# RMPP
# Bomberman

Etienne PENAULT

Semestre 4

# SOMMAIRE:

# Introduction

Bomberman est un jeu créé dans les années 80. Son premier éditeur fut Hudson Soft et fut lancé sur MSX avant d'être démocratisé sur la NES. Le principe du jeu est simple. Plusieurs joueurs s'affrontent en même temps afin d'être le dernier en vie. Il faudra alors tuer ses ennemis avec des bombes, casser des murs pour pouvoir retrouver les adversaires, etc. Tout au long de la partie, des bonus seront à disposition afin de pouvoir augmenter le nombre de bombes, leur puissance, la vitesse du joueur et bien d'autre. Bomberman est un jeu avec un principe simple, pourtant il plait et perdure malgré les âges. Nous pouvons noter qu'il y a toujours des rééditions de ce jeu presque 40 ans après sa création (CF: Nintendo Switch).

Au cours de ce rapport, nous allons voir comment nous avons réussi à recréer un jeu similaire à ce dernier, plus particulièrement en langage GO (golang), son fonctionnement, comment l'implémentation réseau fut possible, par quel protocole, les différents choix que le groupe a pu faire, et enfin les implémentation futures.

# Langage & Bibliothèque

Dans le cadre de notre projet, chaque membre s'est vu imposé de créer une version du jeu, dans des langages distincts et qu'ils soient tous compatible entre eux.

Pour ma part, j'ai choisi le langage GO, qui est un langage compilé, haut niveau, développé par Google, se rapprochant du C et du Pascal. GO me semblait cohérent car c'est un langage prenant de plus en plus d'ampleur de nos jours, du fait de sa puissance.
En plus de GO, il fallait quelque chose pour gérer le GUI (Graphic User Interface). Ce langage étant assez récent, il ne possède pas un catalogue aussi fourni que d'autres langages en terme de bibliothèque pour les interfaces graphiques. Le choix qui me semblait le plus cohérent était donc un "bindage" de la fameuse bibliothèque graphique pour le C: SDL2 (ainsi que SDL2_mixer pour la musique, SDL2_ttf pour l'affichage de texte). Voici le github de la librairie concernée: https://github.com/veandco/go-sdl2.

# Fonctionnement Utilisateur

Lorsque nous lançons le jeu, nous arrivons sur le menu. Sur ce menu sont proposés deux choix. Le premier permet de créer une partie, c'est à dire que nous souhaitons héberger la partie (et par conséquent devenir l'hôte de la partie), tandis que le second permet de rejoindre une partie hébergée par quelqu'un d'autre (voir se connecter à un serveur lancé indépendamment). Une fois un de ces choix sélectionné, l'utilisateur sera invité à entrer l'adresse du serveur à rejoindre dans le cas ou il n'héberge pas la partie (nous pouvons noter que l'adresse local du pc s'affiche quand on héberge la partie, question de facilité pour donner l'ip aux autres joueurs), le port du serveur, et pour finir son nom d'utilisateur.

Une fois cette étape passée, nous arrivons dans le lobby. Le lobby permet d'attendre avant que la partie soit lancée. Un système de chat est intégré afin de se mettre d'accord pour le choix de la carte, ou encore savoir quand il faut lancer la partie.
L'hôte (ou le premier joueur connecté si le serveur est détaché du client) dans le lobby possède tous les droits. Il peut décider de lancer la partie à tout moment en utilisant la commande "/start" qui permet de lancer la partie s'il y a au moins deux joueurs dans le lobby (nous pouvons noter que la commande "/start debug" permet d'outrepasser cela afin de pouvoir tester le programme tout seul). De plus, l'hôte peut également changer la carte à l'aide des flèches directionnelles (le numéro de la carte sera incrémenté si on appuye sur la flèche droite, décrémenté sur la flèche gauche).
Tout le monde dans le lobby a accès à la commande "/nbplayer" afin de connaître le nombre de joueurs actuellement dans le lobby.

Une fois dans la partie, l'utilisateur se déplace par les flèches directionnelles, et pose les bombes par la touche espace.
Les murs cassables sont les murs bleus. Lorsqu'un joueur casse un mur, aléatoirement, un bonus peut apparaître, le "S" correspond à un gain de vitesse, le "B" correspond au gain d'une bombe en plus, et pour finir, le "P" correspond a un gain d'une case en ampleur.
La partie s'arrête dès qu'un joueur vient de mourir et que le nombre de joueur en vie est égal à un.

Le seul moyen, durant une partie (même finie), pour revenir au lobby, est que l'hôte de la partie appuie sur la touche "R" de son clavier.

# Fonctionnement Client:

Le projet contient 11 fichier de code source:

bomb.go client.go display.go main.go menu.go server.go
bonus.go config.go explosion.go map.go player.go

Ainsi qu'un dossier contenant toute sorte de média (image,son,données de cartes etc.).

Tout d'abord, nous allons nous pencher sur le fichier main.go qui va permettre d'orchestrer le lancement du menu ainsi que le client. Logiquement nous allons appeler notre fonction "launchMenu()" se trouvant dans le fichier menu.go. Cette fonction repose principalement sur de l'affichage et sur de l'entrée de textes. Notons tout de même que c'est cette fonction qui va donner l'ordre au serveur de se lancer si l'utilisateur a choisi d'héberger la partie. Cette fonction retourne donc l'adresse et le port saisi afin de les passer à notre fonction "launchClient()" qui va donc se connecter à ces derniers.

La fonction "launchClient()" va tout d'abord lancer la fonction "clientLoop()" en thread. Cette fonction est basée sur une boucle infinie et une fonction "Read()" qui va nous permettre de récupérer les messages à un port et une adresse donnés. Pour chaque "type" de donnée lu (définie par un header de 3 lettres capitales), nous allons traiter l'information de manière spécifique pour les faire communiquer avec notre client.
Ensuite on lance notre fenêtre et on affiche le lobby. On envoie "LOG" suivi de notre nom au server pour lui indiquer qui nous sommes. Les messages que nous pouvons voir dans le lobby sont définis par le header "MSG" pour un message normal. Si quelqu'un vient de se connecter ("ARV"), nous allons afficher le message de connection, et de la même manière si quelqu'un part ("LFT"), on affichera un message de déconnection. Au delà du visuel, lorsqu'un joueur se connecte ou part, cela provoque l'ajout/la suppression d'un "Player" à notre map de Joueur. Les données commencent par "LST" quant à elle on pour but de lister le nombre de joueur déjà présent sur le serveur et donc de les ajouter à notre map de "Player". "CFG" avant que la partie soit lancer, permet de définir une vitesse, un nombre de bombes, etc, soit la configuration des joueurs. Pour finir avec le lobby, lorsque le client reçoit "ACT", cela indique au client qu'on change d'index de carte. Si nous somme l'hôte, on envoie "NXT" ou "PRV" au changement de carte.

Au commencement de la partie, le client reçoit le header “MAP” qui va lui donner, sous forme de nombres, la largeur, la hauteur, et l’apparence de la carte, suivi de “ALL” qui indiquera au client quelle position est attribuée à tel joueur. Enfin, avant que la partie commence, on envoie “STR” suivi du nombre afin d’indiquer au client combien, de temps il y a attendre avant le début de la partie et “SPD” qui donne le délai de mouvement initial du joueur afin d’avoir un bel affichage.

Une fois la partie débutée, le client peut recevoir tout type d’information:

-“POS” suivi de l’index et des coordonnées du joueur.
-”BMB” suivi des coordonnées et de la puissance de la bombe quand une bombe est posée.
-”BRK” suivi des coordonnées des murs cassés
-”BNS” suivi des coordonnées du bonus ainsi que son type.
-”GOT” suivi des coordonnées quand quelqu’un récupère un bonus.
-”DTH” suivi de l’index du joueur tué ainsi que du tueur.
-”END” suivi du numéro du gagnant quand la partie est terminée.
-”RST” pour retourner au lobby.

Toutes ces informations sont traitées en parallèle de la boucle graphique. Il suffit donc juste de dessiner à l'écran ces informations (par exemple dessiner tout myBonus, myPlayer, myExplosions)

# Fonctionnement Serveur

C’est dans le serveur que repose le coeur du fonctionnement du jeu. En effet, c’est ce dernier qui exécutera tous les calculs. Cela a pour but d'alléger le client, et ainsi éviter toute sorte de triche côté client (au cas ou un client soit défectueux ou incompatible par exemple).

Tout d’abord, on commence par lancer deux fonctions sous forme de thread. Une s’appelant “holdingExplosions()” et l’autre “holdingMap()”.

La première va permettre de regarder 60 fois par secondes si un joueur n’est pas dans une explosion. Si il y en a un, on envoie à tous les clients que ce joueur est mort par le header “DTH”.
La seconde fonction permet d’actualiser les cartes présentes coté serveur. Cela permet par exemple d'ajouter des cartes aux serveurs malgré que le serveur soit en train de tourner, on actualise toutes les 10 secondes car ce n’est pas autant vital que la gestion des explosions.

Une fois ces deux fonctions lancées en arrière plan on rentre dans une boucle infinie qui va permettre d'accepter chaque nouvelle connexion. Pour chaque nouvelle connexion, nous allons lancer "handleConnection()" en thread, c'est ici que repose le coeur du server.

La fonction "handleConnection()" est basée sur une boucle infinie et un appel à la fonction "Read()", tout comme la fonction "clientLoop()". De la même manière, nous allons récupérer les informations d'une connexion donnée, les traiter, et dans certains cas renvoyer d'autres informations au client concerné ou encore à tous les clients.

Tout d'abord, à chaque tour de boucle, nous allons vérifier si la connexion est toujours existante, sinon on renvoie "LFT" à tous les clients afin de les avertir du départ d'un joueur.
Si l'on reçoit "NXT" ou "PRV", alors nous renvoyons "ACT" afin d'avertir tous les clients que la carte a changé.

Lorsqu'un joueur se connecte, le serveur reçoit "LOG" suivi de son nom. Cela permet au serveur de prévenir tout le monde de l'arrivée d'un joueur par les données "CND","LST", "ARV", et "CFG".

Si le serveur reçoit "RDY", alors le serveur, à partir du fichier concerné transforme la carte sous forme de chaîne de caractère à envoyer, l'envoie, envoie la donnée "ALL" avec les positions de chaque joueur sur la carte, puis envoie "STR" et "SPD" à tous les joueurs.

Lorsque le serveur reçoit "MOV" suivi de la direction, il va calculer les collisions entre le joueur concerné et les murs, s'il peut bouger, alors le serveur envoie à tout le monde "POS" suivi de l'index du joueur, et de ces coordonnées. Notons que c'est à ce moment que nous allons vérifier si un joueur est sur un bonus. Si c'est le cas, nous envoyons "GOT" suivi des coordonnées du bonus en question.

Pour finir, lorsque le serveur reçoit "BMB", nous allons tout d'abord vérifier que l'emplacement de la bombe n'est pas pris. S'il ne l'est pas, nous renvoyons "BMB" à tous les joueurs, on décrémente le stock de bombe du joueur, puis lançons un "Sleep" afin d'attendre l'explosion de la bombe. Une fois ce "Sleep" fini, on lance un thread qui va permettre d'attendre que l'explosion finisse avant de redonner la bombe posée au joueur.
Ensuite, nous allons lancer la fonction "calculBreakedBombs()" qui va nous permettre de récupérer tous nos blocs cassés, ainsi que récupérer les endroits ou il y a explosion pour notre fonction "holdingExplosions()". Une fois cela fini, on envoie "BRK" suivi de la liste des blocs cassés à tous les clients.

# Difficultés

Lors de ce projet, je n’ai pas forcément rencontré de difficultés majeur malgré l’apprentissage d’un nouveau langage. Le seul problème que j’ai pu rencontré fu les différences de noms de fonction ou encore de prototype du bindage de SDL en go, et SDL original en C.

# Conclusion

Le projet est encore améliorable malgré qu’il soit jouable. Je continuerai à l’améliorer par le futur car il manque des choses essentielles selon moi, comme le score de joueur, l’affiche du nombre de bombes en direct, faire un système de classement, ou encore plein de petites choses qui font que le jeu soit gage de qualité.
Je suis plutôt fier du rendu pour un langage qui m'était extérieur avant ce projet. L’entraide du groupe était quelque chose qui, selon moi, nous a poussé vers le haut.

# Sources

https://fr.wikipedia.org/wiki/Bomberman
https://fr.wikipedia.org/wiki/Go
https://github.com/veandco/go-sdl2
https://golang.org/

# Remerciement

M. Pablo Rauzy (professeur de la matière)
Mme. Alix Houel (Graphisme)