Projets IESE4 2024

Rapport final IESE4 Écrans tactiles STM32

Polytech Grenoble - Fablab MASTIC

Étudiants :

BOUCHE Romain

CHEVALLIER Bastien

COSNAY Antony

Porteur de projet :

LEMASSON Germain

Enseignants tuteurs :

GUYADER Nathalie

ZALKIND Madeleine

Glossaire

GUI (Graphical User Interface) :

C’est une interface visuelle qui permet aux utilisateurs d'interagir avec des programmes informatiques à l'aide d'éléments graphiques tels que des fenêtres, des boutons, des menus déroulants, des icônes, plutôt que de devoir saisir des commandes textuelles.

SPI (Serial Peripheral Interface) :

C’est est un bus de données série synchrone. Les circuits communiquent selon un schéma maître-esclave, où le maître contrôle la communication.

I2C (Inter-Integrated Circuit)

C’est un bus série synchrone bidirectionnel half-duplex, où plusieurs équipements, maîtres ou esclaves, peuvent être connectés au bus.

RMII (Reduced Media Independent Interface) **/ MII** (Media Independent Interface) **:**

RMII et MII sont deux interfaces utilisées dans les communications entre un contrôleur Ethernet et un transceiver Ethernet (PHY) dans les systèmes informatiques.

USB (Universal Serial Bus) :

L'USB est un protocole de communication série entre entités.

UART (Universal Asynchronous Receiver Transmitter)

L’UART est un émetteur-récepteur asynchrone universel. C'est le composant utilisé pour faire la liaison entre l'ordinateur et le port série. L'ordinateur envoie les données en parallèle (autant de fils que de bits de données).

RS232

RS-232 est une norme standardisant une voie de communication de type série., il est communément appelé le « port série ».

GPIO (General Purpose Input/Output)

Un connecteur GPIO offre à une carte électronique la possibilité de communiquer avec d'autres circuits électroniques.

Table des matières

I Introduction

Les écrans tactiles sont devenus omniprésents dans notre quotidien, offrant une interface intuitive et interactive pour une multitude d'applications. Dans ce contexte, le projet proposé par Germain Lemasson, en collaboration avec le Fablab MASTIC, vise à explorer les capacités de cartes de la série STM32H7 équipées d'écrans tactiles. Cette initiative s'inscrit dans une démarche visant à exploiter ces technologies pour le développement d'applications fonctionnelles, artistiques ou ludiques.

Premièrement, nous allons présenter le lieu où nous allons travailler, puis nous explorerons plus en détails les aspects du projet. Ensuite, nous aborderons les méthodes de gestion de projet utilisées, et enfin, le déroulement du projet et passant par la programmation des interfaces utilisateur graphiques (GUI), le développement de l'application et la mise en réseau.

II Fablab MASTIC

Le Fablab MASTIC (site internet) est un espace universitaire rattaché à l'Université Grenoble Alpes pour permettre aux étudiants, enseignants et chercheurs de formaliser des projets ou idées pour du prototypage. MASTIC est l'acronyme de Mathématiques Appliquées, Sciences et Technologies de l'Information et de la Communication.
Situés 740 Rue de la piscine à Saint martin d'Hères, les locaux disposent de différentes machines-outils de fabrication (Imprimante 3D, découpeuse laser, découpeuse vinyle, …), ainsi qu'une possibilité de formation sur place pour leur utilisation. L'équipe fournit également un support pour la conception électronique et mécanique, ainsi que pour de la programmation embarquée microcontrôleur.
Notre porteur de projet est Germain Lemasson, Fablab Manager et responsable fablab MASTIC, également Ingénieur Recherche pour le laboratoire LIG Grenoble INP.

III Contexte du projet

III.1 Présentation du sujet

Notre projet a été initié par Germain Lemasson cette année, pour proposer aux étudiants de Polytech Grenoble de donner une utilité à des cartes de développement STM32H7. Reposant actuellement dans le stockage du Fablab, celles-ci possèdent des écrans tactiles, permettant de développer des applications ludiques ou artistiques interactives. Leur caractéristiques techniques seront détaillées ensuite. Nos travaux s'inscrivent dans le projet d'installer ces cartes dans un café étudiants, pour permettre la création d'un espace de détente en jouant à des jeux sur les temps de pause.

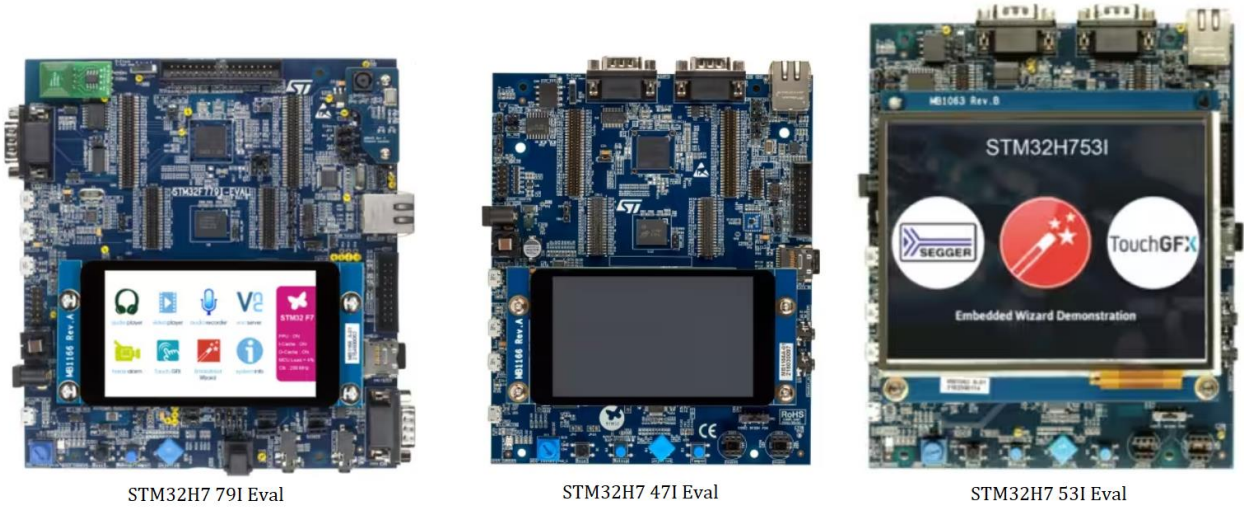
L'objectif technique du projet est de découvrir les fonctionnalités des cartes à disposition, tout en proposant des idées à développer pour tirer profit de leur versatilité. Nous avons également une demande de fonctionnalité de cartes en réseau, énoncée par le porteur de projet. En dehors de ces deux objectifs, les attentes du projet ne sont pas précises, et sont laissées libres et déterminées par notre avancement et nos idées. Visant un public large comme un café étudiant, donc potentiellement international, nous décidons d'utiliser le plus possible l'anglais dans le code et l'affichage de notre application. Cela permet

également la reprise du projet et du code produit par des étudiants internationaux pour sa continuation, la continuité étant un des objectifs du projet.

III.2 Présentation des cartes STM32 à disposition

Pour ce projet, le Fablab MASTIC a dans son stockage trois modèles de cartes STM32. On a donc à disposition les cartes d'évaluation suivantes, dont les images de présentation sont en figure 1.

- 5 modèles de STM32H7 79I Eval
- 13 modèles de STM32H7 47I Eval (1 sortie)
- 8 modèles de STM32H7 53I Eval (3 sorties)



STM32H7 79I Eval | STM32H7 47I Eval | STM32H7 53I Eval

Figure 1 : Cartes STM32H7 à disposition

A première vue de ces cartes en figure 1, deux d'entre elles ont des écrans de taille réduite par rapport à la STM32H7 53I à droite. En effet, cette dernière a un écran 5,3 pouces, contre 4 pour les autres. Ces trois cartes sont équipées de cœurs microprocesseur Arm de la série M7, une des plus performante et efficace en énergie du catalogue Arm pour des applications embarquées. Ces microcontrôleurs H7 sont également dans le catalogue ST les plus performants, ce qui justifie leur utilisation dans des systèmes demandant beaucoup de ressources comme un affichage avec écran. Leur fréquence mémoire peut atteindre 480 MHz, pour des tailles de mémoire flash et RAM de 1 MBytes à 2 MBytes. Des programmes complexes peuvent alors être exécutés avec ces microcontrôleurs.

Visibles en figure 1, les cartes proposent de nombreux périphériques. Ces périphériques additionnels au microcontrôleur leur valent leur distinction "Evaluation Board", signifiant la présence de tous les outils importants pour du prototypage. On peut notamment citer la présence sur toutes les cartes de 168 pins d'entrées et sorties, de connecteurs Ethernet, USB, CAN et de liaison série, en plus de boutons et potentiomètres. Le microcontrôleur propose en plus la gestion de protocoles de communication comme l'I2C, SPI, USART ou Modbus. Lors de nos travaux, notre utilisation de ces cartes se restreint à ces périphériques présents par défaut afin de limiter les distinctions entre les cartes présentes au Fablab pour une utilisation future modulable.

III.3 Démarrage du projet et objectifs

Comme dit dans la liste d'inventaire en début de partie précédente, trois des STM32H7 53I Eval et une STM32H7 47I Eval ont été au préalable sorties de leur emballage. Les autres sont encore dans leur boîte

d'origine, dans un emballage antistatique. Lors de notre découverte de ces cartes à disposition, nous avons pu utiliser le programme de démonstration des capacités des cartes présent par défaut. Avec ces exemples, dont un lien vers leur accès est disponible en référence bibliographique, nous nous sommes rendu compte de leur potentiel dans des applications de contrôle de procédé industriel, de domotique, de communication ou de jeu tactile, en utilisant la majorité des fonctionnalités des cartes.

Comme visible sur la représentation de la STM32H7 53I Eval en figure 1, trois moteurs de rendu sont disponibles sur le programme de démonstration pour l'affichage tactile sur l'écran. On a donc une librairie de Segger, Embedded Wizard et Touch GFX. Cette dernière librairie est développée et maintenue à jour par ST, c'est pourquoi nous avons choisi de l'utiliser pour notre projet.

Avec la récupération des documentations des cartes sur le site internet de STMicroelectronics, nous avons pu voir qu'il n'existait pas de réelle différence en dehors de l'écran entre la carte à petit écran disponible et les trois à grand écran. Nous avons donc choisi de travailler sur les cartes STM32H7 53I Eval durant notre projet, laissant plus de place sur écran à notre future application, la rendant plus agréable à utiliser.

Le projet étant à ses débuts, sans travaux d'étudiants précédents sur lesquels s'appuyer, nous nous sommes d'abord intéressés aux différents tutoriels d'utilisation des outils de ST. Les logiciels de programmation STM32CubeIDE et STM32CubeMX ont donc dû être pris en main. En cherchant différentes applications des cartes STM32H7 et leur possibles utilisations, nous ne sommes pas parvenus à trouver des travaux disponibles en libre accès autre que les codes exemples et de démonstration ST à intégrer sur les cartes. Ils ont tout de même représenté une aide précieuse à l'établissement du premier programme personnalisé, soit un simple compteur modifiable au toucher de boutons d'incrémentation et décrémentation.

Notre objectif est donc de développer un jeu simple, dans l'optique d'une fonctionnalité en réseau entre plusieurs cartes. Après réflexion, nous avons pensé à réaliser un jeu Morpion pour un jeu à deux joueurs. Nous visons ensuite le développement d'un jeu plus complexe à définir, ayant une possibilité de multijoueur plus complexe.

IV Gestion de projet

Pour ce projet, nous sommes un groupe de trois étudiants de IESE4, dont les 100 heures de travail en 25 séances sont généralement organisées en 4 heures les jeudis matin, puis lundis matin à partir de mars. Nous travaillons principalement au Fablab, là où les cartes sont disponibles pour leur programmation. Les séances de projet à Polytech sont rendues possibles avec un déplacement des cartes pour travailler. Nous avons passé une majorité des séances au Fablab notamment aux débuts du projet. En effet, la direction des travaux n'était jusqu'à février pas complètement définie. Les séances restantes, soit lorsque l'on a pu établir un objectif pour nos séances, ont été effectuées à Polytech. On note également que les horaires d'ouvertures du fablab les lundi après-midi et jeudi matin ne nous permettait pas de passer 4 heures consécutives sur le projet. Les jeudis midi, nous avons donc régulièrement commencé à 9h, horaire d'ouverture, et fini à 13h.

A partir de nos connaissances sur le sujet du projet et des membres du groupe, nous pouvons utiliser différents outils de gestion de projet pour cerner nos possibles facilités ou difficultés.

IV.1 Matrice SWOT

Nous construisons alors la matrice SWOT en tableau 1 suivant, répertoriant les différents aspects du projet :

Tableau 1 : Matrice SWOT

Strength	Compétences Motivation Liberté de conception	Weakness	Pas d'objectif/cahier des charges précis
Opportunities	Environnement de laboratoire	Threat	Dispersion Disponibilité et bon fonctionnement des cartes pour les séances

Dans cette matrice SWOT, on trouve d'abord des forces. On peut citer notre motivation pour mener à bien le projet. On voit également une certaine liberté de conception de notre application, qui vient avec une faiblesse, soit l'absence de cahier des charges précis de la part de notre porteur de projet. Nos menaces sont alors une dispersion des tâches réalisées n'allant pas vers notre objectif.

IV.2 Analyse des Risques

Une analyse des différents risques auxquels nous sommes exposés est essentielle pour réduire la probabilité d'échec du projet. En fonction des différents risques, nous pouvons attribuer une probabilité d'arriver et leur impact direct sur le projet. A partir des faiblesses et risques présentés dans la matrice SWOT précédente, on peut alors construire la matrice en tableau 2 suivant, présentant les risques s'appliquant au projet, associés à leur probabilité et impact sur nos réalisations.

Tableau 2 : Analyse des risques

Risque	Probabilité	Impact
Endommagement de l'ordinateur d'un membre du groupe	Faible	Fort
Perte du code d'un membre du projet	Très faible (GIT)	Moyen
Ecart aux objectifs	Moyen	Moyen
Endommagement d'une carte STM32	Faible	Faible
Absence prolongée du porteur de projet	Faible	Faible

Nous avons alors pensé à plusieurs risques de probabilités faibles, ayant un impact plus ou moins fort sur notre avancement du projet. En effet, l'endommagement de l'ordinateur d'un membre du projet entraînerait un empêchement temporaire de travailler ainsi qu'une perte de données potentielle.

Pour pallier ce risque fort, nous avons mis en place l'utilisation de Git pour le stockage en ligne du code du projet. Son accès est disponible sur internet au lien dans les parties V.2.1 Utilisation de GIT et VIII Références Bibliographiques. Nous utilisons également un fichier partagé sur Google Drive, dont le lien se trouve dans les références Bibliographiques.

IV.3 Tableau de compétences

Au démarrage du projet, les connaissances applicables au déroulé du projet nous semblent intéressantes à développer, pour identifier les possibles blocages ou facilités. On construit alors le tableau de compétences suivantes, présentant le niveau de connaissances de chacun des membres du projet sur les différents sujets relatifs au projet.

Tableau 3 : Compétence du projet

Compétence		BOUCHE Romain	CHEVALLIER Bastien	COSNAY Antony
Communications interpersonnels		Moyen	Moyen	Moyen
Compétences rédactionnelles		Fort	Fort	Fort
Créativité		Fort	Moyen	Moyen
Lecture de documentation		Fort	Fort	Fort
Développement sur microcontrôleurs		Fort	Fort	Fort
Connaissances TouchGFX		Nulle, Fort atteint	Nulle, Fort atteint	Nulle, Moyen atteint
Connaissances en langage C++	Programmation	Moyen, Fort atteint	Moyen, Fort atteint	Moyen, Fort atteint
	Structuration	Moyen, Fort atteint	Faible, Fort atteint	Moyen, Fort atteint
Connaissances en Réseaux		Moyen	Moyen	Moyen, Fort atteint

On note ici que tous les membres du groupe ont des connaissances solides en programmation C et développement sur microcontrôleur. L'apprentissage du C++ ne sera alors pas un problème, étant donné que c'est un enseignement prévu du second semestre de IESE4. Sur ce point, les séances du second semestre nous ont permis de prendre de l'avance sur les cours, notamment sur la structuration et l'héritage de classes. Concernant les connaissances en réseaux, nous avons tous suivi des cours de TCP/IP en DUT, en plus de la connaissance des bus de données comme l'I2C, SPI et Modbus.

IV.4 Motivations

Pour ce projet, nous sommes très motivés car c'est un projet utilisant un environnement de programmation propriétaire de ST très demandé pour le développement sur STM32. Les parties graphique et de programmation orienté objets sont pour nous de nouvelles compétences à apprendre pour notre connaissance personnelle. De plus, c'est un moyen de développer notre créativité pour la création d'interface graphique, à considérer lors de la création de procédés automatisés.

IV.5 Impact Environnemental

Ce projet a pour but de récupérer des cartes de développement inutilisées de STMicroelectronics stockées au Fablab. De plus, les câbles nécessaires à la communication entre cartes peuvent également donner une utilisation aux nombreux câbles disponibles au Fablab L'impact environnemental du projet est donc très faible, car ne demandant aucune ressource supplémentaire pour réaliser les différentes tâches proposées. On ne peut donner qu'une préoccupation quant à la consommation électrique des cartes utilisées. Nous devrons donc penser à produire un code le plus efficace possible pour économiser de l'énergie électrique.

V Déroulement du projet

V.1 TouchGFX

Ayant choisi le moteur de rendu TouchGFX pour l'affichage graphique, nous nous sommes vite intéressés aux différents moyens de production de code pour les cartes STM32H7 équipés d'écrans tactiles. Nous nous sommes ainsi familiarisés avec le logiciel TouchGFX Designer, développé par STMicroelectronics. Nous avons pu récupérer différents codes exemples disponibles sur le site de STMicroelectronics et TouchGFX designer lui-même afin d'observer et comprendre les différentes fonctionnalités du logiciel.

Nous avons pu suivre des tutoriaux officiels ainsi que non officiels pour apprendre les bases du logiciel. Certaines fonctionnalités plus avancées feront l'objet d'études plus complètes lors de la réalisation du projet, comme l'utilisation de containers, contenant divers objets configurés par l'utilisateur.

Lors de la génération du code de l'interface créée avec TouchGFX Designer, celui-ci génère une arborescence de fichiers C++ pouvant être ouvert et éditée via STM32Cube IDE (fichier STM32CubeIDE -> .cproject). Dans l'arborescence ouverte sur IDE, on peut voir les objets C++ créés depuis l'interface du logiciel dans le dossier TouchGFX, puis generated ou gui. Il génère alors divers fichiers contenant des classes, contenant des objets C++ correspondant aux différents éléments générés sur TouchGFX. Par exemple, un bouton ou une image est défini par une classe, avec des fonctions membres pour sa manipulation. Ces objets sont regroupés par écrans, dont les classes sont définies dans les fichiers portant leur nom.

Comme dit précédemment, TouchGFX Designer crée des éléments dans les deux dossiers generated ou gui, contenant les classes des objets de l'interface. Il est important de noter que les noms des fichiers dans le dossier generated ont un suffixe en "Base". Pour TouchGFX, cela signifie que l'utilisateur ne doit pas les modifier, et ne sont disponible qu'en lecture seule. Toutes les modifications nécessaires aux besoins de l'application seront effectuées dans les fichiers sans le préfixe Base, situés dans le dossier TouchGFX/gui. Les classes définies dans gui héritent de celles de generated, permettant en C++ la redéfinition ou création des fonctions virtuelles de l'application. Par exemple, on redéfinit dans *MorpionMenuView* la fonction C++ d'initialisation créée par TouchGFX dans le fichier MorpionMenuViewBase.

Les fonctions virtuelles jouent un rôle majeur dans l'implémentation de fonctionnalités plus avancées, qui ne peuvent pas être implémentées depuis TouchGFX Designer. Lors de l'appel d'une fonction virtuelle depuis une interaction créée dans TouchGFX, sa déclaration est créée si inexistante dans le fichier Base de l'écran, et peut être redéfinie dans le header View et dans le fichier source View.

Le fonctionnement de TouchGFX repose sur l'utilisation de FreeRTOS. Comme son nom l'indique, c'est un Real Time OS, ou système d'exploitation développé pour une utilisation dans les systèmes embarqués à temps réel. TouchGFX commence par définir dans le fichier main.c du programme une tâche ou *thread* "TouchGFXTaskHandle" gérant tout l'affichage sur l'écran. RTOS alloue aux différentes tâches la ressource microprocesseur en fonction de leur activité et priorité. L'utilisation d'un OS est ici justifiée, car la gestion d'un affichage sur un écran est marquée par une majorité de temps d'attente entre les mises à jour de l'affichage, ou des interactions de l'utilisateur. Nous avons dans le cadre de ce projet eu l'occasion d'utiliser les fonctionnalités de RTOS pour la définition d'autres tâches, notamment pour la communication entre cartes.

V.2 Développement d'un jeu : Morpion

V.2.1 Utilisation de GIT

Comme annoncé dans la partie gestion de projet et les risques associés au déroulement du projet, nous avons mis en place un dépôt GitHub spécialisé pour le stockage du code du projet. Il est disponible publiquement au lien suivant : https://github.com/antcsny/Projet_Ecrantactile. GitHub permet une interface graphique de git, un solution de versionnage de toute sorte de projet, dont les modifications sont apportées une à une par ses contributeurs. Durant le projet, git nous a permis à trois de travailler en parallèle sur le code de notre application, et a rendu simple la fusion de nos versions avec la fonction *merge*. Comme son accès est fait en ligne, cela a été un outil plus pratique qu'une clé USB en transit, utilisée aux débuts du projet. Nous avons senti son importance dès la construction d'une base solide pour notre application.

Le dépôt GitHub contient l'entièreté du dossier ouvrable par STM32Cube IDE pour l'édition du code. Nous nous sommes assurés que le projet soit utilisable dès son téléchargement fini et ouvert avec CubeIDE. Pour intégrer le projet au Workspace CubeIDE, il faut ouvrir dans l'explorateur du système d'exploitation de l'ordinateur le fichier .cproject au chemin Projet_Ecrantactile\STM32CubeIDE\.cproject. Dans le cas d'une erreur lors de la compilation portant sur le dossier STM32CubeIDE/Debug, regénérer le code avec TouchGFX designer suffit à régler le problème. Nous conseillons d'utiliser l'application GitHub Desktop disponible gratuitement au téléchargement.

V.2.2 Écran d'accueil et menu

Pour la réalisation d'un premier jeu, nous avons choisi d'opter pour un jeu très simple : Le jeu du Morpion. On a devant nous une grille de trois fois trois cases, où l'on vient au toucher placer des croix ou des ronds. Le joueur 1 commence par placer une croix, et vient le tour du joueur 2 qui place un rond, et ainsi de suite. Le premier joueur qui aligne trois de ses formes a gagné la partie. Ce jeu nous permet de mettre en pratique des éléments basiques sur un fonctionnement de jeu simple.

Pour sa réalisation avec TouchGFX designer, nous avons tout d'abord établi un écran d'accueil par défaut *MorpionMenu* (figure 2), qui permettra de lancer le jeu sur une grille sur une seule carte, ou bien de jouer

Figure 3 : Écran d'accueil du Morpion

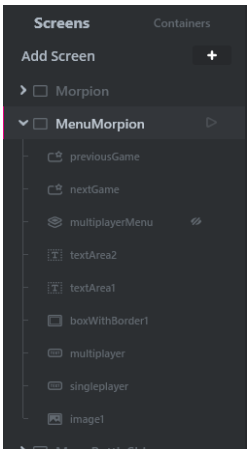


Figure 2 : Hiérarchie des différents écrans créés

en local avec une communication avec une seconde carte. Nous prévoyons dès maintenant cette fonctionnalité, que nous verrons plus tard dans le projet.

Depuis TouchGFX, il est possible de créer plusieurs écrans, qui possèdent chacun leur propre fichier d'exécution, ainsi qu'une classe définissant chaque objet présent dans l'écran. Il est à noter que chaque objet créé depuis l'interface TouchGFX sera lié à la classe TouchGFX, qui contient diverses fonctions permettant de manipuler ces objets, comme modifier leur position, leur visibilité, leur couleur et d'autres propriétés qui deviendront utiles dans la réalisation du code. On pourra retrouver la liste de ces fonctions dans le fichier Drawable.hpp.

On retrouve d'abord en figure 3 les différents écrans et containers créés sur TouchGFX Designer pour l'application. D'autres écrans que ce menu ont été créés pour d'autres fonctionnalités, comme le Morpion, qui contient les différents éléments de jeu du Morpion.

L'affichage du contenu du menu principal MorpionMenu (correspondant à l'écran figure 2) a été en figure 3 étendu pour permettre de voir les différents éléments composant l'écran. On retrouve ici les textes, boutons, et image de fond, ainsi que le conteneur multiplayerMenu, dont nous aurons l'occasion de détailler ensuite.

L'appui sur le bouton *Singleplayer* du *MorpionMenu* exécute une commande générée par TouchGFX qui permet de passer de l'écran *MorpionMenu* à l'écran *Morpion.* Celle-ci est directement implémentée depuis TouchGFX à travers la liste d'interactions de l'écran, comme montré en figure 4 ci-contre.

On remarque également en figure 4 les actions exécutées lors d'appui sur les boutons de l'écran. En effet, comme visible en figure 2, l'écran *MorpionMenu* possède des boutons en flèche gauche et droite. On retrouve alors les interactions "rightGame" et "leftGame"

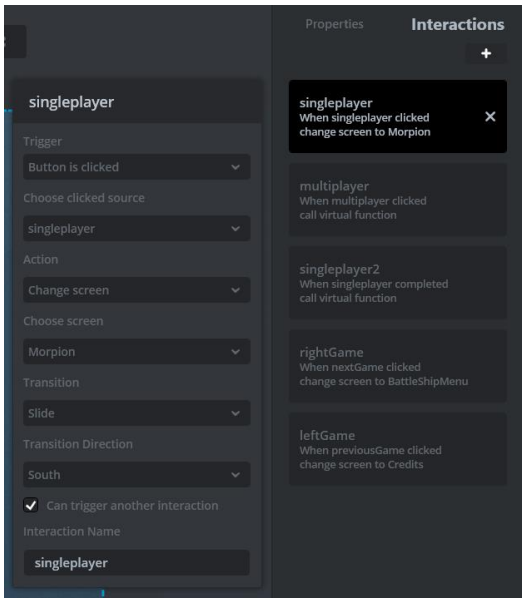


Figure 4 : Liste des interactions de l'écran

Diverses actions peuvent alors être implémentées à l'aide des interactions, comme le déplacement d'un objet dans l'écran, changer le texte d'une zone de texte ou bien appeler une fonction virtuelle que l'on pourra déclarer et définir depuis STM32Cube IDE. On peut également écrire directement du code dans les interactions, ou définir des actions. Ce sont des fonctions virtuelles C++ par défaut vides, à redéfinir.

Il faut noter que chaque écran possède ses propres variables globales et propres classes. Un écran ne pourra pas faire appel à une fonction ou une variable d'un autre écran si son fichier hpp n'est pas inclus. Les variables et fonctions communes aux écrans sont directement déclarées dans le fichier main.cpp du projet, synonyme d'un accès global si nécessaire et facile à en retrouver la source.

V.2.3 Processus de jeu

Une fois le jeu lancé à travers le menu principal, le programme lance l'écran *Morpion* (figure 5 en page suivante) qui contient l'intégralité du jeu dont la partie graphique et la partie code. Nous avons défini la transition vers cet écran pour qu'il exécute automatiquement la fonction initialisation() située dans le fichier de l'écran MorpionView.cpp.

Figure 5 : Écran de jeu du Morpion

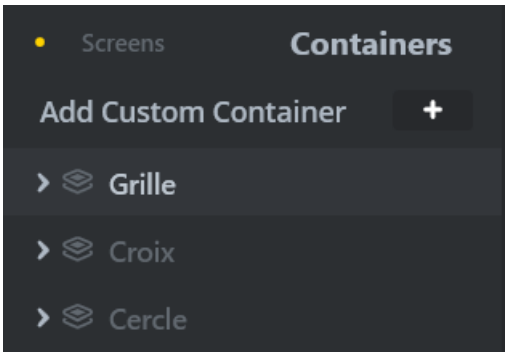


Figure 6 : Hiérarchie des différents conteneurs créés

On distingue la grille du jeu, sur laquelle plusieurs boutons invisibles ont été judicieusement placés à chaque position des formes du jeu. Ces boutons permettent au joueur de placer une croix ou un rond en fonction du tour à l'emplacement du toucher. Le bouton Quit est simplement configuré pour changer d'écran vers le MenuMorpion depuis les interactions TouchGFX. Quant au bouton Restart, il fait appel à la fonction virtuelle d'initialisation du plateau de jeu initialisation().

On note en figure 5 la présence de deux textes superposés. Ils indiquent le tour du joueur par des textes TouchGFX à afficher ou à faire disparaître en fonction de l'état de la partie. Cela pourrait être réalisé plus proprement avec un seul texte et des champs textuels variables. Cette solution nous a paru trop complexe pour un affichage de deux textes différents.

La grille ainsi que les pièces de jeu (croix et ronds) sont créées à partir d'un Container, que l'on peut retrouver directement dans la hiérarchie des Containers sur TouchGFX (figure 6). Tracer la grille du jeu dans un container permet également dans l'écran *Morpion* de modifier sa position et sa taille sans avoir à modifier chacune des lignes tracées. Pour les pièces de jeu, cela nous permet de créer plusieurs containers Croix et Cercle dans l'écran *Morpion* sans avoir à recréer les pièces une par une. Cela permet d'utiliser moins d'espace mémoire et est généralement plus pratique d'utilisation. Pour notre jeu, nous avons créé 9 pièces, 5 croix et 4 cercles, qui sont placés initialement hors de l'écran. Elles n'auront plus qu'à être déplacées au besoin dans la grille du jeu.

Comme dit précédemment, des boutons invisibles se trouvent dans chaque case de la grille. Chacun appelle une fonction virtuelle différente, créée par TouchGFX. Ils sont configurés par nos soins de sorte à appeler en suite une seule et même fonction virtuelle PlayMove, qui gère le fonctionnement global du jeu. Celle-ci prend en paramètre le bouton de type Drawable, et sait ainsi quel bouton l'a appelé. Les

boutons sont tous positionnés à des coordonnées précises sur l'écran, et multiples d'un nombre commun. Pour obtenir l'indice du bouton cliqué sur la grille, une simple division est effectuée au début de la fonction PlayMove. Son appel permet alors en bref de placer une pièce de jeu, une croix ou un cercle en fonction du tour actuel. Si une pièce de jeu est placée, l'autre joueur ne doit pas pouvoir en placer une par-dessus. PlayMove désactive alors le bouton passé en paramètre afin d'éviter que le joueur l'actionne à nouveau. Cette fonction tient également à jour l'affichage alterné du texte appelant le joueur 1 ou 2 à jouer, visible superposé en figure 5, et la vérification des conditions de victoire.

Chaque modification visuelle sur un objet nécessite une actualisation de celui-ci sur l'écran, afin que les modifications qui lui sont apportées soient appliquées La fonction membre de la classe TouchGFX::invalidate() s'applique alors à une majorité des objets créés par TouchGFX pour leur mise à jour sur l'écran.

V.2.4 Condition de victoire

L'une des fonctions principales du jeu du Morpion est la vérification de la condition de victoire, qui met fin au jeu si elle est remplie. Elle est gérée par la fonction *verifier_victoire()*, appelée lors de chaque passage dans la fonction PlayMove à partir du tour 5, où le joueur 1 a la possibilité de placer sa dernière croix restante pour gagner. Lorsque la fonction PlayMove est appelée, elle reprend l'adresse et les coordonnées du bouton appuyé, et stocke dans une matrice de caractère 3x3 la pièce qui a été jouée, sous la forme du caractère 'x' pour les croix et d'un 'o' pour les ronds.

La matrice est déclarée de manière globale dans le fichier Morpionview.cpp, et initialisée dans la fonction d'initialisation, afin que chacune des fonctions puisse y avoir accès. Ainsi, la fonction n'aura plus qu'à vérifier les valeurs de la matrice pour déterminer si une ligne, une colonne ou une diagonale est complétée par un seul et même caractère, ou bien si le nombre de tour atteint son maximum, pour mettre fin au jeu.

Elle retourne ainsi une valeur à la fonction PlayMove pour lui indiquer l'état du jeu actuel, si l'un des joueurs a gagné, si la partie continue ou si une égalité a lieu. Lorsque le jeu prend fin, la fonction PlayMove bloque les interactions avec les boutons de la grille, à l'aide de la fonction membre *setTouchable()* et affiche à l'écran un container initialement invisible en fonction de si l'un des deux joueur a gagné ou si aucun n'a gagné. Nous les avons montrées en figure 7 suivante.

Figure 7 : Objets affichés lors de la fin d'une partie

Comme visible en figure 7, TouchGFX Designer permet d'importer et afficher des images de type PNG dans un projet, donnant une certaine liberté de création et personnalisation à l'utilisateur. Les containers de victoire utilisent cette fonctionnalité pour afficher un message de victoire avec la fonction setVisible() (ainsi que invalidate() pour actualiser) en fonction du joueur qui a gagné la partie.

On pourra à partir d'une victoire interagir uniquement avec les boutons Restart et Quit, pour relancer une partie ou revenir au menu d'accueil du jeu. Cela complète le mode de jeu du Morpion où deux joueurs jouent sur une seule carte.

V.3 Mise en réseau des cartes STM32H7

L'un des souhaits communiqué par M. Lemasson était de donner une seconde vie aux câbles Ethernet disponibles en grande quantité au Fablab MASTIC. Le but du projet étant d'utiliser du matériel déjà disponible au Fablab, il nous été demandé d'implémenter la mise en réseau des cartes STM32H7 par protocole TCP/IP à l'aide des câbles Ethernet RJ45 fournis (visualisation du connecteur en figure 8 ci-contre).

Figure 8 : Connecteur RJ45

Ce type de câble est composé de 8 connexions électriques dans une gaine robuste aux déformations et interférences électriques. C'est donc un choix pour une communication sûre et très rapide (jusqu'à plusieurs GBit/s) et entre plusieurs systèmes communicants, notamment avec le protocole Ethernet.

V.3.1 Détermination du support de communication

Pour implémenter cette fonctionnalité aux cartes, nous avons utilisé le logiciel STM32CubeMX, qui nous permet de visualiser et configurer les différentes Pins et fonctionnalités et horloges du microcontrôleur. Il suffit d'ouvrir le fichier de configuration IOC (Integrated Development Environment Output

Configuration) généré par TouchGFX Designer à la racine du projet. Voici en figure 9 suivante un aperçu de l'interface.

A l'aide de cette interface, il est possible d'accéder à toutes les différentes fonctions disponibles sur les cartes STM32H7 en naviguant dans le menu de gauche. On retrouve ici des fonctionnalités cœur du

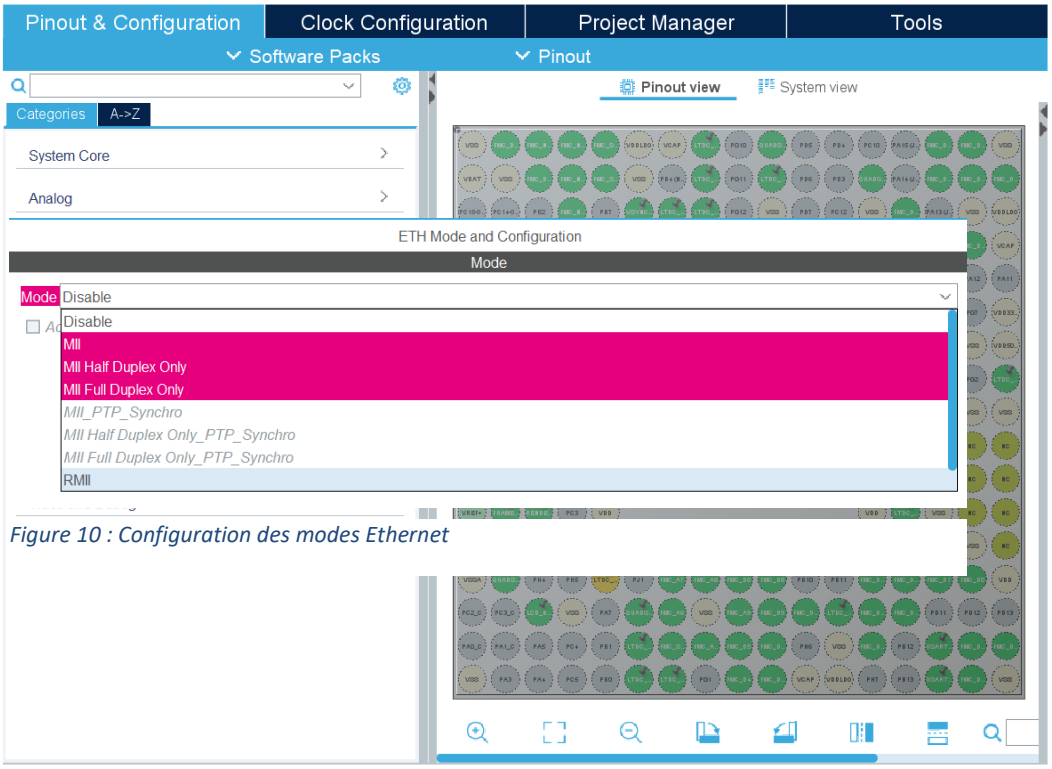


Figure 10 : Configuration des modes Ethernet

Figure 9 : Interface de STM32CubeMX

microcontrôleur, de communication ou d'alimentation. On retrouve par exemple l'activation de FreeRTOS dans le sous menu *Middleware and software Packs.* La sélection d'un pin sur la représentation du STM32 permet de voir quelle fonction microcontrôleur peut être assignée au pin. CubeMX permet alors d'activer la communication Ethernet sur les STM32 le supportant ce qui est le cas de celui sur notre carte. On voit cela sur le choix possible en figure 10 suivante, capture d'écran du menu de sélection après un clic sur Connectivity -> ETH.

On distingue ici deux modes d'Ethernet : Le Media Independent Interface (MII) et le Reduced Media Independent Interface (RMII). En repérant les pins utilisés pour la fonction Ethernet de la carte sur la schématique des cartes STM32H7 sur la figure 11 ci-dessous, on peut justifier le choix de l'utilisation du RMII par le fait qu'il est le seul mode présent et utilisable de la carte. CubeMX connaît les spécificités de la carte, et ne permet pas d'utiliser les autres modes.

Les labels de signaux dans la partie gauche de la figure 11 appariassent connectés aux pins du microcontrôleur sur une autre page de cette schématique. On les retrouvera dans la suite de cette partie du rapport.

Nous nous sommes ensuite servis de différents tutoriaux disponibles sur internet pour configurer une carte capable de transmettre des données via le câble Ethernet RJ45. Malheureusement, l'implémentation de différents codes en suivant divers tutoriaux ne virent aucun succès, la carte ne transmettant dans chaque essai aucune réponse au ping envoyé par le PC. Nous avons persévéré dans nos recherches durant plusieurs séances pour l'implémentation de différents codes exemples proposés par STMicroelectronics. Cependant, nous avons dans chaque cas rencontré différents problèmes, et aucune donnée n'a été transmise ni reçue entre le PC et la carte.

Nous avons donc décidé après deux séances de projet de faire appel à Mme Lavault, enseignante à Polytech Grenoble dans l'utilisation de bus de transmission et d'interfaçage, avec qui nous avons étudié différents types de protocole de transmission de données. Nous avons pensé que son aide pourrait nous éclaircir un peu plus le sujet de la configuration d'un tel protocole de communication, dans le cas d'une utilisation passée avec STM32.

Pour donner suite à cette demande, nous nous sommes de nouveau penché sur les problèmes rencontrés

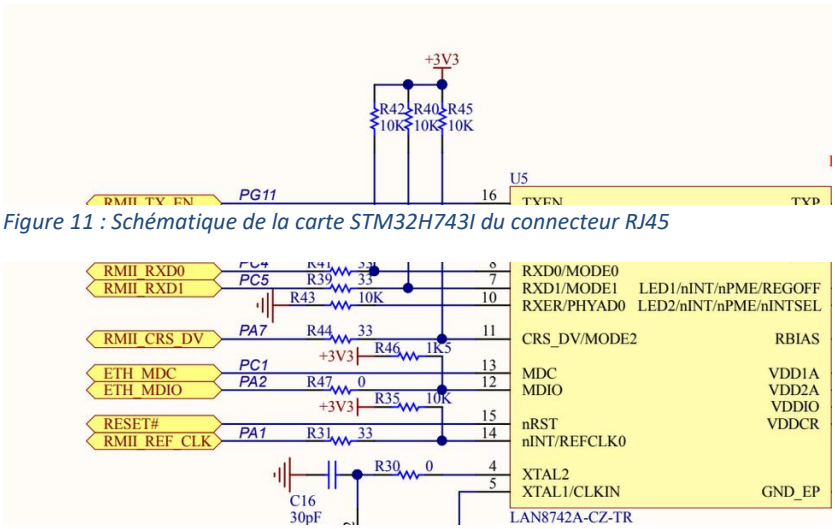


Figure 11 : Schématique de la carte STM32H743I du connecteur RJ45

lors de l'implémentation de la configuration Ethernet sur la carte. Pour faire table rase, nous sommes partis de notre code du Morpion pour tenter une communication à partir du code existant. La simple configuration des pins nécessaires au fonctionnement du port Ethernet avec CubeMX a mené à l'arrêt de l'écran.

En se penchant sur les pins utilisés par la fonction Ethernet, on retrouve les pins PB8, PC1, PC2 et PA2, qui sont également utilisés par l'écran tactile. On peut retrouver la liste en ouvrant CubeMX puis System Core dans le menu à gauche, GPIO -> Group by Peripherials -> Ethernet. De la même façon, Group by

Peripherials -> GPIO permet de voir les pins utilisés par l'écran tactile. Parmi elles, on retrouve la pin de réinitialisation de l'écran LCD sur PA2 (LCD_RESET) qui permet d'initialiser l'affichage de l'écran, et la pin MCU_ACTIVE sur PB8 qui quant à elle indique l'état d'activité du microcontrôleur. On retrouve également les pin FRAME_RATE sur PC1 et RENDER_TIME sur PC2, qui gèrent respectivement la fréquence de rafraîchissement de l'écran et la mesure temporelle du rendu de l'image.

En cherchant à configurer la fonction Ethernet à partir de notre projet Morpion, c'est à ce moment que l'on entre en conflit avec les pins configurés pour le fonctionnement de l'écran tactile. On obtient alors la capture d'écran de CubeMX en figure 12 suivante.

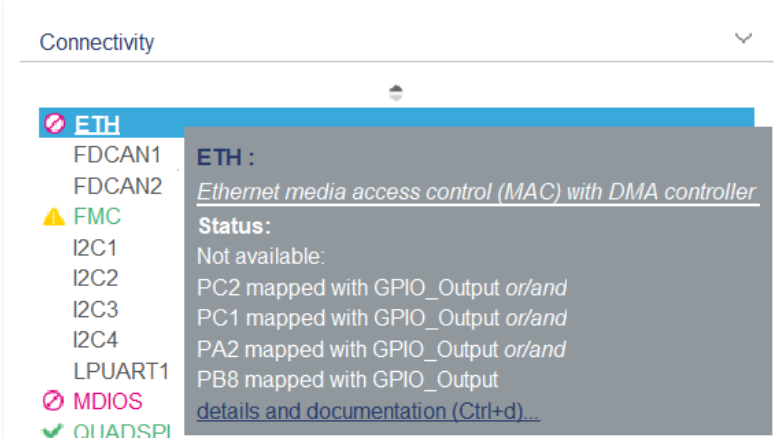


Figure 12 : Incompatibilité de la Fonction Ethernet avec la configuration de TouchGFX

La carte étant configurée pour utiliser les pins PC2, PC1, PA2 et PB8 pour différents signaux de l'écran tactile et de la fonction Ethernet, nous sommes dans l'incapacité de compléter la demande de notre porteur de projet pour l 'utilisation des câbles RJ45.

Nous avons ainsi revu les demandes du projet avec M. LEMASSON concernant le protocole de communication entre carte à mettre en place. Plusieurs nouvelles options se sont alors offertes à nous, nous laissant libres d'utiliser ce que l'on trouve le plus pratique pour réaliser la communication entre cartes. Parmi elles figurent :

- Le protocole SPI (Serial Peripheral Interface)
- Le protocole I2C (Inter-Integrated Circuit)
- La liaison série UART (Universal Asynchronous Receiver/Transmitter)
- La protocole USB (Universal Serial Bus)

Les protocoles SPI, I2C et USB nécessitent qu' une carte joue le rôle de Maître de transmission et une autre à jouer le rôle Esclave. À la suite d'un conseil de M. LEMASSON, ces protocoles ont été écartés car ils viendront limiter la communication entre plusieurs cartes, car proposant une configuration statique de la communication.
Nous avons donc proposé l'utilisation de la communication UART ou liaison série, qui permettrait d'envoyer et recevoir des données plus librement. Elle permettrait de récupérer directement les signaux de transmission et réception sur les pins PB14 et PB15, liés avec le connecteur DE9 sur présent sur la carte. Ce support de communication présente cependant des inconvénients. Parmi eux figure l'indisponibilité des câbles RS232 dans les réserves du Fablab MASTIC. Cela limite également la communication à deux communicants, la liaison série étant une liaison point à point.

Pour la suite du projet, nous utiliserons donc la liaison série RS232 pour les communication entre cartes. Avec l'indisponibilité des câbles au fablab, nous en emprunterons chaque séance en salle 306 de Polytech, ou nous avons nous même réalisé des travaux pratiques avec ces câbles.

V.3.2 Adaptation du jeu Morpion

A ce stade du projet, nous nous sommes fixé la contrainte de produire un code identique sur toutes les cartes sur lesquelles on souhaite utiliser notre application. Cela nous demandera la réalisation d'un code général, pour permettre une installation rapide sur un grand nombre de cartes.

Après plusieurs tests de la liaison série RS232 entre les deux cartes, nous choisissons d'utiliser la fonction de réception de trame en interruption implémentée par la librairie STM32 HAL. Pour l'envoi de caractère, cette librairie et sa fonction d'envoi ne nous a pas permis d'obtenir un résultat convenable. Nous nous sommes donc tournés vers une fonction écrite en séance pour l'envoi d'un seul caractère, utilisant directement les registres du microcontrôleur prévus pour la communication UART.

Pour l'établissement de la communication entre deux cartes, il faut qu'elles s'envoient mutuellement des codes pour se retrouver finalement dans le même état, et se synchroniser. Nos premiers tests ont consisté à attendre avec une boucle while la réception d'un caractère. Cependant, cette méthode bloque l'affichage de TouchGFX à l'écran, n'étant plus disponible pour sa mise à jour.

Sur l'interface de l'application, on sait que l'utilisateur est amené à choisir s'il veut jouer au Morpion sur une seule carte ou sur des cartes différentes. Avec un appui sur Multijoueur, on a alors implémenté le choix d'être "host" ou rejoindre la requête d'un host. Cette action est annulable avec le bouton Cancel, pour interrompre à tout moment le processus de communication si aucune carte répond. Ces fonctionnalités d'envoi et de réception de requêtes sont implémentés par les fonctions *Host_Task* et *Join_Task* dans le fichier TouchGFX/gui/menumulti.cpp, et contiennent une boucle while. Lors de l'appui sur les boutons correspondants, on crée des tâches RTOS exécutant le code de ces fonctions. La définition des tâches est réalisée de la même manière que celle de TouchGFX, qui est plutôt intuitive. L'utilisation de tâches RTOS ne bloquera pas l'affichage de l'écran, la ressource microcontrôleur étant allouée à TouchGFX en parallèle de ces fonctions d'initiation et attente de communication. Les deux tâches *Host_Task* et *Join_Task* s'arrêteront et seront détruites lorsque la communication est établie.

Dès que les deux cartes ont été synchronisées, on peut commencer à jouer. On effectue une transition d'écran vers Morpion, initialisant dans le même temps l'affichage et la mémoire du jeu.

Morpion étant un écran TouchGFX unique pour le jeu sur une seule carte et plusieurs cartes, nous avons naturellement modifié le code écrit jusqu'à présent pour son adaptation au mode de jeu. Cela commence par la définition d'une variable globale playerID, définissant l'état Host ou Join du joueur sur une carte dès l'appui sur les boutons correspondants, visibles en figure 2. On note ici que le joueur Host commence à jouer par une croix. Avec une séparation en if appliqué à playerID du code de chaque fonction du jeu, nous arrivons à exécuter une partie de code supplémentaire pour l'état multicartes.

Pour la même raison que lors de la synchronisation des cartes, nous définissons une tâche RxTTTTask (pour Rx, signal de réception UART, Tic-Tac-Toe étant le nom anglais du jeu du Morpion). Celle-ci est responsable de l'association des caractères reçus par liaison série à une commande de mise à jour de l'affichage sur l'écran. On s'assure que cette tâche est détruite par les deux cartes lorsque l'un des utilisateurs appuie sur le bouton Quitter, affichant à nouveau l'écran MenuMorpion de l'application

TouchGFX. En effet, l'envoi d'un caractère 0x4A est associé au bouton quitter, et supprime la tâche dans la méthode quit_game().

L'appui sur un bouton du jeu par un joueur doit permettre l'affichage de la modification sur les deux cartes en réseau, mais aussi la modification de la mémoire. Cela permet un suivi du jeu par les deux cartes, pour qu'elles restent d'une certaine manière indépendantes. Cette fonctionnalité est implémentée par l'exécution de la fonction PlayMove sur les deux cartes, avec un paramètre supplémentaire *rxcmd*. Celui-ci notifie si la fonction PlayMove a été appelée par réception sur liaison série ou non. De plus, un joueur venant de jouer ne doit pas être en mesure de jouer à nouveau. Ses boutons de jeu seront donc désactivés jusqu'à la prise en compte d'une action du joueur sur l'autre carte.

Lors de l'exécution de la fonction PlayMove à partir d'un caractère reçu, l'écran ne se met pas à jour de la même façon que lors d'un appui sur un bouton, et une croix ne sera pas affichée sur la carte Join même si le code est bien exécuté. En effet, la tâche TouchGFX n'est pas notifiée de la nécessité de mettre à jour l'écran et ne prend pas en compte le changement. Cette mise à jour forcée peut être implémentée dans la fonction *tick* générée par TouchGFX dans le fichier TouchGFX/gui puis Model.cpp, exécutée à chaque frame à 60 mises à jour par secondes de l'écran.
Avec cette particularité, son exécution doit être le plus rapide possible pour ne pas altérer les performances de l'application. Nous définissons donc une variable globale notifiant la nécessité d'une mise à jour, modifiée depuis la tâche RxTTTTask, qui sera testée à chaque tick. Cette fonction gère également une demande de réinitialisation du jeu et de l'écran par liaison série, mettant à jour l'affichage de la totalité des éléments Croix ou Cercle défini pour le jeu.

VI Diagramme de Gantt du projet

Durant ce projet, nos réalisations étaient archivées au jour le jour dans un journal de bord. Cela nous a permis de bien comprendre l'état du projet chaque semaine et ce qu'il reste à accomplir pour son avancement. Nous avons alors pu définir les étapes importantes du projet, et leur répartition dans le temps avec un diagramme de GANTT. Dans le diagramme de Gantt (Figure 13) fait sur Excel, les tâches communes sont en rouge, les tâches de Bastien, Antony, Romain sont respectivement en jaune, vert et bleu.

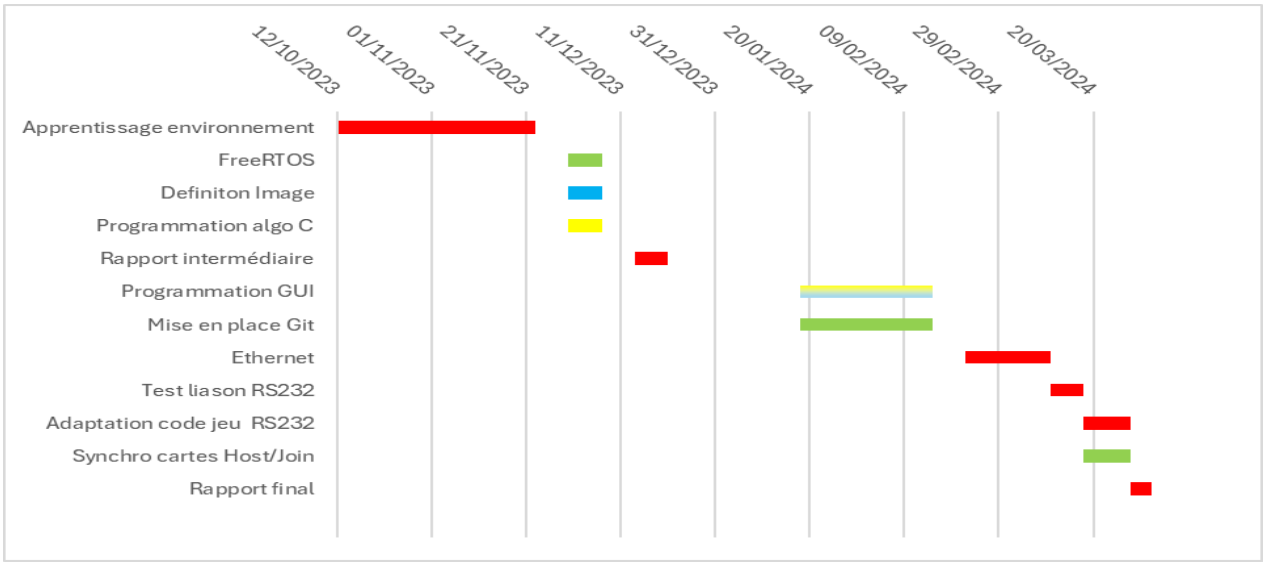


Figure 13 : Diagramme de Gantt du projet

Au début du projet nous avons dû apprendre un nouvel environnement de travail avec de nombreux logiciels tels que Stm32Cube IDE et TouchGFX. Cela est visible sur le diagramme de GANTT en figure 13, marqué par une longue période de recherche de l'utilisation des logiciels. Ensuite, nous nous sommes séparé naturellement les tâches pour avancer plus rapidement sur le jeu du morpion, son algorithme et la définition des images et objets qu'il utilise. Le dépôt Git a été mis en place dès lors que le jeu du Morpion avait une base solide. Cela nous a pris du temps en parallèle du développement du code, mais a représenté finalement un bon investissement pour la progression et la continuité du projet. Vient ensuite l'établissement de la communication des cartes, avec un début par la communication Ethernet difficile. Ce type de communication n'a pas abouti, et nous avons poursuivi nos recherches sur la liaison RS232. On voit également que les rapports intermédiaire et final ont été discutés en séances, avec une préparation en dehors des séances de projet.

En complément du diagramme de Gantt, on donne en figure 14 la représentation graphique sur Git Kraken de chaque modification apportée au code sur le dépôt GitHub. On peut y voir tout le travail effectué en parallèle, notamment avec l'ajout de l'écran Menu Morpion au cours du projet.

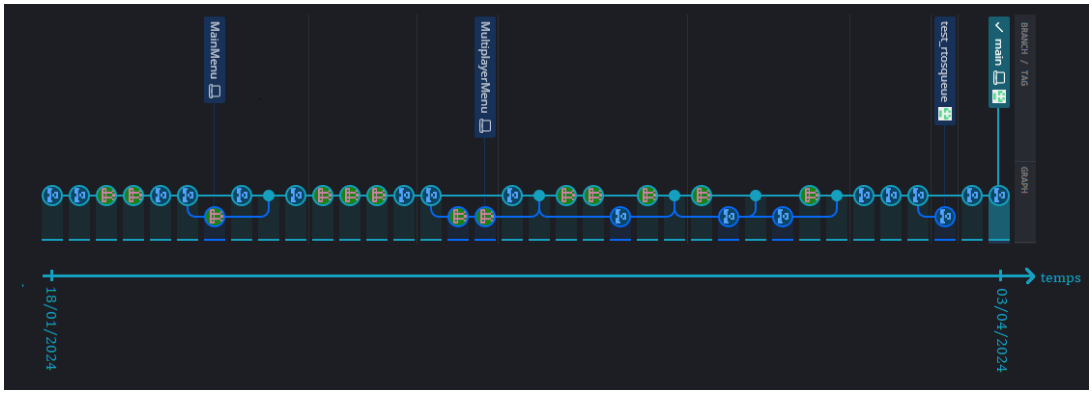


Figure 14: Historique du Git

VII Bilan

VII.1 Application finale au 05/04/2024 et bilan technique

Le 5/04/2024, soit la date de rendu de ce rapport, nous avons sur notre application les fonctionnalités effectives et testées suivantes :

- Navigation entre les écrans
- Jeu du Morpion sur une seule carte
- Synchronisation des cartes pour le démarrage du jeu sur deux cartes différentes
- Réception et mise à jour de l'écran pour l'appui d'**un** bouton
- Le jeu sur deux cartes différentes peut être quitté par les deux parties à tout moment

Il reste donc à faire :

- Processus du jeu sur deux cartes différentes : échange de la possibilité de jouer entre les cartes
- Réinitialisation du jeu : à l'état de prototype

La complétion de ces deux derniers points nous permettra de compléter nos objectifs fixés en début de projet, soit le développement d'un jeu simple pour se familiariser avec l'utilisation des cartes. Nous sommes confiants sur notre capacité à expliquer notre code en détail pour la reprise du projet, et le développement d'applications plus complexes.

La fin du projet en lui-même est définie par sa présentation à la journée des anciens IESE le 11/04/2024. D'ici là, nous comptons bien finir le projet pour présenter une version du code complet lors de sa démonstration. En effet, nous nous estimons proches du but, et quelques heures de travail supplémentaires nous permettront de mener le projet à son terme.

VII.2 Livrables

Nos rendus lors de ce projet sont notre code produit et ce rapport. Le code de notre application est disponible au lien GitHub en partie V.2.1 de ce rapport ou en référence bibliographique.

VII.3 Ressenti collectif

Le projet de programmation de cartes à écran tactiles est un sujet donnant une grande liberté dans la création d'applications et apporte de la motivation dans l'idée de pouvoir faire évoluer le projet. Il nous offre la possibilité de mettre en pratique des connaissances acquises auparavant et durant l'année en C++ et en transmission de données. L'implication de chacun dans le projet a pu permettre à chaque membre du groupe d'apporter une plus-value, en se focalisant naturellement sur les tâches à réaliser qui nous faisaient envie. En résumé, nous sommes satisfaits de notre travail accompli pour ce projet, en étant convaincu qu'il nous a apporté beaucoup de connaissances en programmation embarquée C++.

VII.4 Perspectives

Nos réalisations cette année ont permis d'ouvrir le projet en éclairant les différents points importants pour la bonne compréhension des différents outils utilisés. Les bases acquises durant cette année pourront servir de référence pour les futurs étudiants qui cherchent à donner vie à leur propre création dans le projet.

Il serait intéressant pour les années à venir de repasser sur le code du Morpion, car il met en œuvre les différentes fonctionnalités offertes par la carte et par TouchGFX sur un jeu simple. Des améliorations du jeu pourraient être apportées, du côté graphique comme du côté programme pour optimiser son fonctionnement tout en apprenant à utiliser la synchronisation de deux cartes.

Plusieurs applications plus complexes peuvent également être développées dans le futur, comme le jeu de bataille navale, qui est un exemple parfait et annoncé sur un écran TouchGFX pour la réalisation d'un jeu en réseau sur deux cartes.

Le projet est relativement libre, ce qui permet à n'importe quel étudiant créatif d'y apporter sa part.

VIII Références bibliographiques

Site internet du Fablab MASTIC : https://fabmstic.imag.fr/

Références des cartes STM32H7 à disposition :

https://www.st.com/en/evaluation-tools/stm32f779i-eval.html

https://www.st.com/en/evaluation-tools/stm32h747i-eval.html

https://www.st.com/en/evaluation-tools/stm32h753i-eval.html

Package de code exemple STM32H7 :

https://www.st.com/en/embedded-software/stm32cubeh7.html

Documentation TouchGFX : https://support.touchgfx.com/docs/development/ui-development/touchgfx-engine-features/backend-communication

Documentation RTOS : https://freertos.gitbook.io/mastering-the-freertos-tm-real-time-kernel

GitHub du code du projet : https://github.com/antcsny/Projet_Ecrantactile

Accès au dossier Google Drive du projet : https://drive.google.com/drive/folders/1TWAGRhnJntIqhIxkdbxocL9K7Bg4dull?usp=sharing

Tutoriels suivis pour la communication Ethernet

https://community.st.com/t5/stm32-mcus/how-to-create-project-for-stm32h7-with-ethernet-and-lwip-stack/ta-p/49308

https://youtu.be/Wg3edgNUsTk

https://youtu.be/8r8w6mgSn1A

https://github.com/stm32-hotspot/STM32H7-LwIP-Examples

Documentation librairie HAL STM32

https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf