Dokumentation Schmugglerspiel

*Anmerkung: Hier werden nur Dinge dokumentiert, die noch nicht in der Beispielimplementierung vorhanden waren*

Klasse User:

Variabeln:

int **role**

*beschreibt die aktive Rolle des Nutzers*

*0: Dealer, 1:Polizist, 2: Admin*

int **inv[x][y]**

*beschreibt das aktive Inventar des Nutzers*

***x*** *beschreibt die Rolle für das das Inventar gilt*

*0: Dealer, 1: Polizist*

***y*** *beschreibt den Slot im Inventar*

*bei Dealer*

*0: Geld, 1: Drogenart, 2: Päckchenanzahl*

*bei Polizist*

*0: Justice Points, 1: am Tag gefangene Dealer, 2: insgesamt gefangene Dealer*

int **lastFence**

*die ID des Hehlers mit dem der User das letzte mal gedealt hat*

boolean **inCatch**

*falls ja, ist grade im Fang-Vorgang*

Boolean **isAdmin**

*falls ja, sieht alles und nicht nur Rollenspezifische Sachen*

Methoden:

**public void setRole(**int **role)**

*setzt die Rolle des Nutzers auf* ***role*** *und aktualisiert den Nutzer auf der DB*

**public Boolean toggleAdmin()**

*macht nicht-Admins zu Admins und andersrum, gibt den* ***isAdmin*** *des Users zurück*

**public Boolean toggleCatch()**

*ändert ob man im Fang-Vorgang ist und gibt* ***inCatch*** *zurück*

**public void setFence(**int **ID)**

*setzt den LastFence auf ID*

**public int[ ] getInv()**

*gibt das Inventar der aktiven Rolle des Nutzers zurück*

**public void setInv(**int [ ] **inv)**

*setzt das Inventar der aktuellen Rolle des Nutzers auf* ***inv*** *und aktualisiert den Nutzer*

*auf der DB*

**public void setInvWRole(**int [ ] **inv**,int **role)**

*setzt das Inventar der Rolle* ***role*** *des Nutzers auf* ***inv*** *und aktualisiert den Nutzer auf*

*der DB*

**public void getJustice(**User **dealer,** int **amount)**

*gibt Justice Points basierend auf der Entfernung zum* ***lastFence*** *von* ***dealer*** *und der*

*Anzahl der entnommenen Pakete* ***amount***

**public int getCaught()**

*setzt Pakete im Inventar auf 0, außerdem Geld auf 100 falls == 0, gibt die Anzahl*

*der Pakete zurück die der User hatte*

Klasse UserRepository:

Variablen:

**public Boolean TestSpawning**

*sagt ob TestUser und Hehler in Datenbank existieren oder nicht*

Methoden:

**private void insertTestUser(**Double[] **loc)**

*fügt einen hardgecodeten TestUser in die Datenbank an Stelle* ***loc*** *ein*

**public Iterable<User>**

**findUsersNearbyByRole(**Double[] **loc,** Double **maxDistance,** int **limit,** int **role)**

*gibt bis zu* ***limit*** *Nutzer der Rolle* ***role*** *mit maximal* ***maxDistance*** *Abstand zu* ***loc*** *zurück*

**public void generateTestStuff(**String **fID)**

falls **TestSpawning** true: *generiert einen TestUser und TestHehler in der Nähe von dem Nutzer mit der FacebookID* ***fID***

*sonst: löscht sie aus Datenbank, falls vorhanden*

**public String getNearestDealer(**String **facebookID)**

*gibt den nähesten Dealer zu dem Nutzer mit* ***facebookID*** *zurück*

**public String getNearestFence(**String **facebookID)**

*gibt den nähesten Hehler zu dem Nutzer mit* ***facebookID*** *zurück*

angepasste Funktionen:

**public User updateUserProfileFromLoginCredentials**

*erstellt jetzt bei jedem Login Test User und Hehler*

**copyUser(**User **user)**

*kopiert alle neuen Variablen*

Klasse Fence

Variabeln:

int **ID**

*eindeutige Identifizierung des Hehlers*

Double[] **loc**

*Position des Hehlers auf der Karte*

Methoden:

**public String toString()**

*gibt Hehler* ***ID*** *und* ***loc*** *als String zurück*

Klasse FenceRepository:

Methoden:

**public void updateLocation(**Fence **fence,** Double **longitude,** Double **latitude)**

*ändert die Position des Hehlers* ***fence*** *in der Datenbank auf {****longitude****,****latitude****}*

**public Fence findByID(**int **ID)**

*gibt den Hehler mit der ID* ***ID*** *zurück*

**public Iterable<Fence>**

**findFencesNearby(**Double[] **loc,** Double **maxDistance,** int **limit)**

*gibt bis zu* ***limit*** *Hehler mit maximal Abstand* ***maxDistance*** *zu* ***loc*** *zurück*

**public void addFence(**Fence **f)**

*fügt den Hehler* ***f*** *in die Datenbank ein*

**public void update(**Fence **fence)**

*updatet den Hehler* ***fence*** *in der Datenbank*

**public void initiateFences(**Fence[] **f,** int **anzahl)**

*fügt* ***anzahl*** *Hehler aus* ***f*** *der Datenbank hinzu, zum hardcoden von Positionen*

**public Fence copyFence(**Fence **f)**

*gibt eine Kopie von* ***f*** *zurück*

Klasse CatchThread:

Variablen:

User **dealer, police**

*Polizist der fängt, Dealer der gefangen wird*

long **catchTime**

*Zeit die es dauert bis das Fangen erfolgreich war*

double **maxDistance**

*Distanz, die der Dealer geflohen sein muss um zu entkommen*

double **catchDistance**

*Distanz, innerhalb der der Polizist den Dealer fängt*

double **runDistance**

*Distanz die der Dealer schon geflohen ist*

long **startTime**

*Startzeit der Fang-Vorgangs*

long **timeWhileCatching**

*Zeit, die der Polizist den Dealer schon gefangen hat*

Methoden:

**public void run()**

*verwaltet den kompletten Fang-Prozess. Jede Sekunde wird entweder*

***timeWhileCatching*** *addiert oder* ***runDistance*** *hinzugefügt. Beiden Clients wird*

*mitgeteilt wie weit das Fangen bzw. das Fliehen ist. Sobald das Fangen*

*abgeschlossen ist, werden die jeweiligen Inventories angepasst und die Clients*

*nochmal darüber informiert.*

**public int getStat()**

*gibt den aktuellen Status des Threads bzw. Fangvorgangs zurück*

**public double getCatchPercentage()**

*gibt den Fang-Fortschritt in Prozent zurück*

**public double getFleeingPercentage()**

*gibt den Flucht-Fortschritt in Prozent zurück*

**private double getDist()**

*gibt die Distanz von Polizist und Dealer zurück*

**public String toString()**

*gibt die Daten über den Thread in einem String zurück*

Klasse Game:

Methoden:

**public static void joinUserToTeam(User user)**

*Fügt einen User zu einem Team hinzu*

**public Iterable<User> computeDist(double maxDist, int limit)**

*Gibt eine Liste von Dealern aus, die sich in der Nähe von Polizisten befinden*

HTTP-Requests in HomeController:

einzustellen in “**conf/routes**” auf Serverseite

**POST /users/:userFacebookID/team** (Serverseite)

**new HttpPoster().execute("users", userFacebookID, "team");** (Clientseite)

AKA **public Result joinUser(**String **facebookID)**

*beantragt eine Teamänderung*

**public Result getTeam(**String **facebookID)**

*schickt das aktuelle Team an den Client - ohne Teamänderung*

**POST /games/deal/data/:userFacebookID** (Serverseite)

**new HttpPoster().execute("games", "deal", "data", act.getFacebookID());** (Clientseite)

AKA **public Result requestDealData(**String **facebookID)**

*beantragt die für das Deal-Tab notwendigen Daten*

**public Result getNearbyFences(**String **facebookID)**

*gibt dem Client im Result alleTeammitglieder auf 50000m Distanz, alle Nutzer auf*

*300m Distanz und falls man Dealer ist alle hehler auf 50000m Distanz zurück*

*[nicht sicher ob das wirklich m sind]*

**public Result buyDrugs(**String **facebookID,** int **amount)**

*Dealer mit* ***facebookID*** *kauft, falls möglich,* ***amount*** *Drogen beim nähesten Hehler*

**public Result sellInformation(**String **facebookID,** int **amount)**

*gibt dem Dealer mit* **facebookID** *zurück, wie der Verkauf von* ***amount*** *Drogen*

*ausgehen würde, falls er es kann*

**public Result sellDrugs(**String **facebookID,** int **amount)**

*Dealer mit* ***facebookID*** *verkauft* ***amount*** *Drogen an den nähesten Hehler, wird dafür*

*basierend auf der Distanz zum urpsrünglichen Verkäufer dieser Drogen bezahlt*

**public Result Catch(**String **PFacebookID)**

*startet einen CatchThread mit Polizist mit* ***PFacebookID*** *und dem ihm nähesten Dealer, falls dieser 30m oder näher ist*

**public Result getUserData(**String **fID)**

*gibt Name, Rolle, Inventar und Position des Users mit facebookID* ***fID*** *zurück*

**Client:**

Klasse MapSectionFragment.java

**public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)**

Implementiert die Funktion des Catchbuttons

Implementiert die Funktion des Lookup/Refresh User Buttons

Alles Was auf der Map angezeigt wird ist hier implementiert.

Implementiert einen Location Listener der bei geändertem GPS automatisch alles auf der Map angezeigte aktualisiert.(Der lookup refresh User Button muss zuvor 1 mal gedrueckt worden sein!)

Klasse fragment\_section\_map.xml

Hier werden alle Textfelder, Buttons und die Map deklariert, sprich angezeigt ohne die Funktionalität

Klasse MainActivity.java

**public void processGameMessageOnUIThread(final Map<String,String> gameMessage)**

Bearbeitet alle eingehenden Firebase Nachrichten

private void **updateGameTab(String teamNr)**

Ändert den Namen und den Inhalt des “Game”-Tab

Klasse GameDialogs.java

**public static Dialog createPopup(MainActivity act, String text)**

Erschafft einen Popup mit der Nachricht “text”

**public static Dialog createTeamDialog(MainActivity act, String team)**

Erschafft einen Popup mit der aktuellen Teamzuteilung

**public static Dialog createSellMeassage(MainActivity act, final String amt, String earnings, final String userFacebookID)**

Erschafft einen Dialog für das Verkaufen

Klasse GameSectionFragment.java

**public static void updateContents()**

aktualisiert den gesamten Tab

**public static void updateDealMenu()**

ändert und aktualisiert den Tab zu einem “Deal”-Tab

Klasse MainSectionFragment

**public static boolean** checkForConnection(**int** warteZeitinMilisekunden){}

Falls eine connection zum Server besteht, liefert sie True, sonst false. warteZeitinMilisekunden gibt an, wie lange versucht wird, eine Connection aufzubauen.