Rapport final de Projet
Table Interactive *PlayBoard*

Théo BERILLON, Jean-Baptiste CHEVRIER, Yohan ISMAEL,
Adrien LENOIR, Alexander LHUILIER

9 janvier 2022

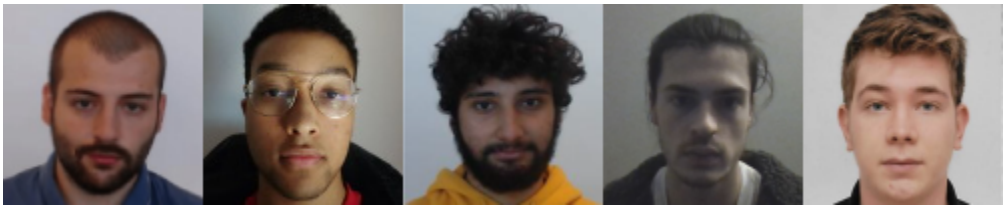
Contents

1 Protocole TUIO - À laisser au début ? **2**

2 Hardware - À déplacer à la fin ? **2**

3 Les petits chevaux - À déplacer où ça ? **3**
3.1 Généralités . . . 3
3.2 La classe Board . . . 3
3.3 La classe Horse . . . 3
3.4 La classe Dice . . . 4
3.5 La classe TUIO . . . 4
3.6 La classe Main . . . 4
3.7 Interaction avec la librairie TUIO . . . 5

ENSEA



Notre équipe se compose de 5 étudiant de 2^e année, Théo BERILLON, Jean-Baptiste CHEVRIER, Yohan ISMAEL, Adrien LENOIR, Alexander LHUILIER, motivés à faire prendre vie au projet ***PlayBoard*** *et doués d'un éventail de compétences relativement large.*

1 Protocole TUIO - À laisser au début ?

Le protocole TUIO fonctionne avec un système de *listeners*, comme pour la bibliothèque "awt" assez classique du langage JAVA.
Une application "TUIO Tracker" s'occupe d'émettre des messages, depuis la table dans notre cas, selon le protocole TUIO (lui-même basé sur le protocole OSC*), à une autre application "TUIO Client", dans notre cas les bibliothèques Tuio_java et Tuio_sharp utilisées dans nos classes.

Dans la pratique, on instancie un *TuioClient*, auquel on ajoute via une méthode dédiée une instance de classe implémentant l'interface *TuioListener*, avant de lancer la communication entre les deux.
L'interface *TuioListener* fournit des méthodes telles que addTuioObject(TuioObject tobj) ou removeTuioObject(TuioObject tobj), appelées automatiquement lorsque l'on ajoute ou retire un tag TUIO sur/de la surface de la table.

Les *TuioObject* en question représentent donc des tags TUIO, et sont munis de méthodes simple d'utilisation telles que getX(), getY() ou getAngle(), renvoyant les informations sur la position et l'orientation desdits tags.
L'objet TuioClient est quant à lui le gestionnaire du groupe de TuioObject et permet d'accéder à un *ArrayList* les contenant tous par exemple.

Le protocole TUIO est issu d'un projet collaboratif et est totalement libre de droit.
*: *Open Sound Control*

2 Hardware - À déplacer à la fin ?

L'intérêt particulier du PlayBoard réside dans l'important sentiment d'interaction des joueurs avec le support et entre eux, et c'est dans cette optique que nous avons développé les interfaces H.-M. des différents jeux implémentés.
Dans le même sens, souhaitant conférer un maximum d'immersion aux utilisateurs, nous avons réalisé des mobiles dans lesquels *clipser* temporairement les tags TUIO, en fonction du jeu lancé. Des formats "carte à jouer", des totems type "jungle speed" et autres pions on pu être designé.

3 Les petits chevaux - À déplacer où ça ?

3.1 Généralités

Nous sommes partis de l'idée que le jeu des petits chevaux, un grand classique des jeux de société, méritais une adaptation numérique, d'autant plus que l'interaction humaine entre le plateau, les pions et le dé est possible grâce à la table que nous possédons.
Pour ce faire, nous avons codé en Java, en utilisant la librairie JavaFx. Le jeu a été mis à jour avec GitHub, ce qui nous permet de garder un historique des versions et des changements apportés au fur et à mesure de l'avancement du projet.
L'objectif est de réussir à, d'une part, coder le jeu des petits chevaux tel qu'il existe aujourd'hui, et d'autre part d'y ajouter toute la partie interaction avec la table connectée, en utilisant le protocole TUIO, et les Tags fournis par l'entreprise Interactive Scape.

3.2 La classe Board

Le principal atout de Java est la possibilité de faire interagir des objets, ce que nous avons fait en distinguant les différentes composantes du jeu. La première et principale classe du jeu est celle du plateau, le Board. C'est dans cette classe que l'on créé les équipes, que l'on initialise le plateau de jeu (l'image), les chevaux de chaque équipe, les positions de départ, et c'est aussi avec cette classe que l'on actualise le plateau après qu'un joueur ait joué.
On y retrouve donc le constructeur Board, qui est appelé en début de partie, qui permet donc d'initialiser tous les objets. Une des principales difficultés rencontrées concerne les positions des images, et leurs dimensions. En effet, le jeu doit être adapté à tous les écrans proposés par Interactive Scape, soit donc en résolution du Full HD ou de la 4K, et en dimension de 32 à 65 pouces. Pour cela, il est indispensable de récupérer les dimensions de l'écran sur lequel le jeu est joué. On utilise ensuite des variables qui seront utilisées pour placer les différentes images, notamment celles des chevaux et des résultats du dé.
La classe Board contient également les trois fonctions principales d'actualisation du jeu : changePosition, associateAction et update. Dans l'ordre d'importance, update vérifie si la partie est finie, c'est-à-dire si un des joueurs placé ses 4 chevaux sur les cases finales, et dans le cas contraire, récupère le résultat du dé si celui-ci a été lancé et cherche à y associer l'action correspondante. C'est alors qu'on fait appel à associateAction. Cette deuxième fonction va vérifier si avec le résultat du dé, un mouvement est possible (pour le joueur actuel). Si c'est le cas, elle cherche à récupérer la position du Tag du joueur concerné, et si celui si est sur un cheval déplaçable, elle fera appel à la fonction changePosition. Si le Tag n'est pas au bon endroit, on attendra de retourner dans cette fonction et de vérifier à nouveau sa position. Si aucun cheval du joueur ne peut être bougé, on passera automatiquement au tour suivant. Enfin, la fonction changePosition fera les interactions graphiques nécessaires, et actualisera au passage la variable position du cheval déplacé. On y vérifiera dedans si la case sur laquelle le cheval va est occupée, au quel cas il faudra en réinitialiser la position.

3.3 La classe Horse

Deuxième classe primordiale, la classe Horse. Elle contient toutes les informations relatives à chaque cheval, notamment sa position, son numéro, et son état. Le constructeur est simple, et n'initialise que la position, l'état et l'image associée au cheval. On retrouve dans la classe la fonction isHorseMovable, qui est appelée par exemple dans la fonction update, et qui vérifie si un cheval en particulier peut être déplacé.

On en déduira une deuxième fonction isOneHorseMovable qui renvoie True si un des 4 chevaux du joueur est jouable.
Deux fonctions ont été pénibles à coder mais sont absolument nécessaires au jeu. Ce sont les fonctions getXPos et getYPos qui, comme leur nom l'indiquent, renvoient des coordonnées correspondant à une position de cheval. C'est là ou le fait d'avoir récupéré les dimensions de l'écran est crucial, puisque ce sont les variables associées que nous utilisons dans cette fonction, à savoir la taille d'une case, la taille d'une ligne, et surtout le milieu horizontal et vertical de l'écran.
Cette classe Horse est associée à la classe Team, mais étant secondaire, on n'y accordera pas tout une partie. Elle regroupe simplement les 4 chevaux d'une même équipe, ainsi que leur couleur.

3.4 La classe Dice

Élément très important de notre intérêt pour les petits chevaux puisqu'il ramène l'interaction humaine avec la table connectée, le dé occupe une part très importante dans le jeu. C'est dans cette classe Dice que nous avons commencé à utiliser des Timers. Ils permettent de ne pas pratiquer la stratégie de polling, c'est-à-dire se placer dans une boucle while et vérifier en permanence si une action à eu lieu. Le timer nous permet de réguler cette vérification, en la faisant que lorsque le timer émet un signal, une action, en Java c'est une ActionEvent.
Lorsque c'est le tour d'un nouveau joueur, on démarre donc le timer (pour ne pas le laisser tourner quand le jeu n'en a pas besoin), et on fait une série de vérifications. Nous entrons alors dans les détails de l'utilisation de la librairie TUIO, que nous détaillerons ultérieurement. La fonction appelée par le timer est actionPerformed. On récupèrera l'angle du Tag associé au dé, et on transformera cette valeur qui est entre 0 et -2π en résultat de lancer de dé entre 1 et 6.
Après avoir actualisé les variables associées au lancer de dé, on traitera ce résultat lors de l'appel de la fonction update et on passera au tour suivant.

3.5 La classe TUIO

Cette classe ne porte pas très bien son nom puisqu'elle ne provient pas de la librairie en elle-même mais nous permet d'initialiser le nombre de joueurs, d'expliquer les règles et de récupérer les données des Tags qui seront utilisés pour le dé et chacun des joueurs. On retrouve également dans cette classe Tuio un timer, qui nous évite de rester coincé dans une boucle while. 3 fonctions principales sont présentes. La première est getNumberOfPlayers, qui nous permet comme son nom l'indique de récupérer via le tactile de l'écran le nombre de joueurs. On y utilise des JFrame, des JLabels et des JButtons afin d'afficher les informations pertinentes comme du texte ou des images pour le nombre de joueurs. Le résultat sera détecté par le timer, qui appellera à son tour la fonction showRules pour afficher les règles à l'écran. Enfin, la fonction getTags récupèrera par le biais du timer les Tags associés aux joueurs.
Nous détaillerons l'utilisation de la librairie TUIO et en particulier le contenu de la classe TestTuio2 qui nous permet d'actualiser et de récupérer toutes les données des différents Tags.

3.6 La classe Main

Cette dernière classe Main est appelée au lancement du jeu. Nous sommes obligés d'y inclure la fonction start de JavaFx, qui s'applique à un Stage, qui nous permet de lancer le jeu et de disposer des nombreuses fonctions fournies par cette librairie. Encore une fois, on utilise un timer dans cette classe. Ceci nous permet de vérifier si toutes les fonctions appelées dans Tuio sont terminées, c'est-à-dire si l'on connait le nombre de joueurs, les Tags, et si les joueurs ont lu les règles.
On attendra donc que toutes les variables soient connues, avant d'appeler le constructeur Board, et de passer le jeu en plein écran pour démarrer la partie.

Enfin, une dernière classe nommée Executable fait simplement appel à la classe Main et nous permet de générer un artéfact, soit dans notre cas un .jar.

3.7 Interaction avec la librairie TUIO

Nous allons détailler dans cette partie comment nous avons relié le code du jeu à l'utilisation des Tags d'Interactive Scape, en utilisant le protocole TUIO. Plusieurs types d'objets sont définissables. Nous n'utilisons que les TuioObject, c'est ainsi qu'ils sont appelés en Java. Chaque objet possède de nombreuses variables associées. Nous n'avons utilisé que les suivantes :

- Le symbol ID, qui est un identifiant propre à chaque Tag, entre 1 et 24 ;
- Le session ID, qui est un identifiant secondaire, qui dépend de l'ordre dans lequel les Tags ont été placés ;
- Le hashcode, qui est un dernier identifiant, plutôt un long entier, associé à chaque Tag, et qui change dès que le Tag n'est plus en contact avec la table puis l'est à nouveau.

Pour récupérer donc un Tag par joueur en plus du dé, on modifie la fonction addTuioObject afin de vérifier que le Tag n'a pas déjà été placé et que nous en avons le bon nombre, et on stocke les objets dans une liste, dont nous auront besoin très régulièrement.
Pour la partie concernant le dé, le code a été fait de manière à détecter le mouvement puis l'arrêt de ce dernier. On vérifie pour cela que le dé a été ôté de la table, c'est même obligatoire avec le lanceur de dé que nous avons développé dans le cadre du projet, grâce au hashcode (il doit être différent du hashcode précédemment gardé en mémoire). Si c'est le cas, on vérifie que le dé est en mouvement (la table étant sensible, un Tag est automatiquement en mouvement lorsqu'il est posé sur la table), avec la fonction fournie par la librairie TUIO getMotionAccel. Par la suite, on attend que cette accélération soit nulle et on récupère alors l'angle, grâce à la fonction getAngle. On transforme enfin cet angle en entier entre 1 et 6 que l'on retourne.
On utilise également la position des Tags avec les fonctions getX et getY, pour notamment déplacer les petits chevaux. Pour cela, on utilise encore une fois les dimensions de l'écran car les positions des objets sont données entre 0 et 1. Par un calcul de distance minimum, on parvient dans la fonction coordToPos (dans la classe Horse) à renvoyer la position correspondant à l'emplacement du Tag.