GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

EDUC'TOI

Intellectus :

Le bot discord Multi-jeux

TORRADO YOAN

GILLIOEN FRANCOIS

MARTIN JEREMY

MANEZ FLORIAN

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

SOMMAIRE

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

Introduction

Pour ce projet, il a fallu choisir une spécialité, entre électronique et informatique, notre groupe était plus à l'aise en informatique, nous avons décidé d'orienter notre projet vers le domaine. L'objectif final du projet est de rendre quelque chose de concret qui serait éventuellement commerçable, une fois terminé. Pour cela nous avons réfléchi à ce qui serait vendeur, principalement à des personnes de notre génération : il nous fallait quelque Chose d'original dans son domaine et d'utile/intéressant pour le client.

Après une analyse de nos activités quotidiennes et de nos besoins, nous avons conclu qu'un bot discord avancé conviendrait, Discord était une plateforme de communication récente, les bots efficaces et appréciés sont rares, ce domaine est donc exploitable. Un administrateur de serveur Discord va avant tout chercher un bot « utile », qui permet de mettre de la musique, d'ajouter des commandes et certains répondent bien à ces attentes et sont très réputés. Nous avons donc choisi de travailler sur un bot divertissant et proposant plusieurs types de jeux, ce qui est plus rare et tout autant apprécié que les bots « utiles ».

Pour se démarquer et réussir à vendre notre projet, il nous faut un bot irréprochable tant sur la forme que sur le fond. Le fond étant quand même le plus important, nous avons choisi d'ajouter tant des petits jeux, tel qu'un QCM, que des plus grosses activités comme le jeu de rôle. Ce dernier a l'avantage d'être bien plus pratique sur discord que sur n'importe quelle autre plateforme car on peut mélanger la communication vocale et écrite ainsi qu'ajouter d'autres salons au serveur pour les discussions et activités annexes.

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

Fonctions réalisées

QCM :

Nous avons quasiment réussi à réaliser toutes les fonctions que nous avions prévu.

La fonction principale de création de Qcm est opérationnelle. Les questions sont créées de quatre manières différentes : question mathématiques, question texte, question images, question image + texte. Nous sommes contents d'avoir pu implémenter différents types de questions car cela apporte de la variété et cela évite la monotonie d'un qcm purement mathématique.

La partie qcm possède aussi des fonctions annexes. La fonction principale est la création/ lancement du Qcm et elle permet de créer un Qcm de 4 questions aléatoires. On peut également rajouter des questions à ce Qcm et mélanger les questions à l'aide de diverses commandes.

Blind test :

Le Blind-test utilise une fonction absente des Qcm le channel vocal. La partie programmation du Blind-test a été particulièrement compliqué à appréhender car il y a des nuances difficiles à comprendre dans la gestion (ouverture/fermeture) des channels vocaux qui entrainent souvent des bugs du Bot.

Notre Blind-test est malgré tout fonctionnel et nous permet de jouer notre musique et de demander à l'utilisateur de deviner la musique qu'il écoute. Le Blind-test peut être lancer également à plusieurs. La faille de nos jeux multijoueur quelles qu'ils soient est que nous n'avons pas réussi à cacher les réponses des joueurs jouant en même temps que nous. Cela peut donc influencer ceux n'ayant pas répondu à la question, mais d'un autre coté cacher les réponses de tout le monde empêche de comparer nos réponses à celles des autres. Ce problème a donc finalement un avantage donc on n'a pas essayé de le fixer.

Les musiques sont pour l'instant pré choisies, l'utilisateur ne peut donc pas choisir une musique à soumettre aux autres utilisateurs.

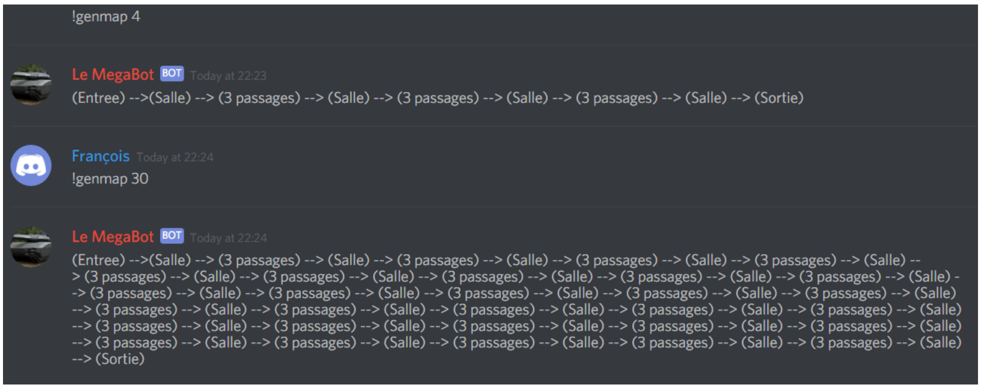
Il est à noter que le QCM et le Blind Test sont compatibles, ce qui fait qu'il est possible de mettre des questions de type Blind Test dans un Qcm ayant par exemple des questions de maths et d'autres visuelles.

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

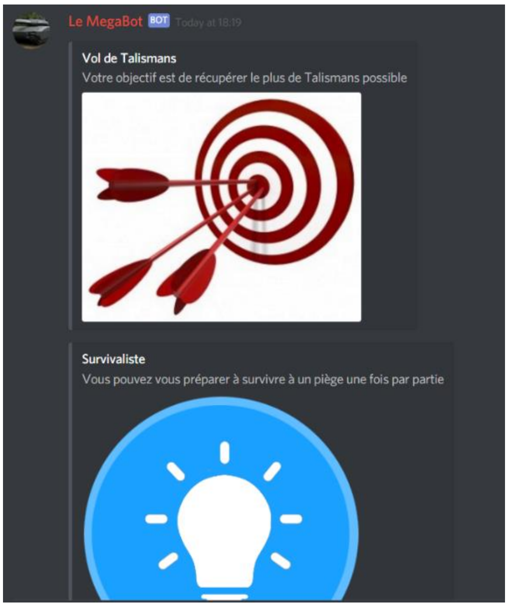
Jeux de rôle :

Le JDR utilise parfaitement les fonctionnalités de Discord en reliant les serveurs textuels et les serveurs vocaux. Il fonctionne selon le même principe qu'un loup garou, les personnes sont déjà créées et lors du début de la partie ils sont répartis aléatoirement entre les joueurs.

Lors de la création de la partie le bot détecte tous les joueurs présents dans le serveur vocal pour déterminer le nombre de joueurs, puis créer les serveurs textuels privés pour chaque joueur.

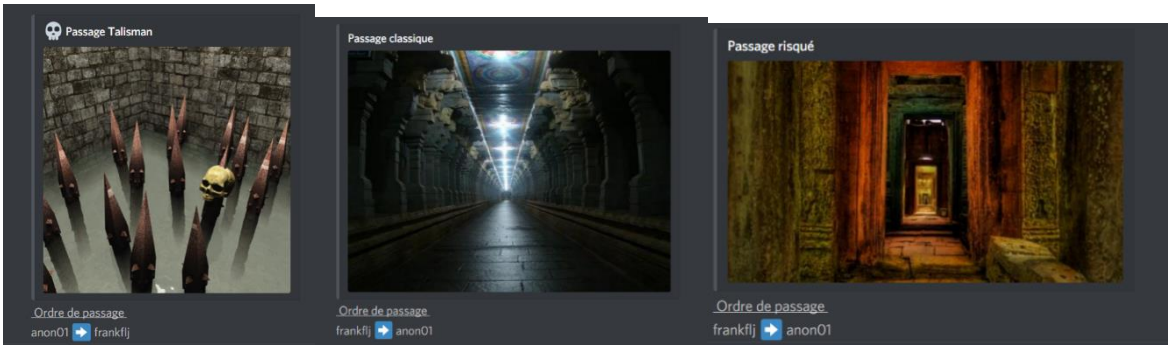


La commande !genmap permet de paramétrer le nombre de salles dans la partie.



Chaque joueur aura un but et une compétence qui sera cachée aux autres joueurs, il pourra utiliser sa compétence pendant la partie.

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY



La partie est donc un enchainement de choix de passage par vote et d'évènement aléatoire créer soit par la découverte d'une arme secrète soit par le sacrifice d'un joueur.

A la fin de la partie le jeu montre qui sont les joueurs qui ont réussi leurs objectifs et ceux qui ont survécu.

Fonctionnement et Code :

Blind test :

Le Blind test fonctionne avec le même système que le QCM, c'est-à-dire qu'il va sélectionner des questions ainsi que les réponses de manière aléatoire. Il va aussi pouvoir récupérer dans un autre dossier les fichiers audios et les lancer en même temps.

Dossier des fichiers audio :

Une fois les fichiers audio récupérés un des noms de fichier audio « file.Name » est pioché aléatoirement dans la liste des musiques existantes dans le dossier ci-dessus.

A partir de cet instant, le nom est envoyé dans un switch afin d'effectuer une liaison entre la musique, les différentes réponses possibles, en indiquant laquelle des réponses est la bonne grâce à la variable « correctAnswer »

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

```
a = b = c = d = null;
string correctAnswer = "Erreur, pas de réponse fournie";

// Génération des reponses
switch (file.Name)
{
    // Exemples de sons pour blindtest
    case "Zombie.mp3":
        a = "Macklemore";
        b = "Zombie";
        c = "Marroon 5";
        d = "Ed Sheeran";
        correctAnswer = b;
        break;

    case "Mackle.mp3" :
        a = "Macklemore";
        b = "B.o.B";
        c = "Vic Mensa";
        d = "Logic";
        correctAnswer = a;
        break;
    case "cWinston.mp3":
        a = "Charlie Winston";
        b = "Yodelice";
        c = "Ben Harper";
        d = "Jehro";
        correctAnswer = a;
        break;

    case "Trophée.mp3":
        a = "Vald";
        b = "Niska";
        c = "Fianso";
        d = "Hugo TSR";
        correctAnswer = a;
        break;
```

Lancement du Bot :

Pour permettre le lancement du bot, en plus du Token et de l'autorisation donnée par un administrateur du serveur au bot, il faut que le bot passe par plusieurs vérifications notamment en asynchrone. On peut voir que la fonction Main(string[] args) permet de lancer le Bot en mode asynchrone. Le mode asynchrone permettant d'effectuer du parallélisme

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

```
public async Task RunBotAsync()
{
    _client = new DiscordSocketClient();
    _commands = new CommandService();            _services = new ServiceCollection(
        .AddSingleton(_client)
        .AddSingleton(_commands)
        .AddSingleton(new AudioService())
        .BuildServiceProvider();
    _client.Log += Log;
    await RegistercommandAsync();

    _client.UserJoined += AnnounceJoinedUser;
    _client.ReactionRemoved += ReactionRemoved;
    Ping p = new Ping(_client);
    _client.ReactionAdded += p.ReactionParse;

    botToken = lines[0];
    await _client.LoginAsync(TokenType.Bot, botToken);

    await _client.StartAsync();

    await Task.Delay(-1);
    // event subscription
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("Starting ...\n");
    new Program().RunBotAsync().GetAwaiter().GetResult();
}
```

Connexion Audio et Envoi :

Pour le fonctionnement de l'audio il faut passer par un programme nommé ffmpeg qui permet l'envoie et la lecture des fichiers audios avec des Streams pour les lancer dans le salon vocal.

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

```
public async Task SendAudioAsync(IGuild guild, IMessageChannel channel, string path)
{
    // Your task: Get a full path to the file if the value of 'path' is only a filename.
    if (!File.Exists(path))
    {

        // Ne fonctionne que si libopus et libsodium sont correctement installés
        await channel.SendMessageAsync(" >> Le fichier n'existe pas à : " + path);
        return;
    }


    if (ConnectedChannels.TryGetValue(guild.Id, out IAudioClient client))
    {
        Console.WriteLine("\n----------------------------ENVOI DE L'AUDIO----------------------------------------------------------\n");
        Console.WriteLine("Client qui va être utilisé : " + client.ConnectionState);
        //HYPER MEGA IMPORTANT, UTILISER LES USING
        using (var ffmpeg = CreateStream(path))
        using (var output = ffmpeg.StandardOutput.BaseStream)
        using (var discord = client.CreatePCMStream(AudioApplication.Mixed))
        {
            try
            {
                await output.CopyToAsync(discord);
                Console.WriteLine("CopyToAsync() terminé: ");
            }
            finally { Console.WriteLine("FlushAsync() va être lancé "); await discord.FlushAsync(); }
            Console.WriteLine("Execution terminée");

        }
    }
    else
         await channel.SendMessageAsync("Merci de d'abord joindre le canal audio avec !join");
    Console.WriteLine("\n-----------------------------------------------------------------------------------------\n");
}
```

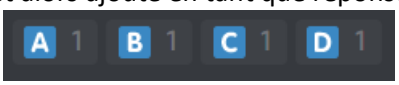
```
private Process CreateStream(string path)
{
    Console.WriteLine("Création d'un stream {" + path+ "}");
     return Process.Start(new ProcessStartInfo
     {
         FileName = "ffmpeg.exe",
         Arguments = $" -i {path} -ac 2 -f s16le -ar 48000 pipe:1",
         UseShellExecute = false,
         RedirectStandardOutput = true
     });
}
```

Système de réponse par Emoji (QCM):

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

L'affiche des émojis et leur utilisation est un des points importants du projet car ils permettent une utilisation plus fluide et ergonomique du bot. Pour fonctionner il faut que le programme détecte en avance la question et peut après cela mettre les émojis sur la question. Et lors de la réponse de l'utilisateur le programme peut identifier les réponses et les stocker.

Le déroulement est le suivant :

1. On regarde si la liste des QCM qcmList n'est pas vide, c'est-à-dire qu'elle possède au moins 1 QCM
2. On utilise ensuite un itérateur foreach qui est permet de parcourir en boucle la liste qcmList, ce qui permet d'analyser la réaction/émoji pour chacun des QCM
3. On vérifie ensuite la condition Bool « HasStarted » qui permet de savoir si le QCM a été démarré au préalable, sinon il n'y a pas raison d'analyser l'émoji davantage.
4. On regarde ensuite si parmi les numéros d'identifiants des questions du questionnaire actuel « qcm », si le numéro d'identifiant est contenu dans la liste des réponses du qcm actuel, alors cela signifie qu'il y'a une correspondance entre un message précis du QCM (le message représentant une question entière) et la réaction, on a donc réalisé un moyen de détecter quelle réaction appartient à quel QCM, la liaison de données est effectuée. Après l'opérateur &&, on peut dénoter que l'on utilise qcm.questionsID.Last() == msg.Id ce qui revient à vérifier que le dernier message apparu sur Discord doit être le message du QCM, cela évite d'analyser les anciens messages et de déclencher certains bugs telles que le passage à la question suivante en cliquant sur une lettre de réponse d'un message bien auparavant.
5. On vérifie enfin si l'utilisateur n'est pas un BOT, car le BOT a pour objectif d'analyser les actions des joueurs, qui sont pour notre QCM tous humains.
6. L'emoji est alors ajouté en tant que réponse puisque les seuls emojis disponibles sont les suivants : 

7. La commande StopAudioCmd est utile pour les Blind Test et permet de passer à la musique suivante en arrêtant la précédente
8. Enfin la fonction StartQCM(string name) est appelée, il s'agit en fait d'envoyer la prochaine question du QCM. Cette fonction prend également en charge le démarrage d'un QCM.

   Ce mode de détection est également présent pour le JDR mais nous n'avons pas choisi de le montrer car cela est basé sur le même principe

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

```
public async Task ReactionParse(Cacheable<IUserMessage, ulong> msg, ISocketMessageChannel msg2, SocketReaction socketReaction)
{
    await Console.Out.WriteLineAsync("\n--------------------------------------------------------------------------\nRéaction détectée!! " + msg.Id);


    if (qcmList != null)
    {
        foreach (var qcm in qcmList)
        {
            if (qcm.HasStarted)
            {
                if (qcm.questionsID.Contains(socketReaction.MessageId) && qcm.questionsID.Last() == msg.Id )
                {
                    if (!socketReaction.User.Value.IsBot)
                    {
                        qcm.allAnswers.Add(socketReaction);
                        await StopAudioCmd();
                        await StartQCM(qcm.name);
                    }

                }
            }
        }
    }
```

Jeu de Rôle :

Le jeu de rôle possède un système de gestion très ressemblant à celui d'un jeu vidéo auquel on aurait lié les fonctionnalités de Discord. On peut paramétrer chaque personnage selon de nombreuses caractéristiques.

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

Un joueur sur l'image ci-dessous possède de très nombreux attributs, on peut d'abord savoir s'il est en vie ou mort, il possède l'utilisateur Discord qui permet d'incarner ce joueur. Ainsi cela est très pratique puisque comme nous stockons un utilisateur, nous pouvons accéder à ses attributs et méthodes qui lui sont propres comme obtenir son Identifiant Discord, lui envoyer un message privé, renommer son pseudonyme etc…

Chaque joueur possède également une chaîne de texte « RestTextChannel », qui lui affiche des messages devant être cachés aux autres joueurs comme ses compétences et capacités.

```
// Comprend le type de personnage, qui définit un personnage fictif pour le joueur, exemple: Soldat Nazi
public class Player
{
    protected bool dead = false;
    public int _type;
    public IUser user;
    public string discordRole;
    public RestTextChannel textChannel;
    public int goal;
    [Required]
    private SocketCommandContext _context;
    public List<Ability> abilities = new List<Ability>();
    private ulong _playerID { get; set; }
    private string _playerName { get; set; }
    public int votePower = 1;
    public bool immuneToTraps = false;
    public int sacrificeCount = 0;


    public Player(int type, IUser u, SocketCommandContext context)
    {
        _type = type;
        user = u;
        _context = context;
        // En fonction du type, on assigne les bonnes compétences et capacités spéciales

        AssignGoalAndAbilities();
        //Console.WriteLine("Assigné : " + Enum.GetName(typeof(CharacterType), _type) + "\npour le joueur: " + u.Username + "\nobjectif: " + goal);
        JDR.allPlayers.Add(this);
    }

    public async Task ShowAbilityInTextChannel(Ability ability)...

    [Command("priv", RunMode = RunMode.Async)]
    public async Task PrivCh(ulong id)...

    public async Task UseAbility(SocketCommandContext context, Ability ab, Player target)...

    private void AssignGoalAndAbilities()...

}
```

Ici nous avons choisi de mettre en avant la fonction AssignGoalAndAbilities(), on peut voir que la Classe Ability() permet une personnalisation des compétences et objectifs de manière séparée.

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

```
private void AssignGoalAndAbilities()
{
    Console.WriteLine("Entrée AssignGoalAndAbilities() ");
    Ability ability  = new Ability(Ability.ID.PrepareSurvival, Ability.UsageType.Single, this);
    switch (_type)
    {
        case CharacterType.CaraLoft:
            goal = Goal.FindTalismans;
            ability = new Ability(Ability.ID.PrepareSurvival, Ability.UsageType.Single,this);
            break;
        case CharacterType.FBI_Inspector:
            goal = Goal.Kill.SerialKiller;
            ability = new Ability(Ability.ID.ShowFBICard, Ability.UsageType.Single,this);
            break;
        case CharacterType.MayaSpirit:
             ability = new Ability(Ability.ID.ActivateTrap, Ability.UsageType.ForEachRoom);
            goal = Goal.Protect.Maya;
            break;
        case CharacterType.MayaSucessor:
            ability = new Ability(Ability.ID.ActivateTrap, Ability.UsageType.Single);
            goal = Goal.Protect.Talismans;
            break;
        case CharacterType.NaziSoldier:
            ability = new Ability(Ability.ID.SecretConversation, Ability.UsageType.Single);
            goal = Goal.Kill.CaraLoft;
            break;
        case CharacterType.SerialKiller:
            ability = new Ability(Ability.ID.PushInTrap, Ability.UsageType.ForEachRoom);
            goal = Goal.Kill.Tourist;
            break;
        case CharacterType.Tourist:
            ability = new Ability(Ability.ID.RevealTrap, Ability.UsageType.Single,this);
            goal = Goal.ExitTemple;
            break;
        default:
            Console.WriteLine("Cas non pris en charge lors de l'attribution des compétences pour la création du joueur : " + user.Username);
            break;
```

GILLIOEN François
TORRADO Yoann
MANEZ Florian
MARTIN JEREMY

Difficultés rencontrées

Nous avons réussi à finaliser notre projet mais l'avancé de celui-ci a parfois été freinée par des difficultés. Certaines de ses difficultés résultaient parfois du manque de documentation concernant la programmation sur Discord. En effet les fonctions de base de Discord sont très rigides ce qui nous a cloisonné dans certaines parties de notre projet.

De plus le peu de documentation que l'on a trouvé sur Internet concernant la programmation sur Discord était parfois rendu obsolète par de nouvelles mises à jour, nous devions donc être très attentif à nos sources et à leur date de publication.

Une autre subtilité que l'on a découvert à travers ce projet est le fait que l'on devait héberger le BOT chacun sur son propre appareil pour pouvoir le tester. Cela peut paraitre trivial mais cela a posé certains soucis notamment pour la déconnexion des canals vocaux qui pouvait prendre plusieurs minutes (pour le blind test cela rendait plus complique l'enchainement de musique).

La dernière difficulté nous l'avons subi sans pouvoir la résoudre car il s'agit du Wi'fi de l'EFREI qui bloquant l'application Discord sur son réseau empechait tout test à l'EFREI sans l'utilisation d'un VPN ou de sa 4G personnel. Il serait agréable pour des projets futurs de pouvoir debrider la connexion pour certains appareils et ainsi nous éviter de devoir passer par un VPN.