Documentation Officielle

DROP DA BOMB

TABLE DES MATIERES

INTRODUCTION

Drop Da Bomb est un jeu de plateau/cartes en ligne.

Il se base sur un ensemble de cartes (*Cartes*) collectionnables (*Inventaire*) par un joueur (*Compte*) afin de composer une ou plusieurs « main de jeu » (*Deck*).

Ces cartes peuvent être obtenues de diverses manières mais principalement via la boutique (*Shop*) qui permet d'acheter avec différentes monnaies des lots de cartes (*Packs*). Il existe différents types et raretés pour les cartes.

Une partie (*Match*) oppose deux joueurs, dans une arène spécifique avec des règles et des modes de jeu précis ; à tour de rôle les joueurs peuvent utiliser les cartes de leur Deck alors en main pour tenter de faire basculer l'objectif (« la bombe ») dans le camp de leur adversaire.

Si la bombe explose ou se trouve dans le camp d'un joueur à la fin du temps imparti, la partie est perdue par celui-ci ; elle est gagnée par l'autre qui pourra alors remporter des récompenses (Cartes, Monnaie Virtuelle, etc..).

Ce jeu s'inspire par moments des jeux du même type devenus aujourd'hui des grands classiques (Yu-Gi-Oh, Clash Royale, HearthStone) mais se démarque par bien des aspects afin de se créer une véritable identité, un gameplay unique et tente de se faire sa place parmi les classiques cités précédemment.

Ce projet concerne donc la réalisation de ce jeu ainsi que de toutes ses composantes à savoir :

- Réalisation du jeu pour clients web
- Réalisation d'une interface de gestion pour clients web
- Réalisation d'une interface web pour administrateurs (optionnel)

A plus long terme, les objectifs sont de développer une version « application » pour smartphones, ainsi que de maintenir et faire évoluer le jeu au travers de nouveaux contenus, nouveaux modes de jeux, évènements, etc...

LE JEU EN DETAIL

LES REGLES

Préparation : Le joueur possède deux Decks, qui sont composés de 8 cartes uniques.

Un seul de ces deux decks est actif. C'est ce deck qui sera utilisé lors d'une partie.

Le joueur peut modifier autant de fois que nécessaire ses decks, depuis son interface web de gestion de l'inventaire et des decks, en échangeant une carte du deck avec une carte disponible dans son inventaire.

Lancement d'un match : Un match est lancé lorsque deux joueurs souhaitant jouer sont trouvés.

Déroulement d'un match : Un match se déroule tour par tour. Chaque joueur démarre avec 4 cartes en main (tirées au hasard dans leurs decks respectifs), un montant de poudre égal et tous les emplacements de jeu vide.

La partie se termine lorsque la bombe « explose » dans le camp d'un joueur. Une barre située en haut de l'écran indique la position de la bombe. Elle se trouve initialement à la position 0. Les bornes indiquent le camp d'un joueur, disons « -100 » pour la borne de mon camp, et « +100 » pour la borne du camp adverse (Les valeurs sont ici données arbitrairement et sont susceptibles d'être différentes de celles réellement appliqués en jeu). Le gagnant est celui qui fait exploser la bombe dans le camp adverse.

A chaque tour (15 secondes par tour) le joueur peut poser autant de cartes qu'il veut, en sachant que chaque carte à un cout en poudre, et que chaque tour ne régénère qu'un montant précis de poudre.

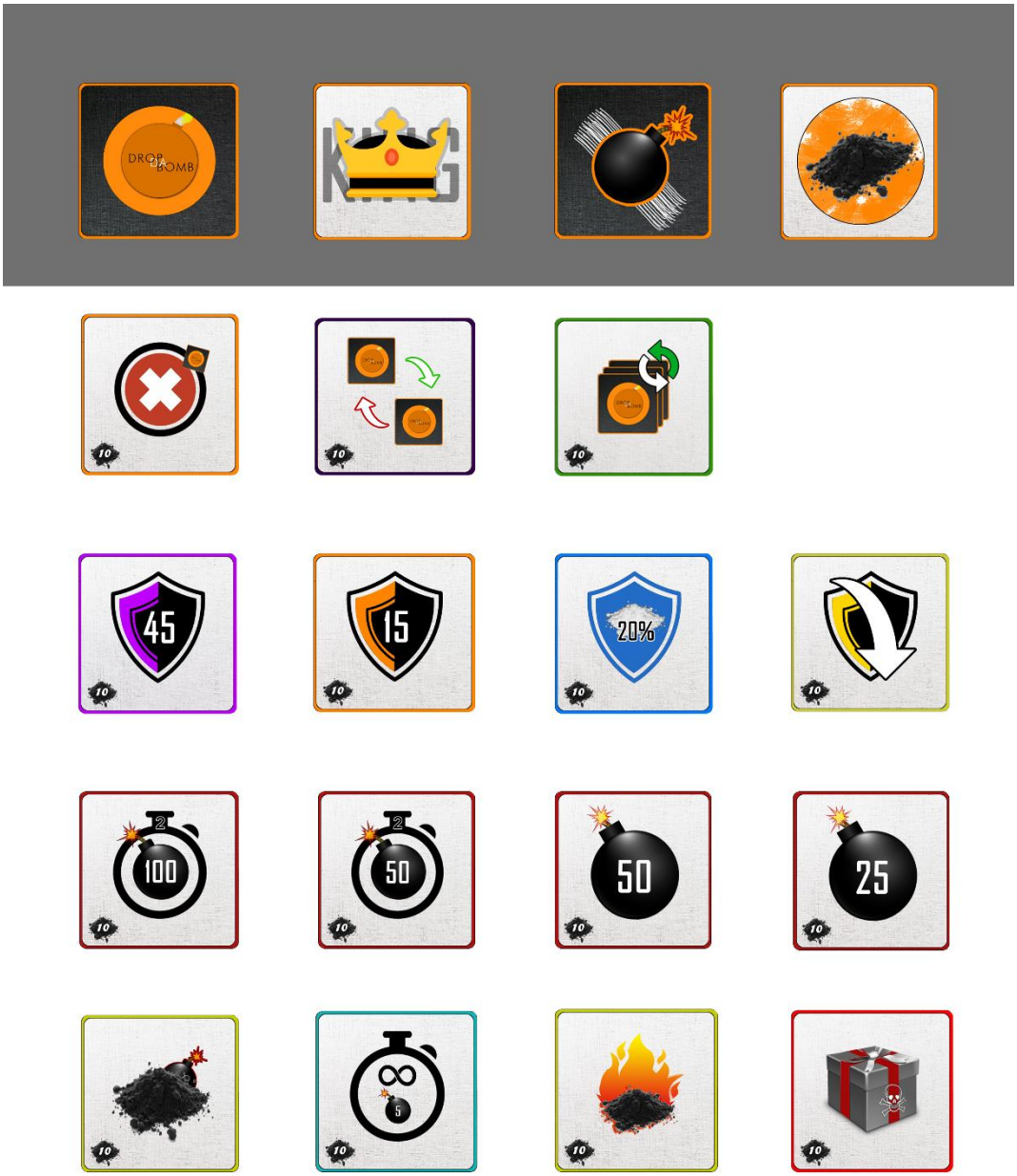
Chaque fois qu'une carte est posée, une nouvelle carte de son deck est piochée, et ceci de manière cyclique (la carte alors posée se place tout en bas du deck des cartes après utilisation).

Il existe différents types de cartes tels que les cartes d'attaque directe (qui font avancer ou reculer la position de la bombe sur la barre), les cartes à effets, les cartes pièges, etc.. Plus de détails ci-après.

La « zone de jeu » d'un joueur comporte 5 emplacements qui peuvent accueillir les cartes à retardements, les cartes à effets et les cartes pièges. Ces emplacements sont à utiliser avec précaution car une fois la zone pleine, il n'existe pas de moyen de supprimer volontairement une carte posée !

Voir également les sections suivantes : Les cartes et Les modes de jeu

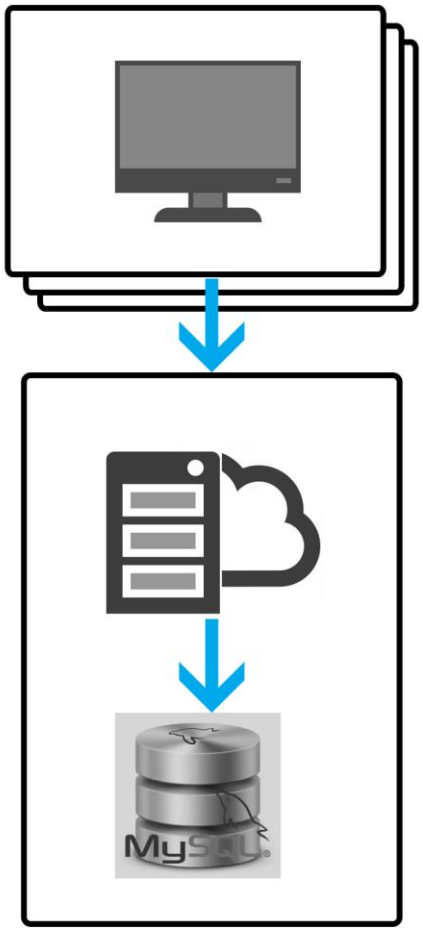
LES CARTES



LES MODES DE JEU

En l'état, aucune limite de temps n'est fixée, et tous les joueurs quelques soit leurs niveaux peuvent se rencontrer. Dans les futures évolutions du jeu, des niveaux de joueurs seront créés, et des « arènes » différentes permettront aux joueurs de jouer uniquement contre des joueurs d'un même niveau. De plus, différents types de rencontres sont imaginables (matchs amicaux, matchs classiques, matchs de tournoi, etc…) avec pour chaque mode des règles spécifiques. Ces fonctionnalités concernent les prochaines versions du jeu, nous parlerons donc pour l'instant du seul mode de jeu disponible.

STRUCTURE DU PROJET



Le projet DropDaBomb est basé sur une architecture Client-Serveur classique.

Le client est un client web, il est donc ouvert à tous les systèmes d'exploitation, tous les navigateurs, quel que soit le support.

Le serveur est un serveur distant, adressé par une IP publique, qui repose sur deux technologies : *Node.js* pour l'aspect purement serveur (détaillé dans les parties suivantes) et *MySQL* pour le serveur Base de Données. Seul le serveur *Node.js* est autorisé à échanger avec la base de données, le client communique donc uniquement avec le serveur *Node.js*.

DANS UN SOUCI DE SIMPLIFICATION ON PARLERA DE « SERVEUR » POUR LE SERVEUR NODE.JS ET DE « BASE DE DONNEES » OU « BDD » POUR LA PARTIE SERVEUR MYSQL.

LA PARTIE CLIENT

LA PARTIE SERVEUR

SERVEUR DISTANT

NODE.JS

MODULE EXPRESS

Le module Express (package npm) est probablement le module le plus populaire et le plus utilisé pour Node.

Nous avons vu dans la section précédente que si la plateforme Node est en soi un noyau simple et aux fonctionnalités limités, elle devient en revanche d'une incroyable puissante si on prend en compte les modules que l'on peut y ajouter. Express apparait comme l'un de ces modules qui rend Node si populaire, puissant et agréable à utiliser ; ce module agit comme une « surcouche » et permet d'ajouter (et/ou remplacer) des fonctionnalités très intéressantes à notre serveur.

NB : Express est lui-même un module « décomposé », qui est livré avec un certain nombre de fonctionnalités et qui peut être « augmenté » par de nombreux et divers sous-modules. Pour plus d'informations au sujet de ce module, ses possibilités et ses « sous-modules » nous vous invitons à consulter la page officielle du package, dont le lien est fourni dans la section finale « Bibliographie ».

Nous présenterons uniquement les deux fonctionnalités utilisées dans le cadre de notre serveur.

- ➔ La première concerne la « gestion des routes ». En effet, Node seul permet de servir les fichiers au client à partir d'une requête précise, mais force est de reconnaitre que cela devient relativement compliqué et peu pratique lorsque les différentes requêtes à traiter se multiplient et se complexifient. Express (et c'est là une des raisons de sa popularité) permet de gérer tout cela de manière simplifiée et automatisée avec un système de routes.

Ainsi en deux lignes seulement on peut indiquer au serveur « Pour telle requête, tu envoi tel fichier ».

Initialisation :

```
var express = require('express');
var app = express();
```

(Exemple) Ensuite lorsque l'utilisateur veut accéder à *monSite/AccountCreate*, on lui renvoi la page *AccountCreate.html.*

```
app.get('/AccountCreate', function (req, res) {
    res.render('AccountCreate');
});
```

On fait cela pour toutes les routes, tous les fichiers, que l'on souhaite desservir au client.

Un autre aspect incroyable du système de routes par Express est la gestion des ressources liées auxdits fichiers. En effet, avec Node seul si on renvoi une page, disons index.html, mais que celle-ci

importe 2 scripts, 2 images et 3 feuilles de styles, on doit alors créer un « chemin » pour chacun de ces fichiers, les ouvrir et les renvoyer en adaptant précisément le contenu de la requête de renvoi.

C'est terriblement lourd et complexe à gérer manuellement.

Avec Express, encore une fois, on peut le faire en… 1 ligne seulement.

```
app.use(express.static(path.resolve('../')));
```

Il suffit de préciser un répertoire par défaut, dans lequel on mettra tous les fichiers que le client est en mesure d'exiger, et Express se chargera de les envoyer si la page demandée par le client est liée à un ou plusieurs de ces fichiers.

Il existe d'autres manières, simples également, de gérer des cas plus complexes (plusieurs dossiers de fichiers à servir par défaut par exemple) mais dans notre cas cette solution est adaptée et idéale.

➔ La deuxième fonctionnalité qui nous intéresse avec Express est liée aux cookies, ou pour être exact aux « variables de session ». En effet, on souhaite pouvoir conserver des informations pour une session d'utilisation (une session commence lors d'une visite depuis une IP **X** avec un navigateur **Y**).

Ainsi, on peut gérer le cas suivant : une fois que l'utilisateur s'est identifié, il peut naviguer sur le site en restant connecté. S'il accède à la section « Compte » on lui affiche non pas la page d'accueil mais la page « Mon compte » avec toutes ses informations personnelles, par exemple.

Initialisation :

```
var bodyParser = require('body-parser');
var expressSession = require('express-session');
var cookieParser = require('cookie-parser');

app.use(bodyParser());
app.use(expressSession({secret:'somesecrettokenhere'}));
app.use(cookieParser());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
```

Cette fonctionnalité repose sur trois « sous-modules » que sont *« body-parser »*, *« express-session »* et *« cookie-parser »*. On indique au serveur Node qu'il va y avoir recours, et on indique à Express qu'il va utiliser ces modules.

On peut ensuite utiliser *req.session* (ou *req* est l'objet en paramètre de chaque fonction de gestion de route, voir l'exemple en page précédente) pour stocker… tout ce que l'on veut (format **JSON**).

Exemple :

```
req.session.account = {'pseudo' : undefined};
req.session.save( (err) => {} );
```

MODULE SOCKET.IO

MODULE MYSQL

Le module MySQL (package npm) est comme son nom l'indique un module qui permet de travailler avec une base de données MySQL.

Dans notre projet nous nous en servons essentiellement pour faire des requêtes à la base de données, notamment dans les situations suivantes :

- Vérifier les informations de connexion d'un client
- Ajouter un compte client après vérification du formulaire d'inscription
- Modification d'un deck
- Afficher les informations d'un compte
- Récupérer le contenu du Deck actif pour la participation d'un joueur à un match

Le fonctionnement d'une transaction est des plus classiques :

(Exemple) On commence par créer une « connexion » :

```
var mysql = require('mysql');
var connection = mysql.createConnection({
  host : [redacted],
  user : [redacted],
  password : [redacted],
  database : [redacted],
});
```

(Exemple) On prépare une requête :

```
var query0 = "SELECT Carte.* FROM CompteJoueur JOIN JoueurCarteDeck USING (Pseudo)
```

(Exemple) Puis on envoi cette requête, on traite les éventuelles erreurs et on traite le résultat (stocké sous forme d'un tableau de lignes) :

```
connection.query(query0, function(err, rows, fields){
  if (err) throw err;
  for(var i=0; i<rows.length ; i++){
    inventaire1.push(rows[i]);
    inventaire2.push(rows[i]);
  }
});
```

MODULE JADE

https://www.npmjs.com/package/jade

Jade (package npm) est un moteur de templating crée essentiellement pour être utilisé en collaboration avec Node. Si Node remplace les langages coté serveur les plus puissants comme PHP, il demeure certain aspect pour lesquels Node est « incompétent ».

Dans notre situation (on rappelle que l'on souhaite développer un client et un serveur uniquement basé sur Javascript) Jade apparait comme une merveilleuse solution. En effet, pour servir au client des pages dynamiques (pas des pages avec du contenu dynamique, on parle ici de pages dont le contenu est différent selon le contexte) Node seul ne propose aucune solution viable.

Jade permet donc d'écrire des pages .jade qui seront compilées par le moteur de « rendering » de Node qui les « traduira » en du pur HTML à renvoyer au client. La syntaxe de Jade intègre donc tous les composants du HTML, en rajoutant la plupart des composantes algorithmiques classiques telles que les structures de contrôle, les boucles et les variables. De plus la syntaxe de Jade est simple, claire et permet même une productivité accrue dans l'écriture de pages HTML.

En guise d'exemple, on va parler du cas de la page (ou plutôt de la « route », voir la section Module Express) Account. La partie de notre site relative aux Comptes comporte deux aspects :

- L'accueil (« se connecter ou créer un compte »)
- L'affichage de « mon compte »

Si on utilise Node seul, l'unique solution qui s'offre à nous est de disposer de différentes routes tels que « *monSite/AccountMenu* » et « *monSite/AccountHome* » avec leurs fichiers propres.

Cette solution n'est pas idéale, on aimerait disposer d'une seule route *« monSite/Account* » qui affiche le contenu adapté selon le contexte (afficher les informations de mon compte quand je suis connecté, une page de sign-in/sign-up sinon).

Avec Jade et l'utilisation des « cookies » (variables de session) on a donc la structure suivante :

La route *Account* qui dispose d'un simple « renvoi » de la page *Account.jade* :

```
app.get('/Account', function (req, res) {
    res.render('Account', req.session.account);
});
```

Cette page contient les quelques lignes suivantes :

```
html
    head
        include templates/template_head

    body
        include templates/template_nav

        #GUI_CONTENT.row

            if pseudo == undefined
                include AccountMenu.jade
            else
                include AccountHome.jade
```

On constate deux choses :

- Avec Jade on peut inclure du code d'autres fichiers .jade, ainsi on peut alléger le code et le rendre plus universel en « exportant » les portions de code communes à toutes les pages, comme c'est le cas pour une partie de la section « head » (importation des feuilles de style, de jquery, etc..)

- Dans la partie « contenu » de notre code HTML (#GUI_CONTENT) à renvoyer on ajoute soit le contenu du fichier AccountMenu.jade soit de AccountHome.jade selon que l'utilisateur s'est déjà connecté ou non.

Dans les pages *AccountMenu.jade* et *AccountHome.jade* on a le code HTML (Jade en l'occurrence, mais qui sera traduit en du HTML) qui correspond aux pages « Se connecter ou créer un compte » et « Mon compte ».

En résumé, les principales caractéristiques de Jade sont :

- Syntaxe très claire et simplifiée
- Traitements algorithmiques tels que les boucles ou les tests
- Passage d'informations utilisables dans le code Jade au format JSON lors du rendering
- Possibilité de découper le code et de créer des portions réutilisables via les « *include* »

SOURCES ET BIBLIOGRAPHIE

EXPRESS : https://www.npmjs.com/package/express

MYSQL : https://www.npmjs.com/package/mysql

JADE : https://www.npmjs.com/package/jade