Centre de formation professionnelle technique

9 juin 2020

Travail Pratique Individuel

MonoBattle

Documentation Technique

Nelson Jeanrenaud

Figure 1 – Logo de l'application

Version 1.0

Table des matières

1 Introduction

MonoBattle est un jeu de bataille navale à 1 ou 2 joueurs utilisant Monogame en Csharp. Ce travail est réalisé dans le cadre de mon *Travail pratique individuel* (TPI) pour valider ma formation d'informaticien orientée développement d'application. Ce document est le manuel technique de mon application, il présente les différentes étapes de conception du projet.

2 Rappel du cahier des charges

2.1 Organisation

Elève :
- Nelson Jeanrenaud, nelson.jnrnd@eduge.ch

Maître d'apprentissage :
- Stéphane Garchery, stephane.garchery@edu.ge.ch

Experts :
- Pierre Conrad, pierre.conrad@etat.ge.ch
- Philippe Bernard, philippe.bernard@etat.ge.ch

2.2 Objectifs

Le but est de concevoir, en 88 heures, une application Csharp orientée objet qui permet de jouer une partie complète de bataille navale. L'adversaire peut être un joueur humain connecté depuis une autre machine en réseau local ou une 'IA' sur la même machine.

2.3 Règles du jeu

La bataille navale est un jeu dans lequel deux joueurs placent leurs bateaux sur une grille. Chaque joueur ne connaît pas la position des navires de l'autre et ils doivent, à tour de rôle, tenter de toucher les navires adverses en annonçant une case. Un navire est alors coulé, quand toutes ces cases ont été touchées par l'adversaire. Le premier qui parvient à faire couler toute la flotte adverse, a gagné.

2.4 Spécifications

L'application permet de jouer à la bataille navale en respectant les règles et les joueurs ne peuvent pas faire d'actions illégales. Un joueur se connecte à l'autre en entrant son adresse IP. Une fois la partie commencée chaque joueur place ses bateaux sur sa grille, ils peuvent également choisir de les placer aléatoirement.

Les éléments de jeux sont affichés et mis à jour sur l'écran de chaque joueur. Chaque joueur peut également abandonner la partie à tout moment avec un bouton prévu pour. Dans le mode solo, l'adversaire est remplacé par un ordinateur qui tir aléatoirement.

2.5 Matériel et logiciels à disposition

— 2 PC standard école, 2 écrans
— Windows 10
— Visual Studio
— Suite Office

2.6 Livrables

Pour les experts et le maître d'apprentissage :
— Documentation (PDF)
— Code source (PDF)
— Manuel utilisateur (PDF)

Pour le maître d'apprentissage uniquement :
— Une archive des sources du projet
— Le journal de bord

3 Méthodologie

Pour pouvoir réaliser ce projet, j'ai utilisé la méthodologie dite "méthode en 6 étapes". Cette dernière se présente sous la forme suivante :

1. **S'informer** : La première étape de mon projet a été de lire en détail l'énoncé qui m'a été fourni pour comprendre l'étendue de la tâche, ainsi que les librairies que je pourais utiliser pour certaines parties comme la connexion entre 2 machines.
2. **Planifier** : Avant de passer à la réalisation du projet j'ai mis en place un planning sur excel pour pouvoir organiser mes efforts. J'ai donc divisé les tâches à effectuer en plusieurs sous-tâches, j'ai estimé leurs durée et je les ai arrangées sur mon emploi du temps en fonction de leur importance et dépendance.
3. **Décider** : Durant l'ensemble de mon travail j'ai dû prendre des décisions à plusieurs reprises (méthode de travail, diagramme de classes, utilisation d'une technique etc). Je les ai alors documentées dans mon journal de bord.
4. **Réaliser** : Je me suis alors mis au développement de l'application et à la rédaction de la documentation.
5. **Contrôler** : Pour chaque fonctionnalitée je l'ai testée avec différents cas d'usage pour confirmer son bon fonctionnement. J'ai également mis en place des tests unitaires dans mon application pour vérifier le fonctionnement de mes fonctions.
6. **Évaluer** : J'ai terminé mon TPI par une conclusion qui auto-évalue mon travail entier.

4 Analyse fonctionnelle

Ce chapitre traite de la partie de l'application visible par l'utilisateur. Tout d'abord l'interface sera présentée en détail puis les fonctionnalités disponibles pour l'utilisateur seront énumérées.

4.1 Présentation de l'interface

L'interface contient un formulaire WindowsForm, la fenêtre de jeu Monogame et de multiple message d'erreur et d'avertissements.

4.1.1 La fiche de connexion

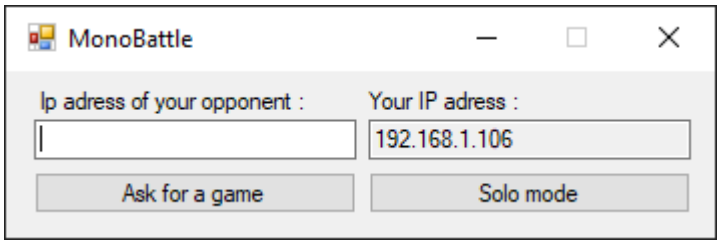


FIGURE 2 – Fenêtre du choix de mode de jeu et de connexion

C'est la fiche qui s'ouvre quand on lance l'exécutable, elle permet à l'utilisateur de choisir le mode de jeu : Réseau ou Local contre une IA. Le choix des modes est effectué a l'aide de 2 boutons.

Si le joueur veut jouer en réseau il est nécessaire qu'il remplisse la textbox avec son adresse IP. La fiche vérifie le format de cette adresse et s'il n'est pas correct la forme en informe l'utilisateur. Une fois le bouton de connexion appuyé la forme contacte l'application à l'adresse IP entrée. Une messagebox s'ouvre alors demandant si l'utilisateur veut accepter l'invitation. En fonction de la décision prise par le joueur, son partenaire en est informé avec une messagebox.

La forme possède également une textbox en lecture seul qui l'informe de son adresse IP.

4.1.2 La fenètre de jeu MonoGame

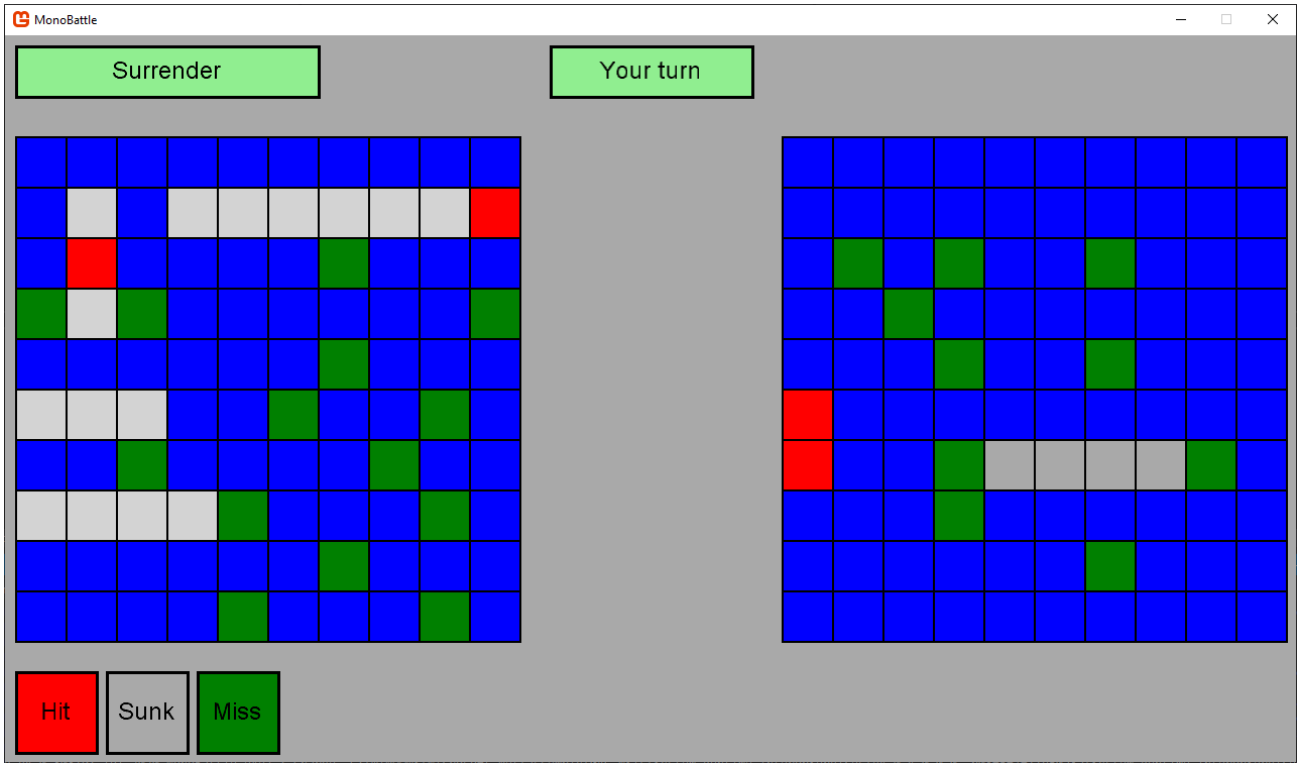


FIGURE 3 – Fenêtre de jeu juste après le placement des bateaux

C'est la fenêtre ou le jeu s'affiche. Lors de la phase de placement des bateaux, les navires du joueur sont placés sur la droite de l'écran. Il peut alors les déplacer sur la grille à la gauche de l'écran. Une fois satisfait du placement de sa flotte il peut appuyer sur le bouton "validate placement" pour passer à l'étape suivante.

Pendant la phase de jeu, la grille du joueur est affichée à gauche et celle de l'adversaire à droite. Si c'est son tour, le joueur peut appuyer sur le quadrillage de l'adversaire pour tenter de trouver les bateaux adverses. Le joueur est informé du tour courant avec une bannière qui se situe en haut de l'écran.

À tout moment, les joueurs peuvent abandonner la partie avec le bouton "surrender" situé en haut a gauche de l'écran.

4.1.3 Format de l'adresse IP incorrect

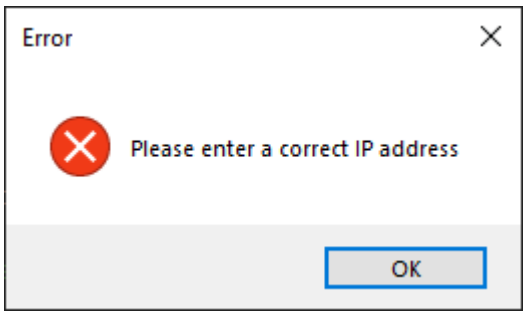


FIGURE 4 – Message informant que l'adresse IP est incorrecte

Ce composant MessageBox informe l'utilisateur que l'adresse IP qu'il a entrée ne peut pas exister. Il s'affiche quand il appuie sur le bouton "Ask for a game".

4.1.4 Invitation reçue

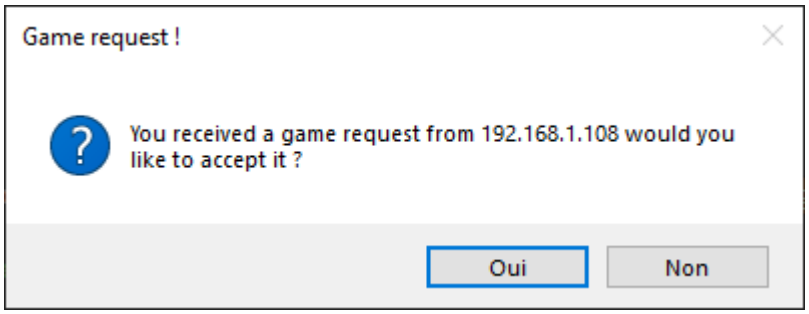


FIGURE 5 – Message informant qu'il a reçu une invitation

Quand un joueur appuie sur le bouton "Ask for a game" l'application ouverte sur la machine possédant l'adresse IP entrée reçois une invitation sous la forme de cette fenêtre. Il peut alors accepter ou refuser et l'autre joueur sera informé de la décision

4.1.5 Invitation refusée

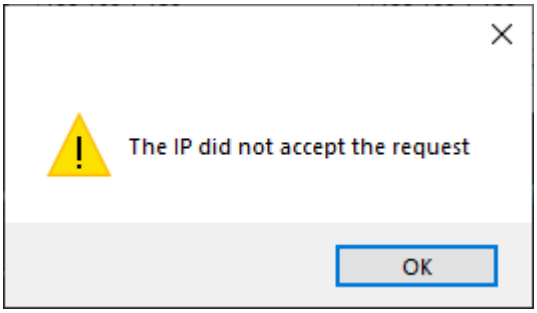


FIGURE 6 – Message informant que l'invitation a été refusée

Après que le joueur ai envoyé une invitation avec le bouton "Ask for a game", si l'invitation est refusé par l'adresse IP, ce message est afficher pour informer l'utilisateur.

4.1.6 Connexion échouée

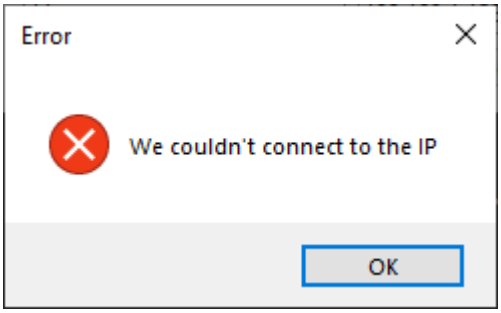


FIGURE 7 – Message informant que la connexion n'a pas pue être établie

Quand le joueur appuie sur le bouton "Ask for a game", si l'adresse IP ne répond pas après 20 secondes ce message s'affiche, l'informant que la connexion n'a pas pu être établie.

4.1.7 La partie va bientôt commencer

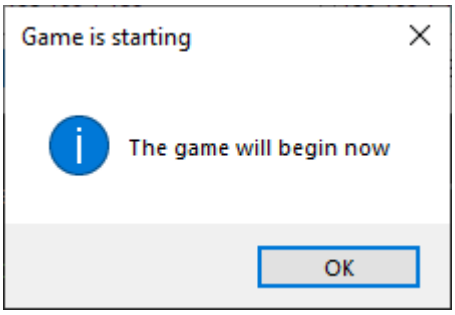


FIGURE 8 – Message informant que la partie va commencer

Cette Messagebox s'affiche quand l'invitation a été acceptée

4.1.8 Placement des bateaux impossible

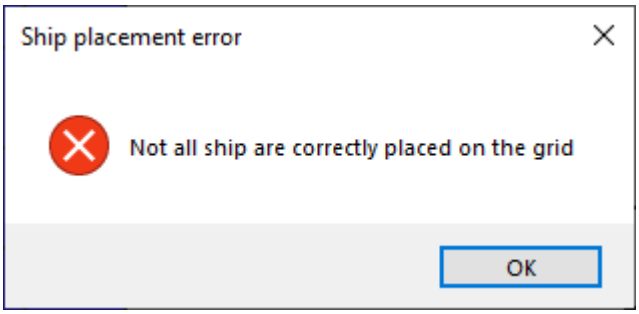


FIGURE 9 – Message informant que le placement des bateaux est impossible

Cette Messagebox apparaît quand l'utilisateur n'a pas placé tous les bateaux sur la grille et qu'il appuie sur le bouton de validation.

4.1.9 Abandonner la partie

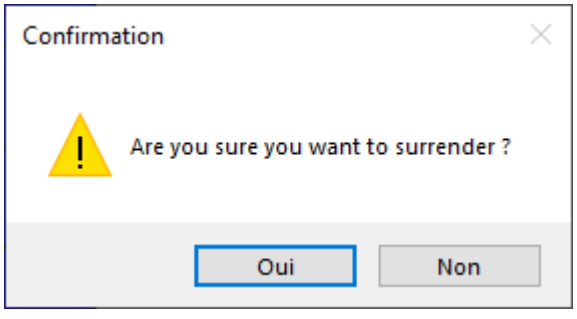


FIGURE 10 – Message demandant la confirmation du joueur avant la reddition

Ce message apparaît quand le joueur appuie sur le bouton de reddition, s'il confirme la partie se terminera, dans le cas contraire elle reprendra.

4.1.10 La fin de partie

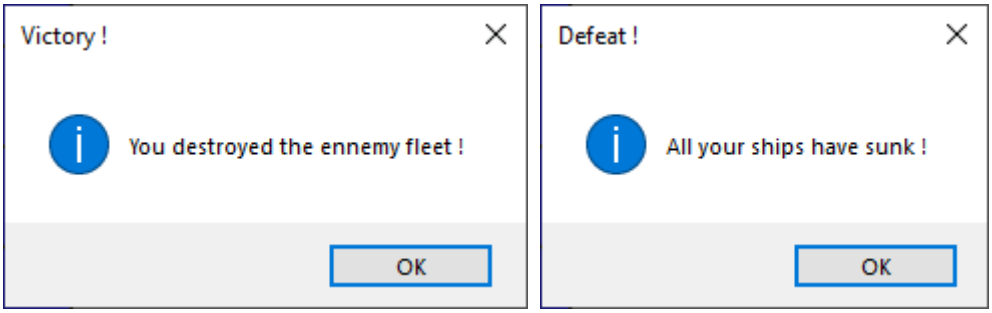


FIGURE 11 – Messages informant de l'issue de la partie

Ces MessageBox s'affichent quand la partie se finit pour informer les joueurs du résultat. Une fois le bouton Ok appuyé le jeu se ferme.

4.2 Fonctionnalités

4.2.1 Choisir le mode de jeu

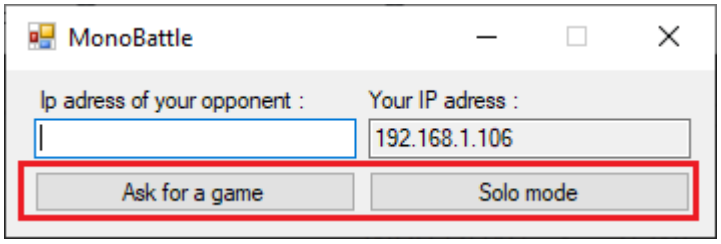


FIGURE 12 – Boutons de choix du mode de jeu

Une fois l'application lancée le joueur aura le choix entre jouer avec un autre joueur ou lancer le mode solo ou il affrontera une IA

4.2.2 Ce connecter avec un autre joueur

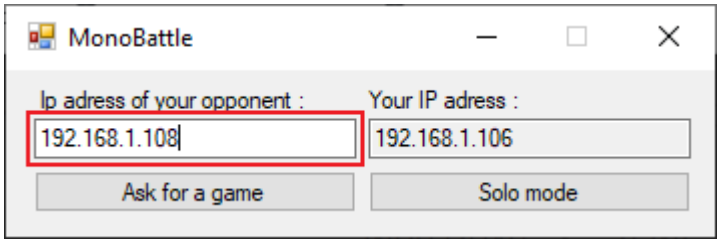


FIGURE 13 – Zone de texte pour choisir son adversaire

Si le joueur veut jouer avec une autre personne il peut entrer l'adresse IP du PC de son adversaire. Une zone de texte informe le joueur de sa propre adresse IP pour qu'il puisse la communiquer sans avoir à ouvrir une invite de commande. Une fois la demande envoyé, l'adversaire reçoit un pop up qui lui demande s'il veut rejoindre la partie.

4.2.3 Placer ses bateaux sur la grille

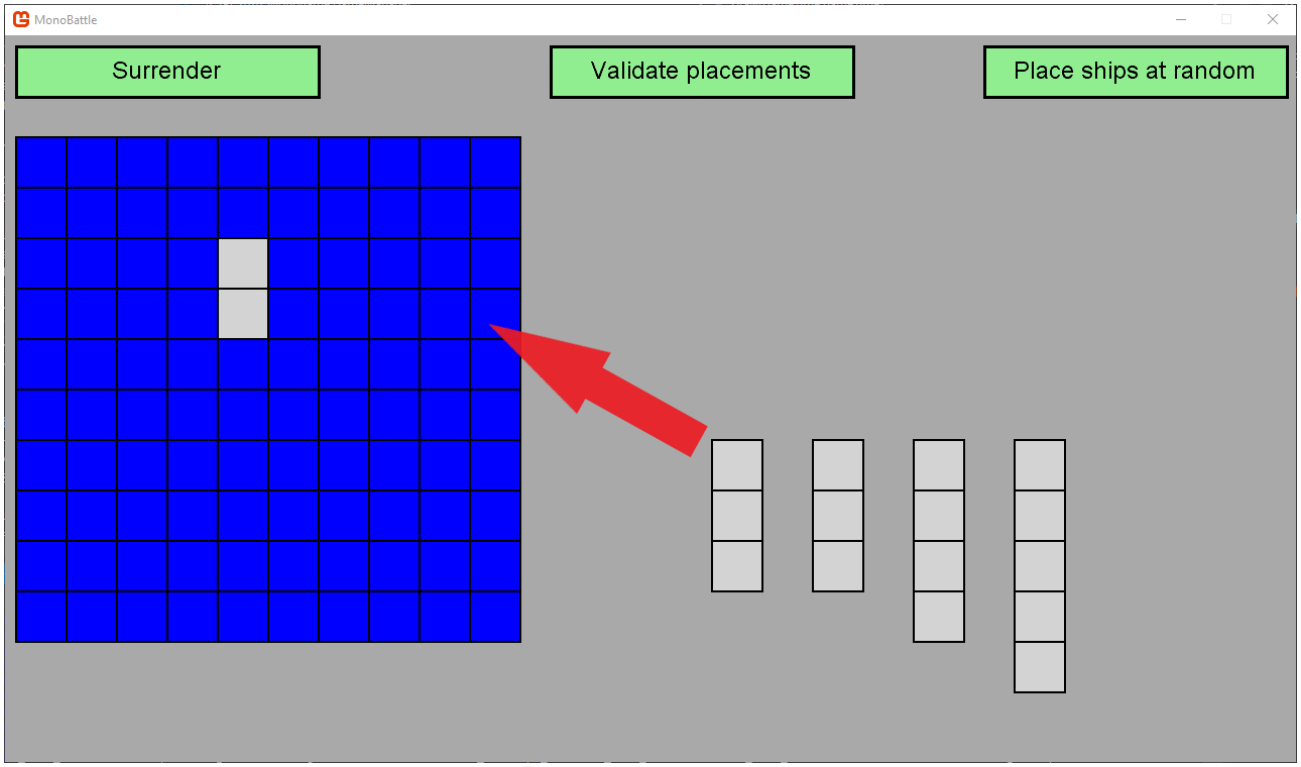


FIGURE 14 – Placement des bateaux

Quand le jeu se lance, les joueurs doivent placer leurs bateaux sur la grille, ils peuvent alors drag and drop les différents bateaux de leur inventaire sur la grille. Une fois tous les bateaux bien placés ils peuvent valider leur placement.

4.2.4 Placer les bateaux aléatoirement

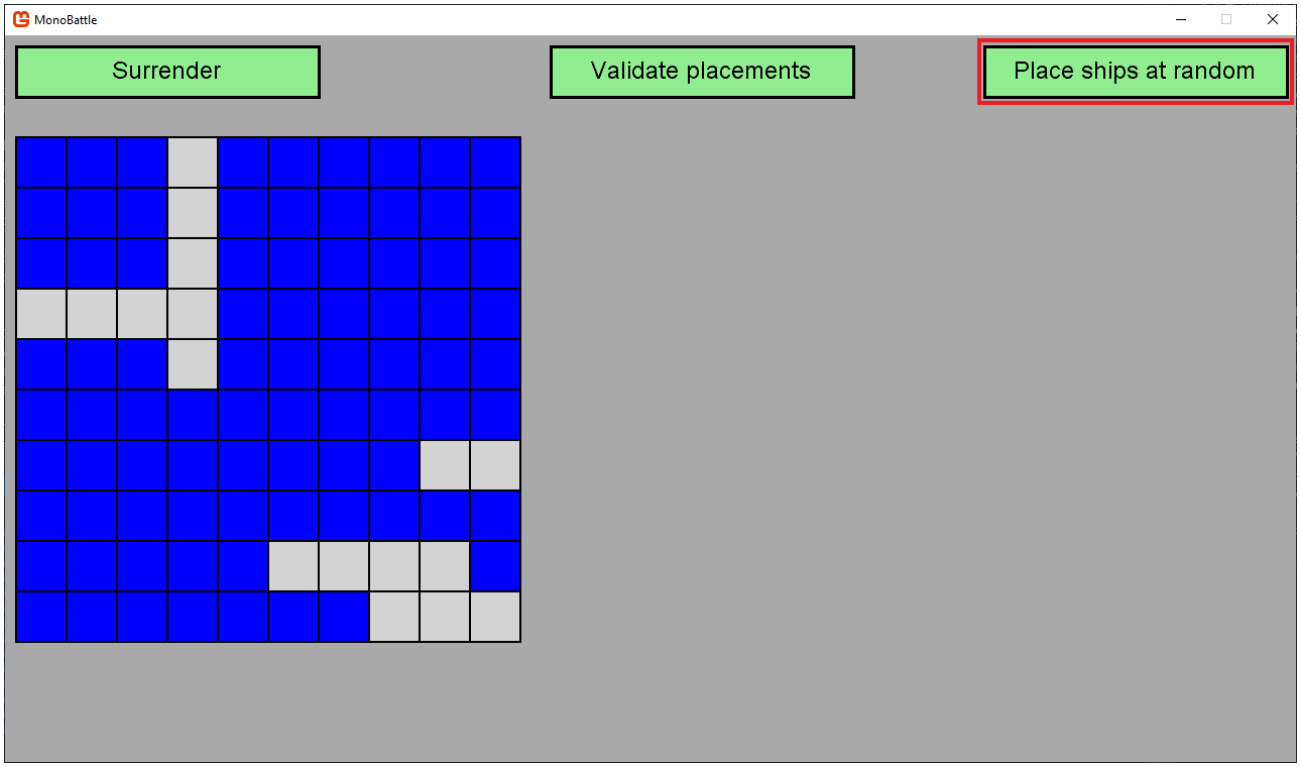


FIGURE 15 – Bouton de placement aléatoire

Durant cette phase, les joueurs peuvent également appuyer sur un bouton pour placer aléatoirement leurs bateaux sur la grille.

4.2.5 Tirer sur une case

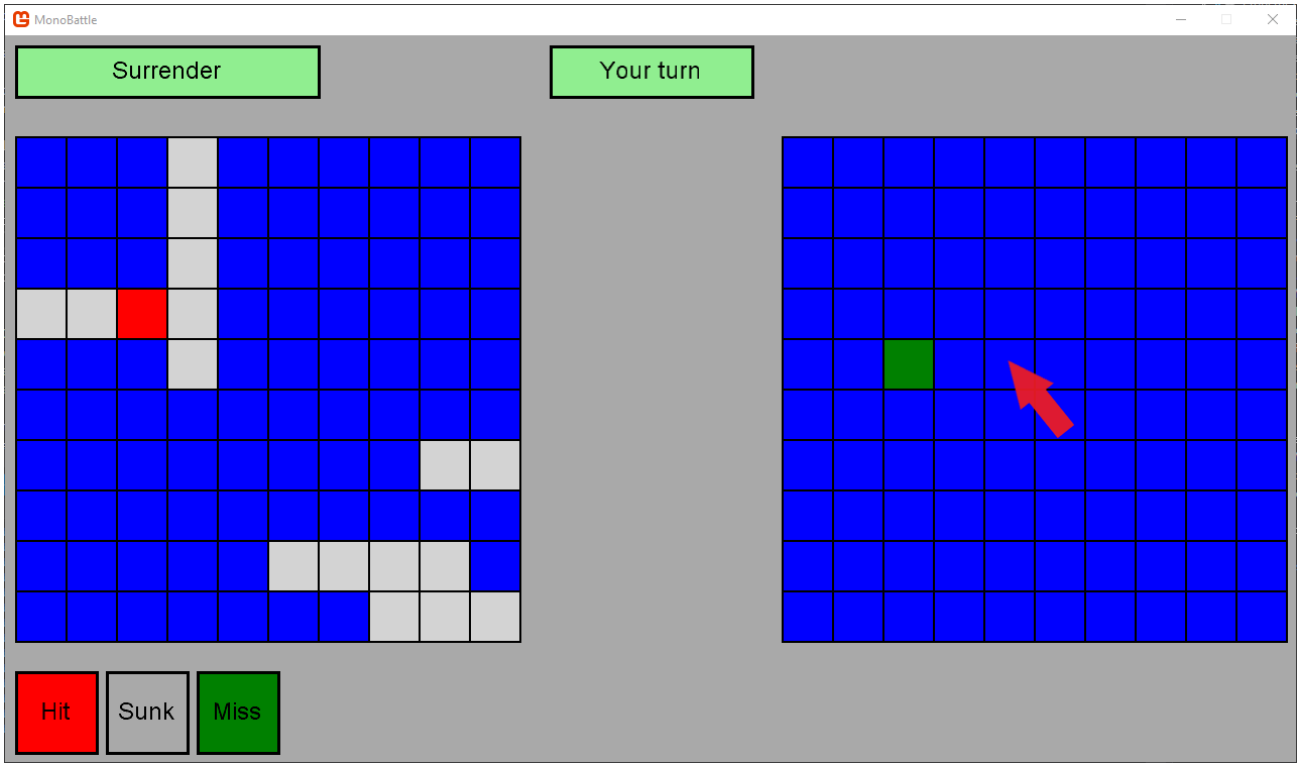


FIGURE 16 – Cliquer sur une case pour tirer

Une fois la phase de placement terminer, si c'est leur tour, les joueurs peuvent viser une case sur la grille adverse en la sélectionnant. Si elle est vide, occupé d'un bateau ou permet de couler un bateau, le plateau sera mis à jour en conséquence.

4.2.6 Les joueurs peuvent abandonner

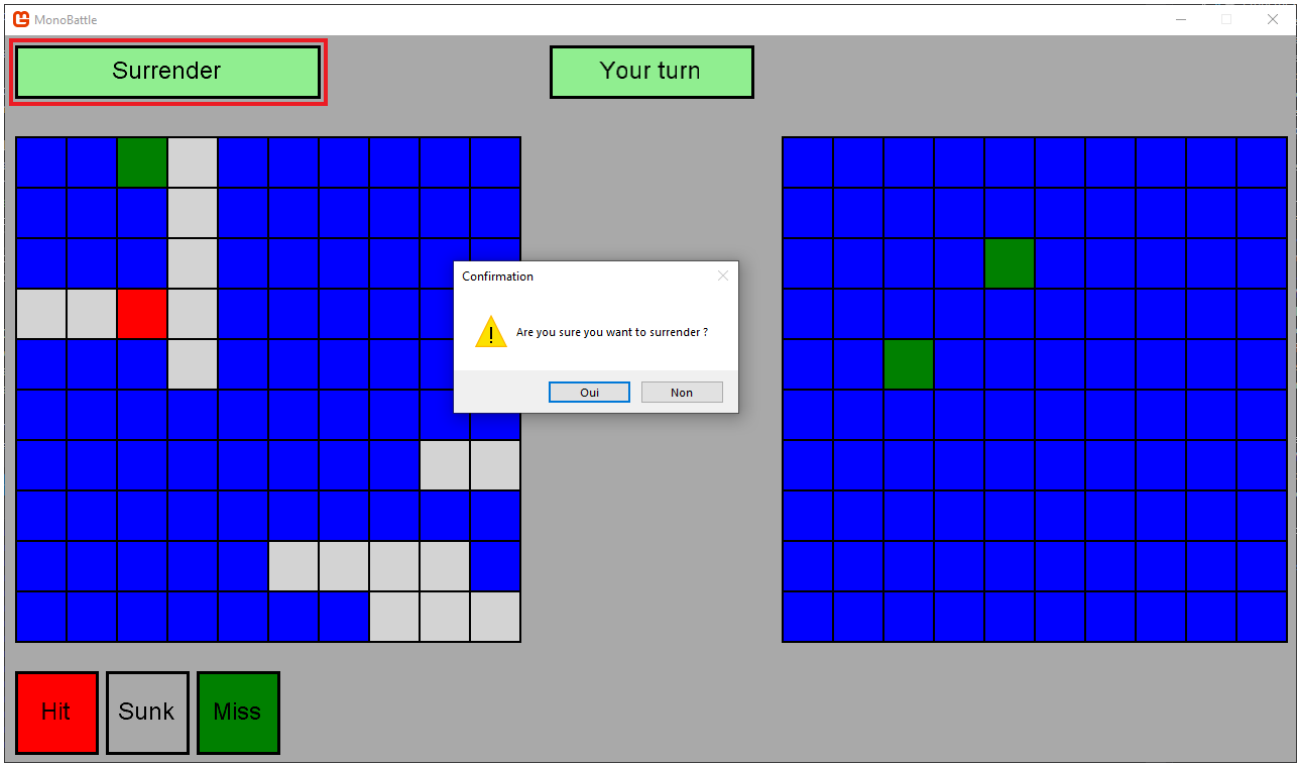


FIGURE 17 – Bouton d'abandon

Un bouton est mis à disposition des joueurs et ils peuvent l'utiliser pour abandonner la partie en cours. Un pop-up leur demande de confirmer leurs actions avant de terminer la partie.

5 Analyse Organique

Ce chapitre-là, au contraire, traite de la partie de l'application vue par le développeur. L'architecture du projet, les actions implicites du programme ainsi que les outils externes utilisés.

5.1 Architecture du projet

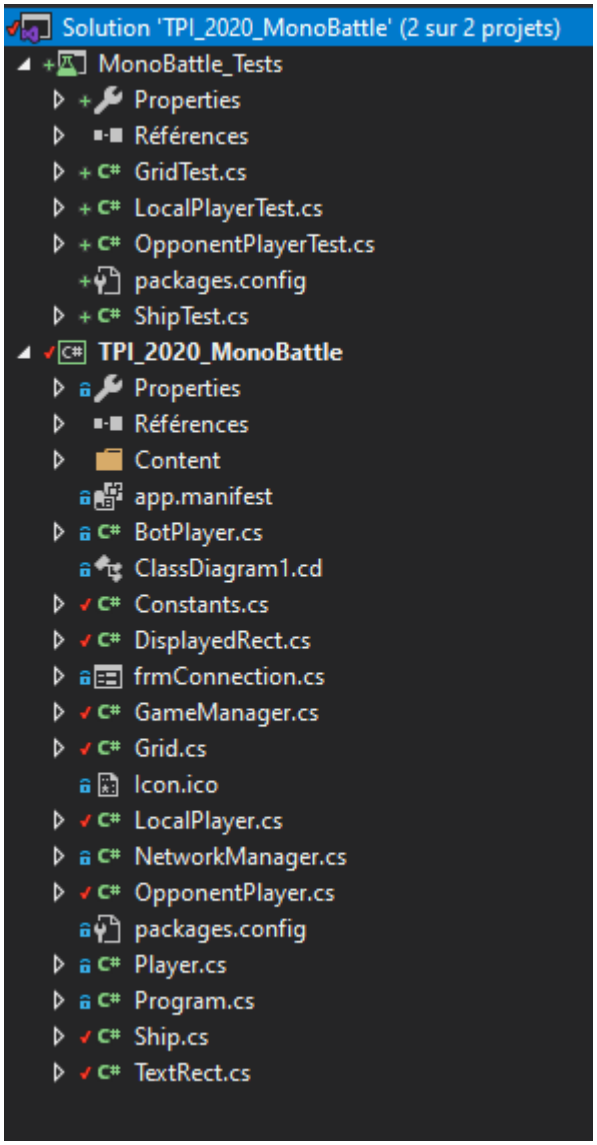


FIGURE 18 – Arborescence du projet

Le programme se trouve dans la solution TPI_2020_MonoBattle. Cette solution contient également le projet de test unitaire. Le projet monogame contient, quand à lui, l'intérgralité des classes de l'application, le formulaire Windows

forms et le dossier Content où est enregistrée la police de caractères pour les éléments de l'interface en jeu.

5.2 Explication de la communication pour le mode 2 joueurs

5.2.1 Introduction

Un des points de l'énoncé était de permettre aux joueurs de jouer ensemble sur 2 machines différentes. Pour ce faire j'ai opté pour un système sans serveur centralisé, les joueurs doivent connaître l'adresse IPv4 de leur adversaire et lui envoyer une invitation directement.

La problématique de cette méthode est qu'elle ne fonctionne qu'en LAN. Effectivement, a moins d'avoir sa machine directement connectée à internet avec une adresse IP publique, sans routeur, il n'est pas possible de connecter les 2 machines. Comme vous le savez surement, le router se charge de distribuer des adresses IP privées a ses ordinateurs et bloque le trafic entrant, à moins que ça ne soit le retour des paquets envoyés par votre ordinateur.

Il existe une méthode pour pallier a ce problème mais elle complexe et au-delà de mes compétences actuelles, en tout cas dans la limite de temps du TPI. Vous pouvez trouver un article très intéressant à ce sujet à **cette adresse**.

5.2.2 Librairie utilisée

J'ai donc utilisé une librairie nommée SimpleTCP pour sa facilité d'utilisation. Sur visual studio, il suffit d'installer le paquet NuGet "SimpleTCP". Vous pouvez trouver un guide d'utilisation en vidéo à l'adresse suivante : `https://www.youtube.com/watch?v=ve2LX1tOwIM`

5.2.3 Pourquoi TCP ?

Le protocole UDP est plus souvent utilisé dans les jeux en ligne mais j'ai préféré utiliser le protocole TCP pour les raisons suivantes :

1. **La connexion** : Le protocole est dit "connection-oriented" ce qui veut dire qu'il ouvre un canal de communication à 2 voie entre les 2 ordinateurs. Contrairement à UDP qui est "connectionless" et envoie directement les données sans se préoccuper d'établir une connexion.
2. **La fiabilité :** Le grand avantage de TCP est que les paquets sont numérotés et séquencés, ce qu'il veux dire que les paquets seront toujours

traités dans le même ordre que celui d'envoi. TCP garantit également l'arrivée des paquets à destination, si un paquet se perd et n'atteint pas le destinataire, il est remplacé et renvoyé.

3. **La vitesse :** La raison pourquoi UDP est plus souvent utilisé dans les jeux en ligne est car il est bien plus rapide que TCP. Mais pour un jeu de bataille navale la latence a beaucoup moins d'impact que pour un jeu de tir par exemple.

5.2.4 Choix du numéro de port

Ensuite, mon application utilise le port 51340 pour communiquer, il à été choisi avec l'aide **d'une base de données de ports utilisés connus**.

5.2.5 Codes de communications

Pour traiter les informations, les messages envoyés sont formatés de la manière suivante : TypeDeMessage%Contenu#. Le serveur qui reçoit coupe alors la chaine de caractère sur le % et le # pour récupérer le contenu et le type du message. Le serveur traite alors différemment le contenu reçu en fonction du type du message :

1. **ConnectionRequest :** C'est le premier message envoyé/reçu par l'application en temps normal. Il contient l'adresse IP de l'expéditeur et permet aux 2 applications d'établir une connexion.
2. **Shoot :** C'est les messages envoyés quand une application tente de deviner la position d'un bateau adverse. Il contient les coordonnées de la case en question au format "X,Y". Le destinataire répond alors avec le status de la case qui se trouve à ses coordonnées (hit, miss, sink).
3. **endgame :** Ce message est envoyé quand la partie est terminée, que ça soit parce que tous les bateaux ont été coulés, que le joueur ait abandonnés ou qu'il ait fermé l'application. L'application passe instantanément à la phase fin de jeu et annonce le vainqueur.

5.3 Description détaillée des méthodes de l'application

Dans ce sous-chapitre seront listé et expliquées les méthodes de l'application. Ainsi qu'une description sommaire de chaque classe.

5.3.1 Classe GameManager

C'est le gestionnaire de jeu, cette classe hérite de la classe Game de Monogame et se charge d'interfacer la fenêtre de jeu avec les autres classes de l'application.

Elle contient l'intégralité des autres classes à l'exception du formulaire Windows forms et des classes statiques.

5.3.1.1 Initialize

Méthode surchargée de Game.cs, elle a pour but d'initialiser l'ensemble des éléments de jeux. Elle est appelée quand le fenêtre frmConnection est fermée et que le script program.cs appelle la méthode Run héritée de Game.cs.

5.3.1.2 Update

Méthode surchargée de Game.cs, elle est appelée environ 60 fois par seconde par monogame. Elle contient la première machine d'état de l'application et selon la phase de jeu, elle vérifie le status de la souris pour relever les cliques sur les éléments de jeux ou l'interface. C'est aussi elle qui se charge de déplacer les bateaux à la position de la souris et de les aligner sur la grille.

5.3.1.3 Draw

C'est aussi une méthode surchargée de Game.cs, elle est appelée à chaque rafraichissement de la fenêtre de jeu. Son but est de dessiner l'écran de jeu, elle contient la seconde machine d'état de l'application et dessine les élements de jeu en fonction de la phase de jeu.

5.3.1.4 Shoot

Cette méthode privée prend en paramètre les coordonnées X,Y d'une case. Elle va tout d'abord vérifier que c'est le tour du joueur local de jouer, car il ne doit pas pouvoir effectuer de tir pendant d'autre phases. Si c'est le cas, elle va envoyer un message à l'adversaire avec la méthode SendShootMessage du NetworkManager. En fonction de la réponse qu'elle obtient sur l'état de la case, elle va mettre à jour le plateau de l'adversaire et passer au tour suivant.
Mais si la partie se termine avec ce tir, elle va appeler la méthode endgame pour passer à la phase de fin de partie. Également, si l'application adverse ne répond pas, elle va avertir l'utilisateur du départ de l'adversaire et fermer l'application.
Cette méthode est appelée depuis la méthode Update quand un clic sur une case est enregistré.
Si l'application est en mode solo, cette méthode n'appelle pas le NetworkManager mais la classe botPlayer.

5.3.1.5 ReceiveShot

Cette méthode publique prend aussi en paramètre des coordonnées X,Y. Comme la méthode précédente, elle vérifie que c'est bien le tour du joueur distant, le joueur ne doit pas recevoir de tir pendant les autres phases.
Si c'est bien le tour du joueur distant, l'application va vérifier le contenu de la case dont les coordonnées sont celles en paramètre avec la méthode ReceiveShot de la classe LocalPlayer et le retourner. Si c'est un bateau qui coule et que c'est le dernier de la flotte, un message de fin de partie sera envoyé à la place. La méthode endgame sera aussi appelée au lieu de passer au tour suivant.
Cette méthode est appelée dans le NetworkManager quand il reçoit un message de type "shoot" et pendant le tour adverse si le mode solo est enclenché.

5.3.1.6 NextTurn

Cette petite méthode publique change simplement la phase de jeu en fonction de celle en cours. Si c'est le tour du joueur local, la méthode change la phase au tour du joueur distant et inversement.
Comme dit plus haut, cette méthode est appelée dans les méthodes ReceiveShot et Shoot.

5.3.1.7 EndGame

Cette méthode publique prend en paramètre un booléen qui répond à la question "est-ce que le joueur local a gagné ?" En fonction de ce dernier, il va mettre à jour le champ _localPlayerWonGame et passer à la phase endgame.
Cette méthode est appelée quand l'utilisateur abandonne, que sont derniers bateaux et détruit, ou quand il reçoit un message du type endgame de l'adversaire.

5.3.1.8 OnExiting

Cet événement, hérité de Game.cs est appelé quand l'application se ferme pour n'importe quelle raison. Si l'application est en mode multijoueur, elle va envoyer un message à l'autre application pour l'avertir de son départ.

5.3.2 Classe NetworkManager

Cette classe sert à se connecter et communiquer avec l'autre application. Elle implémente le design patern singelton pour s'assurer qu'une seule instance tourne de la classe soit instanciée.

5.3.2.1 GetInstance

Comme dit juste avant, cette classe implémente le design patern singelton, cette méthode statique, retourne l'instance de la classe et si elle n'a pas encore été instanciée, elle l'instancie et l'enregistre dans le champ statique _instance.

5.3.2.2 StartServer

Cette méthode publique démarre le serveur local s'il ne l'est pas déjà et initialise tous ses variables. Elle est appelé au tout début du cycle de vie de l'application, avant que la forme de connexion s'ouvre.

5.3.2.3 Server_DelimiterDataReceived

Méthode privée "subscribed" à l'événement DelimiterDataReceived du serveur, elle est donc appelée à chaque fois que le serveur reçoit un message complet.
Elle découpe les messages reçu pour pouvoir les traités en fonctions de leur type (Voir **5.2.5**).

- **ConnectionRequest :** Dans ce cas, l'application a reçu une invitation, tout d'abord elle va vérifier si un client est déjà connecté. Pour éviter les problèmes si une 3ème machine essaie de se connecter. Une fenêtre Messagebox est alors ouverte, pour demander à l'utilisateur s'il veut accepter cette invitation. Si oui, l'application se connecte à l'IP qui a été envoyé avec l'invitation en appelant la méthode ConnectClient. Le NetworkManager va ensuite utiliser l'événement "ReceivedInvite" pour avertir la forme qu'une invitation a été reçue et si elle a été acceptée, de même le NetworkManager va répondre au message pour préciser la réponse à la demande.
- **Shoot :** Les coordonnées de la case visée sont envoyées au format X,Y, elles sont alors converties en Points et envoyé à la méthode ReceiveShot du GameManager. Le statut de la case sera également renvoyé en réponse.
- **EndGame :** La méthode engame du GameManager est appelé, avertissant le joueur, qu'il à gagné.

5.3.2.4 ConnectClient

Cette méthode publique permet de connecter se connecter à une adresse IP envoyé en paramètre. L'adresse est d'abord vérifié avec la méthode ValidateIPv4 et si le retour est positif, un canal de communication est ouvert avec l'adresse.
Cette méthode est appelée dans la méthode Server_DelimiterDataReceived et dans SendConnectionRequest pour pouvoir envoyer l'invitation.

5.3.2.5 SendConnectionRequest

Avec cette méthode publique l'application se connecte à l'adresse IP envoyé en paramètre et envoie un message du type connectionRequest (Voir **5.2.5**). Si l'application ne reçoit pas de réponse dans les 20 secondes elle considère l'invitation comme refusée. Elle retourne ensuite un booléen en fonction de la réponse reçue.
Cette méthode est utilisée dans la méthode SendRequest liée au clique du bouton de connexion sur la forme FrmConnection.

5.3.2.6 SendEndGameMessage

Similairement à la méthode précédente, cette méthode publique envoie un message de type endgame au client connecté.
Elle est appelée depuis le GameManager quand le joueur abandonne ou qu'il a quitté l'application prématurément.

5.3.2.7 SendShootMessage

Cette méthode publique envoie un message du type Shoot au client connecté avec les coordonnées en données paramètre. Elle retourne alors le status de la case communiqué par l'autre application.
Cette méthode est appelée dans la méthode Shoot du GameManager.

5.3.2.8 GetLocalIpAddress

Cette méthode statique et publique retourne l'adresse IP local de la machine. Elle est utilisée pour envoyer l'adresse Ip local avec le message du type connectionRequest et pour afficher l'adresse IP local sur le formulaire windows forms.

5.3.2.9 ValidateIPv4

Cette méthode statique et publique vérifie si une adresse IP est valide, elle va d'abord vérifier si elle contient 4 parties séparées par des points puis si ces chaines de caractères sont valides. J'ai trouvé ce code de vérification sur **ce thread**. Cette vérification est loin d'être parfaite mais combinée à un IPAdress.Parse, je l'ai jugé suffisante pour ce projet.
Cette méthode est utilisé dans la méthode ConnectionClient avant de tenter une connexion à une adresse.

5.3.3 Classe DisplayedRect

Cette classe représente un rectangle sur l'écran de jeu.

5.3.3.1 Draw

Cette méthode publique permet de dessiner le rectangle et ses bordures sur le spriteBatch en paramètre. Elle est utilisée partout dans l'application que ça soit pour les grilles, les bateaux ou les boutons.

5.3.3.2 CreateRectBorder

Cette méthode privée permet de créer les bordures du rectangle donné en paramètre en fonction de la valeur du champ _borderThickness. Elle retourne ces bordures sous la forme d'un tableau de rectangle.
Cette méthode est utilisée à la création du DisplayedRect et à chaque fois que son rectangle central est modifié.

5.3.3.3 IntersectWith

Cette méthode publique retourne un booléen en fonction de si un point donnée en paramètre est contenu dans le rectangle.
Cette méthode est utilisé pour vérifier les cliques sur les boutons et autre rectangle dans le GameManager.

5.3.4 Classe TextRect

Cette classe héritée de DisplayedRect permet de dessiner un rectangle avec du texte sur l'écran. Elle est utilisée pour afficher les boutons

5.3.4.1 Draw

Cette méthode publique surcharge la méthode draw du DisplayedRect. Elle fait appel a ce dernier pour dessiner le rectangle mais rajoute du texte en fonction des champs.

5.3.5 Classe Ship

Cette classe héritée de DisplayedRect représente un bateau.

5.3.5.1 Draw

Cette méthode publique surcharge la méthode draw du DisplayedRect. Elle dessine un carrée pour chaque case de longueur du bateau à des positions différentes en fonction de la rotation.

5.3.5.2 IntersectWith

Cette méthode publique qui surcharge le IntersectWith de DisplayedRect, retourne un booléen en fonction de si un point donné en paramètre est contenu dans l'hitbox du bateau c'est a dire l'ensemble de ces carrés.

5.3.5.3 Rotate

Cette méthode publique permet d'inverser le sense du booléen _isVertical et changer l'orientation du bateau. Cette méthdoe est appelée dans le gameManager.Update quand le joueur clique sur le bouton droit de la souris.

5.3.6 Classe Grid

Cette classe représesente une grille de bataille navale et permet de noter et afficher l'état de ses cases.

5.3.6.1 InitializeGrid

Cette méthode privée est appelée à la création d'un objet grid. Elle va créer un tableau a 2 dimensions de "SquareType" (type utiliser pour les différents états d'une case : empty, miss, hit, ship, sunk) en fonction des constantes de tailles du tableau.

5.3.6.2 Draw

Cette méthode publique dessine le quadrillage sur le spriteBatch donné en paramètre, chaque case est représentée par un DisplayedRect et la couleur de cette dernière dépend de son SquareType.

5.3.6.3 FindClosestSquareAnchor

Cette méthode publique retourne les coordonnées du coin en haut à gauche de la case dans laquelle se trouvent le point donnée en paramètre. Pour trouver à quel case le point appartient, cette méthode fait appel à FindSquare. Dans le cas ou un point n'appartient pas à une case du quadrillage, la méthode retourne le même point.

5.3.6.4 FindSquare

Cette méthode publique retourne les coordonnées X,Y dans la grille de la case qui contient un point donné en paramètre. Cette case est trouvé en divisant la

position du point relative au coin en haut à gauche de la grille par le nombre de case total.

5.3.7 Classe PlayerLocal

Cette classe qui hérite de la classe abstraite Player représente le joueur Local avec son quadrillage et ses bateaux.

5.3.7.1 ReceiveShot

Cette méthode publique retourne l'état de la case du quadrillage du joueur dont les coordonnées sont celles donnée en paramètre. Si la case est vide, elle est marquée comme raté. Si la case est occupée par un bateau, elle est marquée comme touché et si la case est la dernière case d'un bateaux alors celui-ci est intégralement marquée comme coulé et la méthode retournera également les coordonnées de ses autres cases.
Cette méthode est utilisée dans la méthode du GameManager du même nom.

5.3.7.2 ValidateShips

Cette méthode publique vérifie si le placement des bateaux du joueur respecte les règles du jeu. C'est a dire, qu'ils ne se chevauchent pas et qu'ils ne sortent pas du quadrillage. La méthode retourne un booléen en fonction de si ces règles sont respectées. Cette méthode possède également un booléen en paramètre qui permet de changer l'état des cases qui contiennent les bateaux.

5.3.7.3 PlaceShipRandom

Cette méthode permet de placer les bateaux aléatoirement dans la grille et en respectant les règles à l'aide de ValidateShips.

5.3.8 Classe OpponentPlayer

Cette lasse qui hérite de la classe abstraite Player représente le joueur distant, elle contient une grille mais pas de bateaux. Elle sert à afficher les tirs du joueur et leur résultat.

5.3.8.1 ReceiveShot

Cette méthode publique va modifier l'état de la case dont les coordonnées sont égales à celles en paramètre en fonction du résultat de tir donné dans l'autre paramètre. Cette méthode va découper la chaine de résultat pour pouvoir en

extraire le bateau coulé ou la case touchée.
Cette méthode est appelé quand le joueur tir avec la méthode shoot du GameMananger.

5.3.9 Classe BotPlayer

Cette hérite de localPlayer et imite le fonctionnement d'un joueur distant. Elle est utilisé dans le mode solo pour permettre au joueur de jouer seul. Les bateaux de ce joueur sont placés aléatoirement, grace à la méthode héritée PlaceShipRandom lors de l'instanciation de ce dernier.

5.3.9.1 Shoot

Cette méthode publique retourne un point aléatoire dans la grille. La classe Botplayer possède un champ _positionAlreadyTested qui lui permet de savoir ou il a déjà tiré. En vérifiant cette liste de points il peut tirer aléatoirement sans jamais viser la même case.
Cette méthode est appelée dans l'update de la classe GameManager pendant le tour du joueur distant, uniquement si le mode solo est enclenché.

5.3.10 FrmConnection

C'est la fiche windows forms de connexion. Elle permet à l'utilisateur de choisir le mode de jeu : Réseau ou Local contre une IA. Le choix des modes est effectué a l'aide de 2 boutons.

5.3.10.1 SendRequest

Cette méthode envoie un message du type connectionRequest au serveur à l'adresse IP entré dans le textbox. Si l'invitation est acceptée elle appelle la méthode LaunchGame. Plusieurs boîtes de messages sont prévues pour les cas ou la connexion est impossible.
Elle est appelée quand l'utilisateur appuie sur le bouton "Ask for game".

5.3.10.2 LaunchGame

Cette méthode ferme la formulaire Windows et change la valeur du champ _needLauchApplication à true pour avertir le script Program.cs de lancer l'application monogame. Il est a noter que cette méthode utilise la méthode Control.Invoke pour pouvoir fermer le formulaire même si cette méthode est appelée depuis un autre thread. (Voir **ce thread**).

5.3.10.3 SettingMenuForm_ReceivedInvite

Cette méthode est "subscribed" à l'événement du même nom du serveur. Une fois l'événement relevé, si le résultat de l'invitation est positif, le jeu est lancé.

5.3.11 Classe Constants

Cette classe ne contient pas de méthode mais uniquement des champs publique et statique ou constante. Tous les strings et nombre de l'application sont enregistrés dans cette classe pour éviter **les magic values**

5.4 Diagrammes de classe

Étant donné de la taille du diagramme j'ai décidé de le diviser en plusieurs parties pour faciliter sa lecture.

5.4.1 Diagramme de classe global

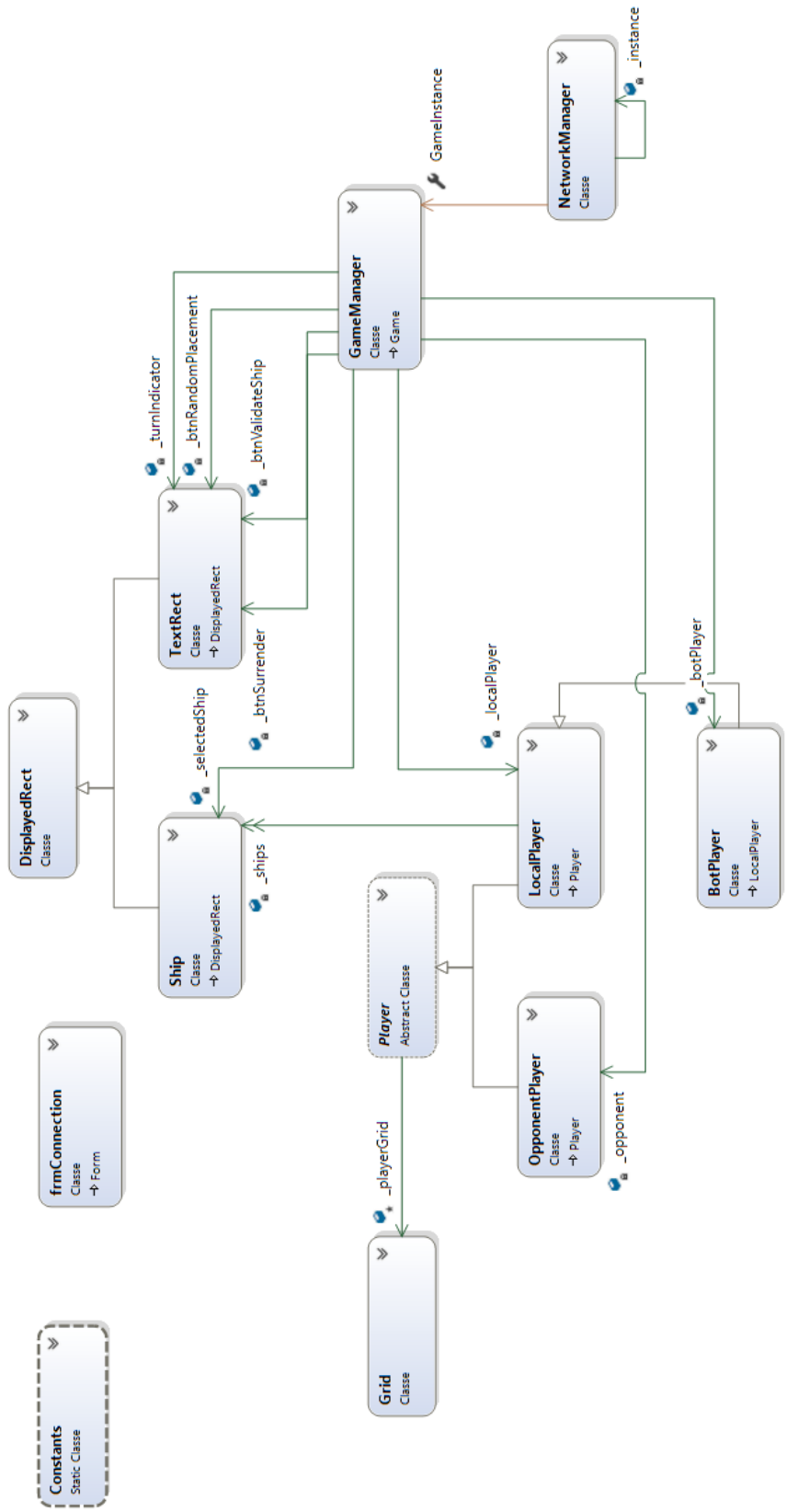


FIGURE 19 – Diagramme de classe global de l'application

5.4.2 Diagramme de classe de Player et sous classes

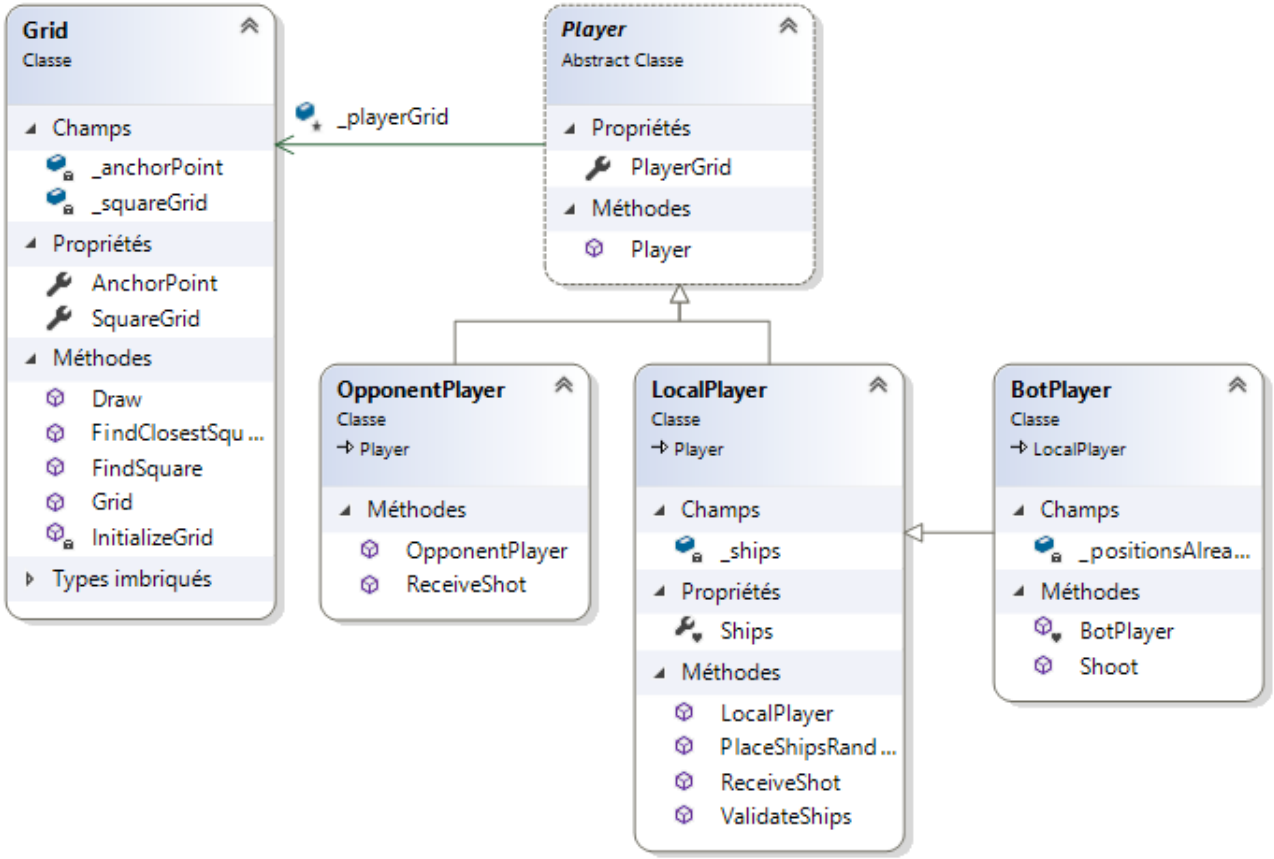


FIGURE 20 – Diagramme de classe de Player et sous classes

5.4.3 Diagramme de classe de DisplayedRect et sous classes

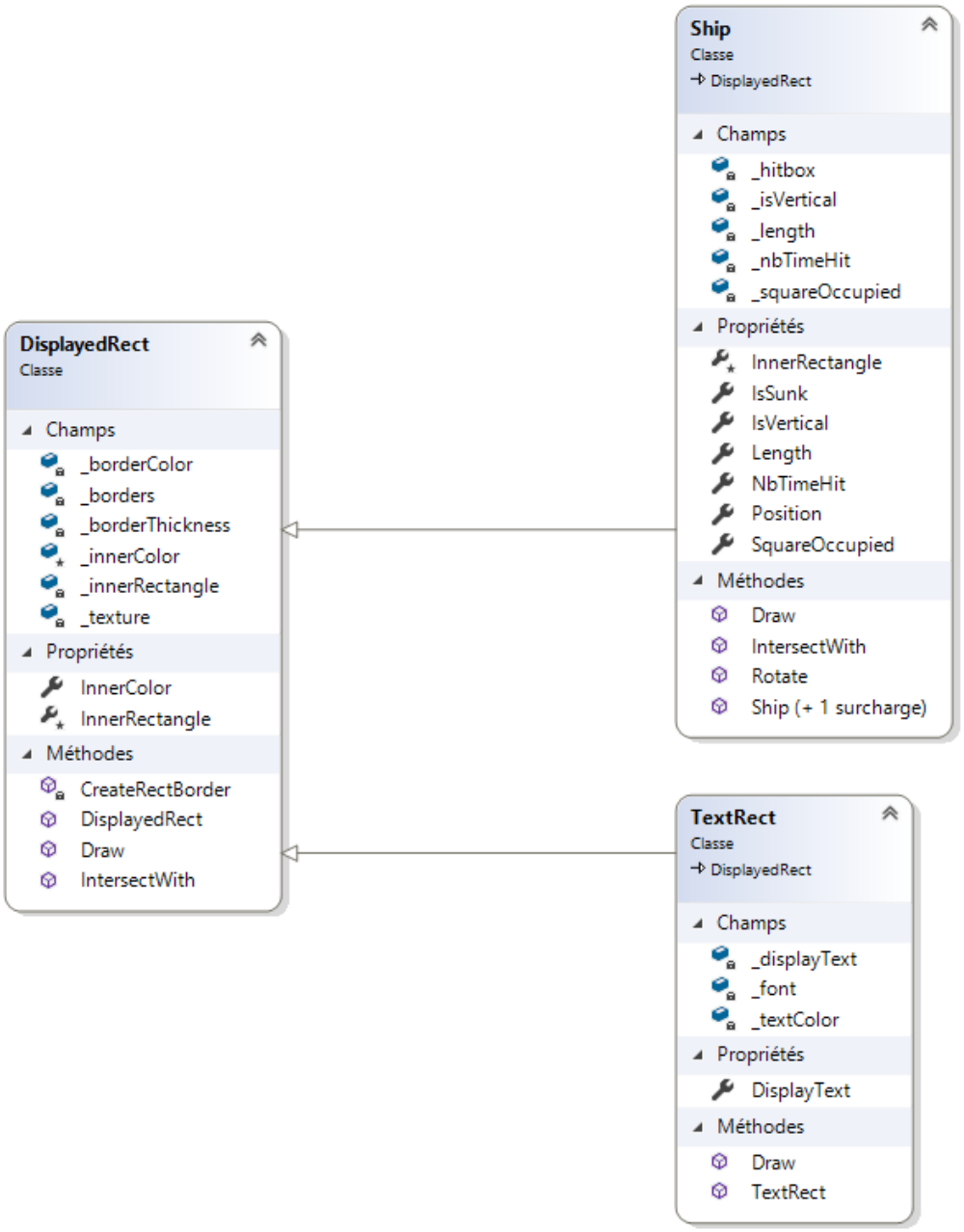


FIGURE 21 – Diagramme de classe de DisplayedRect et sous classes

5.4.4 Diagramme de classe de GameManager et NetworkManager

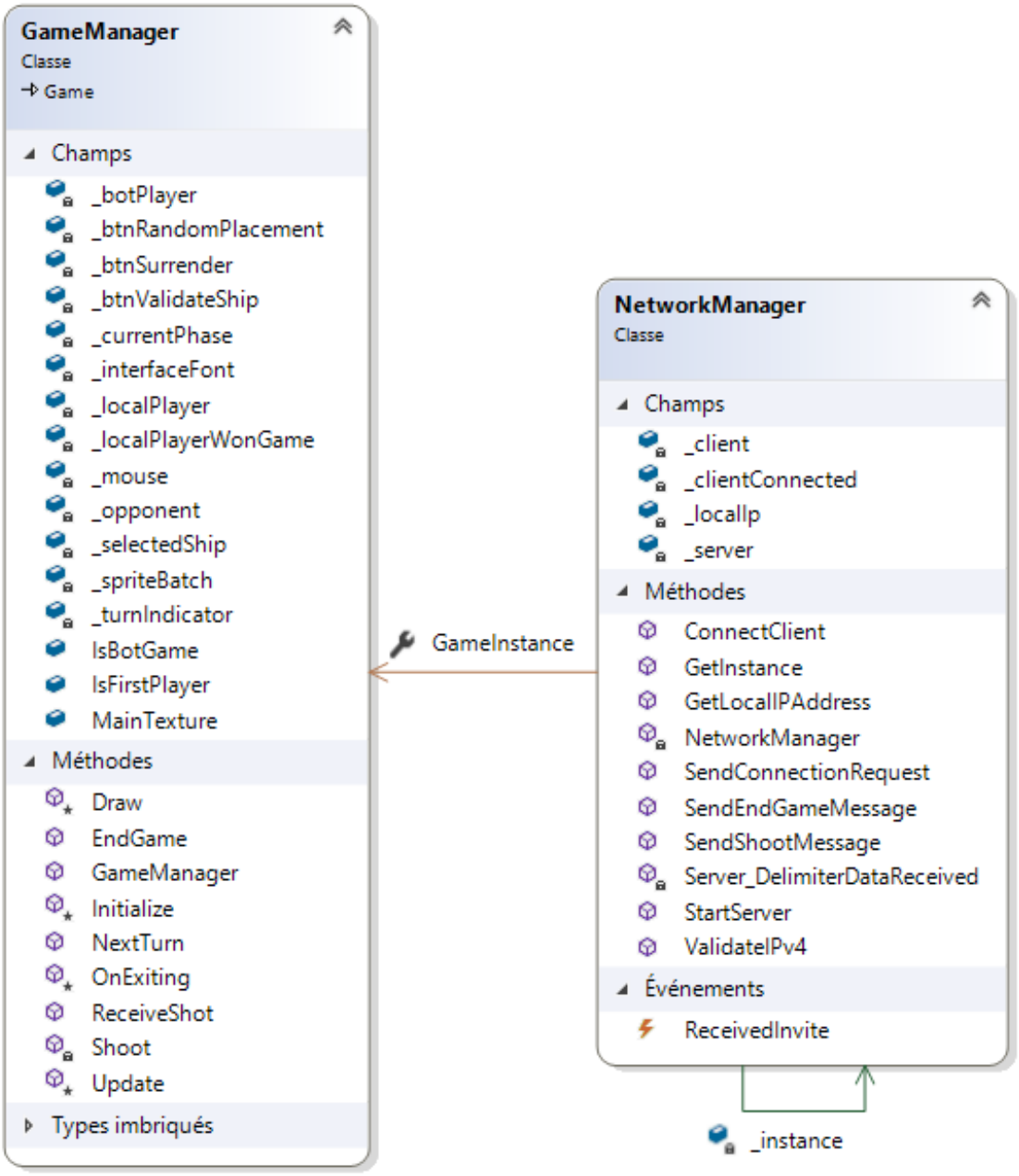


FIGURE 22 – Diagramme de classe de GameManager et NetworkManager

5.5 Outils externes

5.5.1 SimpleTCP

Comme expliqué plus haut, j'ai utilisé la librairie NuGet SimpleTCP pour la connexion réseau, elle permet de gérer la communication entre 2 applications de manière simple avec des classes serveurs et clients faciles à prendre en main. Vous pouvez trouver un guide d'utilisation en vidéo à l'adresse suivante : `https://www.youtube.com/watch?v=ve2LX1tOwIM`

5.5.2 GitHub

Github est un service de version pour le développement de logicels, il m'a permis de sauvegarder mon projet quotidiennement tout en gardant une trace des nouvelles fonctionnalités apportées.

5.5.3 Overleaf

Overleaf est un outil en ligne d'édition et de compilation de LaTeX. Il permet d'écrire de la documentation en automatisant la mise en page à l'aide de commandes.

6 Tests unitaires

La solution contient également un projet de tests unitaires. Comme l'énoncé l'indique, les tests n'ont été effectué que sur les classes de bases. Ils n'ont pas pu être implémentés sur la classe NetworkManager car les tests ne fonctionnent pas pour les fonctionnalités réseaux. Je n'ai aussi pas testé les fonctions de dessin parce qu'il n'est pas possible de vérifier le contenu d'une image.

7 Plan de test

En plus des tests unitaires 1 plan de tests des fonctionnalités de l'application a été mis en place.

Scénario de tests 1 - Connexion		
N°	**Scénario**	**Résultat attendu**
1	Lancer l'application	Affichage d'un formulaire qui permet de selectionner le mode de jeu et de choisir un partenaire de jeu
2	Sélection du mode de jeu solo	La partie se lance en mode solo
3	Appuyer sur le bouton de jeu en ligne sans préciser une adresse IP	Une pop up averti l'utilisateur qu'il doit entrer une adresse IP
4	Appuyer sur le bouton de jeu en ligne en entrant une adresse IP incorrecte	Une pop up averti l'utilisateur qu'il doit entrer une adresse IP correcte
5	Appuyer sur le bouton de jeu en ligne avec une adresse IP correcte	Le jeu envoie une invitation à l'application ouverte sur la machine possèdant l'IP
6	Appuyer sur le bouton de jeu en ligne avec une adresse IP possible mais elle ne possède pas l'application ouverte	Après 20 secondes, le joueur reçois un message comme quoi l'adresse IP n'a pas pu être contactée
7	Appuyer sur le bouton de jeu en ligne avec ça propre adresse Ip d'entré	Une pop up averti l'utilisateur qu'il doit entrer une adresse IP correcte
8	Refuser l'invitation	Le partenaire est informé du refu de son invitation
9	Accepter l'invitation	Les 2 joueurs sont informés du début imminent de la partie et le jeu se lance

FIGURE 23 – Plan de tests 1

Scénario de tests 2 - Jeu			
N°	**Scénario**	**Résultat attendu**	**Mode de jeu**
10	Déplacer les bateaux avec la souris	Ils sont bien déplacable en leur cliquant dessus et quand ils sont déposés sur la grille, il s'aligne sur elle	Les deux
11	Tourner les bateaux avec le clic droit de la souris	Le bateaux change d'orientation correctement	Les deux
12	Valider le placement des bateaux alors qu'ils ne sont pas tous placés, qu'ils sortent de la grille ou qu'ils se chevauchent	L'utilisateur est averti du mauvais placement de ces bateaux	Les deux
13	Valider le placement des bateaux quand ils sont bien placés	Le jeu passe au tour du joueur ou de l'adversaire	Les deux
14	Le joueur clique sur le bouton placer aléatoirement	Les bateaux se place aléatoirement sur la grille dans une configuration possible	Les deux
15	Le joueur clique sur une case du tableau avant que ce dernier a fini de placer ces bateaux	Il est averti que sont adversaire n'as pas fini de placer ces bateaux	Mode multijoueur
16	Le joueur clique sur une case du tableau adverse, elle est vide	Elle s'illumine en vert pour indiquer qu'elle n'a rien touché	Les deux
17	Le joueur clique sur une case du tableau adverse, elle touche un bateau	Elle s'illumine en rouge pour indiquer qu'un bateau à été touché	Les deux
18	Le joueur clique sur la dernière case non touché d'un bateau	L'intégralité des cases du bateaux deviennent grises pour indiquer qu'il à coulé	Les deux
19	Le joueur clique sur un case qui à déjà été sélectionnée	Rien ne se passe	Les deux
20	Le joueur joue son tour, c'est le tour de l'adversaire	L'adversaire tir sur une case aléatoire, on peut voir le résultat de son tir sur le tableau de gauche	Mode solo
21	Le joueur clique sur le bouton de reddition	Il est averti par un message de confirmation, si il l'accepte la partie ce termine	Les deux
22	L'adversaire se rend	Un écran de fin de partie l'averti que l'adversaire s'est rendu et la partie se termine	Mode multijoueur
23	Le joueur coule le dernier bateau adverse	Un écran de fin de partie l'averti qu'il a gagné et la partie se termine	Les deux
24	Le dernier bateaux du joueur coule	Un écran de fin de partie l'averti qu'il a perdu et la partie se termine	Les deux
25	Le joueur quitte la partie	Un écran de fin de partie l'averti que l'adversaire s'est rendu et la partie se termine	Mode multijoueur

FIGURE 24 – Plan de tests 2

8 Rapport de tests

Rapport de test				
N°	Date	Scénario	Résultat obtenu	Réussi ?
1	08.06.2020	Lancer l'application	Affichage d'un formulaire qui permet de selectionner le mode de jeu et de choisir un partenaire de jeu	OUI
2	08.06.2020	Sélection du mode de jeu solo	La partie se lance en mode solo	OUI
3	08.06.2020	Appuyer sur le bouton de jeu en ligne sans préciser une adresse IP	Une pop up averti l'utilisateur qu'il doit entrer une adresse IP	OUI
4	08.06.2020	Appuyer sur le bouton de jeu en ligne en entrant une adresse IP incorrecte	Une pop up averti l'utilisateur qu'il doit entrer une adresse IP correcte, mais pour certaine entrée comme 4 chiffre avec 1 point (12.15 ou 111.1) L'application tente quand même de se connecter. Pas de crash mais une pérdiode d'attente de 20 secondes	NON
5	08.06.2020	Appuyer sur le bouton de jeu en ligne avec une adresse IP correcte	Le jeu envoie une invitation à l'application ouverte sur la machine possèdant l'IP	OUI
6	08.06.2020	Appuyer sur le bouton de jeu en ligne avec une adresse IP possible mais elle ne possède pas l'application ouverte	Après 20 secondes, le joueur reçois un message comme quoi l'adresse IP n'a pas pu être contactée	OUI
7	08.06.2020	Appuyer sur le bouton de jeu en ligne avec ça propre adresse Ip d'entré	Une pop up averti l'utilisateur que l'utilisateur n'as pas pu être contacté	OUI
8	08.06.2020	Refuser l'invitation	Le partenaire est informé du refu de son invitation	OUI
9	08.06.2020	Accepter l'invitation	Les 2 joueurs sont informés du début imminent de la partie et le jeu se lance	OUI
10	08.06.2020	Déplacer les bateaux avec la souris	Ils sont bien déplacable en leur cliquant dessus et quand ils sont déposés sur la grille, il s'aligne sur elle	OUI
11	08.06.2020	Tourner les bateaux avec le clic droit de la souris	Le bateaux change d'orientation correctement	OUI
12	08.06.2020	Valider le placement des bateaux alors qu'ils ne sont pas tous placés, qu'ils sortent de la grille ou qu'ils se chevauchent	L'utilisateur est averti du mauvais placement de ces bateaux	OUI
13	08.06.2020	Valider le placement des bateaux quand ils sont bien placés	Le jeu passe au tour du joueur ou de l'adversaire	OUI
14	08.06.2020	Le joueur clique sur le bouton placer aléatoirement	Les bateaux se place aléatoirement sur la grille dans une configuration possible	OUI
15	08.06.2020	Le joueur clique sur une case du tableau avant que ce dernier a fini de placer ces bateaux	Rien ne se passe	NON
16	08.06.2020	Le joueur clique sur une case du tableau adverse, elle est vide	Elle s'illumine en vert pour indiquer qu'elle n'a rien touché	OUI
17	08.06.2020	Le joueur clique sur une case du tableau adverse, elle touche un bateau	Elle s'illumine en rouge pour indiquer qu'un bateau à été touché	OUI
18	08.06.2020	Le joueur clique sur la dernière case non touché d'un bateau	L'intégralité des cases du bateaux deviennent grises pour indiquer qu'il à coulé	OUI
19	08.06.2020	Le joueur clique sur un case qui à déjà été sélectionnée	Rien ne se passe	OUI
20	08.06.2020	Le joueur joue son tour, c'est le tour de l'adversaire	L'adversaire tir sur une case aléatoire, on peut voir le résultat de son tir sur le tableau de gauche	OUI
21	08.06.2020	Le joueur clique sur le bouton de reddition	Il est averti par un message de confirmation, si il l'accepte la partie ce termine	OUI
22	08.06.2020	L'adversaire se rend	Un écran de fin de partie l'averti qu'il a gagné et la partie se termine	OUI
23	08.06.2020	Le joueur coule le dernier bateau adverse	Un écran de fin de partie l'averti qu'il a gagné et la partie se termine	OUI
24	08.06.2020	Le dernier bateaux du joueur coule	Un écran de fin de partie l'averti qu'il a perdu et la partie se termine	OUI
25	08.06.2020	Le joueur quitte la partie	Un écran de fin de partie l'averti qu'il a gagné et la partie se termine	OUI

FIGURE 25 – Rapport de tests du 8/6/2020

9 Conclusion

9.1 Améliorations possibles

Il y a plusieurs points qu'on pourrait grandement améliorer le premier serait les graphismes : pour le moment l'application utilise des carrés pour représenter les bateaux. Il serait beaucoup plus agréable pour les yeux d'avoir des bateaux représentés par différents sprites et des petites icônes ou même animations pourraient être affichées à chaque tir.
Ensuite la fin de partie pourrait être amélioré pour l'instant l'application se ferme après l'affichage d'un message mais il serait possible d'afficher une jolie fenêtre de vicoire avant de revenir à l'écran de connexion ou demander aux joueurs s'ils veulent rejouer.
Aussi, il y a des améliorations au code qui peuvent être effectué afin de le rendre plus lisible et/ou plus performant. Au niveau du découpage des classes j'ai dû prendre des décisions avec laquelle je ne suis plus forcément d'accord, comme le fait que la méthode Shoot soit dans le GameManager et pas dans le LocalPlayer.
Également, je pense qu'il pourait être intéressant d'enregistrer les constantes du fichier Constants.cs au format XML et de désérialiser ces valeurs pour pouvoir modifier les paramètres sans devoir rebuild l'exécutable. Finalement, il est toujours possible d'ajouter les améliorations du jeu physique. Comme le tir en salve pour rendre le jeu plus technique.

9.2 Comparaison Plannings

En regardant mon TPI sur une vue d'ensemble j'ai clairement été dans les temps. Évidemment, j'ai fini certaines tâches rapidement et d'autres m'ont pris beaucoup plus de temps.
Dans l'ensemble, le développement de l'application m'as pris beaucoup moins de temps que prévu. Au contraire, la documentation, particulièrement la documentation technique m'a pris beaucoup plus de temps que prévu, ce qui à contrebalancer l'avance que j'avais prise sur le développement.

9.3 Bilan personnel

Ce TPI m'a permis de finalement vivre une expérience d'un développement complet, documentation comprise. seul et avec une deadline. Contrairement, aux exercices de TPI en groupe que nous faisons en atelier, l'expérience en solitaire est beaucoup plus stressante. J'ai aussi apprécié pouvoir développer un jeu, car c'est un de mes hobbies. J'ai déjà participé a des game jams, qui sont des concours de développement de jeux dans un temps très limité (environ

2-3 jours en général). Mais là encore, en équipe, vous pouvez d'ailleurs trouver notre site à **cette adresse** (J'ai participé au développement de Atomic Ball et Marius The plumber).
Je suis donc très heureux du résultat de ce Travail et des connaissances qu'il m'a apportées, particulièrement dans le domaine du développement de jeux en réseau.

10 Glossaire

IA	Intelligence artificielle
IP	Internet protocol
LAN	Local area network (Réseau local)
Monogame	Framework C# de création de jeu
NuGet	Gestionnaire de paquet pour visual studio
TCP	Transmission control protocol
TPI	Travail pratique individuel
UDP	User datagram protocol

11 Bibliographie

11.1 Code repris

— Fonction GetLocalIpAddress() de **ce thread Stackoverflow**
— Fonction ValidateIPv4() de **ce thread Stackoverflow**

11.1.1 Sites utilisés

— **Documentation .net microsoft**
— **Stackoverflow.com**, pour les questions techniques
— **Overleaf.com**, pour les questions sur LaTeX
— **Speedguide.net**, pour la listes des ports utilisés connus

11.1.2 Réference d'image

Logo de l'application

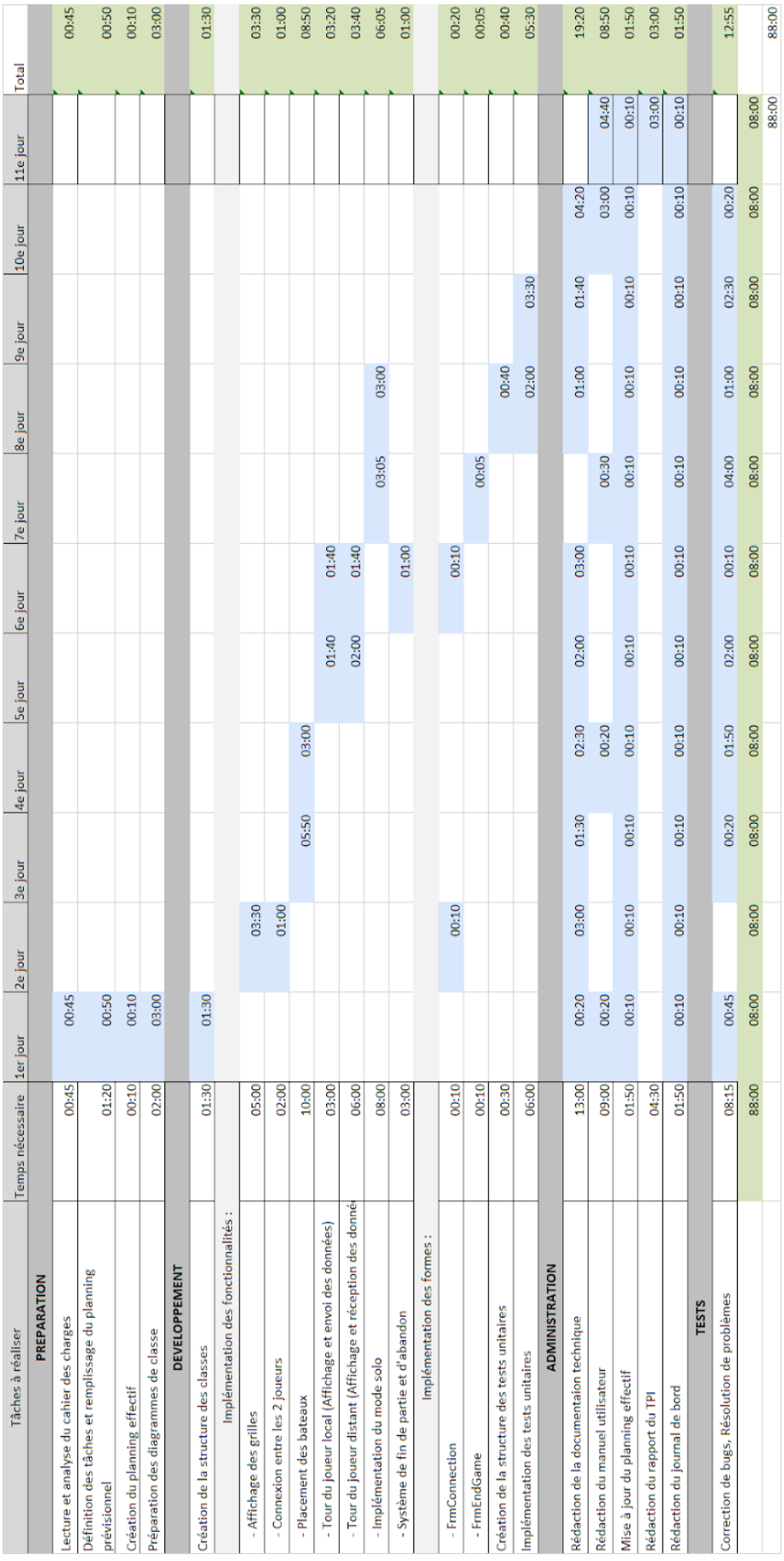
12 Table des figures

Table des figures

13 Annexes

Tâches à réaliser	Temps nécessaire	1er jour	2e jour	3e jour	4e jour	5e jour	6e jour	7e jour	8e jour	9e jour	10e jour	11e jour	Total
PREPARATION													
Lecture et analyse du cahier des charges	00:45	00:45											00:45
Définition des tâches et remplissage du planning prévisionnel	01:20	01:20											01:20
Création du planning effectif	00:10	00:10											00:10
Préparation des diagrammes de classe	02:00	02:00											02:00
DEVELOPPEMENT													
Création de la structure des classes	01:30	01:30											01:30
Implémentation des fonctionnalités :													
- Affichage des grilles	05:00		05:00										05:00
- Connexion entre les 2 joueurs	02:00		00:30	01:30									02:00
- Placement des bateaux	10:00			04:30	05:30								10:00
- Tour du joueur local (Affichage et envoi des données)	03:00					03:00							03:00
- Tour du joueur distant (Affichage et réception des donné	06:00					03:00	03:00						06:00
- Implémentation du mode solo	08:00							05:30	02:30				08:00
- Système de fin de partie et d'abandon	03:00						03:00						03:00
Implémentation des formes :													
- FrmConnection	00:10		00:10										00:10
- FrmEndGame	00:10							00:10					00:10
Création de la structure des tests unitaires	00:30	00:30											00:30
Implémentation des tests unitaires	06:00									04:00	02:00		06:00
ADMINISTRATION													
Rédaction de la documentaion technique	13:00	00:20	00:45	00:35	00:50	00:45	00:30	00:50	02:55	02:00	03:30		13:00
Rédaction du manuel utilisateur	09:00	00:20	00:30	00:20	00:35	00:10	00:15	00:35	01:30	00:55	01:25	02:25	09:00
Mise à jour du planning effectif	01:50	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	01:50
Rédaction du rapport du TPI	04:30											04:30	04:30
Rédaction du journal de bord	01:50	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	01:50
TESTS													
Correction de bugs, Résolution de problèmes	08:15	00:45	00:45	00:45	00:45	00:45	00:45	00:45	00:45	00:45	00:45	00:45	08:15
	88:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	88:00

FIGURE 26 – Planning prévisionnel



Tâches à réaliser	Temps nécessaire	1er jour	2e jour	3e jour	4e jour	5e jour	6e jour	7e jour	8e jour	9e jour	10e jour	11e jour	Total
PREPARATION													
Lecture et analyse du cahier des charges	00:45	00:45											00:45
Définition des tâches et remplissage du planning prévisionnel	01:20	00:50											00:50
Création du planning effectif	00:10	00:10											00:10
Préparation des diagrammes de classe	02:00	03:00											03:00
DEVELOPPEMENT													
Création de la structure des classes	01:30	01:30											01:30
Implémentation des fonctionnalités :													
- Affichage des grilles	05:00		03:30										03:30
- Connexion entre les 2 joueurs	02:00		01:00										01:00
- Placement des bateaux	10:00			05:50	03:00								08:50
- Tour du joueur local (Affichage et envoi des données)	03:00					01:40	01:40						03:20
- Tour du joueur distant (Affichage et réception des donné	06:00					02:00	01:40						03:40
- Implémentation du mode solo	08:00							03:05	03:00				06:05
- Système de fin de partie et d'abandon	03:00						01:00						01:00
Implémentation des formes :													
- FrmConnection	00:10		00:10				00:10						00:20
- FrmEndGame	00:10							00:05					00:05
Création de la structure des tests unitaires	00:30								00:40				00:40
Implémentation des tests unitaires	06:00									02:00	03:30		05:30
ADMINISTRATION													
Rédaction de la documentaion technique	13:00	00:20	03:00	01:30	02:30	02:00	03:00			01:00	01:40	04:20	19:20
Rédaction du manuel utilisateur	09:00	00:20			00:20				00:30			03:00	04:40	08:50
Mise à jour du planning effectif	01:50	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	01:50
Rédaction du rapport du TPI	04:30												03:00	03:00
Rédaction du journal de bord	01:50	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	01:50
TESTS													
Correction de bugs, Résolution de problèmes	08:15	00:45		00:20	01:50	02:00	00:10	04:00	01:00	02:30	00:20		12:55
	88:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	88:00	88:00

FIGURE 27 – Planning éffectif

13.1 Code source

13.1.1 GameManager.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 25.05.2020
9  *
10 * File          : GameManager.cs
11 */
12 using Microsoft.Xna.Framework;
13 using Microsoft.Xna.Framework.Graphics;
14 using Microsoft.Xna.Framework.Input;
15 using System;
16 using System.Collections.Generic;
17 using static TPI_2020_MonoBattle.Constants;
18 // We rename it because Forms is ambiguous XNA.Framework
19 using WF = System.Windows.Forms;
20
21 namespace TPI_2020_MonoBattle
22 {
23     /// <summary>
24     /// The main class of the project, it gather the inputs of the player ↩
           to transfer it to the classes.
25     /// It's also the only class that can communicate with the ↩
           NetworkManager.
26     /// </summary>
27     public class GameManager : Game
28     {
29         /// <summary>
30         /// The different phases of the game
31         /// </summary>
32         public enum Phase
33         {
34             shipsPlacement,
35             turnLocalPlayer,
36             turnOpponent,
37             endGame
38         }
39
40         #region Fields
41         /// <summary>
42         /// True if the local player plays first
43         /// </summary>
44         public static bool IsFirstPlayer = false;
45         /// <summary>
46         /// True if the game is local with a bot
47         /// </summary>
48         public static bool IsBotGame = false;
49         /// <summary>
50         /// Texture used for all the UI elements
51         /// </summary>
52         public static Texture2D MainTexture;
53         /// <summary>
54         /// Represent the phase the game is currently in.
55         /// We use this variable in the update and draw loop to determine ↩
               actions to take.
56         /// </summary>
57         private Phase _currentPhase;
```

```
58          /// <summary>
59          /// Is used to draw sprites and text onto the game screen
60          /// </summary>
61          private SpriteBatch _spriteBatch;
62          /// <summary>
63          /// The font used for the UI.
64          /// </summary>
65          private SpriteFont _interfaceFont;
66          /// <summary>
67          /// A box on top of the screen to indicate which player turn it is.
68          /// </summary>
69          private TextRect _turnIndicator;
70          /// <summary>
71          /// Button on top of the screen to valdiate the placement of the ↩
               ships
72          /// If they are not all placed, or missplaced the user is told to ↩
               retry.
73          /// Otherwise it ends the ship placement phase.
74          /// </summary>
75          private TextRect _btnValidateShip;
76          /// <summary>
77          /// Button on the top left of the screen.
78          /// It's used to end the game at any moment.
79          /// </summary>
80          private TextRect _btnSurrender;
81          /// <summary>
82          /// Button on the top right of the screen during the ship ↩
               placement phase.
83          /// It places the ship on the grid randomly
84          /// </summary>
85          private TextRect _btnRandomPlacement;
86          private TextRect _tooltipHit;
87          private TextRect _tooltipMiss;
88          private TextRect _tooltipSink;
89          /// <summary>
90          /// Represents the player that is on the computer running this ↩
               application
91          /// </summary>
92          private LocalPlayer _localPlayer;
93          /// <summary>
94          /// Represent the other player, be it an AI or a real human.
95          /// </summary>
96          private OpponentPlayer _opponent;
97          /// <summary>
98          /// The bot used to replace the opponent decision making in a bot ↩
               game
99          /// Is null when IsBotGame is false
100         /// </summary>
101         private BotPlayer _botPlayer;
102         /// <summary>
103         /// This variable is used in the ship placement phase.
104         /// It makes it so only one ship can be moved at a time to stop ↩
               them from pilling up.
105         /// </summary>
106         private Ship _selectedShip = null;
107         /// <summary>
108         /// The state of the mouse, we keep this global to remember the ↩
               last state beyond one update.
109         /// </summary>
110         private MouseState _mouse;
111         /// <summary>
112         /// True if the local player won the game
113         /// </summary>
114         private bool _localPlayerWonGame;
115         #endregion
```

```
116
117         #region Constructors
118         /// <summary>
119         /// Create the GameManager and initialize some necessary elements ↩
                of the game
120         /// </summary>
121         public GameManager()
122         {
123             // Name of the folder with all the images and fonts
124             Content.RootDirectory = "Content";
125
126             // Apply some changes to the monogame window (name, visible ↩
                mouse, resolution...)
127             Window.Title = DISPLAY_NAME;
128             IsMouseVisible = true;
129             GraphicsDeviceManager _graphics = new ↩
                GraphicsDeviceManager(this);
130             _graphics.PreferredBackBufferHeight = SCREEN_SIZE_HEIGHT;
131             _graphics.PreferredBackBufferWidth = SCREEN_SIZE_WIDTH;
132             _graphics.ApplyChanges();
133
134             NetworkManager.GetInstance().GameInstance = this;
135         }
136         #endregion
137
138         #region Methods
139         /// <summary>
140         /// Initialization of monogame variables
141         /// </summary>
142         protected override void Initialize()
143         {
144             base.Initialize();
145
146             // Initialize the variables for the start of the game.
147             _currentPhase = Phase.shipsPlacement;
148             _spriteBatch = new SpriteBatch(GraphicsDevice);
149             MainTexture = new Texture2D(_spriteBatch.GraphicsDevice, 1, 1);
150             MainTexture.SetData(new[] { Color.White });
151             _localPlayerWonGame = false;
152             // We create all the boats for the placement phase and asign ↩
                them to the player
153             List<Ship> _boats = new List<Ship>() {
154                 new Ship(new Point(600, 400), 2),
155                 new Ship(new Point(700, 400), 3),
156                 new Ship(new Point(800, 400), 3),
157                 new Ship(new Point(900, 400), 4),
158                 new Ship(new Point(1000, 400), 5),
159             };
160
161             _localPlayer = new LocalPlayer(new Grid(new ↩
                Point(PLAYER_GRID_X, PLAYER_GRID_Y)), _boats);
162             _opponent = new OpponentPlayer(new Grid(new ↩
                Point(OPPONENT_GRID_X, OPPONENT_GRID_Y)));
163             if (IsBotGame)
164             {
165                 List<Ship> _botBoats = new List<Ship>() {
166                 new Ship(new Point(600, 400), 2),
167                 new Ship(new Point(700, 400), 3),
168                 new Ship(new Point(800, 400), 3),
169                 new Ship(new Point(900, 400), 4),
170                 new Ship(new Point(1000, 400), 5),
171                 };
172                 _botPlayer = new BotPlayer(new Grid(new ↩
                    Point(OPPONENT_GRID_X, OPPONENT_GRID_Y)), _botBoats);
173             }
```

```
174
175             // Initialisation of all the UI elements
176             _interfaceFont = Content.Load<SpriteFont>("UiFont");
177             _turnIndicator = new TextRect(new Rectangle(TURN_INDICATOR_X, ↩
                    TURN_INDICATOR_Y, TURN_INDICATOR_WIDTH, ↩
                    TURN_INDICATOR_HEIGHT),
178                 TURN_INDICATOR_BORDER_WIDTH, ↩
                        TURN_INDICATOR_LOCAL_TURN_BACKCOLOR, ↩
                        TURN_INDICATOR_BORDER_COLOR,
179                 TURN_INDICATOR_LOCAL_TURN_TEXT, _interfaceFont, ↩
                        TURN_INDICATOR_TEXT_COLOR
180                 );
181             _btnValidateShip = new TextRect(new Rectangle(BTN_VALIDATE_X, ↩
                    BTN_VALIDATE_Y, BTN_VALIDATE_WIDTH, BTN_VALIDATE_HEIGHT), ↩
                    BTN_VALIDATE_BORDER_WIDTH,
182                 BTN_VALIDATE_BACKCOLOR, BTN_VALIDATE_BORDER_COLOR, ↩
                        BTN_VALIDATE_TEXT, _interfaceFont, BTN_VALIDATE_TEXT_COLOR
183                 );
184             _btnSurrender = new TextRect(new Rectangle(BTN_SURRENDER_X, ↩
                    BTN_SURRENDER_Y, BTN_SURRENDER_WIDTH, ↩
                    BTN_SURRENDER_HEIGHT), BTN_SURRENDER_BORDER_WIDTH,
185                 BTN_SURRENDER_BACKCOLOR, BTN_SURRENDER_BORDER_COLOR, ↩
                        BTN_SURRENDER_TEXT, _interfaceFont, ↩
                        BTN_SURRENDER_TEXT_COLOR
186                 );
187             _btnRandomPlacement = new TextRect(new Rectangle(BTN_RANDOM_X, ↩
                    BTN_RANDOM_Y, BTN_RANDOM_WIDTH, BTN_RANDOM_HEIGHT), ↩
                    BTN_RANDOM_BORDER_WIDTH,
188                 BTN_RANDOM_BACKCOLOR, BTN_RANDOM_BORDER_COLOR, ↩
                        BTN_RANDOM_TEXT, _interfaceFont, BTN_RANDOM_TEXT_COLOR
189                 );
190             _tooltipHit = new TextRect(new Rectangle(TOOLTIP_HIT_X, ↩
                    TOOLTIP_Y, TOOLTIP_WIDTH, TOOLTIP_HEIGHT), ↩
                    TOOLTIP_BORDER_WIDTH,
191                 TOOLTIP_HIT_BACKCOLOR, TOOLTIP_BORDER_COLOR, ↩
                        TOOLTIP_HIT_TEXT, _interfaceFont, TOOLTIP_TEXT_COLOR);
192             _tooltipMiss = new TextRect(new Rectangle(TOOLTIP_MISS_X, ↩
                    TOOLTIP_Y, TOOLTIP_WIDTH, TOOLTIP_HEIGHT), ↩
                    TOOLTIP_BORDER_WIDTH,
193                 TOOLTIP_MISS_BACKCOLOR, TOOLTIP_BORDER_COLOR, ↩
                        TOOLTIP_MISS_TEXT, _interfaceFont, TOOLTIP_TEXT_COLOR);
194             _tooltipSink = new TextRect(new Rectangle(TOOLTIP_SINK_X, ↩
                    TOOLTIP_Y, TOOLTIP_WIDTH, TOOLTIP_HEIGHT), ↩
                    TOOLTIP_BORDER_WIDTH,
195                 TOOLTIP_SINK_BACKCOLOR, TOOLTIP_BORDER_COLOR, ↩
                        TOOLTIP_SINK_TEXT, _interfaceFont, TOOLTIP_TEXT_COLOR);
196             _mouse = Mouse.GetState();
197         }
198         /// <summary>
199         /// Main loop of the application. It's constantly called to check ↩
                for inputs.
200         /// </summary>
201         /// <param name="gameTime">Provides a snapshot of timing ↩
                values.</param>
202         protected override void Update(GameTime gameTime)
203         {
204             // Keeping track of the old state for flank detection
205             MouseState oldMouseState = _mouse;
206             _mouse = Mouse.GetState();
207             // Check if the surrender button is clicked
208             if (_btnSurrender.IntersectWith(_mouse.Position) && ↩
                    _mouse.LeftButton == ButtonState.Pressed && ↩
                    oldMouseState.LeftButton == ButtonState.Released)
209             {
210                 // Prompt the user and ask for confirmation
```

```
211                 if (WF.MessageBox.Show("Are you sure you want to surrender?", "Confirmation", WF.MessageBoxButtons.YesNo, WF.MessageBoxIcon.Warning) == WF.DialogResult.Yes)
212                 {
213                     try
214                     {
215                         // Send the message that we surrendered to the opponent
216                         NetworkManager.GetInstance().SendEndGameMessage();
217                         EndGame(false);
218                     }
219                     catch (NullReferenceException)
220                     {
221                         WF.MessageBox.Show("Connection to opponent lost, game is closing now", "Connection error", WF.MessageBoxButtons.OK, WF.MessageBoxIcon.Warning);
222                         Exit();
223                     }
224 
225                 }
226             }
227             // Different actions depending on the game phase
228             switch (_currentPhase)
229             {
230                 case Phase.shipsPlacement:
231                     // If this is the first frame of the mouse being pressed
232                     if (_mouse.LeftButton == ButtonState.Pressed && oldMouseState.LeftButton == ButtonState.Released)
233                     {
234                         // We check if the buttons are hovered by the mouse, if so they are clicked.
235                         if (_btnValidateShip.IntersectWith(_mouse.Position))
236                         {
237                             // We check the position of all the ships, if they are correct we start the game
238                             if (_localPlayer.ValidateShips(true))
239                             {
240                                 if (IsFirstPlayer)
241                                 {
242                                     _currentPhase = Phase.turnLocalPlayer;
243                                 }
244                                 else
245                                 {
246                                     _currentPhase = Phase.turnOpponent;
247                                 }
248                             }
249                             else
250                             {
251                                 WF.MessageBox.Show("Not all ship are correctly placed on the grid", "Ship placement error", WF.MessageBoxButtons.OK, WF.MessageBoxIcon.Error);
252                             }
253                         }
254                         if (_btnRandomPlacement.IntersectWith(_mouse.Position))
255                         {
256                             try
257                             {
258                                 _localPlayer.PlaceShipsRandom();
259                             }
260                             catch (StackOverflowException)
261                             {
```

```
262                                 WF.MessageBox.Show("Error", "No possible configuration");
263                             }
264
265                         }
266
267                         // The ship that is hovered by the mouse is the one we want to interact with
268                         foreach (Ship boat in _localPlayer.Ships)
269                         {
270                             if (boat.IntersectWith(_mouse.Position))
271                             {
272                                 _selectedShip = boat;
273                             }
274                         }
275                     }
276                     // We reset the selected ship only if the mouse click is not held.
277                     // It stops the game from selecting other ships while draging moving them around
278                     // And it let move the ships in a top left direction, where they are not hovered by the mouse.
279                     else if (_mouse.LeftButton == ButtonState.Released && _selectedShip != null)
280                     {
281                         // Once the ship is dropped we try to align it with the grid if possible
282                         _selectedShip.Position = _localPlayer.PlayerGrid.FindClosestSquareAnchor(_selectedSh
283                         _selectedShip = null;
284                     }
285
286                     // If a ship is held we can rotate it and it follows the mouse movements
287                     if (_selectedShip != null)
288                     {
289                         _selectedShip.Position = _mouse.Position;
290                         if (_mouse.RightButton == ButtonState.Pressed && oldMouseState.RightButton == ButtonState.Released)
291                         {
292                             _selectedShip.Rotate();
293                         }
294                     }
295                     break;
296                 case Phase.turnLocalPlayer:
297                     // If this is the first frame of the mouse being pressed
298                     if (_mouse.LeftButton == ButtonState.Pressed && oldMouseState.LeftButton == ButtonState.Released)
299                     {
300                         try
301                         {
302                             Shoot(_opponent.PlayerGrid.FindSquare(_mouse.Position));
303                         }
304                         catch (ArgumentOutOfRangeException)
305                         {
306                             // Means the user clicked outside of the grid, it's not an issue so we don't prompt the user
307                         }
308                     }
309                     break;
310                 case Phase.turnOpponent:
311                     // If the game is solo, the bot plays otherwhise we wait for the opponent to respond
312                     if (IsBotGame)
313                     {
```

```
314                         ReceiveShot(_botPlayer.Shoot());
315                     }
316                     break;
317                 case Phase.endGame:
318                     // We show the correct end of game message
319                     if (_localPlayerWonGame)
320                     {
321                         WF.MessageBox.Show("You␣destroyed␣the␣ennemy␣fleet␣↩
                               !", "Victory␣!", WF.MessageBoxButtons.OK, ↩
                               WF.MessageBoxIcon.Information);
322                         Exit();
323                     }
324                     else
325                     {
326                         WF.MessageBox.Show("All␣your␣ships␣have␣sunk␣!", ↩
                               "Defeat␣!", WF.MessageBoxButtons.OK, ↩
                               WF.MessageBoxIcon.Information);
327                         Exit();
328                     }
329                     break;
330                 default:
331                     break;
332             }
333         }
334 
335         /// <summary>
336         /// This is called when the game should draw itself.
337         /// </summary>
338         /// <param name="gameTime">Provides a snapshot of timing ↩
                values.</param>
339         protected override void Draw(GameTime gameTime)
340         {
341             // Clears the last frame
342             GraphicsDevice.Clear(BACKGROUND_COLOR);
343 
344             // Open the spriteBatch to draw all the sprites at the same time
345             _spriteBatch.Begin();
346             _btnSurrender.Draw(_spriteBatch);
347 
348             // Draw the scene depending on the phase
349             switch (_currentPhase)
350             {
351                 // We draw only the local player grid, his boats and the ↩
                    buttons
352                 case Phase.shipsPlacement:
353                     _localPlayer.PlayerGrid.Draw(_spriteBatch);
354                     foreach (Ship boat in _localPlayer.Ships)
355                     {
356                         boat.Draw(_spriteBatch);
357                     }
358                     _btnValidateShip.Draw(_spriteBatch);
359                     _btnRandomPlacement.Draw(_spriteBatch);
360                     break;
361                 case Phase.turnLocalPlayer:
362                     _localPlayer.PlayerGrid.Draw(_spriteBatch);
363                     _opponent.PlayerGrid.Draw(_spriteBatch);
364                     // We change the colors/ text of the turn indicator.
365                     if (_turnIndicator.DisplayText != ↩
                            TURN_INDICATOR_LOCAL_TURN_TEXT)
366                     {
367                         _turnIndicator.DisplayText = ↩
                                TURN_INDICATOR_LOCAL_TURN_TEXT;
368                         _turnIndicator.InnerColor = ↩
                                TURN_INDICATOR_LOCAL_TURN_BACKCOLOR;
369 
```

```
370                     }
371                     _turnIndicator.Draw(_spriteBatch);
372                     _tooltipHit.Draw(_spriteBatch);
373                     _tooltipMiss.Draw(_spriteBatch);
374                     _tooltipSink.Draw(_spriteBatch);
375                     break;
376                 case Phase.turnOpponent:
377                     _localPlayer.PlayerGrid.Draw(_spriteBatch);
378                     _opponent.PlayerGrid.Draw(_spriteBatch);
379                     if (_turnIndicator.DisplayText != ↩
                           TURN_INDICATOR_OPPONENT_TURN_TEXT)
380                     {
381                         _turnIndicator.DisplayText = ↩
                               TURN_INDICATOR_OPPONENT_TURN_TEXT;
382                         _turnIndicator.InnerColor = ↩
                               TURN_INDICATOR_OPPONENT_TURN_BACKCOLOR;
383 
384                     }
385                     _turnIndicator.Draw(_spriteBatch);
386                     _tooltipHit.Draw(_spriteBatch);
387                     _tooltipMiss.Draw(_spriteBatch);
388                     _tooltipSink.Draw(_spriteBatch);
389                     break;
390                 case Phase.endGame:
391                     _localPlayer.PlayerGrid.Draw(_spriteBatch);
392                     _opponent.PlayerGrid.Draw(_spriteBatch);
393                     _turnIndicator.Draw(_spriteBatch);
394                     _tooltipHit.Draw(_spriteBatch);
395                     _tooltipMiss.Draw(_spriteBatch);
396                     _tooltipSink.Draw(_spriteBatch);
397                     break;
398             }
399             // Close the sprite batch to draw all the sprites on the screen
400             _spriteBatch.End();
401 
402         }
403         #endregion
404 
405         /// <summary>
406         /// Sends the information to the opponent that we guess the point ↩
                in parameter.
407         /// If the game is in solo Mode we send the data to the bot,
408         /// otherwise we use the NetworkManager to send the data to the ↩
                other application.
409         /// Then, depending on the response, we update the opponent grid.
410         /// </summary>
411         /// <param name="location">The point we want to guess</param>
412         private void Shoot(Point location)
413         {
414             if (_currentPhase == Phase.turnLocalPlayer)
415             {
416                 // shotResult is what the cell contains : Ship, Nothing, ↩
                       Last part of a ship...
417                 string shotResult = "";
418                 if (IsBotGame)
419                 {
420                     shotResult = _botPlayer.ReceiveShot(location);
421                 }
422                 else
423                 {
424                     try
425                     {
426                         shotResult = ↩
                               NetworkManager.GetInstance().SendShootMessage(location);
427                     }
```

```
428                     catch (NullReferenceException)
429                     {
430                         WF.MessageBox.Show("Connection to opponent lost, game is closing now", "Connection error", WF.MessageBoxButtons.OK, WF.MessageBoxIcon.Warning);
431                         Exit();
432                     }
433                 }
434 
435                 if (shotResult.Contains(GAME_FINISHED_MESSAGE))
436                 {
437                     EndGame(true);
438                 }
439                 else if (!shotResult.Contains(ERROR_MESSAGE))
440                 {
441                     // We apply the changes to the local grid and pass on the next turn
442                     _opponent.ReceiveShot(location, shotResult);
443                     NextTurn();
444                 }
445             }
446         }
447         /// <summary>
448         /// When the local player receive a shot guess from the opponent.
449         /// We apply the shot to the display grid and return the result
450         /// </summary>
451         /// <param name="location">Coordinates of the guess</param>
452         /// <returns>Result of the shot, see Constant.cs</returns>
453         public string ReceiveShot(Point location)
454         {
455             // If it's not the opponent turns, he shouldn't send shots
456             if (_currentPhase == Phase.turnOpponent)
457             {
458                 string result = _localPlayer.ReceiveShot(location);
459                 if (result.Contains(GAME_FINISHED_MESSAGE))
460                 {
461                     EndGame(false);
462                     return GAME_FINISHED_MESSAGE;
463                 }
464                 else if (!result.Contains(ERROR_MESSAGE))
465                 {
466                     NextTurn();
467                 }
468                 return result;
469             }
470             return ERROR_MESSAGE;
471         }
472         /// <summary>
473         /// Change the phase to the next turn
474         /// </summary>
475         public void NextTurn()
476         {
477             if (_currentPhase == Phase.turnLocalPlayer)
478             {
479                 _currentPhase = Phase.turnOpponent;
480             }
481             else
482             {
483                 _currentPhase = Phase.turnLocalPlayer;
484             }
485         }
486         /// <summary>
487         /// Enter the end game phase
488         /// </summary>
```

```
489         /// <param name="didLocalPlayerWin">true if the local player ↩
                won</param>
490         public void EndGame(bool didLocalPlayerWin)
491         {
492             _localPlayerWonGame = didLocalPlayerWin;
493             _currentPhase = Phase.endGame;
494         }
495         #region Events
496         /// <summary>
497         /// We override the OnExit method of Game.cs to tell the other ↩
                player we left the game
498         /// </summary>
499         /// <param name="sender"></param>
500         /// <param name="args"></param>
501         protected override void OnExiting(object sender, EventArgs args)
502         {
503             if (!IsBotGame)
504             {
505                 try
506                 {
507                     NetworkManager.GetInstance().SendEndGameMessage();
508                 }
509                 catch (NullReferenceException)
510                 {
511                     // Opponent already left, no need to send a message
512                 }
513             }
514             base.OnExiting(sender, args);
515         }
516         #endregion
517     }
518 }
```

Listing 1 – Sources/gameManager.cs

13.1.2 NetworkManager.cs

```
1   /*
2  * Author       : Nelson Jeanrenaud
3  *
4  * Teacher      : Stéphane Garchery
5  *
6  * Experts      : Pierre Conrad, Philippe Bernard
7  *
8  * Date         : 06.06.2020
9  *
10 * File         : NetworkManager.cs
11 */
12 using Microsoft.Xna.Framework;
13 using SimpleTCP;
14 using System;
15 using System.Diagnostics;
16 using System.Linq;
17 using System.Net;
18 using System.Net.Sockets;
19 using static TPI_2020_MonoBattle.Constants;
20 // We rename it because Forms.Message is ambiguous with SimpleTCP.Message
21 using WF = System.Windows.Forms;
22
23 namespace TPI_2020_MonoBattle
24 {
25     /// <summary>
26     /// Singleton class that uses simpleTCP NuGet to connect the players
27     /// </summary>
28     public class NetworkManager
29     {
30         #region Fields
31         private static NetworkManager _instance;
32
33         /// <summary>
34         /// Returns the ip of the user
35         /// </summary>
36         private IPAddress _localIp;
37         /// <summary>
38         /// Server that receive data from the other machine
39         /// </summary>
40         private SimpleTcpServer _server;
41         /// <summary>
42         /// Represents the other application, if it's null then there is ↩
               no app connected.
43         /// </summary>
44         private SimpleTcpClient _client;
45         /// <summary>
46         /// If a client connected to the application
47         /// </summary>
48         private bool _clientConnected;
49         #endregion
50         #region Propriétés
51         public GameManager GameInstance { get; set; }
52         #endregion
53         #region Events
54         /// <summary>
55         /// Event that is called whenever we reply to a game request
56         /// It informs the local form about if it should close to launch ↩
               the game
57         /// </summary>
58         public event EventHandler<bool> ReceivedInvite;
59         #endregion
60
61         /// <summary>
```

```
62          /// Create a new instance of the class
63          /// ! Only to be called from GetInstance !
64          /// </summary>
65          private NetworkManager()
66          {
67              _localIp = GetLocalIPAddress();
68              _server = new SimpleTcpServer();
69              _client = new SimpleTcpClient();
70              _clientConnected = false;
71          }
72
73          /// <summary>
74          /// Returns the single instance of the class
75          /// </summary>
76          public static NetworkManager GetInstance()
77          {
78              if (_instance == null)
79              {
80                  _instance = new NetworkManager();
81              }
82              return _instance;
83          }
84          /// <summary>
85          /// Start the local server and initialize all the variables, ↩
                Delimiter...
86          /// </summary>
87          public void StartServer()
88          {
89              if (!_server.IsStarted)
90              {
91                  // Start the server and indicates which port he needs to ↩
                        listen on
92                  _server.Start(_localIp, CONNECTION_PORT);
93                  // Everytime the server receive a \n it calls the ↩
                        DelimiterDataReceived event.
94                  _server.Delimiter = 0x13;
95                  _server.DelimiterDataReceived += ↩
                        Server_DelimiterDataReceived;
96              }
97          }
98          /// <summary>
99          /// Called everytime the server receive 1 full data package ↩
                (indicated by a \n),
100         /// it paints the cell who's location correspond with the data ↩
                received.
101         /// </summary>
102         private void Server_DelimiterDataReceived(object sender, Message e)
103         {
104             // The message received is formatted as such "TypeOfMessage" + ↩
                    Separator + "Data" + EndOfMessageSeparator
105             // We split the messaged based on those separators
106             // 0 is the type of the message and 1 is the data
107             string[] messageSplited = e.MessageString.Split(new string[] { ↩
                    TYPE_SEPARATOR, MESSAGE_END_SEPARATOR }, ↩
                    StringSplitOptions.RemoveEmptyEntries);
108
109             // We convert the string message received to a MessageType ↩
                    variable
110             Enum.TryParse(messageSplited[0], out MESSAGE_TYPE messageType);
111
112             switch (messageType)
113             {
114                 // If we received a request we prompt the user about it ↩
                        then send back the response to the other app.
115                 case MESSAGE_TYPE.connectionRequest:
```

```
116                     // In the case of a request the data is always the ↩
                           adress of the other computer
117                     string ipAdress = messageSplited[1];
118                     // We prompt the user about the message
119                     // In case we receive a request while already in game
120                     if (!_clientConnected)
121                     {
122                         if (WF.MessageBox.Show("You␣received␣a␣game␣↩
                               request␣from␣" + ipAdress + "␣would␣you␣like␣to␣↩
                               accept␣it␣?", "Game␣request␣!", ↩
                               WF.MessageBoxButtons.YesNo, ↩
                               WF.MessageBoxIcon.Question) == ↩
                               WF.DialogResult.Yes)
123                         {
124                             // Since we want to be able to communicate ↩
                                   with the other app in the future we create ↩
                                   a communication canal
125                             ConnectClient(ipAdress);
126                             // Trigger the form event to warn the view ↩
                                   about the result of the connection.
127                             ReceivedInvite.Invoke(this, true);
128                             // We send back the response 1 is for yes
129                             GameManager.IsFirstPlayer = false;
130                             e.Reply(ACCEPT_INVITE);
131                         }
132                         else
133                         {
134                             // Trigger the form event to warn the view ↩
                                   about the result of the connection.
135                             ReceivedInvite.Invoke(this, false);
136                             // We send back the response 0 is for no
137                             e.Reply(REFUSE_INVITE);
138                         }
139                     }
140                     e.Reply(REFUSE_INVITE);
141                     break;
142                 case MESSAGE_TYPE.shoot:
143                     if (GameInstance != null)
144                     {
145                         // Separates the data
146                         char[] separators = { ',' };
147                         string[] stringPosition = ↩
                               messageSplited[1].Split(separators);
148                         // Convert the positions from string to int so it ↩
                               can be used
149                         int positionX = Convert.ToInt32(stringPosition[0]);
150                         int positionY = Convert.ToInt32(stringPosition[1]);
151                         e.Reply(GameInstance.ReceiveShot(new ↩
                               Point(positionX, positionY)));
152                     }
153                     break;
154                 case MESSAGE_TYPE.endgame:
155                     GameInstance.EndGame(true);
156                     break;
157                 default:
158                     break;
159             }
160         }
161         /// <summary>
162         /// Connects the user to set peer
163         /// </summary>
164         public void ConnectClient(string ip)
165         {
166             // We check the syntax of the ip and if it's correct create a ↩
                   TCP connection
```

```
167             if (ValidateIPv4(ip.ToString()))
168             {
169                 try
170                 {
171                     _client.Connect(ip.ToString(), CONNECTION_PORT);
172                     _clientConnected = true;
173                 }
174                 catch (SocketException)
175                 {
176                     throw new SocketException(61);
177                 }
178             }
179         }
180         /// <summary>
181         /// Sends a request to the given Ip and return bool coresponding ↩
                to the state of the result
182         /// </summary>
183         /// <param name="message"></param>
184         public bool SendConnectionRequest(IPAddress ip)
185         {
186             try
187             {
188                 ConnectClient(ip.ToString());
189                 if (_client.TcpClient.Connected)
190                 {
191                     // Send a request type message to the ip with our ip ↩
                        so he can send us back a response.
192                     // End the message with \n to indicate the end of the ↩
                        message.
193                     Message reply = ↩
                        _client.WriteLineAndGetReply(MESSAGE_TYPE.connectionRequest ↩
                        +
194                         TYPE_SEPARATOR + _localIp.ToString() + ↩
                            MESSAGE_END_SEPARATOR, TimeSpan.FromSeconds(20));
195                     if (reply != null)
196                     {
197                         if (reply.MessageString == ACCEPT_INVITE)
198                         {
199                             return true;
200                         }
201                     }
202                 }
203                 return false;
204             }
205             catch (SocketException)
206             {
207                 throw new SocketException(61);
208             }
209 
210         }
211         /// <summary>
212         /// Sends a message to the peer to indicate the game's finished ↩
                (and he won)
213         /// </summary>
214         public void SendEndGameMessage()
215         {
216             if (_client.TcpClient.Connected)
217             {
218                 // End the message with \n to indicate the end of the ↩
                    message.
219                 try
220                 {
221                     _client.WriteLine(MESSAGE_TYPE.endgame.ToString() +
222                         TYPE_SEPARATOR + MESSAGE_END_SEPARATOR);
223                 }
```

```
224                 catch (System.IO.IOException)
225                 {
226                     throw new NullReferenceException("Client is not ↩
                           connected");
227                 }
228 
229             }
230             else
231             {
232                 throw new NullReferenceException("Client is not connected");
233             }
234         }
235         /// <summary>
236         /// Sends a shot message at the coordiantes given in parameter
237         /// </summary>
238         /// <param name="coordinates"></param>
239         /// <returns></returns>
240         public string SendShootMessage(Point coordinates)
241         {
242             if (_client.TcpClient.Connected)
243             {
244                 // End the message with \n to indicate the end of the ↩
                       message.
245                 try
246                 {
247                     Message reply = ↩
                           _client.WriteLineAndGetReply(MESSAGE_TYPE.shoot.ToString() ↩
                           +
248                           TYPE_SEPARATOR + coordinates.X.ToString() + "," + ↩
                               coordinates.Y.ToString() + ↩
                               MESSAGE_END_SEPARATOR, TimeSpan.FromSeconds(20));
249 
250                     return reply.MessageString;
251                 }
252                 catch (System.IO.IOException)
253                 {
254                     throw new NullReferenceException("Client is not ↩
                           connected");
255                 }
256 
257             }
258             else
259             {
260                 throw new NullReferenceException("Client is not connected");
261             }
262         }
263         /// <summary>
264         /// Returns the current IP adress of the machine
265         /// </summary>
266         public static IPAddress GetLocalIPAddress()
267         {
268             var host = Dns.GetHostEntry(Dns.GetHostName());
269             foreach (var ip in host.AddressList)
270             {
271                 if (ip.AddressFamily == AddressFamily.InterNetwork)
272                 {
273                     return ip;
274                 }
275             }
276             throw new Exception("No network adapters with an IPv4 address ↩
                   in the system!");
277         }
278 
279         /// <summary>
```

```
280         /// We check the syntax of the IP adress, first if it contains ↩
                whitespaces or is null, then if there is 4 parts and finaly we ↩
                parse the 4 strings
281         /// From : ↩
                https://stackoverflow.com/questions/11412956/what-is-the-best-way-of-valida
282         /// </summary>
283         /// <param name="ipString"></param>
284         /// <returns></returns>
285         public static bool ValidateIPv4(string ipString)
286         {
287             if (string.IsNullOrWhiteSpace(ipString))
288             {
289                 return false;
290             }
291 
292             string[] splitValues = ipString.Split('.');
293             if (splitValues.Length != 4)
294             {
295                 return false;
296             }
297 
298             byte tempForParsing;
299 
300             return splitValues.All(r => byte.TryParse(r, out ↩
                    tempForParsing));
301         }
302     }
303 }
```

Listing 2 – Sources/NetworkManager.cs

13.1.3 DisplayedRect.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 06.06.2020
9  *
10 * File          : DisplayedRect.cs
11 */
12 using Microsoft.Xna.Framework;
13 using Microsoft.Xna.Framework.Graphics;
14 
15 namespace TPI_2020_MonoBattle
16 {
17     /// <summary>
18     /// Class that represent a rectangle that is drawable on screen.
19     /// </summary>
20     public class DisplayedRect
21     {
22         #region Fields
23         /// <summary>
24         /// Represent the rectangle that fills the inside space
25         /// </summary>
26         private Rectangle _innerRectangle;
27         /// <summary>
28         /// The 4 lines that create the borders
29         /// </summary>
30         private Rectangle[] _borders;
31         /// <summary>
32         /// The color inside the rectangle
33         /// </summary>
34         protected Color _innerColor;
35         /// <summary>
36         /// The color of the borders
37         /// </summary>
38         private Color _borderColor;
39         /// <summary>
40         /// The thickness in pixels of the borders
41         /// </summary>
42         private int _borderThickness;
43         /// <summary>
44         /// The texture in which the rectangle is drawn in
45         /// </summary>
46         private Texture2D _texture;
47         #endregion
48 
49         #region Properties
50         /// <summary>
51         /// Represent the rectangle that fills the inside space
52         /// Automatically recalculates the borders when changed
53         /// </summary>
54         protected virtual Rectangle InnerRectangle
55         {
56             get => _innerRectangle;
57             set
58             {
59                 _innerRectangle = value;
60                 _borders = CreateRectBorder(_innerRectangle);
61             }
62         }
63         /// <summary>
```

```
64          /// The color inside the rectangle
65          /// </summary>
66          public Color InnerColor { get => _innerColor; set => _innerColor = ↩
                value; }
67          #endregion
68          #region Constructors
69          /// <summary>
70          /// A rectangle drawable on screen.
71          /// </summary>
72          /// <param name="texture">The texture in which the rectangle is ↩
                drawn in</param>
73          /// <param name="innerRectangle">Represent the rectangle that ↩
                fills the inside space</param>
74          /// <param name="borderThickness">The thickness in pixels of the ↩
                borders</param>
75          /// <param name="innerColor">The color inside the rectangle</param>
76          /// <param name="borderColor">The color of the borders</param>
77          public DisplayedRect(Rectangle innerRectangle, int ↩
                borderThickness, Color innerColor, Color borderColor)
78          {
79              _texture = GameManager.MainTexture;
80              InnerRectangle = innerRectangle;
81              _innerColor = innerColor;
82              _borderColor = borderColor;
83              _borderThickness = borderThickness;
84
85              _borders = CreateRectBorder(InnerRectangle);
86          }
87          #endregion
88
89          #region Methods
90          /// <summary>
91          /// Draws the rectangle on the screen,
92          /// the spriteBatch must be started before calling this method
93          /// </summary>
94          /// <param name="spriteBatch"></param>
95          public virtual void Draw(SpriteBatch spriteBatch)
96          {
97              spriteBatch.Draw(_texture, InnerRectangle, _innerColor);
98              foreach (Rectangle border in _borders)
99              {
100                 spriteBatch.Draw(_texture, border, _borderColor);
101             }
102         }
103         /// <summary>
104         /// Create the 4 borders depending on the rectangle given
105         /// </summary>
106         /// <param name="rect"></param>
107         /// <returns></returns>
108         private Rectangle[] CreateRectBorder(Rectangle rect)
109         {
110             Rectangle[] borders = {
111                 new Rectangle(rect.Left, rect.Top, _borderThickness, ↩
                        rect.Height + _borderThickness),
112                 new Rectangle(rect.Right, rect.Top, _borderThickness, ↩
                        rect.Height + _borderThickness),
113                 new Rectangle(rect.Left, rect.Top, rect.Width + ↩
                        _borderThickness, _borderThickness),
114                 new Rectangle(rect.Left, rect.Bottom, rect.Width + ↩
                        _borderThickness, _borderThickness)
115             };
116             return borders;
117         }
118         /// <summary>
119         /// Returns true if the given point is in the rectangle hitbox
```

```
120         /// </summary>
121         /// <param name="position">Point to test</param>
122         public virtual bool IntersectWith(Point position)
123         {
124             return InnerRectangle.Contains(position);
125         }
126         #endregion
127     }
128 }
```

Listing 3 – Sources/DisplayedRect.cs

13.1.4 TextRect.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 06.06.2020
9  *
10 * File          : TextRect.cs
11 */
12 using Microsoft.Xna.Framework;
13 using Microsoft.Xna.Framework.Graphics;
14 
15 namespace TPI_2020_MonoBattle
16 {
17     /// <summary>
18     /// Derived class of displayRect that let the user insert text
19     /// </summary>
20     class TextRect : DisplayedRect
21     {
22         #region Fields
23         /// <summary>
24         /// The text to display
25         /// </summary>
26         private string _displayText;
27         /// <summary>
28         /// The font used to display the text
29         /// </summary>
30         private SpriteFont _font;
31         /// <summary>
32         /// The color of the text displayed
33         /// </summary>
34         private Color _textColor;
35         #endregion
36 
37         #region Properties
38         /// <summary>
39         /// The text to display
40         /// </summary>
41         public string DisplayText { get => _displayText; set => ↩
               _displayText = value; }
42         #endregion
43 
44         #region Constructor
45         public TextRect(Rectangle innerRectangle, int borderThickness, ↩
               Color innerColor, Color borderColor, string displayText, ↩
               SpriteFont font, Color textColor)
46             : base( innerRectangle, borderThickness, innerColor, borderColor)
47         {
48             _displayText = displayText;
49             _font = font;
50             _textColor = textColor;
51         }
52         #endregion
53 
54         #region Methods
55         /// <summary>
56         /// Draws the square on the spriteBatch with the text in the middle
57         /// </summary>
58         /// <param name="spriteBatch"></param>
59         public override void Draw(SpriteBatch spriteBatch)
60         {
```

```
61              base.Draw(spriteBatch);
62              // We determine the position of the text using vectors
63              spriteBatch.DrawString(_font, _displayText, ↩
                    InnerRectangle.Location.ToVector2() + ↩
                    InnerRectangle.Size.ToVector2() / 2 - ↩
                    _font.MeasureString(_displayText) / 2, _textColor);
64          }
65
66          #endregion
67      }
68  }
```

Listing 4 – `Sources/TextRect.cs`

13.1.5 Ship.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 06.06.2020
9  *
10 * File          : Ship.cs
11 */
12 using Microsoft.Xna.Framework;
13 using Microsoft.Xna.Framework.Graphics;
14 using static TPI_2020_MonoBattle.Constants;
15 
16 namespace TPI_2020_MonoBattle
17 {
18     /// <summary>
19     /// Class derived from DispalyedRect that represent a ship
20     /// </summary>
21     public class Ship : DisplayedRect
22     {
23         #region Fields
24         /// <summary>
25         /// The rectangle of the entire ship
26         /// </summary>
27         private Rectangle _hitbox;
28         /// <summary>
29         /// Length of the ship in squares
30         /// </summary>
31         private int _length;
32         /// <summary>
33         /// How the ship is positionned spatialy on the grid
34         /// </summary>
35         private bool _isVertical;
36         /// <summary>
37         /// The coordinates of all the square occupied by the ship
38         /// </summary>
39         private Point[] _squareOccupied;
40         /// <summary>
41         /// The number of time the ship has been hit by a player.
42         /// If this number is higher or equal than his length, the ship sink
43         /// </summary>
44         private int _nbTimeHit;
45         #endregion
46         #region Properties
47         /// <summary>
48         /// The "head" rectangle of the ship
49         /// </summary>
50         protected override Rectangle InnerRectangle
51         {
52             get => base.InnerRectangle;
53             // We recalculate the hitbox everytime the base rectangle changes
54             set
55             {
56                 base.InnerRectangle = value;
57                 if (_isVertical)
58                 {
59                     _hitbox = new Rectangle(InnerRectangle.X, ↩
                           InnerRectangle.Y, InnerRectangle.Width, ↩
                           InnerRectangle.Height * _length);
60                 }
61                 else
```

```
62                  {
63                      _hitbox = new Rectangle(InnerRectangle.X, ↩
                            InnerRectangle.Y, InnerRectangle.Width * _length, ↩
                            InnerRectangle.Height);
64                  }
65              }
66          }
67          /// <summary>
68          /// The top left position of the ship
69          /// </summary>
70          public Point Position
71          {
72              get
73              {
74                  return InnerRectangle.Location;
75              }
76              set
77              {
78                  InnerRectangle = new Rectangle(value, InnerRectangle.Size);
79              }
80          }
81          /// <summary>
82          /// Returns if the ship has sunk already,
83          /// if the nbOfTimeHit is higher or equal than his length, the ↩
                ship sink
84          /// </summary>
85          public bool IsSunk
86          {
87              get
88              {
89                  return _nbTimeHit >= _length;
90              }
91          }
92          /// <summary>
93          /// The coordinates of all the square occupied by the ship
94          /// </summary>
95          public Point[] SquareOccupied { get => _squareOccupied; set => ↩
                _squareOccupied = value; }
96          /// <summary>
97          /// How the ship is positionned spatialy on the grid
98          /// </summary>
99          public bool IsVertical { get => _isVertical; }
100         /// <summary>
101         /// Length of the ship in squares
102         /// </summary>
103         public int Length { get => _length; }
104         /// <summary>
105         /// The number of time the ship has been hit by a player.
106         /// If this number is higher or equal than his length, the ship sink
107         /// </summary>
108         public int NbTimeHit { get => _nbTimeHit; set => _nbTimeHit = ↩
                value; }
109         #endregion
110         #region Constructor
111         public Ship(Rectangle innerRectangle, int borderThickness, Color ↩
                innerColor, Color borderColor, int length) : ↩
                base(innerRectangle, borderThickness, innerColor, borderColor)
112         {
113             _length = length;
114             _hitbox = new Rectangle(InnerRectangle.X, InnerRectangle.Y, ↩
                    InnerRectangle.Width, InnerRectangle.Height * _length);
115             _isVertical = true;
116             _squareOccupied = new Point[_length];
117             _nbTimeHit = 0;
118         }
```

```
119         public Ship(Point position, int length) :this(new ↩
                Rectangle(position.X, position.Y, SQUARE_SIZE, SQUARE_SIZE), ↩
                BORDER_WIDTH, SHIP_COLOR, Color.Black, length)
120         {
121
122         }
123         #endregion
124         #region Methods
125         /// <summary>
126         /// Draws the ship on the spriteBatch given in parameter
127         /// </summary>
128         /// <param name="spriteBatch"></param>
129         public override void Draw(SpriteBatch spriteBatch)
130         {
131             // We loop through his length and determine the position of ↩
                    his body to draw each of the squares.
132             Rectangle originalRectangle = InnerRectangle;
133             base.Draw(spriteBatch);
134             for (int i = 1; i < _length; i++)
135             {
136                 if (_isVertical)
137                 {
138                     InnerRectangle = new Rectangle(InnerRectangle.X, ↩
                            InnerRectangle.Y + InnerRectangle.Height, ↩
                            InnerRectangle.Width, InnerRectangle.Height);
139                 }
140                 else
141                 {
142                     InnerRectangle = new Rectangle(InnerRectangle.X + ↩
                            InnerRectangle.Width, InnerRectangle.Y, ↩
                            InnerRectangle.Width, InnerRectangle.Height);
143                 }
144                 base.Draw(spriteBatch);
145             }
146             InnerRectangle = originalRectangle;
147         }
148         /// <summary>
149         /// Returns true if the given point is in the ship hitbox
150         /// </summary>
151         /// <param name="position">Point to test</param>
152         public override bool IntersectWith(Point position)
153         {
154             return _hitbox.Contains(position);
155         }
156         /// <summary>
157         /// Change the rotation of the ship
158         /// </summary>
159         public void Rotate()
160         {
161             _isVertical = !_isVertical;
162         }
163         #endregion
164     }
165 }
```

Listing 5 – Sources/Ship.cs

13.1.6 Grid.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 06.06.2020
9  *
10 * File          : Grid.cs
11 */
12 using Microsoft.Xna.Framework;
13 using Microsoft.Xna.Framework.Graphics;
14 using System;
15 using static TPI_2020_MonoBattle.Constants;
16 
17 namespace TPI_2020_MonoBattle
18 {
19     /// <summary>
20     /// Class that represent a grid used by a player to mark their ships ↩
       and/or hits.
21     /// </summary>
22     public class Grid
23     {
24         /// <summary>
25         /// Different possible square types
26         /// </summary>
27         public enum SquareType
28         {
29             empty,
30             ship,
31             hit,
32             miss,
33             sunk
34         }
35         #region Fields
36         /// <summary>
37         /// Array of square type that represent the status of all the squares
38         /// </summary>
39         private SquareType[,] _squareGrid;
40         /// <summary>
41         /// Top left point of the grid
42         /// </summary>
43         private Point _anchorPoint;
44         #endregion
45         #region Properties
46         /// <summary>
47         /// Top left point of the grid
48         /// </summary>
49         public Point AnchorPoint { get => _anchorPoint; set => ↩
               _anchorPoint = value; }
50         /// <summary>
51         /// Array of square type that represent the status of all the squares
52         /// </summary>
53         public SquareType[,] SquareGrid { get => _squareGrid; set => ↩
               _squareGrid = value; }
54         #endregion
55         #region Constructor
56         /// <summary>
57         /// Create a an empty grid at the given location
58         /// </summary>
59         /// <param name="texture">texture used to draw the grid</param>
60         /// <param name="anchorPoint">Top left point of the grid</param>
```

```
61          public Grid(Point anchorPoint)
62          {
63              _anchorPoint = anchorPoint;
64              // Initialize the cells
65              InitializeGrid();
66          }
67          #endregion
68          #region Methods
69          /// <summary>
70          /// Set squareGrid to a X by X array where all the squares are empty
71          /// X being the nb of square rows in constants.cs
72          /// </summary>
73          private void InitializeGrid()
74          {
75              SquareGrid = new SquareType[NB_SQUARE_ROW, NB_SQUARE_ROW];
76
77              for (int line = 0; line < NB_SQUARE_ROW; line++)
78              {
79                  for (int row = 0; row < NB_SQUARE_ROW; row++)
80                  {
81                      SquareGrid[line, row] = SquareType.empty;
82                  }
83              }
84          }
85          /// <summary>
86          /// Draws the entire grid on the spriteBatch given in parameter
87          /// </summary>
88          /// <param name="spriteBatch"></param>
89          public void Draw(SpriteBatch spriteBatch)
90          {
91              for (int line = 0; line < NB_SQUARE_ROW; line++)
92              {
93                  for (int row = 0; row < NB_SQUARE_ROW; row++)
94                  {
95                      Color squareColor = new Color();
96                      // Foreach squares we fill it with the right color for ↩
                            it's status
97                      switch (SquareGrid[row, line])
98                      {
99                          case SquareType.empty:
100                             squareColor = EMPTY_COLOR;
101                             break;
102                         case SquareType.ship:
103                             squareColor = SHIP_COLOR;
104                             break;
105                         case SquareType.hit:
106                             squareColor = HIT_COLOR;
107                             break;
108                         case SquareType.miss:
109                             squareColor = MISS_COLOR;
110                             break;
111                         case SquareType.sunk:
112                             squareColor = SUNK_COLOR;
113                             break;
114                     }
115                     Color borderColor = GRID_BORDER_COLOR;
116
117                     Rectangle innerRectangle = new Rectangle(row * ↩
                            SQUARE_SIZE, line * SQUARE_SIZE, SQUARE_SIZE, ↩
                            SQUARE_SIZE);
118                     innerRectangle.Location += _anchorPoint;
119
120                     new DisplayedRect(innerRectangle, BORDER_WIDTH, ↩
                            squareColor, borderColor).Draw(spriteBatch);
121                 }
```

```
122             }
123         }
124         /// <summary>
125         /// Finds the top left coordinates of the square in which the point is in
126         /// </summary>
127         /// <param name="inputPoint">The input point</param>
128         /// <returns></returns>
129         public Point FindClosestSquareAnchor(Point inputPoint)
130         {
131             try
132             {
133                 Point SquareCoordinates = FindSquare(inputPoint);
134 
135                 Point anchorPoint = new Point(SquareCoordinates.X * SQUARE_SIZE + AnchorPoint.X, SquareCoordinates.Y * SQUARE_SIZE + AnchorPoint.Y);
136                 return anchorPoint;
137             }
138             catch (ArgumentOutOfRangeException)
139             {
140                 // Point is not in grid
141                 return inputPoint;
142             }
143 
144         }
145         /// <summary>
146         /// Find the square that contains the inputed point
147         /// </summary>
148         /// <param name="pointInSquare"></param>
149         /// <returns></returns>
150         public Point FindSquare(Point pointInSquare)
151         {
152             // We find the coordinates relative to the grid
153             int pointRelativeX = pointInSquare.X - AnchorPoint.X;
154             int pointRelativeY = pointInSquare.Y - AnchorPoint.Y;
155 
156             // We check if the point is in the grid
157             int furthestPointXY = NB_SQUARE_ROW * SQUARE_SIZE;
158             if (pointRelativeX > furthestPointXY || pointRelativeY > furthestPointXY || pointRelativeX < 0 || pointRelativeY < 0)
159             {
160                 throw new ArgumentOutOfRangeException("The point is not in the Grid");
161             }
162             // We calculate the position of the cell the point is in by dividing his relative position by the size of a square
163             double nbSquareX = Math.Ceiling(Convert.ToDouble(pointRelativeX / SQUARE_SIZE));
164             double nbSquareY = Math.Ceiling(Convert.ToDouble(pointRelativeY / SQUARE_SIZE));
165 
166             return new Point(Convert.ToInt32(nbSquareX), Convert.ToInt32(nbSquareY));
167         }
168         #endregion
169     }
170 }
```

Listing 6 – Sources/Grid.cs

13.1.7 Player.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 06.06.2020
9  *
10 * File          : Player.cs
11 */
12 namespace TPI_2020_MonoBattle
13 {
14     /// <summary>
15     /// Abstract Class that represent a player and contains his grid
16     /// </summary>
17     public abstract class Player
18     {
19         #region Fields
20         protected Grid _playerGrid;
21         #endregion
22         #region Properties
23         public Grid PlayerGrid { get => _playerGrid; }
24         #endregion
25         #region Constructors
26         public Player(Grid playerGrid)
27         {
28             _playerGrid = playerGrid;
29         }
30         #endregion
31     }
32 }
```

Listing 7 – Sources/Player.cs

13.1.8 LocalPlayer.cs

```
1   /*
2  * Author       : Nelson Jeanrenaud
3  *
4  * Teacher      : Stéphane Garchery
5  *
6  * Experts      : Pierre Conrad, Philippe Bernard
7  *
8  * Date         : 06.06.2020
9  *
10 * File         : LocalPlayer.cs
11 */
12 using Microsoft.Xna.Framework;
13 using System;
14 using System.Collections.Generic;
15 using System.Linq;
16 using static TPI_2020_MonoBattle.Constants;
17 
18 namespace TPI_2020_MonoBattle
19 {
20     /// <summary>
21     /// Class that represent the player that is playing on the machine the ↩
       app is running.
22     /// </summary>
23     public class LocalPlayer : Player
24     {
25         #region Fields
26         /// <summary>
27         /// All the ships the player have
28         /// </summary>
29         private List<Ship> _ships;
30         #endregion
31 
32         #region Properties
33         internal List<Ship> Ships { get => _ships; set => _ships = value; }
34         #endregion
35         #region Constructor
36         /// <summary>
37         /// represent the player that is playing on the machine the app is ↩
           running.
38         /// </summary>
39         /// <param name="playerGrid">Grid where the player place his ↩
           boats</param>
40         /// <param name="ships">Ships that the palyer places</param>
41         public LocalPlayer(Grid playerGrid, List<Ship> ships) : ↩
           base(playerGrid)
42         {
43             _ships = ships;
44         }
45         #endregion
46 
47         #region Methods
48         /// <summary>
49         /// The opponent player shot at the given location.
50         /// Looks at the square at this location and return the state of it.
51         /// Marks the square as shot by opponent on the grid.
52         /// And if a ship is shot check if it has sunk
53         /// </summary>
54         /// <param name="location">Coordiantes the opponent shot</param>
55         /// <returns></returns>
56         public string ReceiveShot(Point location)
57         {
58             try
59             {
```

```
60                  if (_playerGrid.SquareGrid[location.X, location.Y] == ↩
                        Grid.SquareType.ship)
61                  {
62                      // If it hit a ship we search which on got hit and ↩
                            adjust his number of time hit.
63                      // If it has too many he sinks.
64                      foreach (Ship ship in _ships)
65                      {
66                          if (ship.SquareOccupied.Contains(location))
67                          {
68                              ship.NbTimeHit++;
69                              if (ship.IsSunk)
70                              {
71                                  if (_ships.Find(s => s.IsSunk == false) == ↩
                                        null)
72                                  {
73                                      return GAME_FINISHED_MESSAGE;
74                                  }
75                                  // We build a string containing the ↩
                                        positions of the other squares that ↩
                                        contains the ship
76                                  // That way, the opponent can marks the ↩
                                        whole ship as sunk on his side.
77                                  string positionsString = TYPE_SEPARATOR;
78                                  foreach (Point squareOccupiedLocation in ↩
                                        ship.SquareOccupied)
79                                  {
80                                      _playerGrid.SquareGrid[squareOccupiedLocation.X, ←
                                            squareOccupiedLocation.Y] = ↩
                                            Grid.SquareType.sunk;
81                                      positionsString += ";" + ↩
                                            squareOccupiedLocation.X + "," + ↩
                                            squareOccupiedLocation.Y;
82                                  }
83                                  return SINK_MESSAGE + positionsString;
84                              }
85                          }
86                      }
87                      _playerGrid.SquareGrid[location.X, location.Y] = ↩
                            Grid.SquareType.hit;
88                      return HIT_MESSAGE;
89                  }
90                  else if (_playerGrid.SquareGrid[location.X, location.Y] == ↩
                        Grid.SquareType.empty)
91                  {
92                      _playerGrid.SquareGrid[location.X, location.Y] = ↩
                            Grid.SquareType.miss;
93                      return MISS_MESSAGE;
94                  }
95                  return ERROR_MESSAGE;
96              }
97              catch (Exception)
98              {
99                  return ERROR_MESSAGE;
100             }
101         }
102
103         /// <summary>
104         /// Check if the ships are correctly placed on the grid
105         /// </summary>
106         /// <param name="imprintOnGrid">If true change the status of cells ↩
                as ships</param>
107         /// <returns></returns>
108         public bool ValidateShips(bool imprintOnGrid)
109         {
```

```
110             // List of all the squares already occupied by ships
111             List<Point> squareAlreadyOccupied = new List<Point>();
112 
113             foreach (Ship ship in _ships)
114             {
115                 try
116                 {
117                     Point shipHead = _playerGrid.FindSquare(ship.Position);
118                     // Determines the coordinates of the squares of the ↩
                        rest of the ship
119                     for (int i = 0; i < ship.Length; i++)
120                     {
121                         Point squareInShip = new Point();
122                         if (ship.IsVertical)
123                         {
124                             squareInShip = new Point(shipHead.X, ↩
                                shipHead.Y + i);
125                         }
126                         else
127                         {
128                             squareInShip = new Point(shipHead.X + i, ↩
                                shipHead.Y);
129                         }
130                         if (squareInShip.X >= NB_SQUARE_ROW || ↩
                            squareInShip.Y >= NB_SQUARE_ROW)
131                         {
132                             return false;
133                         }
134                         if (squareAlreadyOccupied.Contains(squareInShip))
135                         {
136                             return false;
137                         }
138                         squareAlreadyOccupied.Add(squareInShip);
139                         ship.SquareOccupied[i] = squareInShip;
140                     }
141                 }
142                 catch (ArgumentOutOfRangeException)
143                 {
144                     // A ship is out of the grid
145                     return false;
146                 }
147             }
148             if (imprintOnGrid)
149             {
150                 foreach (Point squarePosition in squareAlreadyOccupied)
151                 {
152                     _playerGrid.SquareGrid[squarePosition.X, ↩
                        squarePosition.Y] = Grid.SquareType.ship;
153                 }
154             }
155 
156             return true;
157         }
158         /// <summary>
159         /// Place the ship randomly
160         /// </summary>
161         public void PlaceShipsRandom()
162         {
163             // As long as the ship placement is not correct we replace the ↩
                ships
164             do
165             {
166                 Random rnd = new Random();
167                 foreach (Ship ship in _ships)
168                 {
```

```
169                     if (rnd.Next(2) == 0)
170                     {
171                         ship.Rotate();
172                     }
173                     ship.Position = ↩
                            _playerGrid.FindClosestSquareAnchor(new Point(
174                         rnd.Next(_playerGrid.AnchorPoint.X, ↩
                                _playerGrid.AnchorPoint.X + ( NB_SQUARE_ROW * ↩
                                SQUARE_SIZE )),
175                         rnd.Next(_playerGrid.AnchorPoint.Y, ↩
                                _playerGrid.AnchorPoint.Y + ( NB_SQUARE_ROW * ↩
                                SQUARE_SIZE ))));
176                 }
177             } while (!ValidateShips(false));
178         }
179         #endregion
180     }
181 }
```

Listing 8 – `Sources/LocalPlayer.cs`

13.1.9 OpponentPlayer.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 06.06.2020
9  *
10 * File          : OpponentPlayer.cs
11 */
12 using Microsoft.Xna.Framework;
13 using System;
14 using static TPI_2020_MonoBattle.Constants;
15 
16 namespace TPI_2020_MonoBattle
17 {
18     /// <summary>
19     /// Class derived of player that represent the opponent of the local ↩
       player
20     /// </summary>
21     public class OpponentPlayer : Player
22     {
23         #region Constructor
24         public OpponentPlayer(Grid playerGrid) : base(playerGrid)
25         {
26         }
27         #endregion
28 
29         #region Methods
30         /// <summary>
31         /// Update the grid with the shot result given in parameter
32         /// </summary>
33         /// <param name="location"></param>
34         /// <param name="shotResult"></param>
35         public void ReceiveShot(Point location, string shotResult)
36         {
37             if (shotResult.Contains(SINK_MESSAGE))
38             {
39                 string data = shotResult.Split(new string[] { ↩
                       TYPE_SEPARATOR}, ↩
                       StringSplitOptions.RemoveEmptyEntries)[1];
40                 string[] positions = data.Split(new string[] { ";" }, ↩
                       StringSplitOptions.RemoveEmptyEntries);
41 
42                 foreach (string position in positions)
43                 {
44                     string[] positionXY = position.Split(new string[] { ↩
                           "," }, StringSplitOptions.RemoveEmptyEntries);
45                     _playerGrid.SquareGrid[Convert.ToInt32(positionXY[0]), ↩
                           Convert.ToInt32(positionXY[1])] = ↩
                           Grid.SquareType.sunk;
46                 }
47             }
48             switch (shotResult)
49             {
50                 case HIT_MESSAGE:
51                     _playerGrid.SquareGrid[location.X, location.Y] = ↩
                           Grid.SquareType.hit;
52                     break;
53                 case MISS_MESSAGE:
54                     _playerGrid.SquareGrid[location.X, location.Y] = ↩
                           Grid.SquareType.miss;
```

```
55                          break;
56                      }
57                  }
58                  #endregion
59              }
60          }
```

Listing 9 – `Sources/OpponentPlayer.cs`

13.1.10 BotPlayer.cs

```
1   /*
2  * Author        : Nelson Jeanrenaud
3  *
4  * Teacher       : Stéphane Garchery
5  *
6  * Experts       : Pierre Conrad, Philippe Bernard
7  *
8  * Date          : 06.06.2020
9  *
10 * File          : BotPlayer.cs
11 */
12 using System;
13 using System.Collections.Generic;
14 using Microsoft.Xna.Framework;
15 using static TPI_2020_MonoBattle.Constants;
16 
17 namespace TPI_2020_MonoBattle
18 {
19     /// <summary>
20     /// Derived class of LocalPlayer that place his ships by himself and ↩
       emulate a player shooting
21     /// </summary>
22     internal class BotPlayer : LocalPlayer
23     {
24         #region Fields
25         /// <summary>
26         /// List of all the points already tried by the bot,
27         /// used so he doesn't shoot twice at the same place
28         /// </summary>
29         private List<Point> _positionsAlreadyTested;
30         #endregion
31         #region Constructor
32         /// <summary>
33         /// Derived class of LocalPlayer that place his ships by himself ↩
           and emulate a player shooting
34         /// </summary>
35         /// <param name="playerGrid">Grid where the bot plays</param>
36         /// <param name="ships">Ships that the bot places</param>
37         internal BotPlayer(Grid playerGrid, List<Ship> ships) : ↩
               base(playerGrid, ships)
38         {
39             _positionsAlreadyTested = new List<Point>();
40             // Place ships randomly on the grid
41             PlaceShipsRandom();
42             ValidateShips(true);
43         }
44         #endregion
45 
46         #region Methods
47         /// <summary>
48         /// Returns a random point in grid limits
49         /// that hasn't already been returned with this function
50         /// </summary>
51         public Point Shoot()
52         {
53             Random rnd = new Random();
54             Point pointToTest;
55             // As long as we don't find a unused point, we loop
56             do
57             {
58                 pointToTest = new Point(rnd.Next(0, NB_SQUARE_ROW), ↩
                       rnd.Next(0, NB_SQUARE_ROW));
59             } while (_positionsAlreadyTested.Contains(pointToTest));
```

```
60
61                  _positionsAlreadyTested.Add(pointToTest);
62                  return pointToTest;
63              }
64              #endregion
65          }
66      }
```

Listing 10 – `Sources/BotPlayer.cs`

13.1.11 frmConnection.cs

```
1   /*
2  * Author       : Nelson Jeanrenaud
3  *
4  * Teacher      : Stéphane Garchery
5  *
6  * Experts      : Pierre Conrad, Philippe Bernard
7  *
8  * Date         : 06.06.2020
9  *
10 * File         : FrmConnection.cs
11 */
12 using System;
13 using System.Net;
14 using System.Net.Sockets;
15 using System.Windows.Forms;
16 
17 namespace TPI_2020_MonoBattle
18 {
19     /// <summary>
20     /// This is the forms that opens up when you launch the application.
21     /// She let the user enter the IP address of the other player and ↩
           launch the game
22     /// </summary>
23     public partial class frmConnection : Form
24     {
25         #region Fields
26         /// <summary>
27         /// Determine if the application will be launched when this forms ↩
               closes.
28         /// </summary>
29         private bool _needlaunchApplication;
30         /// <summary>
31         /// Delegate for the LaunchGame Method
32         /// </summary>
33         private delegate void SafeClosingDelegate();
34         #endregion
35         #region Properties
36         /// <summary>
37         /// Determine if the application will be launched when this forms ↩
               closes.
38         /// </summary>
39         public bool NeedlaunchApplication { get => _needlaunchApplication; }
40         #endregion
41         #region Constructors
42         /// <summary>
43         /// Constructor of the setting form : Initialize the variables and ↩
               start the local server.
44         /// </summary>
45         public frmConnection()
46         {
47             InitializeComponent();
48             // By default the game doesn't start when the form closes
49             _needlaunchApplication = false;
50             // Start the local server to receive game request
51             NetworkManager.GetInstance().StartServer();
52             // Add the FormRequest event from the NetworkManager Class, ↩
                   it's called everytime the server receive a game request.
53             NetworkManager.GetInstance().ReceivedInvite += ↩
                   SettingMenuForm_ReceivedInvite;
54             // Display the local ip in a textbox for the user
55             tbxSelfIp.Text = NetworkManager.GetLocalIPAddress().ToString();
56         }
57         #endregion
```

```
58
59          #region Methods
60          /// <summary>
61          /// Sends a connection request to the server at the given IP, if ↩
                the connection is accepted the game will be launched.
62          /// Otherwise a corresponding error message is displayed.
63          /// </summary>
64          /// <param name="ipPeer"></param>
65          private void SendRequest(string ipPeer)
66          {
67              try
68              {
69                  // Parse the entered adress and sends a connection request ↩
                        message to it.
70                  // This if block is executed only if the adress reply ↩
                        positivly to the request.
71                  if ↩
                        (NetworkManager.GetInstance().SendConnectionRequest(IPAddress.Parse
72                  {
73                      // Protecting from the issue where the player enters ↩
                            his own adress
74                      if (ipPeer != ↩
                            NetworkManager.GetLocalIPAddress().ToString())
75                      {
76                          GameManager.IsFirstPlayer = true;
77                          LaunchGame();
78                      }
79                      else
80                      {
81                          throw new FormatException();
82                      }
83                  }
84                  else
85                  {
86                      MessageBox.Show("The␣IP␣did␣not␣accept␣the␣request", ↩
                            "", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
87                  }
88              }
89              catch (FormatException)
90              {
91                  MessageBox.Show("Please␣enter␣a␣correct␣IP␣address", ↩
                        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
92              }
93              catch (SocketException)
94              {
95                  MessageBox.Show("We␣couldn't␣connect␣to␣the␣IP", "Error", ↩
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
96              }
97          }
98          /// <summary>
99          /// When called, this methods prompt the user about the game ↩
                starting before forcing the form to close.
100         /// </summary>
101         private void LaunchGame()
102         {
103             // When accessing windows form properties from another thread ↩
                    you need to make it safe
104             // Source : ↩
                    https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how
105             if (InvokeRequired)
106             {
107                 // Call this method from the current thread instead of the ↩
                        NetworkManager's thread
108                 Invoke(new SafeClosingDelegate(LaunchGame));
109             }
```

```
110             else
111             {
112                 MessageBox.Show("The game will begin now", "Game is starting", MessageBoxButtons.OK, MessageBoxIcon.Information);
113                 // By setting this to true the game will start when the form closes.
114                 _needlaunchApplication = true;
115                 Close();
116             }
117         }
118         #endregion
119 
120         #region Events
121         /// <summary>
122         /// Event called when the button sendRequest is clicked, it sends a request to the given ip adress
123         /// </summary>
124         /// <param name="sender"></param>
125         /// <param name="e"></param>
126         private void BtnSendRequest_Click(object sender, EventArgs e)
127         {
128             SendRequest(tbxIpAddressOpponent.Text);
129         }
130         /// <summary>
131         /// Starts the game in local with the bot activated
132         /// </summary>
133         /// <param name="sender"></param>
134         /// <param name="e"></param>
135         private void btnConnectLocal_Click(object sender, EventArgs e)
136         {
137             MessageBox.Show("The game will begin now", "Game is starting", MessageBoxButtons.OK, MessageBoxIcon.Information);
138             // By setting this to true the game will start when the form closes.
139             _needlaunchApplication = true;
140             GameManager.IsBotGame = true;
141             GameManager.IsFirstPlayer = true;
142             Close();
143         }
144         /// <summary>
145         /// Called when the NetworkManager triggers a FormRequest action (when he receive a request response).
146         /// </summary>
147         /// <param name="sender"></param>
148         /// <param name="result">The result of the request, we only launch the game if the result is true</param>
149         private void SettingMenuForm_ReceivedInvite(object sender, bool result)
150         {
151             // If true forms need to launch application
152             if (result)
153             {
154                 LaunchGame();
155             }
156         }
157         #endregion
158     }
159 }
```

Listing 11 – Sources/frmConnection.cs

13.1.12 Program.cs

```
   /*
 * Author       : Nelson Jeanrenaud
 *
 * Teacher      : Stéphane Garchery
 *
 * Experts      : Pierre Conrad, Philippe Bernard
 *
 * Date         : 06.06.2020
 *
 * File         : Program.cs
 */
using System;
using System.Windows.Forms;

namespace TPI_2020_MonoBattle
{
#if WINDOWS || LINUX
    /// <summary>
    /// The main class.
    /// </summary>
    public static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
             Application.EnableVisualStyles();
             Application.SetCompatibleTextRenderingDefault(false);
             frmConnection menuForm = new frmConnection();
             Application.Run(menuForm);
             if (menuForm.NeedlaunchApplication)
             {
                 GameManager game = new GameManager();
                 game.Run();
             }
            /*GameManager game = new GameManager();
            game.Run();*/
        }
    }
#endif
}
```

Listing 12 – `Sources/Program.cs`

13.1.13 Constants.cs

```
1   /*
2  * Author       : Nelson Jeanrenaud
3  *
4  * Teacher      : Stéphane Garchery
5  *
6  * Experts      : Pierre Conrad, Philippe Bernard
7  *
8  * Date         : 06.06.2020
9  *
10 * File         : Constants.cs
11 */
12 using Microsoft.Xna.Framework;
13 
14 namespace TPI_2020_MonoBattle
15 {
16     /// <summary>
17     /// Class containing all of the parameters for the game
18     /// </summary>
19     public static class Constants
20     {
21         // Window parameters
22         public const string DISPLAY_NAME = "MonoBattle";
23         public const int SCREEN_SIZE_WIDTH = 1280;
24         public const int SCREEN_SIZE_HEIGHT = 720;
25         public static Color BACKGROUND_COLOR = Color.DarkGray;
26 
27         // Grid parameters
28         public const int NB_SQUARE_ROW = 10;
29         public const int SQUARE_SIZE = 50;
30         public const int BORDER_WIDTH = 2;
31 
32         public const int PLAYER_GRID_X = 10;
33         public const int PLAYER_GRID_Y = 100;
34         public const int OPPONENT_GRID_X = SCREEN_SIZE_WIDTH - ↩
               (SQUARE_SIZE * NB_SQUARE_ROW) - 10;
35         public const int OPPONENT_GRID_Y = 100;
36 
37         public static Color EMPTY_COLOR = Color.Blue;
38         public static Color SHIP_COLOR = Color.LightGray;
39         public static Color HIT_COLOR = Color.Red;
40         public static Color MISS_COLOR = Color.Green;
41         public static Color SUNK_COLOR = Color.DarkGray;
42         public static Color GRID_BORDER_COLOR = Color.Black;
43 
44         // UI parameters
45 
46         public const int TOOLTIP_WIDTH = 80;
47         public const int TOOLTIP_HEIGHT = 80;
48         public const int TOOLTIP_Y = SCREEN_SIZE_HEIGHT - TOOLTIP_HEIGHT - ↩
               10;
49 
50         public const int TOOLTIP_BORDER_WIDTH = 3;
51         public static Color TOOLTIP_BORDER_COLOR = Color.Black;
52         public static Color TOOLTIP_TEXT_COLOR = Color.Black;
53 
54         public const string TOOLTIP_HIT_TEXT = "Hit";
55         public static Color TOOLTIP_HIT_BACKCOLOR = HIT_COLOR;
56         public const int TOOLTIP_HIT_X = 10;
57 
58         public const string TOOLTIP_SINK_TEXT = "Sunk";
59         public static Color TOOLTIP_SINK_BACKCOLOR = SUNK_COLOR;
60         public const int TOOLTIP_SINK_X = TOOLTIP_HIT_X + TOOLTIP_WIDTH + 10;
61 
```

```
62          public const string TOOLTIP_MISS_TEXT = "Miss";
63          public static Color TOOLTIP_MISS_BACKCOLOR = MISS_COLOR;
64          public const int TOOLTIP_MISS_X = TOOLTIP_SINK_X + TOOLTIP_WIDTH + ↩
                10;
65
66
67          public const int TURN_INDICATOR_WIDTH = 200;
68          public const int TURN_INDICATOR_HEIGHT = 50;
69          public const int TURN_INDICATOR_X = SCREEN_SIZE_WIDTH / 2 - ↩
                TURN_INDICATOR_WIDTH / 2;
70          public const int TURN_INDICATOR_Y = 10;
71          public const int TURN_INDICATOR_BORDER_WIDTH = 3;
72          public static Color TURN_INDICATOR_BORDER_COLOR = Color.Black;
73          public static Color TURN_INDICATOR_LOCAL_TURN_BACKCOLOR = ↩
                Color.LightGreen;
74          public static Color TURN_INDICATOR_OPPONENT_TURN_BACKCOLOR = ↩
                Color.Red;
75          public static Color TURN_INDICATOR_TEXT_COLOR = Color.Black;
76          public const string TURN_INDICATOR_LOCAL_TURN_TEXT = "Your␣turn";
77          public const string TURN_INDICATOR_OPPONENT_TURN_TEXT = ↩
                "Opponent's␣turn";
78
79          public const int BTN_VALIDATE_WIDTH = 300;
80          public const int BTN_VALIDATE_HEIGHT = 50;
81          public const int BTN_VALIDATE_X = SCREEN_SIZE_WIDTH / 2 - ↩
                TURN_INDICATOR_WIDTH / 2;
82          public const int BTN_VALIDATE_Y = 10;
83          public const int BTN_VALIDATE_BORDER_WIDTH = 3;
84          public static Color BTN_VALIDATE_BORDER_COLOR = Color.Black;
85          public static Color BTN_VALIDATE_BACKCOLOR = Color.LightGreen;
86          public static Color BTN_VALIDATE_TEXT_COLOR = Color.Black;
87          public const string BTN_VALIDATE_TEXT = "Validate␣placements";
88
89          public const int BTN_SURRENDER_WIDTH = 300;
90          public const int BTN_SURRENDER_HEIGHT = 50;
91          public const int BTN_SURRENDER_X = 10;
92          public const int BTN_SURRENDER_Y = 10;
93          public const int BTN_SURRENDER_BORDER_WIDTH = 3;
94          public static Color BTN_SURRENDER_BORDER_COLOR = Color.Black;
95          public static Color BTN_SURRENDER_BACKCOLOR = Color.LightGreen;
96          public static Color BTN_SURRENDER_TEXT_COLOR = Color.Black;
97          public const string BTN_SURRENDER_TEXT = "Surrender";
98
99          public const int BTN_RANDOM_WIDTH = 300;
100         public const int BTN_RANDOM_HEIGHT = 50;
101         public const int BTN_RANDOM_X = SCREEN_SIZE_WIDTH - ↩
                BTN_RANDOM_WIDTH - 10;
102         public const int BTN_RANDOM_Y = 10;
103         public const int BTN_RANDOM_BORDER_WIDTH = 3;
104         public static Color BTN_RANDOM_BORDER_COLOR = Color.Black;
105         public static Color BTN_RANDOM_BACKCOLOR = Color.LightGreen;
106         public static Color BTN_RANDOM_TEXT_COLOR = Color.Black;
107         public const string BTN_RANDOM_TEXT = "Place␣ships␣at␣random";
108
109         // Network parameters
110         public const int CONNECTION_PORT = 51340;
111         public const string TYPE_SEPARATOR = "%";
112         public const string MESSAGE_END_SEPARATOR = "#";
113         public const string ACCEPT_INVITE = "1";
114         public const string REFUSE_INVITE = "0";
115         public const string HIT_MESSAGE = "hit";
116         public const string MISS_MESSAGE = "miss";
117         public const string SINK_MESSAGE = "sink";
118         public const string GAME_FINISHED_MESSAGE = "finish";
119         public const string ERROR_MESSAGE = "error";
```

```
120         public enum MESSAGE_TYPE
121         {
122             connectionRequest,
123             shoot,
124             endgame
125         }
126     }
127 }
```

Listing 13 – `Sources/Constants.cs`