est l'importance de considérer le support pour le développement d'applications. En effet, étant habitués à programmer pour des applications sur ordinateur, nous n'avons pas saisi dès le départ l'importance de respecter l'esprit de collaboration de la table décrit plus tôt.
Or cet aspect est crucial dans la réalisation des widgets, afin qu'ils correspondent à cette idée de collaboration autour de la table.

La deuxième est l'importance de baser ses choix sur des arguments pertinents. De manière générale, les choix de design et d'architectures doivent être justifiés. Cependant, durant la phase de développement, il était crucial pour nous de considérer les options disponibles et évaluer les solutions alternatives lorsque nous travaillions sur des widgets. En effet, l'aspect de généricité et réutilisabilité est central au projet, nous devions faire en sorte que notre travail soit réutilisable et facilement compréhensible. Nous avons ainsi passé un certain temps à débattre quant à la réalisation de certains widgets ainsi que les permissions que nous laissions aux utilisateurs. Nous avons de ce fait appris à rester ouvert aux options qui s'offrent à nous et argumenter nos choix avec des exemples concrets.