l'école d'ingénierie
informatique

EPSI BORDEAUX - I1 EISI

Module Atelier - Architecture applicative (1/9)

Code Module	Durée	Titre Diplôme	Bloc de Compétences	Promotion	Auteur
ARCE842 - DEVE702	20h	EISI / RNCP 35584	Concevoir & Développer des solutions applicatives métiers	2023/2024	Julien COURAUD

1. Le jeu du "Yam Master"

D'abord connu sous le nom de La Linotte, Yam Master est un agréable successeur du fameux Yam's. Plus besoin de stylo : les résultats sont immédiatement transformés en jetons sur le tableau de marque.

Pour chaque combinaison de dés réussie, placez un de vos pions sur la grille Yam Master. Pour l'emporter, réalisez le plus possible
d'alignements de pions et bloquez ceux de votre adversaire.

Que le duel commence !

1.1 But du jeu

Marquer plus de points que son adversaire, ou réaliser un alignement horizontal, vertical ou en diagonale de cinq pions.

1.2 Déroulement du jeu

Le Yam Master est un jeu pour deux joueurs avec 5 dés au tour par tour.

À son tour, un joueur peut lancer les dés à trois reprises, afin de réaliser une des combinaisons présentes sur le plateau.

Après chaque lancer, il peut écarter autant de dés qu'il le souhaite et relancer les autres. Tout dé écarté peut être relancé dans les jets suivants.

Les combinaisons réalisables sont les suivantes :

- Brelan : trois dés identiques (ex. : case "2" : réalisation de trois "2")
- Full : un brelan et une paire
- Carré : quatre dés identiques
- Yam : cinq dés identiques
- Suite : combinaisons 1, 2, 3, 4, 5 ou 2, 3, 4, 5, 6
- ≤8 : la somme des dés ne doit pas excéder 8

- Sec : une des combinaisons ci-dessus, sauf le brelan, dès le premier lancer
- Défi : avant le deuxième lancer, le joueur relève un défi.

Au cours des deux lancers suivants, il doit impérativement réaliser une des combinaisons ci-dessus (sauf le brelan). Il n'a pas besoin de s'engager sur une combinaison précise.

A noter que les dés peuvent former plusieurs combinaisons simultanément (ex : un Yam est aussi un brelan, un carré, un full) parmi lesquelles le joueur choisit une combinaison.

Dès qu'un joueur réussit une combinaison, il peut (s'il le souhaite) poser un pion sur une des cases libres du plateau correspondant à sa combinaison.

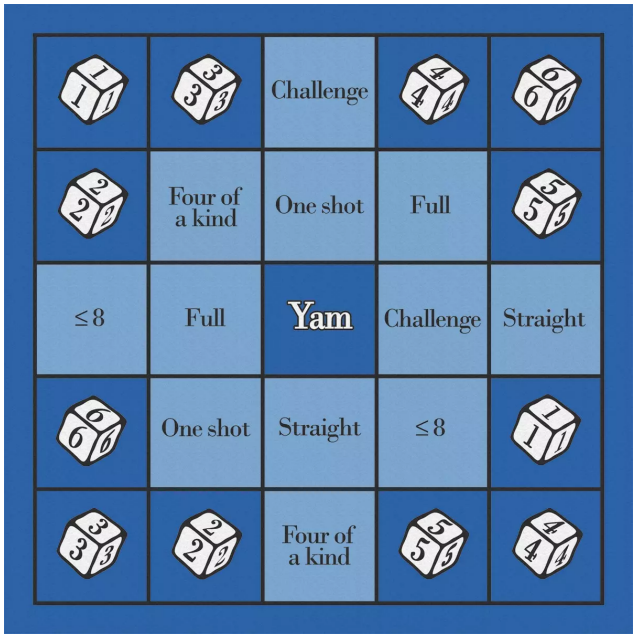
À tout moment, il est possible d'utiliser le Yam Predator : faire un Yam pour retirer n'importe quel pion adverse au lieu d'en poser un des siens.

1.3 Décompte des points

- Un alignement horizontal, vertical ou en diagonale de trois pions rapporte 1 point.
- Un alignement de quatre pions rapporte 2 points.

La partie s'achève lorsqu'un des joueurs n'a plus de pions (le joueur avec le plus de points est alors le vainqueur, les joueurs commencent la partie avec 12 pions à leur disposition) ou lorsqu'un des joueurs réalise un alignement de cinq pions (il gagne instantanément la partie).

1.4 Grille de jeu



2. Projet en binôme: jeu mobile en ligne du Yam Master.

2.1 Déroulé du projet

Dans un premier temps, vous suivrez le déroulé du setup de l'environnement de développement, de la prise en main du starter-kit Expo et de la mise en place du moteur de jeu et de la communication Client / Server via le protocole WebSocket.

Par la suite, sur cette base, vous produirez des développements pour enrichir l'application selon plusieurs aspects:

- Exercice algorithmique: Complétion du moteur de jeu et validation des combinaisons de jeu.
- Exercice architectural: Mise en place d'une base de données prête à stocker les informations relatives aux parties jouées par les joueurs connectés à la plateforme.
- Exercice de développement: Ajout de fonctionnalités essentielles telles que l'authentification et la connexion à un profil authentifié sur la plateforme, le replay de parties ou encore l'ajout d'un score MMR pour faire un classement des joueurs.
- Exercice compétitif: Développement du mode de jeu VsBot et gagnez des points bonus si votre BOT remporte le tournoi où tous les groupes sont invités à participer.

Ce module pose un cadre applicatif fonctionnel mais de faible qualité (code, arborescence des fichiers, factorisation, etc…) et vous serez jugés sur la pertinence des améliorations en terme de qualité dans le respect du cahier des charges.

2.2 Cahier des charges et travaux attendus

Fonctionnalités obligatoires:

- Moteur de jeu et validation des combinaisons.
- Finalisation du mode de jeu en ligne.
- Reprise de la partie en cours de jeu si navigation dans l'application.
- Sauvegarde des parties.
- Login / Logout (au plus simple, si utilisateur non trouvé, directement créé en base avec le mot de passe).
- Contexte utilisateur authentifié côté Frontend et possibilité d'avoir accès à son historique de parties.
- Développement du mode de jeu "Vs Bot".

Fonctionnalités complexes (au moins une à faire parmi les trois):

- 3 niveaux de difficultés dans le mode de jeu "VsBot" (Facile / Intermédiaire / Pro).
- Replay des parties déjà jouées (tour par tour).
- Amélioration significative de l'interface graphique ("effet whouaaaaaa" requis).

Idées de fonctionnalités créatives (non obligatoires):

- Mode de jeu classé basé sur un score MMR des joueurs pour les parties en ligne.
- Grille plus grande et mode de jeu à 4.
- Animation graphique des dés.
- Ajout de données interactives à l'écran (nombre de joueurs en ligne, ou le ratio de parties gagnées/perdues par les joueurs, etc..).
- Un bouton modal pour l'affichage des règles à tout moment pendant la partie.
- Implémentation de notifications mobiles natives quand l'adversaire à finit son tour et que l'on est hors de l'application.
- Mettre en place une interface Shi/Fu/Mi pour savoir qui commence la partie.

3.3 Explication de l'architecture applicative

Une partie de l'architecture applicative vous est imposée mais vous serez également libre de choisir certains aspects de votre conception.

Partie Frontend:

- Stack: Expo / React Native

- Setup développement: Web / Emulation Android Studio / QRCode vers mobile physique

- Couches applicatives:

  - `<App>` : Conteneur principale de l'application, contiendra à terme également les informations de contexte utilisateur.
  - `<MenuScreen>` : Boutons vers les modes de jeu et paramètres.
  - `<OnlineGameScreen>` : Mode de jeu en ligne.
  - `<VsBotGameScreen>` : Mode de jeu contre l'ordinateur.
  - `<AuthScreen>` : Ecran d'authentification.

- Chaque écran de jeu (`<OnlineGameScreen>` et `<VsBotGameScreen>`) sont responsables de check la connexion et délèguent au sein de composants `Controller` (exemple: `OnlineGameController`) la gestion des évènements de parties. Ces controleurs instancient un composant `<Board>` et prévoient les écrans de file d'attentes. Le composant `<Board>` à la simple fonction d'instancier les sous-composants graphiques interactifs et d'appliquer un layout global, il est le composant parent de notre logique graphique liée à la partie en cours.

- Les sous-composants graphiques du composants `<Board>` peuvent être interactifs (clic sur la grille de jeu, clic sur un choix parmi les combinaisons, verrouillage de dés) et écoutent le serveur pour mettre à jour leur interface. Ils reçoivent des objets conçus spécialement pour eux en terme de de nécessités graphiques et peuvent émettre des sockets pour notifier le serveur qu'une action a été réalisée par un joueur.

  - `<OpponentInfos>`
  - `<OpponentTimer>`
  - `<OpponentScore>`
  - `<OpponentDeck>`
  - `<Grid>`
  - `<Choices>`
  - `<DeckPlayer>`
  - `<PlayerInfos>`
  - `<PlayerTimer>`
  - `<PlayerScore>`

- Librairies externes notables:

  - `socket.io-client` : pour la communication bidirectionnelle Client / Serveur
  - `@react-navigation` pour la gestion du routing et des écrans.

Partie Backend:

Pour ce projet, nous aurons trois entités serveurs de type web-services (certains WebSocket, d'autres potentiellement des API Restful). Ces trois entités de notre architecture peuvent être réalisées au sein d'un même repertoire mais vous êtes également libre d'externaliser une partie de la logique, notamment pour le serveur lié au stockage des données.

1. ***WebSocket Server - Game Manager*** :

- Stack (imposée): Express / Node.js

- Librairies externes notables:

  - `express`
  - `socket.io`
  - `uniqid`

- Contrats API WebSocket

  - Client can listen on: `queue.added`
  - Client can listen on: `game.start`
  - Client can listen on: `game.end`
  - Client can listen on: `game.timer`

    - Client can listen on: `game.opponent.leave`
    - …
    - Client can emit on: `disconnect`
    - Client can emit on: `queue.join`
    - Client can emit on: `game.leave`
    - Client can emit on: `game.end-turn`
    - Client can emit on: `game.dices.roll`
    - Client can emit on: `game.dices.lock`
    - Client can emit on: `game.choices.selected`
    - Client can emit on: `game.grid.selected`
    - …

Le déroulé d'une interaction entre deux clients et notre `WebSocketServer` est le suivant:

- Client1 - emit on `queue.join`
- Client2 - emit on `queue.join`
- Server - can response with:
    - emit on `queue.added` (if only one in queue)
    - emit on `game.start` (if already somebody in queue, or can be sent later when other player queued up)
- Server - emit on `game.timer` every second during all game
- Client1 / Client2 - listen on `game.timer` to update timer components.
- Server - listen on `game.dices.roll`, `game.dices.lock`, `game.choices.selected` and `game.grid.selected` during all game and is updating game model while making verifications about plays and timer.
- Server - emit updates for both views, at every step of the game (only needed datas).
- Client1 / Client2 - Keep the views synchronized with server model while receiving sockets.
- Server - can end the game when it's a victory or a loose and emit on `game.end`. Timer for this game is ended at this moment.
- Client1 / Client2 - Display end game screen and can root on main menu.

1. ***Database Server - Stockage Service*** :

- Stack (au choix) - Possibilité de faire au sein du serveur Express mais vous pouvez aussi créer une nouvelle entité Serveur qui communique avec notre WebSocket Server via une API Restful, uniquement aux moments où il est nécessaires d'accéder aux données stockées.

- Relier le socle existant à cet entité Serveur au minimum pour les fonctionnalités de sauvegardes de parties et d'authentification.

1. ***BOT Server - BOT Game Manager*** :

- Stack (au choix) - tant que ce web service peut communiquer via WebSocket avec le serveur de jeu. Peut être fait dans le socle existant du WebSocket Server, mais la logique devra être clairement externalisée du moteur de jeu.

- Lorsqu'une partie VsBot est lancée, le WebSocket Serveur de jeu créé une connexion avec le BOTServer qui interagit selon la même API WebSocket que l'interface Client.

- Cette structure de communication permettra à votre BOT de participer au tournoi.