Magix API v1.1.0

## Objet du document

Ce document contient toutes les informations afin d’être en mesure de se connecter au serveur du jeu Magix.

## Table des matières

[Objet du document 1](#_Toc38618417)

[Table des matières 1](#_Toc38618418)

[Le jeu Magix 2](#_Toc38618419)

[Page d’authentification 2](#_Toc38618420)

[Page « lobby » 3](#_Toc38618421)

[Page de jeu 3](#_Toc38618422)

[L’enregistrement 4](#_Toc38618423)

[Créer son « Deck » 4](#_Toc38618424)

[Le jeu 4](#_Toc38618425)

[L’objectif 4](#_Toc38618426)

[Les cartes 4](#_Toc38618427)

[Temps accordé pour jouer son tour 4](#_Toc38618428)

[Se connecter à l’API 5](#_Toc38618429)

[La boîte de chat (clavardage) 6](#_Toc38618430)

[Styliser votre boîte de chat 6](#_Toc38618431)

[Les services disponibles (l’API) 7](#_Toc38618432)

[Connexion au serveur 7](#_Toc38618433)

[Déconnexion du serveur 7](#_Toc38618434)

[Créer/Joindre une partie 8](#_Toc38618435)

[L’état de la partie en cours 9](#_Toc38618436)

[Pour faire une action (jouer une carte, terminer son tour, attaquer) 11](#_Toc38618437)

[Autres notes importantes 12](#_Toc38618438)

[Le quasi temps-réel 12](#_Toc38618439)

[Distinguer un message texte d’un objet dans une variable 12](#_Toc38618440)

[Observer une partie (optionnel) 12](#_Toc38618441)

## Le jeu Magix

Magix est un serveur d’un jeu de cartes basé sur la technologie PHP, HTML, Node, JavaScript, MongoDB et AJAX. Celui-ci peut être joué contre l’ordinateur ou un autre joueur.

L’API, qui est disponible aux étudiants, permet de concevoir un jeu avec une interface graphique au choix. Voici un exemple d’interface.

### Page d’authentification

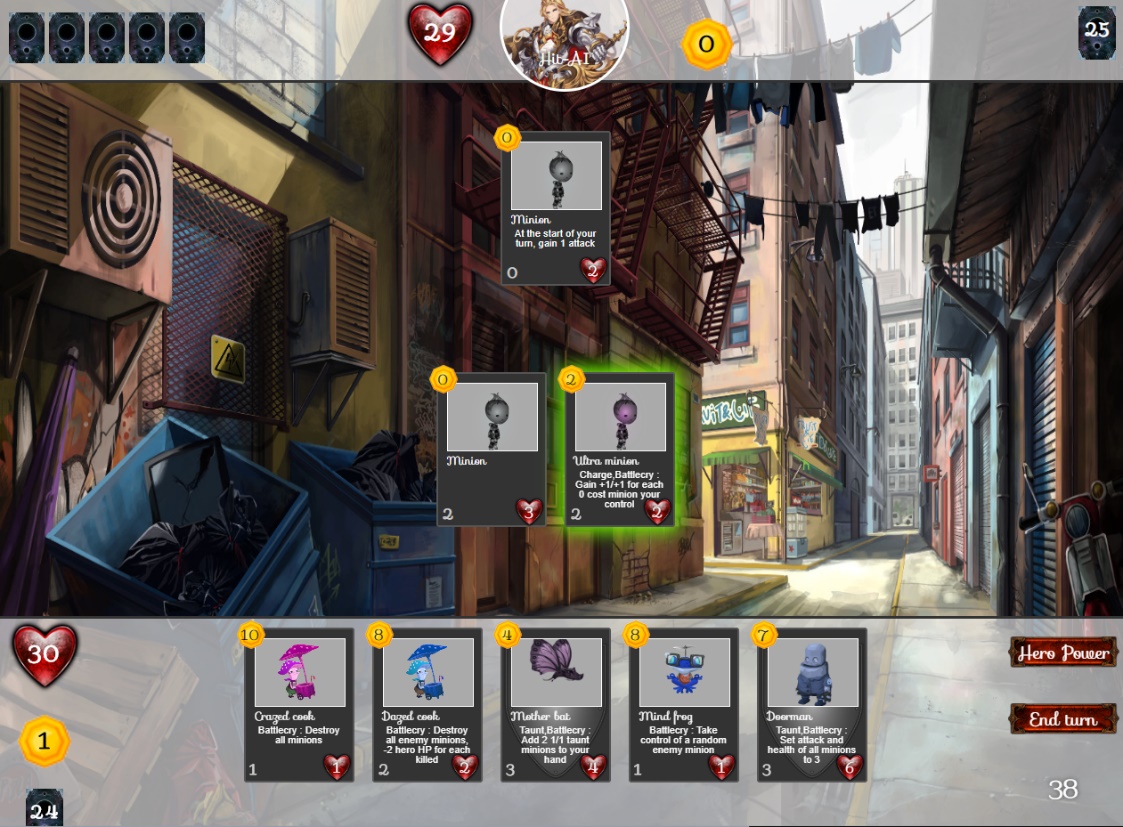


**Note importante** : Vous ne devez pas ajouter le champ « Clé du prof ». Cela ne sert qu’à mon thème à moi, empêchant les étudiants de se connecter dans mon thème et d’analyser/copier tout mon code JavaScript.

### Page « lobby »



### Page de jeu



Le commencement

Avant même de se connecter à l’API, l’étudiant doit avoir terminé de créer son compte sur le serveur Magix.

### L’enregistrement

Pour ce faire, il doit aller au lien suivant, et remplir le formulaire d’inscription :

<https://magix.apps-de-cours.com/server/#/signup>

### Créer son « Deck »

Un deck est un ensemble de 30 cartes sélectionnés par l’utilisateur. Il faut donc se connecter sur le site Web, puis aller dans la section « Deck » pour créer son deck. Lorsque votre deck est terminé, vous pouvez développer votre version du jeu.

## Le jeu

### L’objectif

Le but du jeu est de réduire la vie de l’adversaire à zéro, en attaquant avec ses cartes. Le jeu est fortement inspiré du jeu « Hearthstone » de la compagnie Blizzard.

### Les cartes

L’état d’une carte sur le jeu

Une carte ayant son state « IDLE » peut être jouée, alors qu’une carte « SLEEP » ne peut être joué avant le prochain tour.

Placer une carte sur le jeu

Pour jouer une carte qui se trouve dans votre main, vous devez avoir suffisamment de « mp ».

Une carte « charge »

Ce type de carte déposée sur le jeu peut immédiatement attaquer, alors que les autres doivent attendre au tour suivant.

Une carte « taunt »

L’adversaire ne peut attaquer le héro où les autres cartes derrières les cartes « taunt ».

Une carte « stealth »

L’adversaire ne peut attaquer une carte « stealth » en la ciblant avec un minion. Lorsqu’une carte « stealth » attaque, elle perd ce statut.

### Temps accordé pour jouer son tour

Un joueur peut prendre jusqu’à 50 secondes pour jouer son tour. Lorsqu’il a terminé de faire ses actions, il peut cependant faire l’action « END\_TURN » pour le terminer plus rapidement.

## Se connecter à l’API

Pour créer un jeu, il est nécessaire de communiquer avec le serveur de Magix. Vous devrez donc faire appel à son API.

Voici un exemple d’appel qui permet de faire l’opération « signin ». Il est **FORTEMENT** conseillé d’utiliser cette fonction tel quel et de la placer dans « CommonAction ».

/\*\*

\* data = array('key1' => 'value1', 'key2' => 'value2');

\*/

public function callAPI($service, array $data) {

$apiURL = "https://magix.apps-de-cours.com/api/" . $service;

$options = array(

'http' => array(

'header' => "Content-type: application/x-www-form-urlencoded\r\n",

'method' => 'POST',

'content' => http\_build\_query($data)

)

);

$context = stream\_context\_create($options);

$result = file\_get\_contents($apiURL, false, $context);

if (strpos($result, "<br") !== false) {

var\_dump($result);

exit;

}

return json\_decode($result);

}

Exemple d’appel (à intégrer dans votre action)

$data = [];

$data["username"] = "Falcor";

$data["password"] = "AAAaaa111";

$result = parent::callAPI("signin", $data);

if ($result == "INVALID\_USERNAME\_PASSWORD") {

// err

}

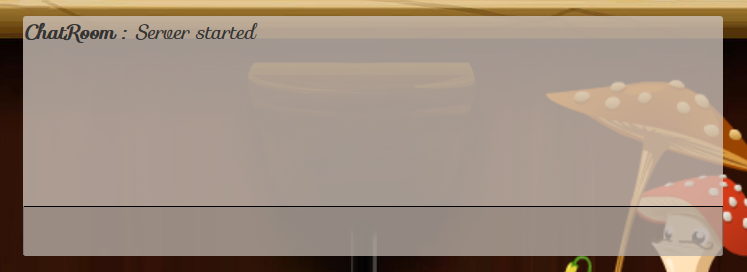
else {

// Pour voir les informations retournées :

$key = $result->key;

}

## La boîte de chat (clavardage)



Pour faire apparaitre la boîte de *chat* dans votre page Web, il s’agit d’injecter votre clé de session (reçu lors de l’authentification) dans l’extrait de code HTML suivant :

<iframe style="width:700px;height:240px;"

src="https://magix.apps-de-cours.com/server/#/chat/**votre-clé-ici**">

</iframe>

### Styliser votre boîte de chat

Afin de modifier le style de votre boîte de chat, il s’agit de faire ce qui suit.

1. Dans la création du iframe, ajouter l’événement *onload*.

<iframe style="width:700px;height:240px;" **onload="applyStyles(this)"**

src="https://magix.apps-de-cours.com/server/#/chat/votre-clé-ici">

</iframe>

1. Dans votre JavaScript, déclarer la fonction « applyStyles() ». Voici un exemple :

const applyStyles = iframe => {

let styles = {

fontColor : "#333",

backgroundColor : "rgba(87, 41, 5, 0.2)",

fontGoogleName : "Sofia",

fontSize : "20px",

}

iframe.contentWindow.postMessage(JSON.stringify(styles), "\*");

}

Notes :

* Si vous désirez modifier la police de caractères, utilisez « fontGoogleName ». Cela doit correspondre au nom d’un font disponible à cet endroit : <https://fonts.google.com/>
* Il n’y a que les styles plus haut qui sont acceptés.

## Les services disponibles (l’API)

### Connexion au serveur

Pour faire une authentification auprès du serveur, il faut avoir complété son inscription (voir plus haut). Lors d’une connexion, le serveur vous retournera une clé. Celle-ci doit être conservée en session car tous les autres appels au serveur en dépendent. Cette clé permet au serveur de « vous reconnaitre » (savoir qui vous êtes).

|  |  |  |
| --- | --- | --- |
| Nom du service | signin |  |
| Paramètres | username  password | Le nom de votre personnage  Votre mot de passe |
| Retour (succès) | Informations de votre personnage et la clé de session | La clé aura cette forme (50 car.):  « afsc9sflasmknc5lkntasd9yhcbasdfnasd9fcn » |
| Retour (erreur) | "INVALID\_USERNAME\_PASSWORD" | - Erreur d’authentification |

Pour savoir comment implémenter ce service, veuillez regarder l’exemple plus haut.

### Déconnexion du serveur

Lorsque vous êtes connectés, vous pouvez vous déconnecter en passant au service votre clef de session.

|  |  |  |
| --- | --- | --- |
| Nom du service | signout |  |
| Paramètres | key | Votre clé  *doit être envoyée dans un tableau, même s’il n’y a que la clé en paramètre. Exemple :*  *$data = [];*  *$data[“key”] = “asfsdafdsf54879y…”;* |
| Retour (succès) | "SIGNED\_OUT" |  |
| Retour (erreur) | "INVALID\_KEY" |  |

### Créer/Joindre une partie

Permet de jouer une partie contre une autre personne (pvp), ou contre l’ordinateur (training).

|  |  |  |
| --- | --- | --- |
| Nom du service | games/auto-match |  |
| Paramètres | key  type  mode (optionnel)  privateKey (optionnel) | Votre clé  "PVP" ou "TRAINING"  "STANDARD" ou "COOP"  Permet de créer une partie privée entre 2 joueurs, en utilisant un mot de passe pour accéder à la partie. *Vous n’avez pas besoin d’intégrer ce paramètre, il est optionnel.* |
| Retour (succès) | "JOINED\_PVP"  "CREATED\_PVP"  "JOINED\_TRAINING" |  |
| Retour (erreur) | "INVALID\_KEY"  "INVALID\_GAME\_TYPE”  "DECK\_INCOMPLETE"  "MAX\_DEATH\_THRESHOLD\_REACHED" | Lors de tournoi seulement. Lorsque vous avez cette erreur, vous êtes mort! |

Après cet appel, vous êtes dans une partie, *game on!*

L’idée est donc de faire en sorte qu’après l’appel de ce service, vous allez sur jeu.php (ou game.php), et commencez à faire des appels pour avoir l’état du jeu (voir page suivante).

### L’état de la partie en cours

Lorsque le personnage est dans la partie, il faut demander régulièrement son état afin de savoir si votre vie a diminué, quelles sont les cartes dans votre main, etc.

Il y doit y avoir un délai de 1 seconde minimum entre chaque appel. Autrement, l’appel pourrait être refusé et vous pourriez même être déconnecté!

|  |  |  |
| --- | --- | --- |
| Nom du service | games/state |  |
| Paramètres | key | Votre clé |
| Retour (succès) | "WAITING"  ou  "LAST\_GAME\_WON"  ou  "LAST\_GAME\_LOST"  ou  Un document JSON contenant l’état de la partie (voir exemple plus bas) | - En attente d’un autre joueur  - La partie n’existe plus, mais la dernière partie jouée a été gagnée  - La dernière partie n’existe plus et vous l’aviez perdue. |
| Retour (erreur) | "INVALID\_KEY" |  |

**Éviter d’être banni du serveur et les délais d’appel**

Pour le service « games/state » : un délai de 1 seconde minimum entre chaque appel

Il est donc nécessaire d’attendre le retour de l’appel du service avant d’en lancer un autre!

Il serait donc une bonne approche de faire un fichier JavaScript (ex : game.js) et d’y mettre le code suivant, qui ira chercher l’état du jeu à chaque seconde.

const state = () => {

fetch("ajax-state.php", { // Il faut créer cette page et son contrôleur appelle

method : "POST", // l’API (games/state)

credentials: "include"

})

.then(response => response.json())

.then(data => {

console.log(data); // contient les cartes/état du jeu.

setTimeout(state, 1000); // Attendre 1 seconde avant de relancer l’appel

})

}

window.addEventListener("load", () => {

setTimeout(state, 1000); // Appel initial (attendre 1 seconde)

});

Exemple de résultat (JSON)

{

"remainingTurnTime":24,

"yourTurn":true,

"heroPowerAlreadyUsed" : false,

"hp":30,

"mp":0,

"maxMp":1,

"hand":[

{"id":4,"cost":2,"hp":3,"atk":2,"mechanics":[], "uid":3,"baseHP":3},

{"id":22,"cost":7,"hp":7,"atk":7,"mechanics":[],"uid":5,"baseHP":7},

{"id":10,"cost":3,"hp":3,"atk":3,"mechanics":["taunt", "charge"],"uid":6,"baseHP":3}

],

"board":[

{"id":2,"cost":1,"hp":1,"atk":2,"mechanics":[],"uid":7,"baseHP":1,"state":"SLEEP"}

],

"welcomeText" : "My life for Aiur!",

"heroClass" : “Warrior",

"remainingCardsCount":24,

"opponent":{

"username":"Dummy-AI",

"heroClass":"Hunter",

"hp":30,

"mp":0,

"board":[],

welcomeText : "Die, maggot!",

"remainingCardsCount":24,

"handSize" : 3

}

}

### Pour faire une action (jouer une carte, terminer son tour, attaquer)

|  |  |  |
| --- | --- | --- |
| Nom du service | games/action |  |
| Paramètres | key  type  ou  type  ou  type  uid  ou  type  uid  targetuid | Votre clé  "END\_TURN"  "HERO\_POWER"  "PLAY" et  ex : 23 (identifiant unique de la carte lors de la partie)  "ATTACK"  uid de la carte  uid de la carte attaquée ou 0 (zéro) pour le héros adverse |
| Retour (succès) | L’état de la partie, voir page suivante |  |
| Retour (erreur) | "INVALID\_KEY"  "INVALID\_ACTION"  "ACTION\_IS\_NOT\_AN\_OBJECT"  "NOT\_ENOUGH\_ENERGY"  "BOARD\_IS\_FULL "  "CARD\_NOT\_IN\_HAND"  "CARD\_IS\_SLEEPING"  "MUST\_ATTACK\_TAUNT\_FIRST"  "OPPONENT\_CARD\_NOT\_FOUND"  "OPPONENT\_CARD\_HAS\_STEALTH"  "CARD\_NOT\_FOUND"  "ERROR\_PROCESSING\_ACTION"  "INTERNAL\_ACTION\_ERROR"  "HERO\_POWER\_ALREADY\_USED" | Action invalide  Mauvaise structure de données  La carte coûte trop cher à jouer  Pas assez de place pour la carte  La carte n’est pas dans votre main  Carte ne peut être jouée ce tour-ci  Une carte taunt empêche ce coup  La carte attaquée n’est pas présente sur le jeu  La carte ne peut être attaquée directement tant qu’elle possède « stealth »  La carte cherchée (uid) n’est pas présente  Erreur interne, ne devrait pas se produire  Autre erreur interne, ne devrait pas se produire  Pouvoir déjà utilisé pour ce tour |

## Autres notes importantes

Voici quelques notes importantes concernant le serveur Magix.

### Le quasi temps-réel

Étant donné qu’AJAX est utilisé au lieu des Web Sockets pour le jeu, le jeu ne fonctionne pas tout-à-fait en temps réel. C’est donc normal que lorsque l’un des joueurs fait une action, un autre joueur ne la voit pas instantanément.

### Distinguer un message texte d’un objet dans une variable

Voici un exemple de code qui permet de voir si la variable est un objet ou une simple chaîne de caractères. Ça peut être utilisé pour savoir si la partie est terminé ou pas

if (typeof maVariable !== "object") {

if (maVariable == "GAME\_NOT\_FOUND") {

// Fin de la partie. Est-ce que j’ai gagné? Je dois appeler user-info

}

}

else {

// maVariable est un objet. On pourrait faire, par exemple, maVariable.game.hp ou

// maVariable.player.mp

}

### Observer une partie (optionnel)

Il est possible d’observer une partie en utilisant le service suivant, qui fonctionne exactement comme le service « games/state », que l’on appelle à chaque seconde.

|  |  |  |
| --- | --- | --- |
| Nom du service | games/observe |  |
| Paramètres | key  username | Votre clé  Nom du joueur à observer |
| Retour (succès) | "WAITING"  ou  "LAST\_GAME\_WON"  ou  "LAST\_GAME\_LOST"  ou  "NOT\_IN\_GAME"  ou  Un document JSON contenant l’état de la partie (voir exemple plus bas) | - En attente d’un autre joueur  - La partie n’existe plus, mais la dernière partie jouée a été gagnée  - La dernière partie n’existe plus et vous l’aviez perdue.  - Le joueur n’est pas dans une partie |
| Retour (erreur) | "INVALID\_KEY" |  |