MINISTÈRE
DE L'ÉDUCATION
NATIONALE
Liberté
Égalité
Fraternité

EDE NUM 1

SESSION 2025

CAPET
CONCOURS EXTERNE ET CAFEP CORRESPONDANT ET TROISIEME CONCOURS

Section : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR

Option : INGÉNIERIE INFORMATIQUE

ÉPREUVE ÉCRITE DISCIPLINAIRE

Durée : 5 heures

Calculatrice autorisée selon les modalités de la circulaire du 17 juin 2021 publiée au BOEN du 29 juillet 2021.

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.

Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.

Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.

NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.
Le fait de rendre une copie blanche est éliminatoire.

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

► **Concours externe du CAPET de l'enseignement public** :

Concours	Section/option	Epreuve	Matière
EDE	14 13 E	101	9311

► **Concours externe du CAFEP/CAPET de l'enseignement privé** :

Concours	Section/option	Epreuve	Matière
EDF	14 13 E	101	9311

►**Troisième concours externe du CAPET de l'enseignement public :**

Concours	Section/option	Epreuve	Matière
EDV	14 13 E	101	9311

IMPRIMERIE NATIONALE – 25 0233 – D'après documents fournis

Ce sujet se décompose de la façon suivante :

- Énoncé, pages 2 à 26 ;
- Documents techniques pages 27 à 33 ;
- Documents réponses (à rendre avec la copie).

Il est demandé aux candidats :

- de rédiger les réponses aux différentes parties sur des feuilles de copie séparées et clairement repérées ;
- de numéroter chaque feuille de copie et indiquer le numéro de la question traitée ;
- d'utiliser exclusivement les notations indiquées dans le sujet lors de la rédaction des réponses ;
- de justifier clairement les réponses ;
- d'encadrer ou souligner les résultats ;
- de présenter lisiblement les applications numériques, sans omettre les unités, après avoir explicité les expressions littérales des calculs ;
- de formuler les hypothèses nécessaires à la résolution des problèmes posés si celles-ci ne sont pas indiquées dans le sujet.

Escape Game

Contexte général Escape Game Kairos

Un jeu d'évasion, plus communément connu sous le nom d'*Escape Game* est une expérience ludique vécue seule ou en petit groupe allant généralement de 2 à 6 joueurs. L'objectif est de résoudre une succession d'énigmes afin de sortir d'une salle en un temps limité (d'où le terme évasion). Dans ce cadre, les énigmes se succèdent et la solution d'un problème permet généralement aux joueurs d'ouvrir une porte ou un coffre qui conduit à l'énigme suivante. Les premiers jeux de ce type reposaient sur des énigmes dont les solutions étaient validées de façon essentiellement mécanique (cadenas à combinaison de chiffres ou à clés) ou manuelles (un opérateur déverrouille manuellement une porte lorsqu'une énigme est résolue). Par ailleurs, l'accompagnement des joueurs est effectué par un opérateur appelé Maître de jeu ou plus souvent *Game Master* qui a pour rôle de donner des indications aux joueurs lorsque ceux-ci ne trouvent pas la solution à une énigme, afin de ne pas gâcher l'expérience ludique.

La société Kairos Escape Game a mis en place un système connecté qui enrichit l'expérience du joueur. Celui-ci repose sur une gamme de systèmes modulaires permettant de valider les réponses aux énigmes à partir de supports d'acquisition originaux. Ces périphériques sont interfacés dans un système de supervision qui centralise les données issues des différents sous-systèmes d'une même salle et permet d'assurer l'accompagnement des joueurs de cette salle de manière distante.

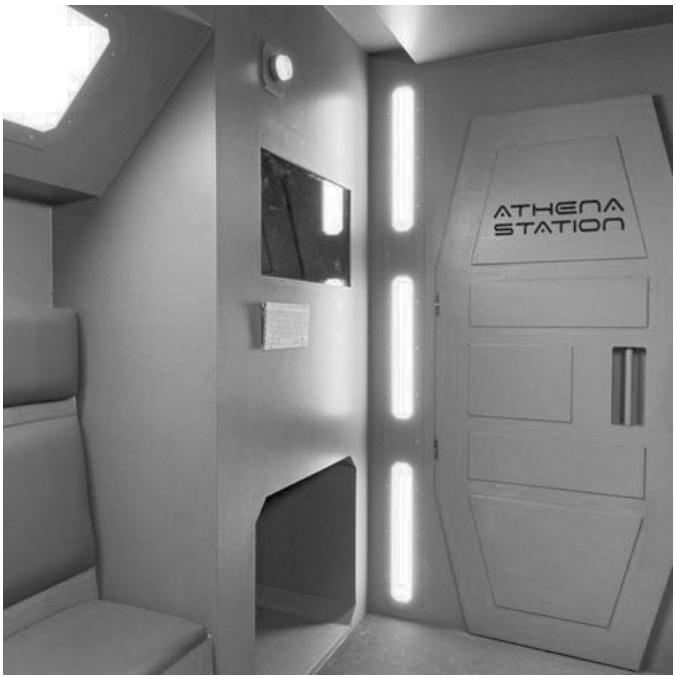


Figure 1 : Exemple de décor d'une salle à l'ambiance futuriste, l'objectif des joueurs est de sortir de la salle. Des claviers permettent de saisir les mots de passe trouvés par les joueurs. Des écrans et des haut-parleurs permettent de fournir des informations aux joueurs

L’objectif du sujet est de mettre en place des modifications sur le déroulement des énigmes d’une salle : *Athena Station*. S’en suit une étude spécifique de la base de données en vue d’inspecter la difficulté des énigmes et d’assurer la promotion des résultats des joueurs. Le sujet se termine par une étude de cas visant à créer une extension de la société Kairos en assurant la supervision de salles de jeu à distance. Cette étude est divisée en 6 parties

- Partie 1 : Analyse structurelle et fonctionnelle d’une salle de l’Escape Game Kairos
- Partie 2 : Architecture logicielle de la gestion des énigmes d’une salle
- Partie 3 : Insertion d’une nouvelle énigme dans la salle *Athena Station*
- Partie 4 : Adaptation de l’interface de supervision du Game Master à une nouvelle énigme
- Partie 5 : Inspection de la base de données et gestion du leaderboard
- Partie 6 : Évolution de l’architecture réseau

Les différentes parties du sujet sont indépendantes, toutefois il est conseillé au candidat de commencer par la partie 1 qui introduit les différents constituants d’une salle et leurs interactions.

Tournez la page S.V.P.

Partie 1 : Analyse structurelle et fonctionnelle d'une salle de l'Escape Game Kairos.

L'objectif de cette partie est d'analyser les composants d'une salle de jeu et de justifier certains choix technologiques permettant d'assurer le déroulement automatique des énigmes d'une salle tout en permettant une supervision par un maître du jeu (*Game Master*).

La société Kairos escape Game héberge des salles de jeu dont les décors et les énigmes traduisent des ambiances et époques variées : une salle à l'ambiance *Pirates des Antilles*, une salle à l'ambiance futuriste nommée *Athena Station (Figure 1)*, et 2 salles traduisant des ambiances de bandes dessinées. Chaque salle comporte des systèmes permettant d'acquérir la solution d'une énigme résolue par un utilisateur. La nature de ces systèmes d'acquisition dépend de la thématique de la salle et repose sur des contraintes artistiques : la salle *Pirates des Antilles* contient des systèmes de déclenchement invisibles pour l'utilisateur, la salle *Athena Station* contient des claviers et des boutons poussoirs permettant de traduire une ambiance futuriste et l'une des salles à la thématique bande dessinée contient un téléphone à cadran analogique apparaissant dans l'œuvre littéraire originale. Chaque salle de jeu permet d'accueillir jusqu'à 6 joueurs simultanément. Ils sont sous la surveillance d'un Game Master qui supervise le bon déroulement de la partie depuis une salle distante via une interface connectée à la salle de jeu. On donne dans le document réponse DR1 le diagramme des cas d'utilisation incomplet de l'Escape Game de la société Kairos.

Question 1 : Compléter le diagramme des cas d'utilisation en ajoutant l'acteur manquant ainsi que les relations avec les différents cas d'utilisation sur lesquels il interagit.

Question 2 : Ajouter au diagramme le cas d'utilisation « Changer le mot de passe d'un terminal ». Ce cas d'utilisation constitue l'une des manières de reconfigurer une énigme.

Dans la suite on se focalisera sur la description structurelle de la salle futuriste *Athena Station* dont certains éléments sont représentés Figures 1,2 et 3.

Description structurelle d'une salle de jeu :

Chaque salle est équipée d'une carte de commande Raspberry-Pi effectuant la centralisation des données saisies par les joueurs sur des systèmes d'acquisition dédiés. La salle futuriste *Athena* Station est munie de 3 **claviers** et de 2 **digicodes** permettant de saisir des mots de passe et des codes. Des informations sont affichées en retour à ces joueurs sur 4 **écrans** situés dans la salle. Les données saisies sur les systèmes d'acquisition sont transmises à la **carte de commande** Raspberry-Pi située dans une **armoire de commande** représentée Figure 2. Cette carte de commande est chargée de :

- Recevoir les données saisies par les joueurs. Lorsqu'un code correct est saisi, la carte de commande doit modifier les images affichées sur les écrans et envoyer un ordre à une **carte de puissance** chargée de déclencher l'ouverture de portes dans la salle via des dispositifs appelés **ventouses électromagnétiques** (constituées d'électro-aimants). Une telle ventouse est représentée Figure 3. Les différents systèmes d'acquisitions ainsi que la carte de puissance sont connectés à la carte de commande via une **Interface carte de commande entrées/sorties**.
- Transmettre des sons aux **haut-parleurs** situés dans la salle. Ces sons sont constitués des indications données par le maître du jeu ainsi que des éventuels bruitages liés à l'ambiance de la salle tels que des bruits d'explosion ou d'ouverture de serrure. Ces sons sont combinés et mixés par une **carte son** dont le signal est amplifié par un **amplificateur son** associé à chaque haut-parleur.

Par ailleurs le suivi des activités des joueurs est effectué par un *Game Master* à distance depuis une pièce de supervision. Les images et les sons de l'activité des joueurs dans la salle lui sont transmis directement grâce à une **caméra IP** telle que celle représentée Figure 3.

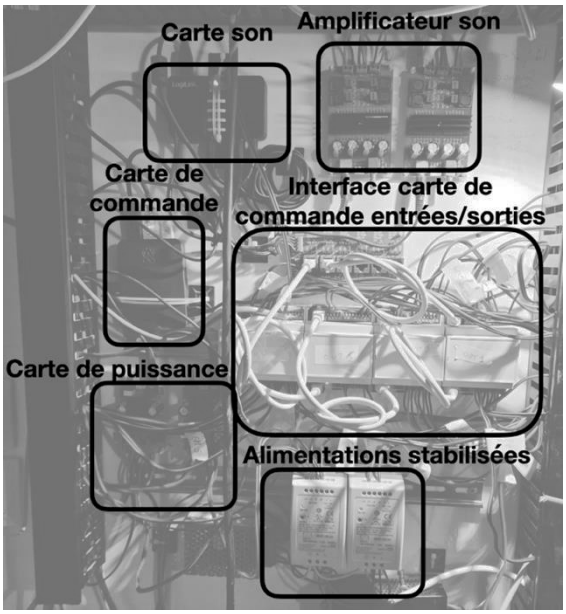


Figure 2 : Contenu de l'armoire de commande d'une salle.

Figure 3 : Ventouse permettant l'ouverture d'une porte (image de gauche) et caméra IP permettant le suivi de la partie par un maître du jeu (image de droite).

Question 3 : Compléter le diagramme de définition de blocs du document réponse DR2 en utilisant les noms des sous-systèmes indiqués en **gras** dans la description structurelle d'une salle de jeu.

Les caméras IP d'une même salle ainsi que la carte de commande sont connectées à un switch lui-même connecté à un réseau interne.

Question 4 : Rappeler la fonction d'un switch.

Tournez la page S.V.P.

L'interface *Game Master* est une page web générée par la carte de commande de la salle lorsque celle-ci reçoit une requête http sur le port 8000. L'adresse IP de la carte de commande a été fixée en mode *static* à la valeur 192.168.17.1 par le concepteur de la salle.

Question 5 : Donner l'url que le *Game Master* doit saisir dans un navigateur web pour accéder à la page web générée par la carte de commande.

On s'intéresse désormais aux flux d'informations échangés entre les différents composants de l'Escape Game de la société Kairos. Afin de simplifier l'étude, on se restreint à une description n'incluant qu'une seule salle de jeu et non pas quatre comme c'est le cas sur le système réel.

Question 6 : Compléter les flux du diagramme de blocs internes fourni sur le document réponse DR3. Pour chacun des flux, **indiquer** par une flèche son sens ainsi que la nature de la grandeur échangée.

Le paramétrage des caméras IP est choisi en mode IP fixe.

Question 7 : Expliquer ce que cela signifie et **préciser** l'intérêt de ce paramétrage. On rappelle que la page web générée par la carte de commande contient les images capturées par les caméras sur IP de la salle.

Le programme assurant la gestion du serveur web, ainsi que la gestion des périphériques avec lesquels les joueurs interagissent sont exécutés sur la carte de commande Raspberry Pi dans un environnement Linux.

Question 8 : Proposer une stratégie de programmation pour que ces deux tâches soient effectuées simultanément du point de vue de l'utilisateur.

Les Escapes Games ont récemment gagné en popularité du fait des avancées technologiques. Les premières entreprises proposaient une expérience ludique constituée d'une série d'énigmes dont la résolution de chaque problème permettait de trouver un code débloquant un cadenas et permettant d'accéder à l'énigme suivante.

Question 9 : Proposer une synthèse récapitulant les différentes solutions technologiques mises en place par l'entreprise Kairos pour améliorer l'immersion du joueur et fluidifier son expérience ludique.

Partie 2 : Architecture logicielle de la gestion des énigmes d'une salle.

L'objectif de cette partie est de s'approprier l'architecture du programme exécuté sur la carte de commande en vue d'y apporter des modifications telles que l'ajout d'une énigme ou la modification d'une énigme existante.

Les différentes salles contiennent des énigmes dont les solutions doivent être transmises à la carte de commande de la salle afin de débloquer l'énigme suivante. La thématique de la salle a une incidence sur la nature des systèmes qui s'y trouvent et donc sur la nature des données transmises à la carte de commande. Malgré la diversité des dispositifs d'acquisition, l'organisation des énigmes de toutes les salles est la même : la résolution d'une énigme permet de déclencher une transition vers la suivante et celles-ci se suivent les unes après les autres. Cependant, il peut arriver que la carte de commande ne parvienne pas à prendre en compte la fin d'une énigme (à cause d'un terminal utilisateur cassé par exemple), c'est pourquoi la transition d'une énigme à la suivante peut être forcée par un maître de jeu communiquant avec la carte de commande via une interface web.

2.1 Architecture logicielle de la gestion des énigmes d'une salle

L'organisation des tâches exécutées par la carte de commande est décrite dans le diagramme d'état Figure 4 (page suivante).

Les différentes transitions *Début maintenance*, *Fin maintenance*, *Début préparation*, *Fin préparation*, *Lancement partie* et *Arrêt carte commande* sont activées par le *Game Master* en cliquant sur des boutons associés dans son interface de commande.

L'interface affichée sur l'écran du *Game* Master lors du lancement du *mode préparation* (pendant lequel un opérateur va cacher des indices dans la salle et y placer des objets) est représentée Figure 5 (page suivante).

Tournez la page S.V.P.

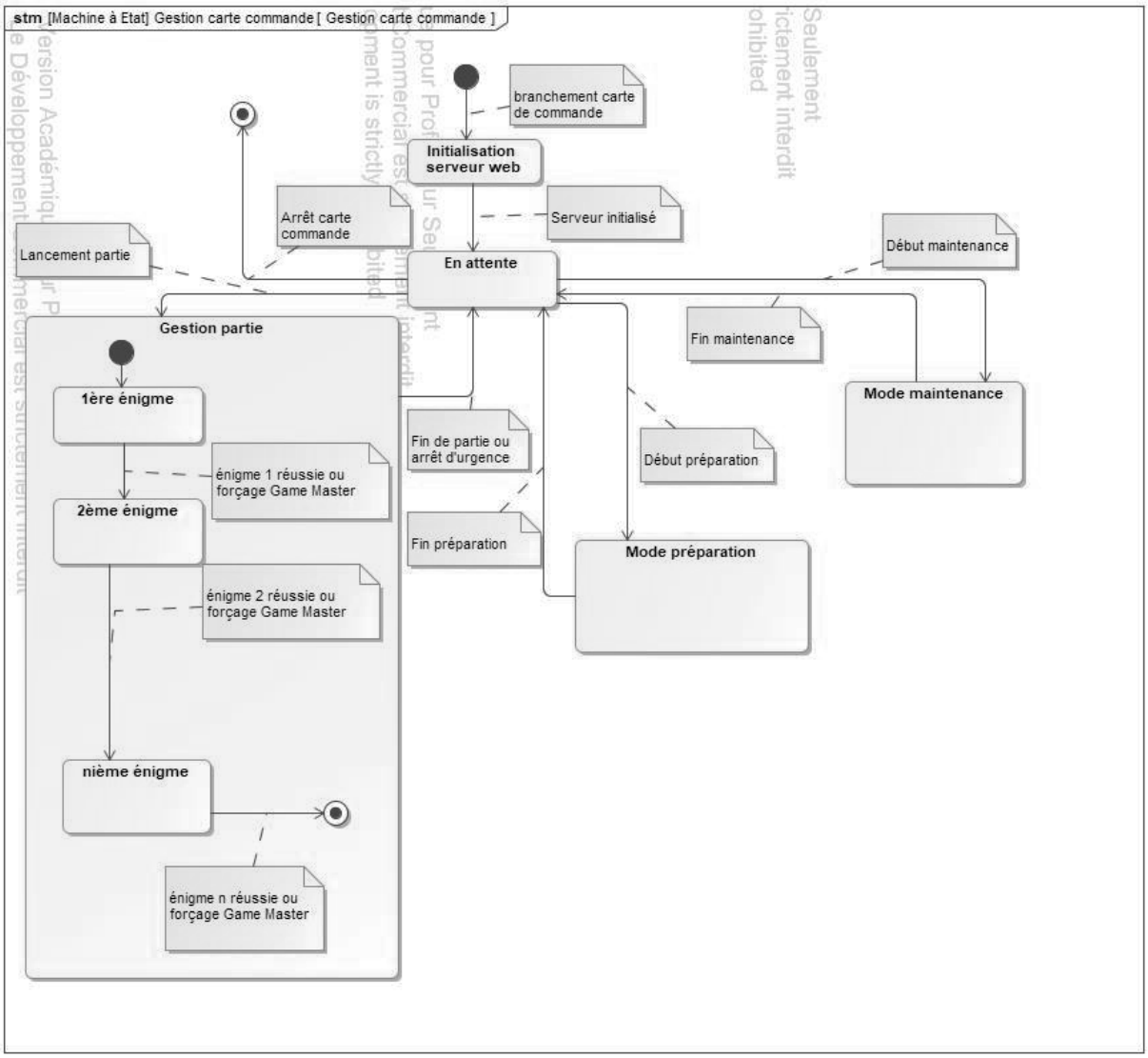


Figure 4 : Diagramme d'état du système

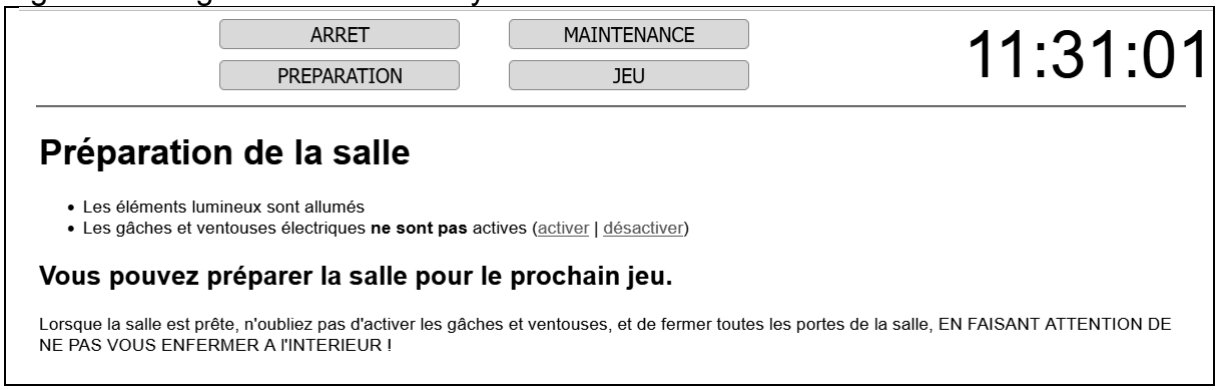


Figure 5 : Interface *Game Master* pendant le *mode préparation*.

Question 10 : Compléter le diagramme d'état DR 4 associé au *mode préparation* en faisant apparaître les états :

- *Gâches et ventouses activées*
- *Gâches et ventouses inactives*
- *Éléments lumineux allumés* (il s'agit des éclairages et éléments de signalisation)

Ainsi que les transitions *activer* et *désactiver* déclenchées par l'appui sur les liens soulignés dans l'interface affichée Figure 5.

On s'intéresse maintenant à l'une des énigmes qui correspond à la recherche d'un code à saisir en un temps limité sur un digicode rétro-éclairé à 10 chiffres initialement éclairés en rouge. Le code à trouver est constitué de 4 chiffres.

- Lorsqu'un chiffre est saisi dans le bon ordre, son rétro-éclairage passe en vert.
- Si un des chiffres est faux dans la composition du code, tous repassent au rouge et il faut recommencer la saisie depuis le début.
- Par ailleurs, si les joueurs mettent plus de 2 secondes à appuyer sur un chiffre, là encore tous repassent au rouge et il faut recommencer la saisie depuis le début.
- Si les joueurs ne parviennent pas à résoudre l'énigme, le *Game Master* dispose d'une commande permettant de valider automatiquement l'énigme.

Question 11 : Proposer un diagramme d'état décrivant le déroulement de cette énigme. On pourra notamment faire figurer un état différent pour chaque numéro correctement identifié.

La machine à état est implémentée dans une fonction en langage C. Celle-ci est codée par une procédure nommée salle_machine_etat() qui est exécutée de manière répétée dans un processus instancié lors du passage dans l'état « *Gestion partie »* représenté Figure 4. Cette procédure est exécutée en parallèle d'une autre procédure gestion_serveur() permettant les interactions entre la carte de commande et le *Game Master* via le serveur web. Un extrait du code permettant d'exécuter ces deux processus indéfiniment est proposé dans le DT2. Celui-ci est accompagné d'une documentation relative à l'utilisation des threads en langage C dans le DT1.

Question 12 : Rappeler brièvement le principe du multithreading en précisant comment il est appliqué ici. On pourra s'appuyer sur un schéma en identifiant les threads maître et esclave.

L'implémentation actuelle instancie 2 threads esclaves dans un thread maître.

Question 13 : Proposer une implémentation simplifiée de la fonction main() ne comportant qu'un thread maître et un thread esclave. Les fonctions callback_machine_etat() et callback_serveur() sont déjà définies.

En pratique, le temps d'exécution des procédures salle_machine_etat() et gestion_serveur() sont très petits. C'est pourquoi il est suffisant de ne les exécuter qu'une fois par seconde pour simuler une exécution en parallèle.

Question 14 : Proposer un code simplifié de la fonction main() précédente qui rappelle indéfiniment les procédures salle_machine_etat() et gestion_serveur() sans passer par des threads et en attendant 1 seconde entre 2 appels successifs.

On donne dans la Figure 6 le code de la procédure salle_machine_etat()d'une machine qui n'est constituée pour des raisons de simplification que de 4 énigmes. Les fonctions _enigme_i() sont chargées de l'activation des différents périphériques liés à l'énigme en cours et renvoient 1 si l'énigme a été réussie. Les fonctions _deblocage_GM_i() consultent les données envoyées par le *Game Master* au serveur web et renvoient 1 si un ordre de déblocage de l'énigme en cours a été reçu.

Tournez la page S.V.P.

```
static u16 cyc ;
static u16 verrou ;
static u32 save_time;
static u16 success_enigme;
static u16 variable; // variable éventuellement utilisable à la question 16
// u16 : unsigned integer sur 16 bits, u32 : unsigned integer sur 32 bits

static void salle_machine_etat ( void ) {
cyc = 0;
verrou = 0;{
switch( cyc )
 { case 0 :
        if (_enigme_0() == 1 || (_deblocage_GM_0() == 1)
        // Exécution des périphériques liés à l'énigme 1 et vérification si réussite
des joueurs
                {cyc++;}
        break;
  case 1 :
                if (_enigme_1() == 1 || _deblocage_GM_1() == 1)
                        {save_time = _GameElapsedTime(); // sauvegarde
de l'instant (en secondes) où l'énigme 1 a été passée
                cyc++;}
        break;
  case 2 : // Au bout de 3 minutes si l'énigme 2 n'est pas résolue, on verrouille la
porte d'entrée et on active la lumière d'urgence. Attention cet évènement ne doit
être déclenché qu'une fois.
        success_enigme = _enigme_2()*_deblocage_GM_2()
        if (
..... )
            { verrou = 1 ;
                    _roomEventAdd( "Verrouillage porte d'entrée après 3 minutes" );
             _outputLevelSet( ESPACE_GACHE_PORTE_ENTREE,
OUTPUT_LEVEL_ON );
             _roomEventAdd( "Changement du mode d'éclairage" );
                _outputLevelSet( LUMIERE_STANDARD_ENTREE,
OUTPUT_LEVEL_OFF );
             _outputLevelSet( LUMIERE_URGENCE_ENTREE,
OUTPUT_LEVEL_ON );
              break;}
        if (
..... )
              {cyc++;}
        break;
  case 3 :
        if (_enigme_3() == 1 || _deblocage_GM_3() == 1)
           {cyc++;}
        break;
}}
```

Figure 6 : Procédure salle_machine_etat()

Question 15 : Proposer sur votre copie des instructions permettant de compléter le code associé à l'énigme activée par l'appel à la fonction _enigme_2(). Cette énigme doit être résolue par les joueurs en un temps suffisamment court conformément aux indications dans les commentaires du code. Dans le cas contraire, l'état des lumières de la salle est modifié.

Afin d'accorder plus de liberté aux joueurs, les scénaristes souhaitent que les joueurs puissent traiter dans l'ordre ou dans le désordre les énigmes 4 et 5 qui sont implémentées par les fonctions _enigme_4(), _enigme_5(), _deblocage_GM_4() et _deblocage_GM_5().

Question 16 : Proposer sur votre copie le code à ajouter à la fin de la procédure salle_machine_etat() pour implémenter ce choix des scénaristes.

Question 17 : Conclure sur l'objectif de cette partie en résumant les différentes modifications à opérer dans le code exécuté par la carte de commande dans le cas où les concepteurs souhaitent introduire une nouvelle énigme et dans le cas où ils souhaitent modifier une énigme existante.

Tournez la page S.V.P.

Partie 3 : Insertion d'une nouvelle énigme dans la salle Athena Station

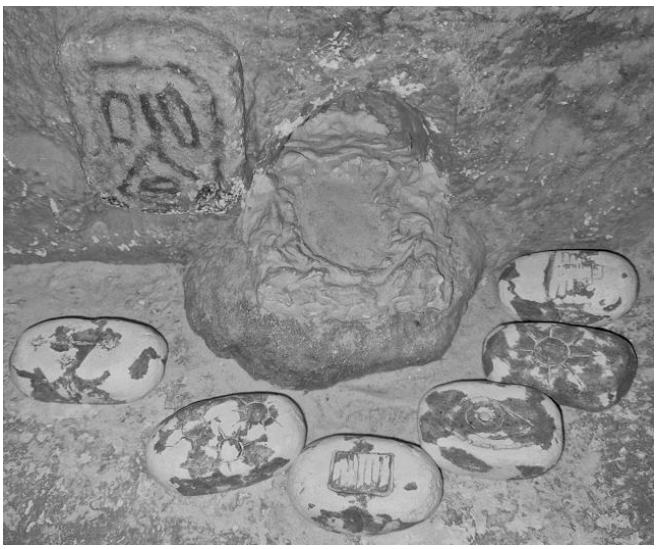
L'objectif de cette partie est d'implémenter une nouvelle énigme dans la salle *Athena Station* en adaptant une énigme issue d'une autre salle.

3.1 Présentation de la nouvelle énigme

La salle *Station Athena* comporte une succession d'énigmes devant être résolues en une heure. La direction artistique souhaite introduire une nouvelle énigme qui repose sur la collaboration des joueurs pour déverrouiller une porte. Les joueurs doivent trouver des badges de l'équipage de la station Athéna et les apposer sur des modules placés dans le poste de commandement de la station. L'objectif de cette partie est d'étudier une énigme similaire présente dans l'une des autres salles et d'intégrer cette énigme dans le code de la carte de commande de la station Athéna.

3.2 Un exemple de système de déclenchement de portes : runes magiques et tags RFID.

Dans un souci de cohérence, les technologies modernes ne peuvent pas être apparentes dans des salles dont la thématique inclut de la magie ou la restitution d'une époque ancienne comme c'est le cas pour la salle *Pirates des Antilles*. Dans de tels cas, il est nécessaire « d'habiller » les systèmes d'acquisition permettant de valider les énigmes. On s'intéresse ici à la reconstitution de pierres runiques (décrites auprès des joueurs comme étant magiques) basées sur l'utilisation de tags RFID contenus dans des pierres en plâtre (Figure 7, photo de gauche). La lecture de ces « tags » est effectuée par des « tags readers » dissimulés dans des socles visibles (Figure 7 photo de droite)



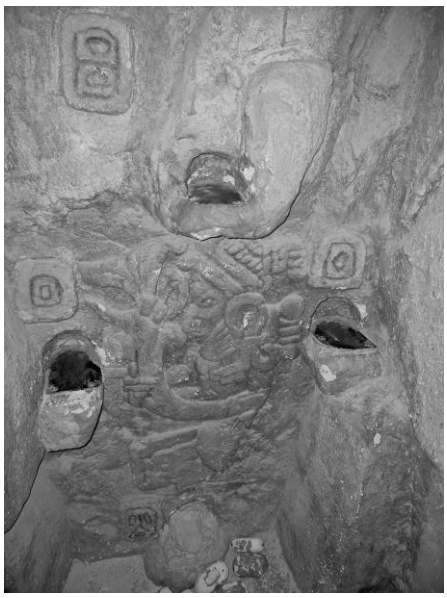


Figure 7 : Photo de gauche : Détail des pierres sur lesquelles un symbole est visible ainsi que l'un des socles près duquel se trouve également un symbole.
Photo de droite : Les 4 socles ainsi que leurs symboles correspondants (les 3 du haut sont noirs et celui du bas est clair).

Question 18 : Expliciter brièvement le fonctionnement d'un « tag » RFID en donnant un exemple usuel d'utilisation de cette technologie.

L'énigme des pierres runiques est la suivante : quatre pierres différentes sur lesquelles un symbole magique différent a été dessiné sont cachées dans une salle. Quatre socles représentant chacun le symbole d'une des pierres sont également disposés dans la salle. Les joueurs doivent retrouver les quatre pierres et les déposer sur leurs socles respectifs pour déclencher l'ouverture d'une porte.

Question 19 : Préciser la nature du système contenu dans chacun des socles. On pourra notamment **expliciter** celui-ci grâce à un schéma.

Question 20 : Pourrait-on implémenter cette énigme en plaçant un aimant dans chacune des pierres et en cachant une sonde à effet Hall dans chacun des socles ? **Justifier** votre réponse.

La partie logicielle repose sur l'implémentation d'une fonction pierres_placees(id_sp_1,id_sp_2, id_sp _3, id_sp_4) qui prend en argument 4 entiers indiquant pour chaque socle, l'identifiant de la pierre placée dessus : si la variable id_sp_1 prend la valeur 2, cela signifie que la pierre numéro 2 est placée sur le socle numéro 1. Cette fonction renvoie le booléen True si toutes les pierres (d'identifiants respectifs 1,2,3 et 4) sont placées sur le bon socle.

Question 21 : Proposer une implémentation en langage Python de la fonction pierres_placees().

Certains joueurs ne remarquent pas au cours de la partie que chaque pierre est munie d'un symbole différent et ceux-ci ne comprennent pas pourquoi rien ne se passe lorsque les 4 pierres sont placées sur les socles. Par ailleurs, le socle du bas qui est clair n'est pas toujours repéré par les joueurs. Il a été décidé d'utiliser des Leds de couleur rouge pour éclairer les socles lorsque les 4 pierres ont été placées mais sur les mauvais socles. Les leds s'allument également lorsque les 3 premières pierres ont bien été placées, mais pas la $4^{ième}$.
Une fonction pierres_incompletes(id_sp_1, id_sp_2, id_sp_3, id_sp_4*)* renvoie le booléen True si toutes les pierres sont bien placées sur un socle mais à des emplacements incorrects ou si les pierres 1,2 et 3 sont bien placées mais pas la $4^{ième}$.
Les variables id_sp_i pour i=1,2,3 ou 4 sont définies de la même manière que dans la question précédente et on supposera par ailleurs que celles-ci peuvent prendre la valeur 0 si aucune pierre n'est placée sur le socle *i*.

Question 22 : Proposer une implémentation en langage Python de la fonction pierres_incompletes().

Tournez la page S.V.P.

3.3 Adaptation à la salle *Athena Station*

La nouvelle énigme implantée dans la salle *Athena Station* repose sur 6 badges munis de tags Rfid qui doivent être déposés sur leurs socles correspondants. Contrairement à l'énigme précédente, les joueurs ne disposent pas d'indication visuelle pour faire correspondre les badges et leurs socles. En revanche lorsque tous les badges sont placés sur un socle, un écran affiche le nombre de tags correctement placés lors de l'appui sur un bouton. L'objectif des joueurs est donc de trouver le plus rapidement possible l'emplacement correct de chacun des badges. La carte de commande collecte les identifiants des tags rfid des badges placés sur les socles dans une liste Python L_id dont l'élément d'indice *i* contient l'identifiant du tag rfid placé sur le socle numéro *i.* On considère que le badge numéro *i* est bien placé si l'identifiant numéro *i* est placé à l'indice *i* de la liste

Question 23 : **Écrire** une fonction nombres_placés(L) qui reçoit en entrée une liste d'identifiants suivant la définition établie ci-dessus et qui renvoie le nombre d'éléments bien placés.

On se place du point de vue des joueurs et on souhaite construire un algorithme permettant d'identifier rapidement la bonne combinaison de tags rfid. On note *n* le nombre de tags rfid à placer sur les socles.

Question 24 : **Exprimer** le nombre de permutations possibles des tags rfid en fonction de *n*. Pour *n*=6 et en supposant que les joueurs prennent en moyenne 10 secondes pour évaluer une permutation, **calculer** la durée maximale mise par les joueurs pour résoudre l'énigme. **Conclure** sur la pertinence de cette énigme.

En vue de simplifier l'énigme, les concepteurs ont choisi d'éclairer en vert les badges correctement placés sur un socle et en rouge ceux qui sont mal placés. L'éclairage ne se fait qu'une fois tous les badges placés sur un socle. On dispose du code fourni Figure 8.

Question 25 : **Préciser** les états successifs de la liste L affichés par la fonction lors d'un appel à cherche_permutation([4,0,1,3,2]).

Question 26 : En suivant la stratégie précédente et en se plaçant dans le pire des cas avec un ensemble de *n* tags à placer correctement, **calculer** le nombre de permutations des tags que les joueurs devront tester avant de trouver la bonne configuration ? En **déduire** le temps nécessaire pour identifier une permutation de 6 tags en supposant que les joueurs prennent 10 secondes pour tester chaque permutation et **conclure** sur le choix d'indiquer aux joueurs les tags correctement placés.

```
def liste_places(L):
    places = []
    for i in range(len(L)):
        if L[i]==i:
            places.append(True)
        else:
            places.append(False)
    return places

def cherche_permutation (L) :
        ‘’’ La fonction reproduit une stratégie de recherche de la permutation des tags
rfid par les joueurs. ‘’’
    while nombre_places(L) != len(L) :
        print(L)
        places = liste_places(L)
        dernier = None
        premier = None
        for i in range(len(L)):
            if places[i] == False:
                if dernier == None:
                    premier = i
                    dernier = L[i]
                else:
                    dernier,L[i] = L[i],dernier # permutation du contenu de la variable dernier
et du i-ème élément de la liste L :  L[i]
        L[premier] = dernier
    return L
```

Figure 8 : Algorithme de recherche de la permutation encodée dans la liste L.

Une fois l’énigme validée, l’ouverture de la porte se fait alors par alimentation d’un électro-aimant dont un extrait de la documentation est fourni DT3. Celui-ci est alimenté au travers de la carte de puissance.

Question 27 : Expliquer pourquoi la ventouse électromagnétique ne peut pas être pilotée directement par la carte de commande.

On souhaite vérifier le dimensionnement de la ventouse. On suppose qu’un adulte exerce un effort de 300 N au milieu d'une porte de largeur 83 cm. La ventouse a été placée sur le dormant de la porte à l’opposé des gonds. La Figure 9 illustre la situation étudiée.

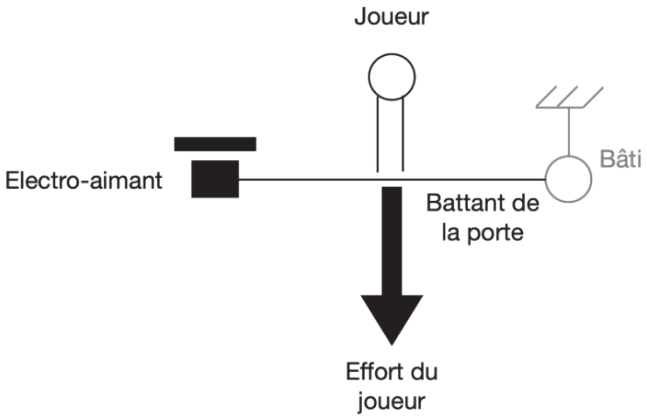


Figure 9 : Dimensionnement de la ventouse.

Tournez la page S.V.P.

Question 28 : Calculer la force que doit développer la ventouse pour maintenir un moment nul autour de l'axe de rotation de la porte puis valider le choix de la ventouse.

Question 29 : En reprenant les éléments étudiés dans cette partie, **indiquer** quelles seront les différentes opérations à réaliser et les composants à mettre en œuvre pour implémenter cette nouvelle énigme.

Partie 4 : Adaptation de l'interface de supervision du Game Master à une nouvelle énigme

L'objectif de cette partie est l'étude de l'interface entre le *Game Master* et la carte de commande d'une salle en vue d'intégrer dans cette salle un nouvel élément de décor qui sera pris en compte dans l'interface *Game Master*.

Les échanges entre la carte de commande et le *Game Master* se font au moyen d'une interface web affichée Figure 9. Lors du démarrage de la carte de commande d'une salle, celle-ci instancie un serveur Mongoose qui génère la page affichée au *Game Master* et collecte les requêtes effectuées par celui-ci pour les exécuter. L'architecture générale est représentée Figure 10.

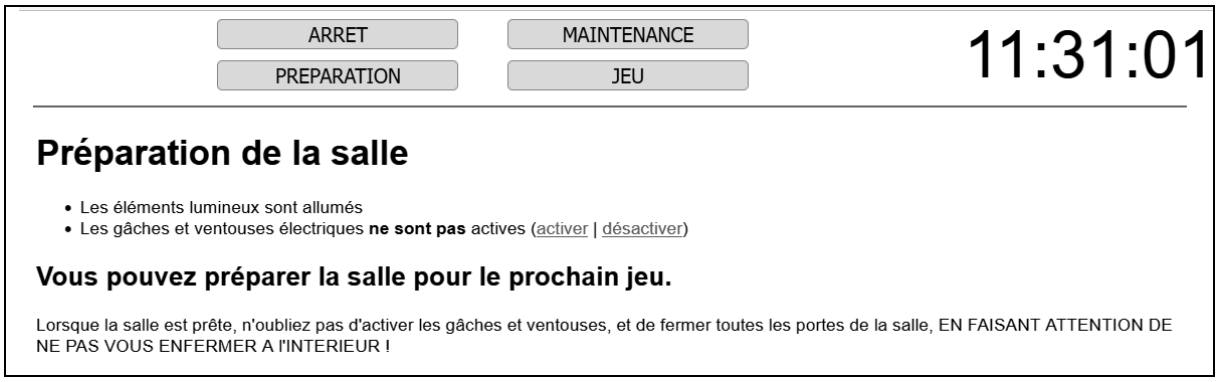


Figure 9 : Interface Game Master en mode 'Préparation de la salle'.

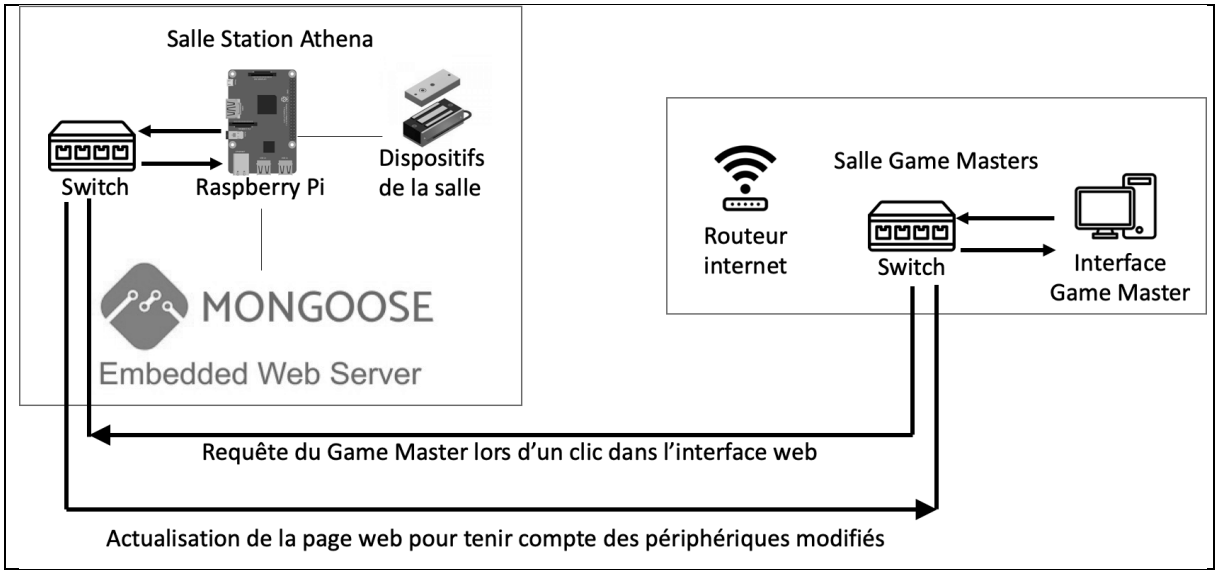


Figure 10 : Communication entre le Game Master et la carde de commande.

On donne un extrait du code du fichier index.html encodant la page web vue par le Game Master dans le DT4.

Question 30 : Indiquer sur quelle machine sont stockés les fichiers index.html ainsi que les différents fichiers .js qui lui sont liés. **Indiquer** également sur quelle machine ils sont exécutés. **Justifier** votre réponse.

Tournez la page S.V.P.

La stratégie de communication entre l'interface *Game Master* et la carte de commande repose sur la technologie Ajax dont la description Wikipedia a été reportée DT5.

Question 31 : Préciser l'intérêt d'intégrer la technologie Ajax dans la page web de l'interface *Game Master*. L'argumentation devra mentionner l'affichage des images issues des caméras IP ainsi que l'interface avec les différents mécanismes de la salle de jeu.

Question 32 : Donner un avantage et un inconvénient liés à l'utilisation d'une méthode de communication asynchrone ?

On a fourni DT6 le code de la fonction ajaxCallUrl() contenue dans le fichier ajax.js.

Question 33 : Expliciter la différence entre une requête http reposant sur la méthode GET et une reposant sur la méthode POST. **Indiquer** le choix qui a été effectué ici.

L'interface Game Master permet d'afficher en temps réel des images issues des caméras de la salle afin de pouvoir guider les joueurs. On a relevé les trames transmises sur le réseau Figure 11.

Time	Source	Destination	Info
16.278246	192.168.1.163	192.168.14.5	GET /image.jpg?t=1713530511898 HTTP/1.1
16.281896	192.168.1.163	192.168.14.6	GET /image.jpg?t=1713530511899 HTTP/1.1
16.288501	192.168.1.163	192.168.14.7	GET /image.jpg?t=1713530511899 HTTP/1.1
16.290334	192.168.1.163	192.168.14.8	GET /image.jpg?t=1713530511900 HTTP/1.1
16.295409	192.168.14.3	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
16.319844	192.168.14.4	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
16.325537	192.168.14.5	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
16.329584	192.168.14.7	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
16.339806	192.168.14.8	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
16.348995	192.168.1.163	192.168.14.1	POST /command HTTP/1.1 (text/plain)
16.351893	192.168.14.6	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
16.369387	192.168.14.1	192.168.1.163	HTTP/1.1 200 OK , JSON (application/json)
17.469436	192.168.1.163	192.168.14.3	GET /image.jpg?t=1713530513003 HTTP/1.1
17.478177	192.168.1.163	192.168.14.4	GET /image.jpg?t=1713530513004 HTTP/1.1
17.480496	192.168.1.163	192.168.14.5	GET /image.jpg?t=1713530513005 HTTP/1.1
17.482976	192.168.1.163	192.168.14.6	GET /image.jpg?t=1713530513005 HTTP/1.1
17.487093	192.168.1.163	192.168.14.7	GET /image.jpg?t=1713530513006 HTTP/1.1
17.496784	192.168.1.163	192.168.14.8	GET /image.jpg?t=1713530513007 HTTP/1.1
17.498455	192.168.1.163	192.168.14.1	POST /command HTTP/1.1 (text/plain)
17.509031	192.168.14.3	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
17.523542	192.168.14.4	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
17.527341	192.168.14.5	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
17.534421	192.168.14.7	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
17.542915	192.168.14.8	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
17.553741	192.168.14.6	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
17.596674	192.168.14.1	192.168.1.163	HTTP/1.1 200 OK , JSON (application/json)
17.999169	192.168.1.163	192.168.11.15	GET /image.jpg?t=1713530513645 HTTP/1.1
18.054771	192.168.11.15	192.168.1.163	HTTP/1.0 200 OK (image/jpeg)
18.462537	192.168.1.163	192.168.14.3	GET /image.jpg?t=1713530514100 HTTP/1.1
18.473009	192.168.1.163	192.168.14.4	GET /image.jpg?t=1713530514101 HTTP/1.1
18.480516	192.168.1.163	192.168.14.5	GET /image.jpg?t=1713530514102 HTTP/1.1
18.484124	192.168.1.163	192.168.14.6	GET /image.jpg?t=1713530514102 HTTP/1.1
18.485890	192.168.1.163	192.168.14.7	GET /image.jpg?t=1713530514103 HTTP/1.1
18.490700	192.168.1.163	192.168.14.8	GET /image.jpg?t=1713530514103 HTTP/1.1

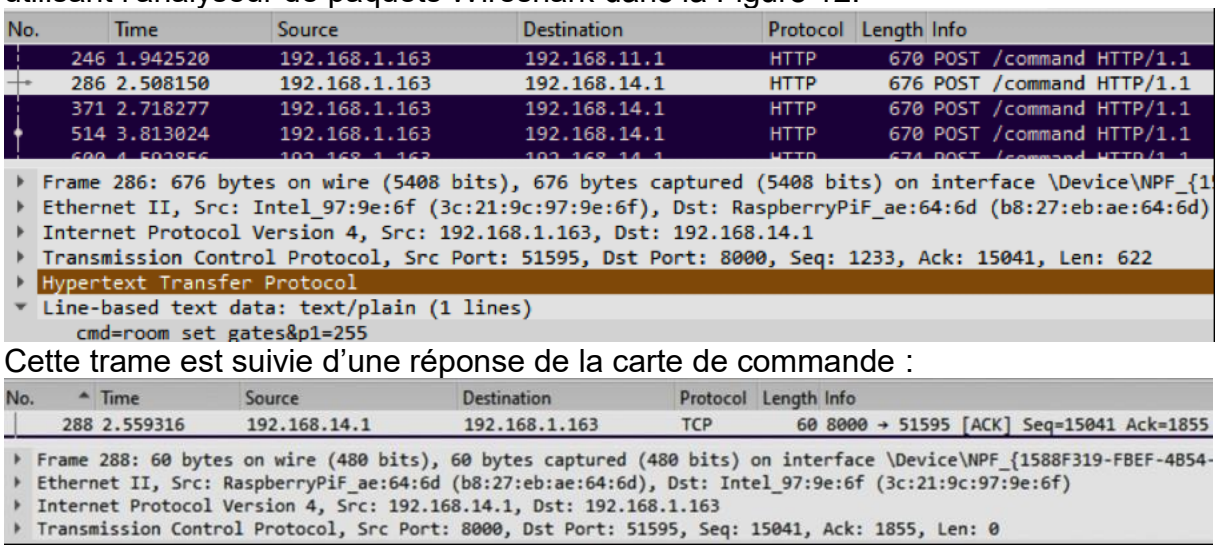
Figure 11 : Trames circulant sur le réseau, relevées grâce à l'analyseur de paquets Wireshark.

Question 34 : **Préciser** le nombre de caméras installées dans la salle et la fréquence de rafraîchissement des images configurées.

L'activation des gâches (fermeture de toutes les portes) se fait lors d'un clic dans l'interface *Game Master* sur le bouton mis en surbrillance. Ce clic déclenche l'exécution de l'instruction :

```
ajaxCallUrl(« /command »,«cmd=room_set_gates&p1=255»,null) ;
```

Cette commande permet de transmettre l'ordre à la carte de commande d'exécuter une fonction. On a relevé les trames transmises par le terminal du Game Master en utilisant l'analyseur de paquets Wireshark dans la Figure 12.



Cette trame est suivie d'une réponse de la carte de commande :

Figure 12 : Lecture de trames transmises lors d'un clic du Game Master sur le bouton *activer* de l'interface Figure 9.

Question 35 : Indiquer les adresse MAC et IP du PC utilisé par le Game Master et de la carte de commande de la salle.

Les requêtes sont reçues par le serveur Mongoose et collectées dans une structure CGI (Common Gateway Interface) qui permet d'interfacer les données reçues par le serveur web avec les fonctions permettant d'actionner les systèmes contenus dans la pièce où se trouvent les joueurs. Le code C permettant de définir cette structure est donné Figure 13 :

```
typedef struct
{
 char name[256] ;   // Nom de la commande CGI, descripteur libre si le nom est vide
 int (*func)(int nbArg, char* args[], char* buffer, u32 clientIp) ;        // Pointeur vers la fonction qui gère cette commande CGI
}
CgiCommand ;
```

Figure 13 : Structure permettant de stocker les différentes commandes cgi.

Le nom d'une commande cgi est stocké dans l'attribut `name` et la fonction correspondante est stockée dans l'attribut `func`.

Tournez la page S.V.P.

Le code permettant d'initialiser cette structure est le suivant :

```
#define CGI_MAX_COMMAND 64
static CgiCommand cgiCommand[CGI_MAX_COMMAND] ;
```

Question 36 : Rappeler la signification de l'instruction #define ainsi que la dénomination de ce type d'instruction en langage C.

Question 37 : Indiquer le nombre de commandes CGI différentes qui peuvent être enregistrées dans la variable cgiCommand. **Préciser** quelles sont les restrictions portant sur les noms de ces commandes.

On fournit la fonction _cgiCommandRegister() Figure 14 qui permet d'actualiser la variable cgiCommand en associant à un nom de commande (commandName) une fonction (cgiFunc()) qui exécutera l'action commandée par le Game Master. Dans la suite on n'analysera pas en détail les arguments et le contenu des fonctions cgiFunc

```
U8 _cgiCommandRegister ( char* commandName, int (*cgiFunc)(int nbArg, char*
args[], char* buffer, u32 clientIp) )
{
 int i ;
 // On scanne pour un emplacement de libre et on enregistre la commande CGI et
son pointeur de fonction
 // La fonction strcpy(s1,s2) permet de recopier la chaîne de caractères s1 dans s2.
 for ( i=0 ; i<CGI_MAX_COMMAND ; i++ )
 { if ( cgiCommand[i].name[0] == '\0' )
      { strcpy( cgiCommand[i].name, commandName );
        cgiCommand[i].func = cgiFunc ;
        debug( « Enregistrement de la commande CGI '%s'\n », commandName) ;
        return TRUE;
      }
 }
 return FALSE ;
}
```

Figure 14 : Fonction effectuant l'enregistrement d'une nouvelle commande cgi dans la structure de stockage.

Question 38 : Écrire le code d'une procédure void _cgiInit(void) qui initialise le champ name de tous les éléments de la variable cgiCommand à la valeur par défaut ('\0').

Un exemple de fonction stockée dans la variable cgiCommand est la fonction roomCgiSetGates (non détaillée dans le sujet). Une telle fonction peut être stockée grâce à l'instruction :

```
_cgiCommandRegister( "room_set_gates", roomCgiSetGates );
```

La fonction roomCgiSetGates sera exécutée lors de la réception de la requête ajax de la Question 35.

Question 39 : Compléter le code de la fonction _cgiExecute() donné DR 5 permettant d'exécuter un ordre reçu du *Game Master*. Le nom de la fonction cgi à utiliser est stocké dans args[0].

La structure cgi étudiée permet donc de faire l'interface entre les commandes du *Game Master* envoyées depuis son navigateur internet et la machine à état étudiée dans la partie 2 qui gère le déroulement des énigmes de la salle.

Question 40 : Proposer un diagramme de séquence faisant intervenir les différents acteurs : *Game Master*, *Navigateur internet*, *Serveur Mongoose* en indiquant les messages échangés entre ceux-ci lorsque la fermeture de toutes les portes est commandée par le Game Master.

Un nouvel élément de décor est inséré dans la salle, celui-ci comprend un écran qui s'allume lorsqu'une énigme est validée ou lorsque le Game Master force sa mise en fonctionnement.

Question 41 : Proposer une synthèse des différentes actions à mener pour que le Game Master dispose d'un bouton permettant d'allumer cet écran. Préciser la nature des modifications à envisager en se référant aux différents codes vus dans cette partie et en précisant quels sont ceux qu'il est nécessaire de modifier.

Tournez la page S.V.P.

Partie 5 : Inspection de la base de données et gestion du leaderboard

Les objectifs de cette partie sont la mise en place des requêtes permettant d'identifier les erreurs d'enregistrement dans la base de données et d'effectuer l'affichage du classement des joueurs les plus rapides.

Les cartes de commande enregistrent les instants auxquels les joueurs terminent les énigmes de leur salle. Cette collecte de données permet de produire un classement des équipes les plus efficaces à destination des joueurs. Ces durées de résolution des différentes énigmes sont également examinées par les concepteurs de salle de la société afin d'ajuster la difficulté des différentes énigmes d'une même salle et d'améliorer l'expérience utilisateur. La base de données est enregistrée au format MySQL et est organisée selon le schéma relationnel suivant :

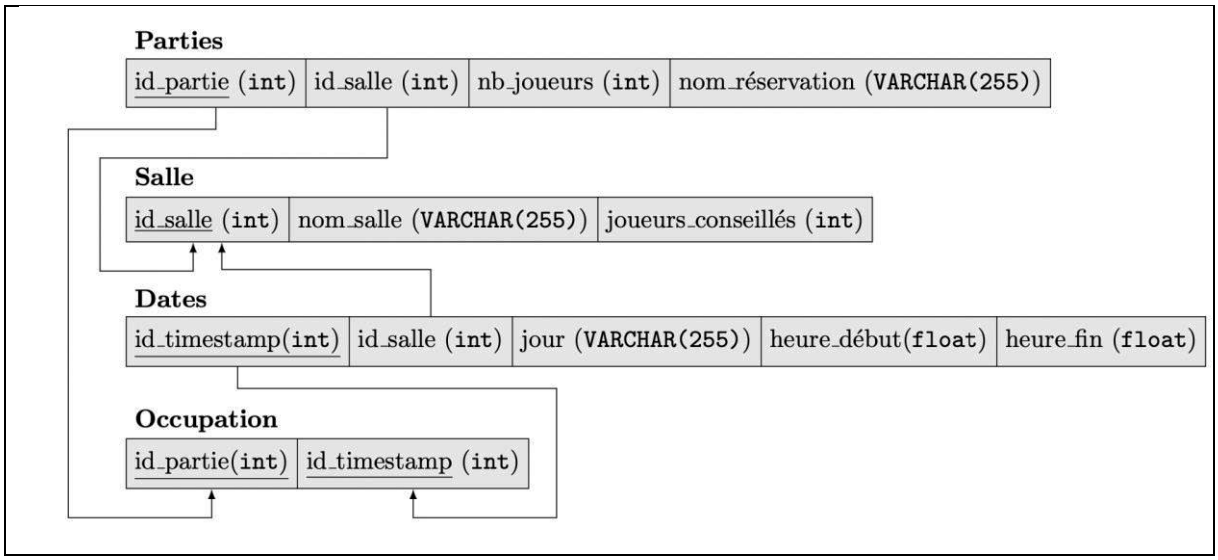


Figure 15 : Schéma relationnel de la base de données étudiée.

Les types des différents attributs des tables Parties, Salle, Dates et Occupation sont indiqués entre parenthèses. La description des différents attributs est disponible dans le document technique DT7. Les clés primaires de chacune des tables sont soulignées. On note les contraintes suivantes dans les données de la table :

- Une seule réservation est possible par salle pour une heure de démarrage donnée.
- L'heure de fin d'une session correspond à l'heure à laquelle les joueurs terminent réellement la partie. Lors de la réservation celle-ci est fixée à la valeur NULL puis remplacée par la véritable heure une fois la session terminée.

Un rappel de la syntaxe des requêtes SQL usuelles est fourni DT8.

Question 42 : Écrire la ou les requêtes SQL permettant d'insérer dans la table Parties une partie à 4 joueurs pour 1 heure réservée par Mr Durand dans la salle *Pirates des Antilles* ayant pour identifiant 3. On suppose que ce créneau a déjà été ajouté dans les tables Dates et Occupation et que l'identifiant de partie id_partie vaut 2030.

Certaines salles sont plus difficiles que d'autres et nécessitent une attention particulière du maître de jeu pour aider les participants notamment lorsqu'ils sont peu nombreux.

Question 43 : Écrire une requête permettant d'extraire les réservations dont le nombre de joueurs est inférieur ou égal à 4.

Le planning de travail des *Game Masters* est constitué à partir de la table Dates qui permet de calculer le nombre de créneaux réservés sur une journée et d'en déduire le nombre de maîtres du jeu à convoquer sur cette journée.

Question 44 : Écrire une requête qui renvoie le nombre de créneaux réservés le 2 mars 2025. Les dates sont encodées dans la base de données sous la forme de chaînes de caractères au format JJ/MM/AAAA.

Question 45 : Écrire une requête qui renvoie le meilleur score de la base de données (c'est-à-dire la durée la plus courte pour terminer la salle). La requête doit afficher le nom de la personne qui a réservé la salle, le nom de la salle terminée ainsi que l'heure de démarrage de la session.

On souhaite vérifier que les différentes salles sont de difficulté sensiblement équivalente et pour cela on choisit d'inspecter la durée moyenne d'une partie dans chacune des salles.

Question 46 : Écrire une requête qui renvoie le nom de chaque salle ainsi que la durée moyenne de la partie calculée sur l'ensemble des groupes qui ont joué dans cette salle. **Indiquer** ce qu'il faudrait modifier dans cette requête pour ne calculer la durée moyenne que sur les parties où les nombre de joueurs est au moins égal à celui recommandé pour la salle.

Une inspection manuelle de la base de données a révélé des erreurs dans l'enregistrement des heures de début et de fin de la table Dates. Dans certains cas, ceux-ci ont été permutés en raison par exemple d'une manipulation du Game Master qui force le redémarrage d'une partie du système de gestion d'une salle.

Question 47 : Écrire une requête qui renvoie l'ensemble des lignes de la Table Dates où une erreur peut être détectée sur les heures de début et de fin, soit parce que celles-ci sont inversées, soit parce que le créneau associé en chevauche un autre de la table.

Afin d'évaluer plus précisément la difficulté des énigmes et d'établir les records de vitesse par énigme et non par salle, la société souhaite modifier la base de données afin de stocker les instants auxquels chacune des énigmes d'une salle sont terminées. On considérera qu'une salle contient au plus 5 énigmes.

Question 48 : Proposer des modifications et/ou ajouts au schéma relationnel précédent afin de stocker les heures de début et de fin de chaque énigme à l'issue d'une session. Cette proposition devra contenir une table dont les attributs seront les heures de début et de fin de chacune des énigmes ainsi que d'autres attributs qui seront précisés.

Un écran affiché en salle d'attente de la société Kairos affiche un classement des meilleures équipes avec les statistiques suivantes :

Nom équipe	Salles terminées	Durée moyenne
Les petits malins	4	47.4
Escape Killer	4	52.5
Athena Rescue	3	49.6
Pirates Forever	3	51.3

Les résultats de toutes les parties effectuées dans l'Escape Game ont été stockés dans une table Résultats contenant les attributs nom_équipe (VARCHAR(255),nom_salle(VARCHAR(255)),durée_partie (float) indiquant respectivement le surnom de chaque équipe, le nom de la salle jouée ainsi que le temps enregistré par l'équipe pour finir les énigmes de celle-ci. Cette table a été constitué à partir de la base de données précédemment étudiée.

Question 49 : Écrire la requête permettant d'extraire les noms d'équipe, le nombre de salles terminées et la durée moyenne pour finir celles-ci des 4 meilleures équipes en respectant l'ordre choisi par la société Kairos : les équipes les mieux classées sont celles qui ont terminé le plus de salle puis la durée moyenne pour terminer les salles est utilisée comme critère secondaire pour trier les équipes.

Partie 6 : Evolution de l'architecture réseau

L'objectif de cette partie est de faire évoluer l'architecture réseau afin de pouvoir accueillir de nouvelles salles éventuellement déportées dans un autre bâtiment.

Actuellement, toutes les salles sont sur un même réseau dont l'adresse est la suivante : 192.168.17.0/24.

Dans l'optique d'accueillir de nouvelles salles qui pourront être sur le même site ou sur un site déporté, il est proposé de faire évoluer le réseau suivant le schéma de la Figure 16 :

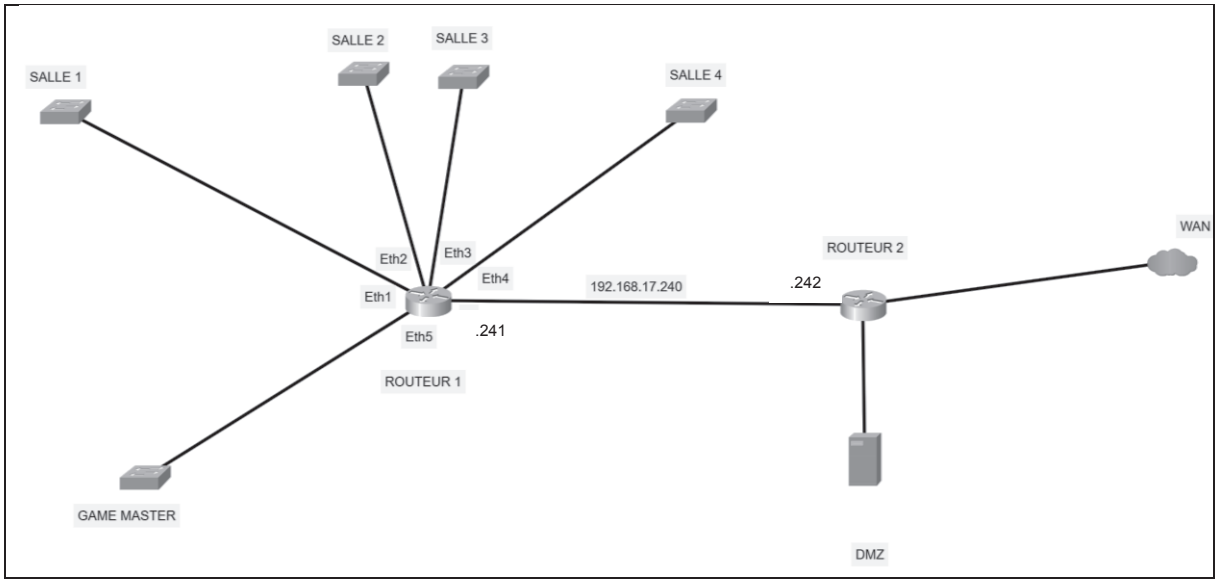


Figure 16 : Nouvelle topologie envisagée du réseau

Des sous-réseaux vont être créés pour chaque salle de jeu ainsi que pour la salle de supervision du Game Master en appliquant un nouveau masque au réseau actuel : 192.168.17.0/28
Dans un souci de protection des échanges avec l'extérieur, on créé une zone DMZ qui accueillera un serveur web (192.168.17.225 /28) et un serveur de données (192.168.17.226/28)

Question 50 : Donner le nombre total de sous réseaux réalisables avec ce nouveau masque.
Préciser combien de ces sous-réseaux vont être utilisés avec la configuration actuelle ainsi que le nombre d'équipements qui pourront être associés à chaque sous réseau.

Question 51 : Compléter la colonne « Adresse du réseau » du document ressource DR6 en indiquant l'adresse du réseau de chaque salle. Le plan d'adressage sera fait dans l'ordre croissant :
1er adresse de réseau disponible = salle 1, 2ème adresse de réseau disponible = salle 2,…, 5ème adresse de réseau disponible = salle Game Master

Tournez la page S.V.P.

Question 52 : Compléter le reste du tableau du document réponse DR6 en indiquant les plages d'adresse qui seront disponibles pour chacune des salles.

Par souci d'uniformisation, on donne la première adresse disponible du réseau aux interfaces du routeur 1

Question 53 : Compléter la table de routage du routeur 1 donnée sur le document réponse DR 7.

Question 54 : Donner le rôle de la DMZ et indiquer quelles sont les architectures de protection possibles.

Question 55 : Le ou les firewall mis en place assureront certaines protections. **Indiquer** les équipements qui seront accessibles depuis l'extérieur ?

Question 56 : Avec la configuration mise en place, **indiquer** le nombre de salles qui pourront être ajoutées sur le site.

Documents Techniques

DT1 : Documentation relative à l'utilisation des threads.

Création et exécution des threads

La routine **pthread_create()** crée un processus léger qui exécute la fonction « start_routine » avec l'argument arg et les attributs attr. Les attributs permettent de spécifier la taille de la pile, la priorité, la politique de planification, etc. Il y a plusieurs formes de modification des attributs. La routine pthread_create fait partie de la librairie **Pthreads** qui est un API de la norme POSIX. Avec cette libraire on crée un thread comme suit:

int pthread_create (pthread_t * thread, pthread_attr_t * attr, void * (* start_routine) (void *), void * arg);

Cette routine crée et lance le thread immédiatement en exécutant la fonction passée en troisième argument. L'exécution du thread se fait soit jusqu'à la fin de l'exécution de sa fonction ou bien jusqu'à son annulation. Il est possible d'utiliser une fonction avec plusieurs threads !

Si la création du thread réussit, la fonction **pthread_create** retourne 0 (zéro) et l'identifiant du thread nouvellement créé est stocké à l'adresse fournie en premier argument. En cas d'erreur, la valeur EAGAIN est retournée par la fonction **pthread_create()** s'il n'y a pas assez de ressources système pour créer un nouveau thread ou bien si le nombre maximum de threads définit par la constante **PTHREAD_THREADS_MAX** est atteint !

les paramètres de la fonction pthread_create

1. Le paramètre *thread est un pointeur de type **pthread_t :** le type pthread est un type opaque, sa valeur réelle dépend de l'implémentation ; sous Linux il s'agit en général du type unsigned long. Ce type correspond à l'identifiant du thread créé ; tout comme les processus Unix, chaque thread a son propre identifiant.
2. Le paramètre *attr_t est un pointeur de type **pthread_attr_t :** ce type est lui aussi un type opaque permettant de définir des attributs spécifiques pour chaque thread. Il faut savoir qu'on peut changer le comportement de la gestion des threads ; exemple, on peut les régler pour qu'ils tournent sur un système temps réel ; en générale on se contente des attributs par défaut et cet argument vaut NULL (valeur par défaut).
3. Chaque thread dispose d'une fonction à exécuter, c'est en même temps sa raison d'être ; Cet argument de la fonction pthread_creat est un pointeur de fonction ; il permet de transmettre un pointeur sur la fonction que le thread devra exécuter.
4. Le dernier argument représente un argument que l'on peut passer à la fonction que le thread doit exécuter.

Tournez la page S.V.P.

DT2 : Implémentation multithreadée des tâches de gestion des énigmes et du serveur.

```
#include <stdio.h>
#include <pthread.h>

// Function prototypes
void *callback_machine_etat(void *arg);
void *callback_serveur(void *arg);

int main() {
    pthread_t thread1, thread2;

    // Create threads for fa() and fb()
    if (pthread_create(&thread1, NULL, callback_machine_etat, NULL) != 0) {
        perror("pthread_create failed");
        return 1;
    }
    if (pthread_create(&thread2, NULL, callback_serveur, NULL) != 0) {
        perror(« pthread_create failed ») ;
        return 1 ;
    }
    while (1) {
        // Les deux threads précédents s'exécutent tant que la fonction main est active
        sleep(1) ; // La tâche du main ne fait rien de spécial.
    }
    // Cette ligne ne devrait jamais s'exécuter :
    return 0;
}

void *callback_machine_etat(void *arg) {
    while (1) {
        // Execute la fonction faisant tourner la machine à états
        salle_machine_etat();
    }
    pthread_exit(NULL);
}

void *callback_serveur(void *arg) {
    while (1) {
        // Exécute la fonction de gestion du serveur
        gestion_serveur();
    }
    pthread_exit(NULL);
}
```

DT3 : Documentation technique d'une ventouse.

BRICARD

95003

Sécurité, Qualité & Design:

95003 fait partie des ventouses exclusives et incontournables pour les professionnels de la sécurité.

Sécurité, qualité et design garantis grâce à l'utilisation de materiaux de haute qualité offrant une résistance maximale, le tout sans magnétisme résiduel ou corrosion.

Spécifications	
Montage	Apparent
Force de maintien réelle (testée) +/-10%:	~2430 N / ~ 545 lbf / ~243 kgf *
Tension:	24/48 V CC
Courant:	250/125 mA
Contrôle de verrouillage:	Contact Reed
NF S61 937:	Oui
Température de fonctionnement:	-40°C à +60°C
Dimensions:	268 x 48 x 25 mm
Poids:	~2 kg

* Suivant tests de gamme réalisés en laboratoire indépendant.

Tournez la page S.V.P.

DT4 : Extrait du fichier index.html

```
<!doctype html>
<html>
<head>
<title></title>
<meta charset="utf-8" />
<link rel="stylesheet" type="text/css" href="style.css"/>
<link rel="stylesheet" type="text/css" href="sweetalert.css"/>
<script type="text/javascript" src="room_description.js"></script>
<script type="text/javascript" src="ajax.js"></script>
<script type="text/javascript" src="camera.js"></script>
<script type="text/javascript" src="page.js"></script>
<script type="text/javascript" src="page_maintenance.js"></script>
<script type="text/javascript" src="page_preparation.js"></script>
<script type="text/javascript" src="page_game.js"></script>
</head>
<body onload="pageInit();">
<table width=100%>
 <tr>
  <td style="border:0px solid blue;">
   <div>
    <table width=100% border=0px>
     <tr align=center>
      <td>
       <button class="menu_button"
onclick="pageChangeRoomMode('standby');">ARRET</button>
       <button class="menu_button"
onclick="pageChangeRoomMode('maintenance');">MAINTENANCE</button>
       <br/>
       <button class="menu_button"
onclick="pageChangeRoomMode('preparation');">PREPARATION</button>
       <button class="menu_button"
onclick="pageChangeRoomMode('game');">JEU</button>
     </tr>
    </table>
   </div>
   <hr width="95%" align="center">
   <div id="pageContent">
   </div>
  </td>
 </tr>
</table>
</body>
</html>
```

DT5 : Technologie Ajax :

AJAX est une méthode utilisant différentes technologies ajoutées aux navigateurs web entre 1995 et 2005, et dont la particularité est de permettre d'effectuer des requêtes au serveur web et, en conséquence, de modifier partiellement la page web affichée sur le poste client sans avoir à afficher une nouvelle page complète. Cette architecture informatique permet de construire des applications web et des sites web dynamiques interactifs. AJAX est l'acronyme d'***asynchronous JavaScript and XML*** : *JavaScript et XML asynchrones*.

La méthode AJAX consiste à utiliser de manière conjointe diverses technologies normalisées ouvertes et disponibles sur la plupart des navigateurs du marché :

- JavaScript, un langage de programmation incorporé dans les navigateurs. Les programmes écrits dans ce langage sont exécutés par le navigateur ; il est utilisé en particulier pour exploiter le *XMLHttpRequest* et le *DOM*. C'est la clé de voûte de l'AJAX.
- Le DOM (sigle de « *Document Object Model* ») c'est une collection d'objets, utilisée pour l'affichage dynamique et l'interaction avec les données, où chaque objet représente un élément structurel ou visuel d'une page web ou d'un document *XML*. Il est utilisé à partir d'un langage de programmation orientée objet tel que JavaScript pour inspecter et modifier le contenu des pages web.
- XMLHttpRequest est un objet de programmation, utilisé dans les programmes en langage JavaScript pour assurer la communication entre le navigateur et un serveur web. Il est utilisé pour la communication asynchrone : envoyer les requêtes vers le serveur et déclencher des opérations lors de la réception de réponses de celui-ci.

XML (sigle de *anglais : Extensible Markup Language*) est un langage de balisage et JSON (sigle de *JavaScript Object Notation*) est un format de données inspirée de la syntaxe du langage JavaScript. Ils sont utilisés pour structurer les informations envoyées par le serveur web. Le format XML est accompagné de XSLT pour sa manipulation.

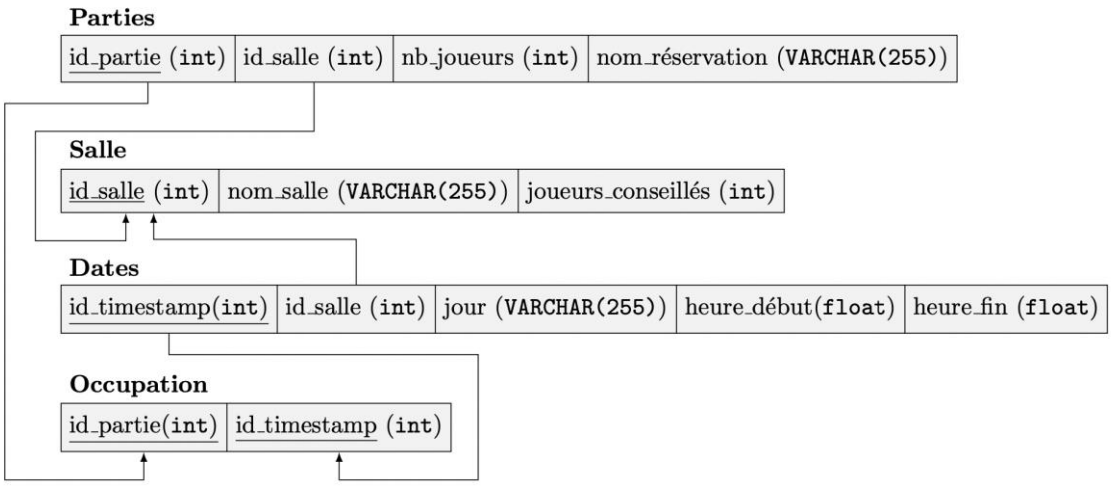
DT6 : Fonction transmettant la requête du Game Master au serveur

```
function ajaxCallUrl ( url, params, callback /*, ... */ )
{
 // url      : url à appeler pour le chargement des données
 // params   : paramètres http éventuels à passer au serveur
 // callback : fonction à appeler quand la réponse du serveur est arrivée
 // ... : paramètres éventuels à faire passer à la callback

 var xhr = new XMLHttpRequest();
 xhr.callback = callback;
 xhr.arguments = Array.prototype.slice.call( arguments, 3);
 xhr.onload = ajaxSuccess;
 xhr.onerror = ajaxError;
 xhr.open( "POST", url, true );
 xhr.send( params );}
```


Tournez la page S.V.P.

DT7 : Détail des attributs du schéma relationnel de la base de données.



- La table Parties contient le nom sous lequel les clients ont déposé une réservation, le nombre de joueurs qui seront présents ainsi que l'identifiant de la salle réservée et un identifiant unique pour chaque réservation.
- La table Salle décrit chaque salle de l'Escape Game en spécifiant son nom, le nombre de joueurs conseillés et un identifiant unique.
- La table Dates répertorie les créneaux horaires correspondant aux différentes réservations par salle.
- La table Occupation répertorie les parties et les créneaux horaires associés

Les heures sont enregistrées au format float : 17.5 code 17 heures et 30 minutes.

On présente ci-dessous les différents attributs des tables de la base de données.
Table Parties :
id_partie : Identifiant unique d'une session de jeu.
id_salle : Identifiant propre à chaque salle. A titre d'exemple la salle *Athena Station* a pour identifiant 1.
nb_joueurs : Nombre de joueurs indiqués dans la réservation.
nom_réservation : Nom de la personne qui a réservé la session dans la salle choisie.

Table Salle :
id_salle : Identifiant propre à chaque salle.
nom_salle : Nom de la salle (exemple : *Athena Station*)
joueurs_conseillés : Nombre de joueurs conseillés pour la salle.

Table Dates :
id_timestamp : Identifiant unique associé à chaque créneau horaire
id_salle : Identifiant de la salle où le créneau est reservé
jour : Jour de la réservation
heure_début : Heure de début de la réservation
heure_fin : Heure de fin de la réservation

Table Occupation :
id_partie : Identifiant de la partie correspondant à la réservation
id_timestamp : Identifiant du créneau où la salle a été réservée.

DT8 : Rappel de requêtes SQL usuelles

Utiliser (rendre active) une base de données existante	**USE** nom_de_la_base;
Créer une base de données	**CREATE DATABASE** nom_de_la_base;
Créer une table dans une base de données	**CREATE TABLE** nomTable (id **INT NOT NULL AUTO_INCREMENT**, champ1 **DOUBLE**, champ2 **VARCHAR**, champ3 **TIMESTAMP NOT NULL**, …, **PRIMARY KEY**(id)) ;
Écrire une nouvelle entrée dans une table de BDD	**INSERT INTO** nomTable(champ1, champ2) **VALUES**('valeur1', 'valeur2') ;
Modifier les informations de l'entrée dont le champ id = 51	**UPDATE** nomTable **SET** nomChamp1=10, valeur2=32 **WHERE** id=51 ;
Ajouter des nouveaux champs (colonnes) dans une table	**ALTER TABLE** nomTable **ADD** champ1 **DOUBLE**, **ADD** champ2 **BOOLEAN DEFAULT FALSE** ;
Sélectionner toutes les informations de la table	**SELECT** * **FROM** nomTable ;
Sélectionner seulement les informations d'un champ	**SELECT** nomChamp **FROM** nomTable ;
Sélectionner tous les champs de la table nomTable correspondant à deux critères	**SELECT** * **FROM** nomTable **WHERE** nomChamp1 = 'poste' **AND** nomChamp3 < 12 ;
Sélectionner sur plusieurs tables (jointure) nomTable1.nomChamp1 est clé primaire. nomTable2.nomChamp4 est une clé étrangère vers nomTable1	**SELECT** * **FROM** nomTable1 **JOIN** nomTable2 **ON** nom_table1.nomChamp1 = nom_table2.nomChamp4 ;
Appliquer une fonction d'agrégation (parmi **MIN**,**MAX**,**COUNT**,**AVG**) sur des valeurs d'un champs nomChamp1 regroupées par ensembles partageant une même valeur dans le champs nomChamp2	**SELECT MAX** (nomChamp1) **FROM** nomTable **GROUP BY** nomChamp2
Ordonner les résultats d'une requête par ordre croissant (**ASC**) ou décroissant (**DESC**) des valeurs d'un champ NomChamp1.	**ORDER BY** nomChamp1 **ASC** (instructions à ajouter en fin de requête)
Limiter le nombre d'entrées renvoyées par une requête en se retreignant aux *k* premières lignes où *k* est un entier.	**LIMIT** k (instruction à ajouter en fin de requête)

Les mots en gras dans la colonne de droite sont des mots réservés par le langage SQL.

Modèle CMEN v3

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Candidat :

Né(e) le : / /

Cadre réservé aux candidats de concours de recrutement et examens professionnels

Concours : ..... **Option / Section :** ..... **N° d'inscription :**

Cocher une seule case parmi les six types de concours suivants :

☐ externe ☐ 3e externe ☐ externe spécial ☐ interne ou 1er interne ☐ 2nd interne ☐ 2nd interne spécial

Cocher public OU privé UNIQUEMENT pour les concours enseignants : ☐ public ☐ privé

Examen professionnel pour l'avancement au grade de : .....

Cadre réservé aux candidats d'examens et du concours général

Examen : ..... **Série / Spécialité :** .....

Epreuve - Matière : ..... **Session :** .....

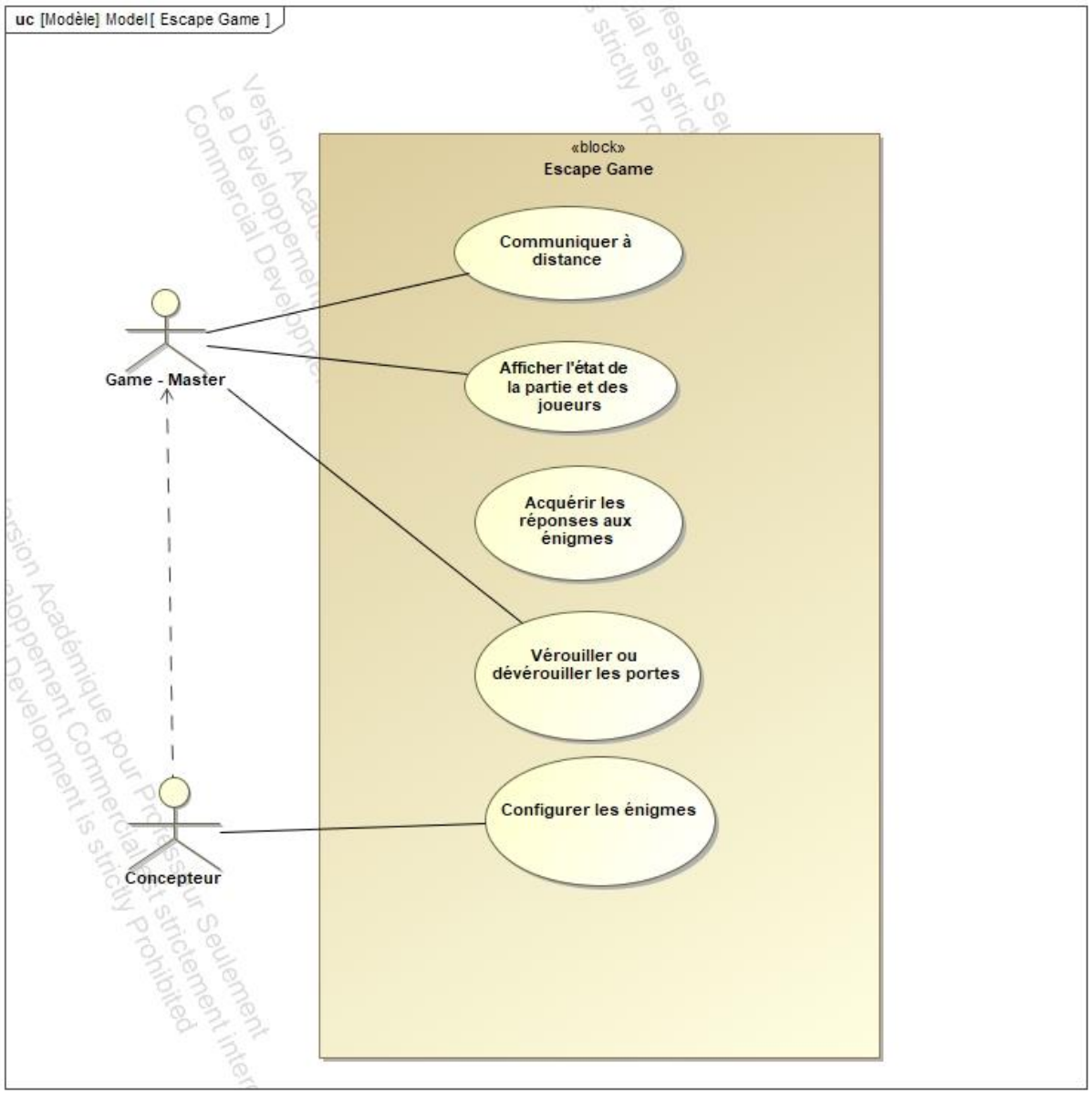
EDE NUM 1

DR1 - DR2

Tous les documents réponses sont à rendre, même non complétés.

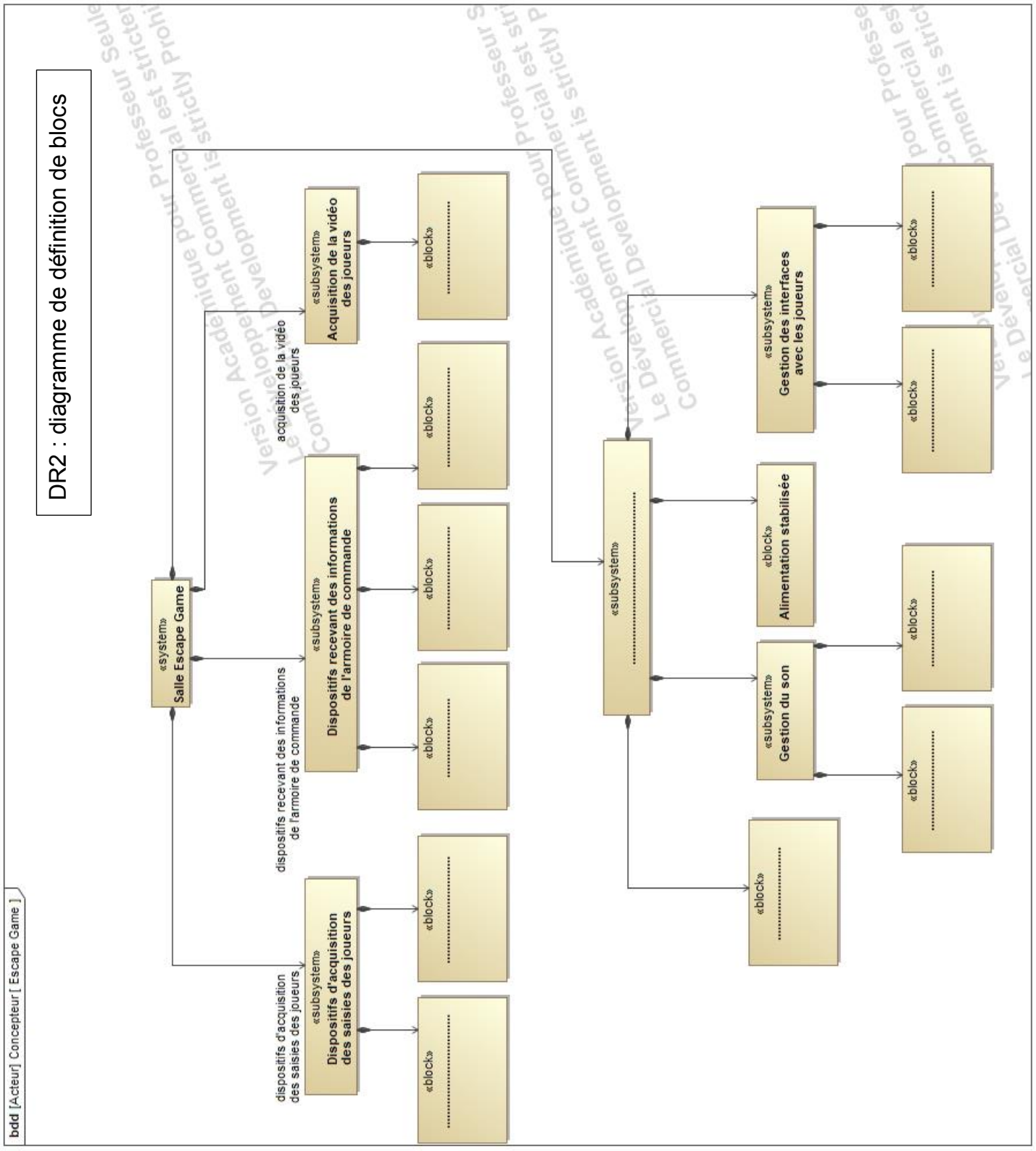
NE RIEN ECRIRE DANS CE CADRE

DR1 : diagramme des cas d'utilisation



Tournez la page S.V.P.

DR2 : diagramme de définition de blocs



Modèle CMEN v3

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Candidat :

Né(e) le : / /

Cadre réservé aux candidats de concours de recrutement et examens professionnels

Concours : ..... **Option / Section :** ..... **N° d'inscription :**

Cocher une seule case parmi les six types de concours suivants :

☐ externe ☐ 3e externe ☐ externe spécial ☐ interne ou 1er interne ☐ 2nd interne ☐ 2nd interne spécial

Cocher public OU privé UNIQUEMENT pour les concours enseignants : ☐ public ☐ privé

Examen professionnel pour l'avancement au grade de : .....

Cadre réservé aux candidats d'examens et du concours général

Examen : ..... **Série / Spécialité :** .....

Epreuve - Matière : ..... **Session :** .....

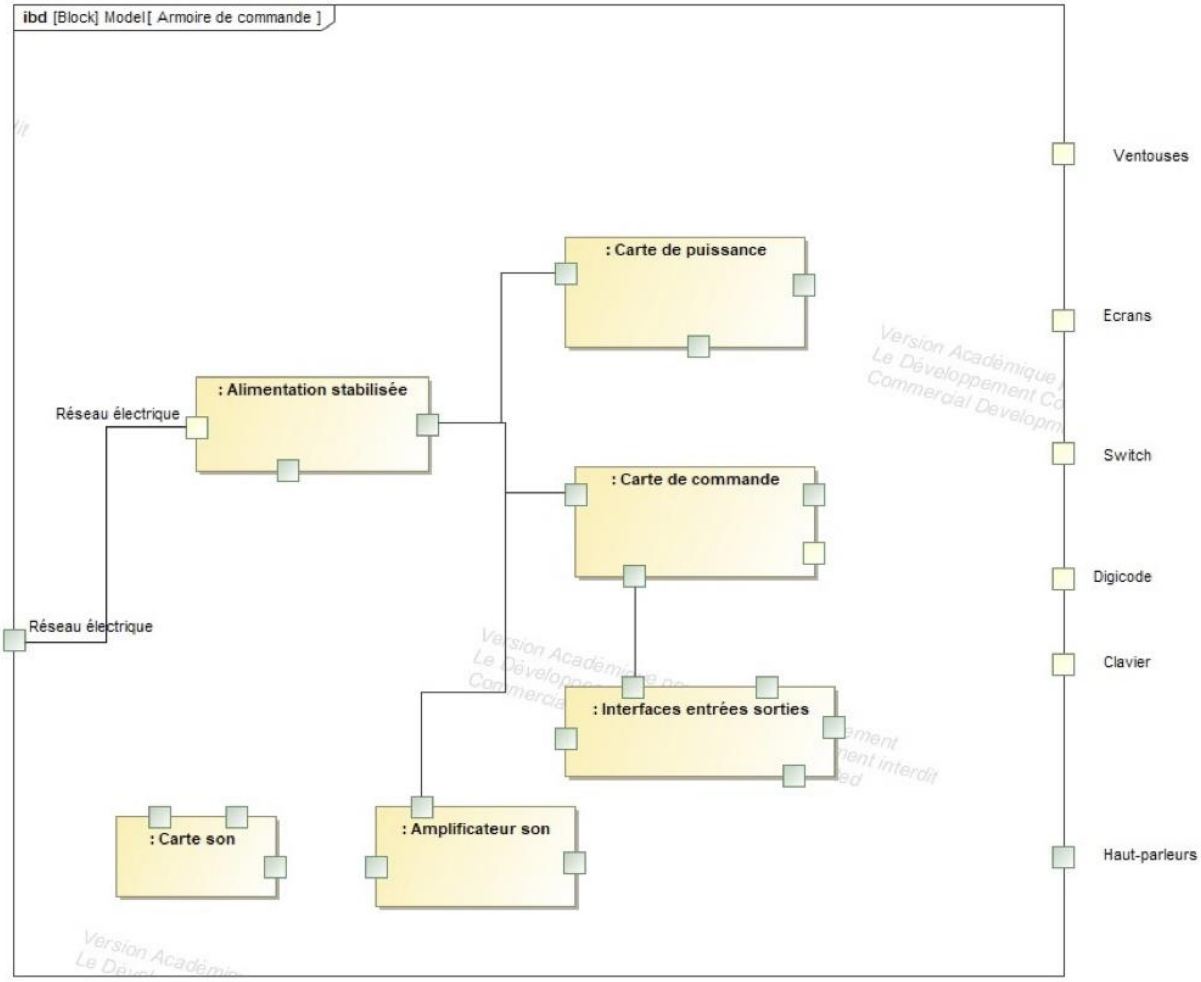
EDE NUM 1

DR3 - DR4

Tous les documents réponses sont à rendre, même non complétés.

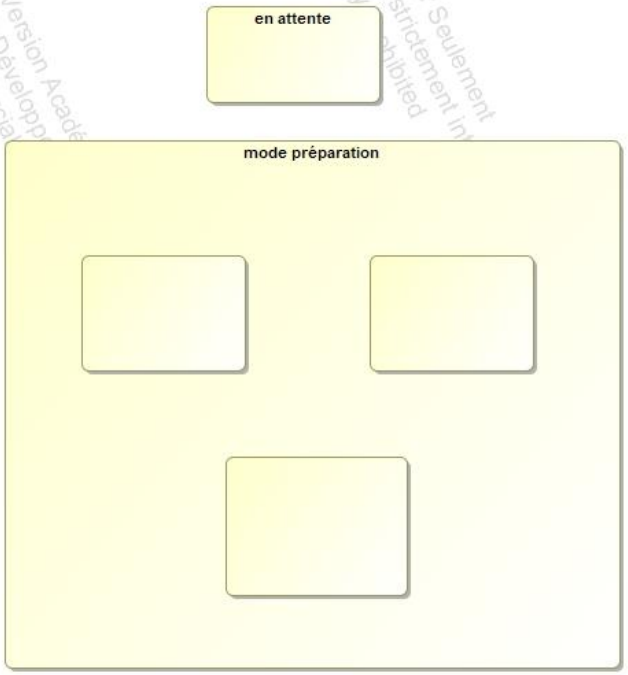
NE RIEN ECRIRE DANS CE CADRE

DR3 : diagramme de blocs internes



Tournez la page S.V.P.

DR4 : Diagramme d’état



Modèle CMEN v3

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Candidat :

Né(e) le : / /

Cadre réservé aux candidats de concours de recrutement et examens professionnels

Concours : ..... **Option / Section :** ..... **N° d'inscription :**

Cocher une seule case parmi les six types de concours suivants :

☐ externe ☐ 3e externe ☐ externe spécial ☐ interne ou 1er interne ☐ 2nd interne ☐ 2nd interne spécial

Cocher public OU privé UNIQUEMENT pour les concours enseignants : ☐ public ☐ privé

Examen professionnel pour l'avancement au grade de : .....

Cadre réservé aux candidats d'examens et du concours général

Examen : ..... **Série / Spécialité :** .....

Epreuve - Matière : ..... **Session :** .....

EDE NUM 1

DR5 - DR6 - DR7

Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

DR 5

```
int _cgiExecute ( int nbArg, char* args[], char* buffer, u32 clientIp )
{
 int resultat; // permet de stocker le résultat renvoyé par les fonctions cgi
 int i;
// Les arguments en entrée de _cgiExecute sont à passer directement en argument
de la fonction cgi Souhaitée.
// Le nom (name) de la commande cgi à exécuter est situé dans args[0]
// On rappelle que la fonction int strcmp(s1,s2) renvoie 0 si deux chaînes de
caractère sont égales et une valeur non nulle sinon.

// Votre code :
```


Tournez la page S.V.P.

DR 6 : Tableau d'adressage

	Adresse du réseau	Adresse mini	Adresse maxi
Salle 1			
Salle 2			
Salle 3			
Salle 4			
Salle Game Master			

DR 7 : Table de routage

Destination	Adresse réseau	Masque	Adresse interface	Passerelle
Salle 1				
Salle 2				
Salle 3				
Salle 4				
Game Master				
DMZ				