Projet Logiciel Transversal - Aide mémoire

Ensea - Option IS - Philippe-Henri Gosselin

Table des matières

1 Installation système

1.1 Système d'exploitation

Le suivi des projets a été préparé sous **Ubuntu 16.04 64-bit**. Tout autre environnement vous privera de garantie de support de la part des encadrants. Pour télécharger Ubuntu 16.04 : `http://releases.ubuntu.com/16.04/`. Choisissez la dernière version "desktop" 64-bit (AMD64).

Vous aurez besoin des paquets suivants (ici pour Ubuntu 16.04, pour les autres distributions, les noms peuvent varier) :

```
sudo apt-get install -y git cmake g++ tar xz-utils gzip bzip2 zip unzip dia
sudo apt-get install -y libpthread-stubs0-dev libsfml-dev libxml2-dev docker.io
sudo apt-get install -y libmicrohttpd-dev
```

1.2 Machine virtuelle

Il est possible de travailler dans une machine virtuelle si vous n'êtes pas sous Linux. Seul VirtualBox donnera lieu à une garantie de support de la part des encadrants. NB : Il n'est pas impératif de travailler sous Ubuntu 16.04 (cf Docker ci-après).

Vous pouvez télécharger virtualbox ici : `https://www.virtualbox.org/`

Créez une nouvelle machine dans virtualbox :
Page 1 du formulation de création : Nom : plt (ou autre), Type : Linux, Version Ubuntu 16.04 64-bit.
Page 2 du formulation de création : Mémoire : 2Go.
Page 3 du formulation de création : 'Créer un disque dur virtuel maintenant'.
Page 4 du formulation de création : Type de fichier disque dur : VMDK.
Page 5 du formulation de création : Allocation : Dynamiquement alloué.
Page 6 du formulation de création : Taille du fichier disque : 10Go.

Une fois la machine virtuelle créée, démarrez la. Une fenêtre s'affiche, choisissez le fichier iso d'Ubuntu 16.04 que vous avez téléchargé. Suivez la procédure d'installation.

Optionnel : installation des pilotes additionnel. Une fois la machine lancée, choisissez dans le menu périphérique[1] : "Insérer l'image CD des Additions invité". Un terminal dans la machine virtuelle va s'ouvrir, entrez le mot de passe que vous avez choisi lors de l'installation. Une fois la procédure terminée, redémarrez la machine virtuelle.

1. de la fenêtre qui contient la machine virtuelle

2 Dépôt versionné (à effectuer sur une seule machine)

2.1 Installation kit de démarrage

Créez un dossier pour votre projet, clonez le kit de démarrage :

```
git clone https://github.com/philippehenri-gosselin/plt.git
```

Vous pouvez tester que cela fonctionne en exécutant `make` dans le dossier de votre projet (doit configurer, compiler, puis afficher à la fin "It works !").

2.2 Création clef ssh pour le projet

Créez une clef SSH spécifique pour le projet (à faire sur une seule machine) :

```
ssh-keygen -t rsa -C "<votre adresse email>" -N "" -f ~/.ssh/<nom groupe>
```

Pour `<nom groupe>`, collez vos noms de famille en utilisant que des minuscules. Vous pouvez choisir des versions plus courtes, l'important est d'avoir un id permettant aux encadrants de facilement faire le lien entre le projet et les personnes associées.

Envoyez par mail à tous les membres du projet, ainsi qu'à `gosselin@ensea.fr` les deux fichiers créés (`~/.ssh/<nom groupe>` et `~/.ssh/<nom groupe>.pub`).

2.3 (Cas Github) Création dépôt git distant pour le projet

Rendez vous sur `https://github.com/`. Créez un compte avec le nom de votre choix, puis créez un nouveau projet dans ce compte avec pour nom `<nom groupe>`.

Envoyez par mail l'url ssh du dépôt (de type `git@github.com:<compte git>/<nom groupe>.git`) à tous les membres du projet ainsi qu'à `gosselin@ensea.fr`.

Dans l'interface github, rendez vous dans les options (icône en haut à droite), menu "Settings", puis onglet à gauche "SSH and GPG keys". Cliquez sur "New SSH key", Title : ce que vous voulez, Key : contenu du fichier `~/.ssh/<nom groupe>.pub`.

2.4 Versionner dépôt local

Initialisez git localement (spécifique pour chaque machine) :

```
git config --global user.email "<email propriétaire machine actuelle>"
git config --global user.name "<nom propriétaire machine actuelle>"
```

Initialisez le dépôt local sur une machine :

```
git init
git add .gitignore CMakeLists.txt Makefile docker src
git commit -m "init"
git remote add origin <url dépôt distant>
git push -u origin master
```

Pour `<url dépôt distant>`, dans le cas github, vous pouvez la retrouver dans l'interface web, bouton "clone or download" (choisissez bien ssh), copiez-collez l'url affichée.

3 Dépôt versionné (à effectuer sur les autres machines)

3.1 Récupération clef ssh pour le projet

Copiez les fichiers de clefs ssh reçues par mail dans `~/.ssh`. Modifiez les droits :

```
cd
mkdir -p .ssh
chmod 700 .ssh
cd .ssh
chmod 600 <nom groupe>
chmod 644 <nom groupe>.pub
```

3.2 Clone local du dépôt distant

Initialisez git localement (spécifique pour chaque machine) :

```
git config --global user.email "<email propriétaire machine actuelle>"
git config --global user.name "<nom propriétaire machine actuelle>"
```

Initialisez le dépôt local sur les autres machines (pensez à copier les clefs ssh) :

```
cd <dossier de votre choix>
export GIT_SSH_COMMAND='ssh -i ~/.ssh/<nom groupe>'
git clone <url dépôt distant>
```

4 Logiciels de développement

4.1 CMake

4.1.1 Linux

Installez le package git.

4.1.2 Windows

Téléchargez et exécutez l'installateur sur `https://git-for-windows.github.io`.

Exécutez l'installeur, et choisissez l'option 'Add CMake to the system path' (pour tous les utilisateur ou uniquement celui-ci, à votre convenance).

Pour l'utiliser, lancer l'application "Git Bash".

4.2 Compilateurs C++

4.2.1 Linux

Installez le package g++.

4.2.2 Windows

Téléchargez Mingw à l'adresse suivante :
`https://sourceforge.net/projects/mingw/files/Installer/mingw-get-setup.exe/download`.

Exécutez le fichier téléchargé `mingw-get-setup.exe`.

Dans le choix des packages, prenez tout sauf gcc-ada, gcc-fortran et gcc-objc. Puis, dans le menu Installation, cliquez sur "Apply Changes".

Cliquez droit sur Ordinateur dans le menu démarrer, puis Paramètres Systèmes Avancés dans la liste à gauche, et enfin le bouton Variables d'environnement. Dans la liste des variables systèmes, étendez la variable PATH avec "`;C:\MinGW\bin`" (remplacez les chemins s'ils ne sont pas corrects). Pour vérifier, ouvrez une invite de commandes, tapez gcc et g++ : les commandes doivent être reconnues.

4.3 CMake

4.3.1 Linux

Installez le package cmake.

4.3.2 Windows

Téléchargez la version compatible avec votre installation de windows (téléchargez l'installeur .msi) : `https://cmake.org/download`.

Exécutez l'installeur, et choisissez l'option 'Add CMake to the system path' (pour tous les utilisateur ou uniquement celui-ci, à votre convenance).

4.4 SFML

Installez CMake et le compilateur C++ (cf sections précédentes).

4.4.1 Linux

Installez le package libsfml-dev.

4.4.2 Windows

Téléchargez les sources de SFML version 2.3.2 : `http://www.sfml-dev.org/files/SFML-2.3.2-sources.zip`.
Décompressez l'archive dans le dossier de votre choix.
Ouvrez une invite de commande, rendez vous ce dossier, puis exécutez les commandes suivantes :

```
cmake -G "MinGW Makefiles" -DCMAKE_BUILD_TYPE=RELEASE .
mingw32-make
```

Pour tous les programmes qui utiliseront SFML, pensez bien à mettre les .dll dans le même dossier que celui de votre exécutable.

4.5 Netbeans

Installez CMake et le compilateur C++ (cf sections précédentes).

4.5.1 Linux

Téléchargez netbeans 64-bit pour C++ à l'adresse suivante : `https://netbeans.org/downloads/`.
Dans le dossier où a été téléchargé le fichier, exécutez une commande du type (ici pour la version 8.1) :

```
sudo sh netbeans-8.1-cpp-linux-x64.sh
```

Netbeans sera alors disponible via le lanceur (icône Ubuntu en haut à gauche de l'écran), ainsi qu'en ligne de commande en tapant 'netbeans'. Vous pouvez supprimer le fichier que vous avez téléchargé.

4.5.2 Windows

Téléchargez la version de Netbeans pour C/C++ compatible avec votre installation de Windows : `https://netbeans.org/downloads`. Exécutez l'installeur.

Lancez Netbeans, allez dans le menu Tools/Options, puis onglet C/C++ : netbeans va chercher les compilateurs, et doit normalement les trouver seul. Si ce n'est pas le cas, il faut indiquer tous les éléments.

4.5.3 Linux et Windows

Pour créer un projet avec netbeans, suivez les étapes suivantes :
Page 1 formulaire : Créez un nouveau projet de catégorie C/C++, projet avec sources existantes.
Page 2 formulaire : Choisissez le dossier de votre projet ; Sélectionnez le mode de configuration "custom".
Page 3 formulaire : "Run in folder" : choisissez le dossier "build" de votre projet (il faudra peut être le créer).
Page 4 formulaire : (suivant).
Page 5 formulaire : Assurez vous que l'unique dossier source est le dossier "src" de votre projet (supprimer ceux proposés, ajoutez celui-ci).
Page 6 formulaire : (suivant).
Page 7 formulaire : (finir).

Si le C++11 ne semble pas être reconnu : dans les propriétés du projet (bouton droit souris sur le projet), section "Code Assistance/C++ compiler", paramètre "C++ Standard" : choisissez "C++11".

Si vous souhaitez toujours compiler avant d'exécuter : propriétés projet / section "Run" / paramètre "Build first" : cochez la case.

Lors de la première exécution, choisissez le binaire "run" dans le dossier "bin" du projet.

4.6 Tiled Map Editor

Ce logiciel vous permet de facilement créer des mondes en grille avec cellules de forme carré, losange et hexagonale.

4.6.1 Linux

```
sudo add-apt-repository ppa:mapeditor.org/tiled
sudo apt-get install tiled
```

4.6.2 Windows

Suivez les instructions à `http://www.mapeditor.org/download.html`

4.7 GProf (pour profiler le code

Disponible que sous Linux.

Ajoutez l'option suivante pour la compilation : -pg.

Par exemple, dans le CMakeLists.txt du kit de démarrage, vous pouvez l'ajouter dans la section suivante :

```
IF(CMAKE_COMPILER_IS_GNUCC)
SET(CMAKE_C_FLAGS   "${CMAKE_C_FLAGS}   -Wall -std=c11   -pthread -g -pg")
SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -std=c++11 -pthread -g -pg")
ENDIF(CMAKE_COMPILER_IS_GNUCC)
```

5 Git

Quelques commandes utiles pour git (a exécuter dans le dépôt local :

`git status` → Affiche l'état actuel du dépôt. Toujours exécuter cette commande avant tout autre !

`git add <fichier/dossier>` → Ajoute le fichier ou le dossier (ainsi que les fichiers qu'il contient) à l'index. Les éléments dans l'index seront ajoutés au prochain commit.

`git commit -m "<message>"` → Forme un nouveau commit avec un message (obligatoire). Le message vous permet de décrire sommairement la nature de vos opérations.

`git pull` → Récupère tous les commits du dépôt distant. Possible que si votre dépôt local est "propre", ie il n'y a aucun candidat au commit.

`git push` → Envoie tous les nouveaux commits vers le dépôt distant. **Toujours faire un git pull avant**.

`git tag [-f] <tag>` → Permet de tagger le dernier commit. L'option -f permet de modifier un tag existant.

`git push [-f] --tags` → Envoie les tags au dépôt distant. L'option -f permet de modifier les tags existants.

`git pull --tags` → Récupère les tags du dépôt distant.

6 Docker

6.1 A propos de docker

Docker permet de travailler dans un environnement très précis, sans avoir à procéder à l'installation ou à l'émulation d'un système complet. Il est disponible pour la plupart des systèmes d'exploitation récent. Docker est utilisé par les encadrants pour tester les projets : à vous de reproduire ces tests afin d'assurer la validité des livrables.

6.1.1 Container

Docker repose sur la notion de container et d'images. Un container est un état particulier d'un système. Chaque commande exécutée donne lieu à un nouveau container.

Commandes utiles :

`sudo docker ps` → Afficher la liste des containers actuels.

`sudo docker rm <container id>` → Supprime un container.

`sudo docker stop $(docker ps -a -q)` → Arrête tous les containers.

`sudo docker rm $(docker ps -a -q)` → Supprimer tous les containers.

6.1.2 Image

Une image est un container statique. Les images ont un nom/id humainement compréhensible. Il existe de nombreuses images automatiquement téléchargées par docker. Tous les containers créés pour fabriquer l'images sont attachés à celle-ci.

Commandes utiles :

`sudo docker build -t plt-initial -f docker/plt-initial .` → Fabrique à partir du fichier docker "docker/plt-initial" une image avec le nom 'plt-initial'. Les containers à partir de cette image auront accès aux données dans le dossier courant de l'hôte.

`sudo docker images` → Affiche la liste des images actuelles.

`sudo docker run -ti plt-initial /bin/bash` → Ouvre une console à partir de l'image 'plt-initial'.

`sudo docker rmi plt-initial` → Supprime l'image nommée "plt-initial" ainsi que les containers attachés.

6.1.3 Problèmes courants

Problème de DNS (ie, mauvais DNS dans le container) : Par défaut, docker utilise ceux de Google (8.8.8.8 et 8.8.4.4) et ignore ceux de votre environnement. Si ceux-ci sont inaccessibles, dans le fichier '/etc/default/docker', changez les url des dns dans le ligne : `DOCKER_OPTS="--dns 8.8.8.8 --dns 8.8.4.4"`

7 Définition des API Web

Pour exprimer chaque service dans votre rapport, les informations suivantes sont demandées :

1. Requête :
   (a) Méthode HTTP : GET, POST, PUT ou DELETE
   (b) URI sans le nom de domaine NB : cette URI peut avoir une partie paramètre, que l'on représentera entre chevrons <>
   (c) Le cas échéant : des données au format JSON. Exprimez le format de ces données avec un schéma de validation JSON.
2. Pour chaque réponse possible :
   (a) Status réponse HTTP : Codes 10x, 20x, 30x, 40x, 50x, ...
   (b) Le cas échéant : des données au format JSON. Exprimez le format de ces données avec un schéma de validation JSON.

Exemple de description pour le service "obtenir le livre d'identifiant id" :

1. Requête : GET /livre/<id>. Pas de données.
2. Réponses :
   — Cas où le livre <id> existe :
      (a) Status 200 (=OK)
      (b) Données :

```
type: "object",
properties: {
  "key": { type:string, pattern: "([a-zA-Z])+([0-9]){2}" },
  "lang": { type:string, pattern: "(fr|en)" },
  "title": { type:string, minLength:1 },
  "author": { type:string, minLength:1 },
  "year": { type:number, minimum:1900, maximum:2100 },
  "url": { type:string, pattern: "http://(www.)?(([a-zA-Z0-9-]){2,}\\.){1,4}([a-zA-Z]){2,6}" },
},
required: [ "key", "lang", "title", "author", "year" ]
```

   — Cas où le livre <id> n'existe pas : Status 404 (=Non trouvé). Pas de données.

8 Format dépôt et validation

8.1 Définition format dépôt

La racine de votre dépôt doit contenir les éléments suivants :
— Fichier Rapport.pdf : votre rapport, conforme au livrable e cours
— Fichier Makefile : le fichier de build principal. Les cibles suivantes sont attendues :
all : clean tout, configure tout, et recompile tout
configure : configure votre projet
extern : configure et compile les dépendances externes
build : compile tous vos exécutables (client, server, ...)
clean : supprime les fichiers compilés, ainsi que les headers générées, relatifs à votre projet
distclean : supprime tous ce qui est généré, y compris les dépendances externes
— Dossier src : les sources de vos programme
— Dossier rapport : contient les sources de votre rapport (.doc, .docx, .odt, .tex, images, ...).
— Dossier res : contient les ressources de votre projet (textures, sons, etc.)
La racine de votre dépôt peut contenir les éléments suivants :
— Dossier lib : contient des librairies précompilées (.so, .dll, .lib) pour vos dépendances externes.
— Fichier .gitignore : indique à git quels fichiers à ignorer
— Dossier .git
— Fichier CMakesList.txt : si vous utilisez CMake

8.2 Procédures de validation

Si vous souhaitez faire une première pré-validation rapide, lancez tout simplement la commande `make` dans le dossier de votre projet. Celle-ci va tout nettoyer, tout reconfigurer, et tout recompiler. Puis, vous pouvez lancer votre exécutable en ligne de commande. Attention : n'utilisez pas cette commande pour travailler au quotidien, préférez celle générée par CMake (dans le dossier build), qui n'effectue que les opérations nécessaires pour la mise à jour de votre exécutable.

La procédure suivie par les encadrants du projet pour valider votre projet sont les suivantes :
— Ouvrir une session invité vierge sur une machine ;
— Cloner votre dépôt git ;
— Lancer la commande `make test` dans le dossier de votre projet. Cette fabrique une première image système initiale via docker, puis une deuxième image avec votre projet compilé, et enfin lance votre exécutable dans cette dernière image.

9 Définitions des Livrables

Les livrables sont incrémentaux : ils doivent contenir les nouveaux éléments qui les définissent, ainsi que l'ensemble des contenus des livrables précédents.

9.1 Livrable 1.1

Environnement de développement. Tous les outils liés au développement doivent être opérationnels au moment de ce livrable :
- Système d'exploitation dédié ou machine virtuelle, avec tous les paquets requis
- Logiciel de développement avancée : Netbeans, eclipse, Visual studio, CLion, ... Doit inclure un debugger opérationnel
- Dépôt git conforme au cahier des charges (cf section 8.1).

Ressources : images, textures, ... à placer dans le dossier 'res' du projet

Rapport section 1 : Présentation générale :
- Présentation rapide des principes et règles du jeu. Partir d'un jeu modèle ou archétype pour en faciliter la compréhension. NB : c'est une présentation rapide, non technique.
- Présentation rapide des ressources du jeu (images, textures, ...)

Rappel : les sources du rapport (.doc, .odt, .tex, ...) sont à placer dans le dossier 'rapport', et le rapport lui-même doit être placé sous le nom 'Rapport.pdf' dans la racine du projet.

Code : affichage message. La commande suivante :

```
./bin/client hello
```

doit afficher à la console un message, par exemple "Bonjour le monde !"

9.2 Livrable 1.final

Rapport section 2 : Description et Conception des états
- Description de tous les éléments qui peuvent composer un état du jeu. La description est exhaustive : toutes les possibilités d'éléments et leur paramètres possibles doivent être énumérés. Bien que très technique et très détaillée, cette description ne dépend pas d'un formalisme informatique et/ou d'un langage de programmation.
- Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : implantation et tests unitaires états. La commande suivante :

```
./bin/client state
```

lance une série de tests très simples qui vérifient le bon fonctionnement de votre implantation des états. Si le professeur encadrant introduit volontairement un bug dans votre implantation, cette procédure doit la détecter, et afficher une message d'erreur.

9.3 Livrable 2.1

Rapport section 3 : Description et Conception du Rendu
- Description de la stratégie de rendu d'un état. Vous devez expliquer comment vous allez rendre les différents états possible. Par exemple, si vous avez un "monde" décrit selon une grille 2D, vous pouvez utiliser un rendu par tuiles [2].
- Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : rendu d'un état La commande suivante :

2. Vous avez un exemple dans la documentation SFML : https ://www.sfml-dev.org/tutorials/2.3/graphics-vertex-array-fr.php.

```
./bin/client render
```

affiche à l'écran le rendu d'un état. Ce rendu doit être fonction des données contenu dans l'état. Celui-ci doit pouvoir être variable, par exemple :

- Le monde est généré aléatoirement. A chaque nouvel exécution, un monde différent est affiché ;
- Le monde est chargé depuis un fichier. Si le professeur modifie ce fichier, il doit voir le changement à l'écran (NB : n'oubliez pas d'expliquer le format du fichier dans votre rapport).
- Le monde est fabriqué à l'exécution par une suite d'appel C++. Si le professeur édite votre fichier source C++, il doit voir le changement à l'écran.

9.4 Livrable 2.2

Rapport section 4 : Règles de changement d'états et moteur de jeu

- Règles de changement d'états. Il faut présenter les événements qui peuvent faire passer d'un état à un autre. C'est une description entièrement en français, indépendante de tout formalisme et langage informatique. La description doit être exhaustive : on doit y trouver toutes les informations qui permettent d'implanter par la suite ces règles.
- Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : changement d'état La commande suivante :

```
./bin/client engine
```

affiche à l'écran une succession d'états différents. Introduisez manuellement des appels aux commandes de votre moteur de jeu. Pensez à laisser une petite pause entre chaque état, afin que l'on puisse voir ce qui se passe. Indiquez également dans la console quelles commandes sont exécutées. Enfin, faites des vérifications sur la nature des états attendus : si le professeur essaye d'introduire un bug dans vos implantations, ces tests doivent les détecter.

9.5 Livrable 2.final

Rapport section 5 : Intelligence Artificielle

- Présentation d'une stratégie aléatoire. Pour ce livrable, une stratégie aléatoire est demandée. Par exemple, à un état donné, vous listez toutes les commandes possibles. Puis, l'IA sélectionne aléatoirement l'une de ces commandes, et l'ajoute à la liste des commandes.
- Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : IA La commande suivante :

```
./bin/client random_ai
```

affiche à l'écran votre jeu joué par une IA aléatoire. Les choix de cette IA ne doivent pas nécessairement mener à une victoire ou quelque chose de cohérent. L'important est de voir le jeu fonctionner seul en suivant ses règles.

9.6 Livrable 3.1

Rapport section 5 : Intelligence Artificielle

- Présentation d'une stratégie basée sur des heuristiques. Pour ce livrable, la stratégie demandée doit être basée sur des règles simples, appelées heuristiques. Vous pouvez partir de l'IA précédente, à savoir lister toutes les commandes possibles. Puis, au lieu de sélectionner aléatoirement, vous effectuer un calcul de probabilité de réussite (ou score, ou toute chose dans cet esprit). Par exemple, dans un jeu de dames, les commandes qui mange un pion ont un meilleur score. Ces règles sont totalement arbitraires et ne garantissent pas la réussite de la partie ; elles doivent juste être bien meilleures que le hasard.

— Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : IA La commande suivante :

```
./bin/client heuristic_ai
```

affiche à l'écran votre jeu joué par une IA basée sur des heuristiques. Les choix de cette IA ne doivent pas nécessairement mener à une victoire, par contre on s'attend à quelque chose de cohérent.

9.7 Livrable 3.final

Rapport section 5 : Intelligence Artificielle

— Présentation d'une stratégie avancée. Pour ce livrable, la stratégie demandée doit être basée sur les arbres de recherche. Pour parvenir à implanter ce type de stratégie, deux pistes sont possibles :
  — Clonage d'état : pour pouvoir passer d'un état à un autre, puis en revenir, vous implanter un "Pattern Clone" sur vos classes d'états. Cela est plus facile mais très rapidement gourmand en mémoire et cpu.
  — Moteur de jeu avec rollback : vous implantez un "Pattern Command" avancé, où vous offrez la possibilité d'annuler des commandes. Cela est plus difficile mais peu consommateur de ressources.
— Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : IA La commande suivante :

```
./bin/client rollback
```

permet de tester votre système de retour en arrière. Par exemple, vous affichez votre jeu qui tournent automatiquement (par exemple, avec l'IA heuristique), puis au bout d'une minute, vous affichez votre jeu en sens inverse.

Code : IA La commande suivante :

```
./bin/client deep_ai
```

affiche à l'écran votre jeu joué par une IA avancée.

9.8 Livrable 4.1

Rapport section 6 : Modularisation

— Moteur de jeu dans un thread séparé. Votre moteur doit fonctionner de manière indépendante dans un autre thread que le thread principal. Suivant la nature de votre jeu, l'exécution des commandes peuvent avoir différentes origines : régulièrement (temps réel), provoqué par l'ajout d'une commande spéciale (ex : fin de tour), ou encore au delà d'un temps limite.
— Serialisation des commandes. Vous devez être en mesure d'encoder et de décoder vos commandes au format JSON. Cela va préparer le livrable suivant, où ces codecs nous permettrons de jouer en réseau.
— Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : Threads La commande suivante :

```
./bin/client thread
```

affiche le jeu joué par des IAs, avec un le moteur de jeu exécuté dans un thread séparé.

Code : Enregistrer La commande suivante :

```
./bin/server record
```

fait jouer les jeu par des IAs pendant un certain temps (environ une minute), et sauvegarde toutes les commandes dans le fichier "replay.txt". Notez bien que c'est en utilisant le binaire server, qui ne doit avoir aucun lien avec le code et les librairies de rendu.

Code : Rejouer La commande suivante :

```
./bin/client play
```

fait jouer les jeu par l'enregistrement stocké dans "replay.txt".

9.9 Livrable 4.2

Rapport section 6 : Modularisation

— API Web pour rassembler les joueurs. Proposez une API Web pour permettre de réunir différents joueurs sur différentes machines (ou process si vous n'avez qu'une machine). On suppose que cette opération est avant le début d'une partie. Puis, une fois tous les joueurs réunis, la partie peut démarrer. Une fois la partie démarrée, il n'est pas nécessaire de pouvoir changer les joueurs (vous pouvez, mais ce n'est pas demandé).
— Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : Serveur La commande suivante :

```
./bin/server listen
```

doit répondre aux services définis par votre API.

Code : Client réseau La commande suivante :

```
./bin/client network
```

permet au client de venir s'ajouter à la liste des joueurs, sous réserve que cela soit possible (il reste des places, partie non démarrée, etc.). Si cela est réussi, affichez la liste de tous les joueurs actuels. Puis, demandez à l'utilisateur de presser une touche pour continuer. Une fois qu'il a pressé cette touche, demandez au serveur de retirer le joueur, puis affichez à nouveau tous les joueurs actuels.

9.10 Livrable 4.final

Rapport section 6 : Modularisation

— API Web pour jouer en réseau. Proposez une API Web pour permettre de jouer en réseau.
— Conception logicielle sous la forme d'un diagramme UML C++ commenté.

Code : Serveur La commande suivante :

```
./bin/server listen
```

doit répondre aux services définis par votre API.

Code : Client réseau La commande suivante :

```
./bin/client network
```

permet au client de venir s'ajouter à la liste des joueurs. Une fois ceci-fait, le client attend que la partie démarre. Une fois la partie démarrée, ce client envoie des commandes au serveur pour le joueur qu'il gère, et applique toutes les commandes reçues du serveur. Utilisez des IA pour automatiser le jeu. Le jeu s'affiche normalement à l'écran.