UNIVERSITÉ DE STRASBOURG

LOUP GAROU

Professeurs référents :
M.CATELOIN Stéphane

Spécification
Equipe 5A

Table des matières

Chapitre 1

Spécification De l'Algorithme du Jeu

1.1 Diagramme de Classes

Ci-joint, Vous trouverez le diagramme de classes qui représente les différentes classes du jeu. Il convient de noter que le diagramme de classes est un outil de modélisation qui permet de visualiser les relations entre les classes et de déterminer les responsabilités et les collaborations entre elles.

1.2 Spécification des fonctions

— Cette fonction a pour objectif de mettre fin à la participation d'un joueur en modifiant son état de vie dans la structutre Joueur en "False".

```
void Kill()
```

— Cette fonction intermédiaire est invoquée en cas où le joueur en question était en relation amoureuse. Elle reçoit en tant que paramètre le partenaire amoureux en question.

```
Joueur Amoureux_Tuer_Joueur(Joueur j)
```

— Cette fonction a pour finalité de décimer un joueur et son amoureux. Elle accomplira également une décrémentation du nombre de personnages en fonction de leur rôle.Cette fonction se sert également des fonctions précédentes.

```
void Tuer_Joueur (Joueur j)
```

— Cette fonction a pour but d'endormir un joueur en modifiant son boolean "est_reveille" à "False", ce qui entraînera une modification de son interface graphique.

```
void endort_joueur(Joueur j)
```

— Cette fonction a pour but de réveiller un joueur en modifiant son boolean "est_reveille" à "True", ce qui entraînera une modification de son interface graphique.

```
void reveil_joueur(Joueur j)
```

— Cette fonction a pour but d'éveiller un joueur afin qu'il puisse accomplir son action déterminée. Elle invoque la fonction précédente et appelle l'action spécifique associée au rôle du joueur en question en appelant la méthode "action()" de la classe "Role".

```
void faire_action(Joueur j)
```

— Cette fonction se charge d'exécuter l'action propre à un rôle donné

```
void action(List<Joueur> listJoueurs)
```

— Cette fonction a pour but de définir un Timer.

```
void set_timer (int seconde)
```

— Cette fonction est conçue pour définir un minuteur et pour observer les résultats produits par la fonction "Vote" pour les Loups - Garous.

```
void action_loup(List<Joueur> listJoueurs)
```

— Cette fonction est conçue pour déclencher un temporisateur et interroger la sorcière quant à son souhait d'utiliser une potion et sur quelle cible. Elle s'assure également que la sorcière dispose toujours de la potion en question.

```
void action_sorciere(List<Joueur> listJoueurs)
```

— "Cette fonction concerne l'utilisation de la potion de résurrection par la sorcière.

```
void potion_vie (Joueur j)
```

— Cette fonction concerne l'emploi de la potion de mortalité de la sorcière.

```
void potion_mort (Joueur j)
```

— Cette fonction définit un chronomètre et requiert l'identification du joueur que la sorcière souhaite scruter.

```
void action_voyante(List<Joueur> listJoueurs)
```

— Cette fonction a pour but de déclencher un temporisateur et de requérir le Cupidon de sélectionner deux joueurs à unir

```
void action_cupidon(List<Joueur> listJoueurs)
```

— Cette fonction a pour but de notifier les deux joueurs choisit par Cupidon qu'ils forment un couple.

```
void annonce_couple(Joueur j1, Joueur j2)
```

— Cette fonction se charge de collecter les communications envoyées par les participants et de les interpréter pour déterminer la cible. Elle établit également un temporisateur et implémente une boucle permettant aux joueurs de modifier leur choix de vote.

```
void Vote()
```

— Cette fonction a pour finalité de déclencher l'algorithme opérant le déroulement du jeu.

```
void Strat()
```

— Cette fonction se charge de vérifier les critères de fin du jeu en comparant les effectifs de villageois et de loups afin de déterminer si les conditions pour une victoire sont réunies.

```
void Check_win(Partie p)
```

Chapitre 2

Spécification Réseau

2.1 Structure d'échange de données

La structure de données "Answer" a été conçue pour permettre une différenciation des messages normaux et des erreurs. La variable booléenne "error" est utilisée pour signaler la survenance d'une erreur, tandis que la variable "errType" permet de classer différents types d'erreurs. En cas de communication réussie, le code correspondant au type de données transmises est enregistré dans la variable "code", et les données à traiter sont rangées dans la variable "message".

```
Struct Answer
{
    bool error;
    int errType;
    int code;
    byte[1024] message;
}
```

2.2 Côté Serveur

— Cette fonction a pour but de vérifier la conformité de la composition d'une chaîne de caractères en examinant sa présence de caractères valides, à savoir les lettres et les chiffres. En retour, elle renvoie une valeur booléenne indiquant si la chaîne est valide (1) ou non (0).

```
bool verifie_composition(string str)
```

2.2.1 Les fonctions qui seront employées précéderont l'inclusion d'un joueur dans une partie.

— Cette procédure se charge de configurer la socket du serveur en définissant le port d'écoute ainsi que le nombre maximal de clients admissibles. Elle retourne la socket configurée pour une utilisation ultérieure.

```
socket SetupSocket(int port)
```

— Cette fonction a pour but de gérer les connexions des clients en les associant à un identifiant spécifié en tant que paramètre. Elle retournera une valeur "0" pour confirmer la connexion réussie d'un client, ou un code d'erreur en cas de non-conformité.

```
int AcceptConnexions(Socket[] clients)
```

— Cette fonction offre la capacité d'envoyer un message à un client. Le message est transmis en tant que tableau d'octets en tant que paramètre. Elle renverra un code de retour "0" pour confirmer l'envoi avec succès, ou un code d'erreur en cas de non-réussite.

```
int SendMessageClient(Socket client,byte[] message)
```

— Cette fonction a pour mission de transmettre au client toutes les informations associées à une partie à laquelle il a participé, notamment le nom de la partie et la liste des joueurs avec leurs pseudonymes. Elle renverra un retour de valeur de 0 pour confirmer la transmission réussie, ou un code d'erreur en cas de difficulté.

```
int SendGameInfo(Socket Client,string name,
            int [] idPlayers,string [] playerNames)
```

— Cette fonction a pour tâche de transmettre l'identifiant unique à un utilisateur qui a réussi sa connexion, ainsi que son pseudonyme. Elle renverra la valeur "0" pour confirmer la transmission avec succès, ou un code d'erreur en cas d'échec.

```
int SendAccountInfo(Socket server,bool answer,
            string username,int id)
```

— Cette fonction a pour but de transmettre à l'utilisateur client la liste complète des parties disponibles. Elle retournera une valeur de 0 pour confirmer la transmission avec succès, ou un code d'erreur en cas d'échec.

```
int SendCurrentGame(Socket server,String [] games ,
            int[] nb_players)
```

— Cette fonction offre la possibilité de transmettre un message à l'ensemble des clients figurant sur la liste spécifiée. Elle sera employée pour annoncer les modifications d'état, les messages envoyés par les utilisateurs d'un salon de discussion, pour notifier les votes et retournera soit un code "0" pour confirmer l'envoi, soit un code d'erreur en cas d'échec.

```
int SendMessageClients(Socket[] clients, byte[] message)
```

2.2.2 Les fonctions qui seront employées pour établir une communication avec le serveur secondaire qui abrite la base de données

— Cette fonction est invoquée lorsqu'un nouveau joueur se connecte. Elle émet une requête de connexion à la base de données, et en fonction de la réponse reçue, elle peut soit confirmer la connexion du joueur au serveur en transmettant ses données correspondantes, soit refuser la connexion. Un retour de 0 indique que la requête a été correctement transmise, tandis qu'un code d'erreur est renvoyé en cas de non-conformité.

```
int ConnexionAttempt(Socket bdd ,int queue,
                     string username,string password)
```

— Cette fonction est activée lors de la nouvelle inscription d'un joueur. Elle envoie une demande d'inscription à la base de données. En fonction de la réponse obtenue, la connexion au serveur sera soit confirmée, soit refusée. Elle renvoie un indicateur de réussite (valeur 0) pour confirmé l'envoi, ou un code d'erreur en cas de non-conformité.

```
int InscriptionAttempt(Socket bdd , string pseudo,
                     string password,string email)
```

— Cette fonction est appelée à chaque fois qu'un client est ajouté à la liste des parties disponibles.

```
int GetCurrentLobbies(Socket bdd)
```

— Cette fonction a pour fonction de requérir les informations des utilisateurs en se basant sur l'identifiant transmis en tant que paramètre. Elle sera mise en œuvre dans la salle d'attente. Elle retournera la valeur "0" pour confirmer l'envoi, ou un code d'erreur en cas d'échec.

```
int GetClientsInfo(Socket bdd ,int[] id)
```

— Cette fonction envoie une requête au serveur de base de données (BDD) afin de procéder à la mise à jour des informations concernant un joueur connu par son identifiant donné en paramètre. Elle retourne une valeur de 0 en cas de confirmation de l'envoi réussi, ou un code d'erreur en cas de non-réussite.

```
int UpdateAccountData(Socket bdd ,int idUser,
                      string username,string password,string email)
```

2.2.3 Les fonctions utiles qui seront mises en œuvre lorsque le serveur recevra un message en provenance d'un client.

— Cette fonction a pour but de recevoir un message envoyé par le client et de retourner une réponse structurée sous forme de "answer". Elle inclura le type d'information ainsi que le message en question, ou bien spécifiera une erreur accompagnée de son code respectif.

```
Answer RecvMessage(Socket client)
```

— Cette fonction constitue l'implémentation algorithmique du jeu du Loup-Garou. Elle est exécutée en parallèle à travers un thread, ce qui permet de relier tous les clients concernés à une instance unique via un nouveau socket. Par la suite, elle interagit directement avec l'algorithme en fonction des messages reçus.

```
void GameInstance()
```

2.2.4 Les fonctions utiles qui seront mises à disposition pour gérer les transmissions d'informations aux joueurs par le biais du thread qui administre la partie.

— Cette fonction a pour fonctionnalité d'envoyer un message émis par l'un des joueurs à tous les autres participants de la partie. Elle retournera une valeur de 0 en confirmation de l'envoi réussi, ou un code d'erreur en cas de problème.

```
int SendChatMessage(Socket [] client , string message)
```

— Cette fonction a pour but de transmettre à un joueur en particulier l'information relative à un vote. Elle retournera la valeur 0 pour confirmer la réussite de l'envoi, ou un code d'erreur en cas de problème technique.

```
int SendVote(Socket [] clients, int id)
```

— Cette fonction a pour but de transmettre à l'ensemble des clients les statistiques de la partie qui vient de se terminer. Elles incluent les rôles attribués à chaque joueur ainsi que les joueurs désignés comme gagnants. La fonction renvoie la valeur 0 en cas de transmission réussie, ou un code d'erreur en cas de problème.

```
int SendEndStats(Socket [] clients, int [] idjoueur ,
            int [] roles,int winner)
```

2.3 Côté Serveur hébergeant la Base de donnée

— Le socket est configuré sur le serveur pour garantir la liaison entre le serveur 1, qui héberge l'algorithme du jeu, et le serveur 2, qui stocke la base de données.

```
Socket SetupSocket()
```

— Cette fonction est conçue pour recevoir des informations à partir du serveur 2, afin d'interroger la base de données correspondante

```
byte[] RecvMessage(Socket serveur)
```

— Ce processus envoie au serveur les données relatives à la demande de connexion du client, ainsi que sa position dans la file d'attente des demandes de connexion. Il convient de prendre en compte que des problèmes de synchronisation peuvent survenir en cas de multiples connexions simultanées.

```
int SendPlayerInfo(Socket serveur,int queue,bool answer,
            int idPlayer,string username)
```

— La fonction envoie des informations au serveur 2 pour interroger la base de données

```
int SendExistingWaitingRooms(Socket serveur,int [] idRoom,
            string [] name)
```

— Cette fonction a pour tâche de lire la structure fournie en tant que paramètre. En se basant sur le contenu initial de celle-ci, elle accomplit les actions appropriées, telles que la récupération des parties disponibles ou des données d'un utilisateur. Par la suite, elle renvoie le message destiné à être transmis au serveur.

```
byte[] interpreterMessageDB(answer message)
```

— Cette fonction analyse le message fourni en tant que paramètre. En se basant sur le contenu initial contenu dans ce message, elle interprète les demandes nécessaires pour le bon fonctionnement de l'algorithme et effectue

les requêtes appropriées auprès de la base de données. Enfin, elle retourne le message à transmettre au serveur.

```
Answer interpreterMessageServer(byte[] message)
```

2.4 Côté Client

— Cette opération consiste à transmettre les informations d'identification, à savoir le nom d'utilisateur et le mot de passe, au serveur. Le serveur, quant à lui, s'occupera de vérifier la présence du compte correspondant dans la base de données. Ensuite, nous attendrons une réponse du serveur afin de confirmer ou non la réussite de la connexion.

```
int Connect(Socket server,string username,string password)
```

— Cette fonction a pour but de déconnecter le compte du client en envoyant une requête de fermeture de socket client au serveur

```
int Disconnect(Socket server)
```

— Cette procédure envoie les informations d'identification telles que le nom d'utilisateur, le mot de passe et l'adresse électronique transmis en tant que paramètres au serveur. Le serveur émet alors une requête d'inscription à la base de données. Ensuite, il est nécessaire d'attendre une réponse du serveur pour confirmer ou refuser l'inscription.

```
int Inscription(Socket server,string username,
        string password,string email)
```

— Cette fonction se consacre à établir une connexion au serveur.

```
socket LinkToServer()
```

— Cette fonction permet de générer une requête d'expulsion, ce qui donne aux joueurs la possibilité de voter pour l'expulsion d'un joueur. Elle transmet la demande au serveur et autorise les autres utilisateurs à voter à leur tour. Il est à noter que le client qui a initié la requête votera automatiquement pour l'expulsion du joueur concerné. Si un joueur tente de se faire exclure lui-même, une erreur sera retournée.

```
int QueryKickVote(Socket server,int idPlayer)
```

— Cette fonction permet aux joueurs de participer au vote en cas de requête de renvoi. Chaque joueur peut exprimer son choix en votant, en utilisant l'identifiant du client (idClient) pour représenter celui qui émet le vote.

```
int int KickVote(Socket server,int idClient)
```

— Cette fonction a pour mission de transmettre les messages saisis dans l'interface de communication au serveur, afin qu'il puisse les confirmer et les diffuser à tous les clients connectés.

```
int SendMessageToServer(Socket server,byte[] message)
```

— Cette fonction envoie au serveur une requête de vote. Le paramètre "idUser" représente l'utilisateur qui effectue le vote, tandis que le paramètre "idVote" désigne la personne cible de l'action.

```
int Voter(Socket server,int idUser , int idVote)
```

— Cette fonction a pour tâche de recevoir un message et de le convertir en une structure "Answer".

```
Answer RecvMessage(Socket server)
```

Chapitre 3

Spécification de la Base de donnée

3.1 Modèle entite-association Modèle relationnel

Ce document est accompagné de deux modèles qui représentent la structure de la base de données. Le premier est un modèle d'entité-association, qui représente les relations entre les différentes entités et les associations entre elles. Le second est un modèle relationnel, qui représente les données sous forme de tables avec des relations définies entre elles. Les deux modèles sont importants pour comprendre la structure de la base de données et comment les différentes informations y sont stockées et reliées les unes aux autres.

3.2 Spécification des tables de la base de donnée

3.2.1 Table Utilisateurs

Cette entité enregistre tous les joueurs inscrits au jeu avec leurs informations personnelles : leurs pseudos, leurs mots de passes et le nom d'utilisateur qui leur serviront pour la connexion.
Les contraintes de la table sont :

- "idUsers" est une clé primaire.
- "nom_utilisateur" est UNIQUE et ne peut pas être vide.
- "pseudo" est UNIQUE et ne peut pas être vide.
- "motdepasse" ne peut pas être vide.
- La longeur de "motdepasse" doit être plus grand que 8 et "motdepasse" doit contenir au moins une lettre minuscule, une lettre majuscule, un nombre et un charactère spécial parmi (!, @, , $, %, ; , *).
- La longeur minimale de "nom_utilis" est 5, maximale est 50. "nom_utilis" doit contenir seulement les chiffres, les nombres et le charactere '_'.

— La longeur minimale de "pseudo" est 5, maximale est 50. "pseudo" doit contenir seulement les chiffres, les nombres et le charactere '_'.

3.2.2 Table Partie

Cette table représente l'ensemble de partie du jeu qui sont créée par un utilisateur. Elle est caractérisé par l'identifiant (la clé primaire), le nom de la partie (l'utilisateur est libre de choisir le nom qu'il veut) , les étapes et les actions réalisées au cours du jeu.

— "id_utilis" et "id_partie" sont les clés primaires.
— "id_utilis" est une clé étrangère qui réference la table "Utilisateur".
— "id_utilis" et "id_partie" ne peuvent pas être vides.

3.2.3 Table SauvegardePartie

Cette table est nécessaire quand un utilisateur décide de sauvegarder la partie du jeu pour pouvoir rejouer
Les contraintes de la table sont :

— "id_partie" est une clé primaire.
— "id_partie" est aussi une clé étrangere qui réference la table "parties"

3.2.4 Table Statistiques

Elle contient des informations nécessaires pour visualiser les statistiques du joueur comme son score, son nombre de parties joué et les victoires obtenues au cours d'une partie.

— "id_utilis" est une clé primaire.
— "id_utilis" est une clé étrangere qui réference la table "utilisateur"
— "score" ne peut pas être vide et a une valeur par défaut : 0
— "nb_joues" ne peut pas être vide et a une valeur par défaut : 0
— "nb_victoire" ne peut pas être vide et a une valeur par défaut : 0

3.2.5 Table Amis

Cette table regroupe un utilisateur et leurs amis. Après la demande d'ajout si elle est acceptée celui-ci est ajouté dans la table.
Les contraintes de la table sont :

— "id_utilis1" et "id_utilis2" sont les clés primaires.
— "id_utilis1" et "id_utilis2" sont aussi les clés étrangères qui réferencent la table "utilisateur".
— "id_utilis1" et "id_utilis2" ne peuvent pas être vides.