DotAI - Projet de semestre

Thomas Ibanez

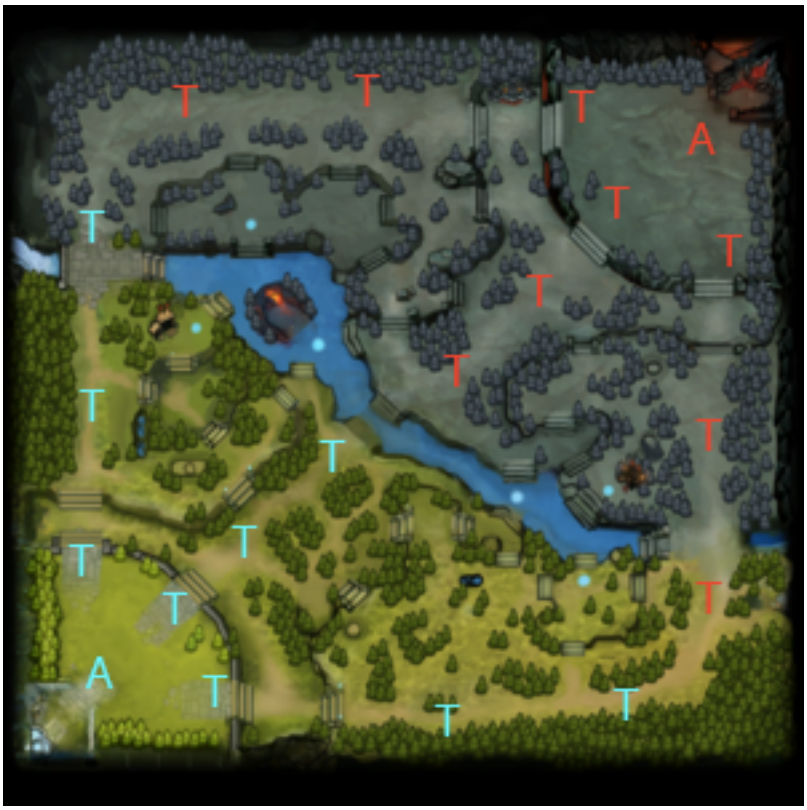
22 février 2018

1 Introduction

Le but de ce projet de semestre est de créer un ensemble de logiciels permettant de présenter les données contenues sur `opendota.com` pour le développement d'un logiciel d'apprentissage automatique sur le jeu DotA 2.

1.1 DotA 2

DotA 2 est jeu-vidéo multijoueur de type bataille en arène (MOBA) où 2 équipes s'affrontent. Chaque équipe est composée de 5 joueurs chacun controllant un héro choisi parmis les héros disponnibles dans le jeu (il ne s'agit donc pas d'un avatar personnalisable comme dans le cas des MMORPG). Il y a à l'heure actuelle le jeu propose 115 héros, chacuns disposant de 5 attaques différentes La partie se déroule sur une carte symétrique constituté de 3 "lanes" sur lesquelles sont disposées des tours dont voici une vue aérienne :



Chaque équipe commence dans un coin de la carte (en bas à gauche et en haut à droite), le but pour chaque héros est de controler une partie de la carte et détruire les tours ennemies (T) afin d'arriver jusqu'a la base adverse (A) et de la détruire.

En plus des héros chaque base va périodiquement créer des "creeps", sortes de petit monstres qui vont se déplacer sur une lane.

2 Recupération des informations d'OpenDota

Opendota collecte une quantitée énorme de parties de DotA 2 jouées par des joueurs du monde entier. Le site présente une API permettant la récupération de matchs selon certains critères. Une documentation sur le genre de requêtes faisable est disponible sur `https://docs.opendota.com/`. Dans le cadre de se projet, nous allons utiliser un filtre permettant de séléctionner un héros dont on voudrait qu'il soit présent dans le match.
Dans un premier temps M. Malaspinas et moi-même pensions que les données étaient directement exploitable depuis opendota, malheureusement c'est plus compliqué. En effet OpenDota stocke uniquement des données statistiques sur la partie jouée mais pas les actions et positions des entitées à chaque instant. Heureusement le résultat contient un lien vers un fichier *.dem* qui lui contient toutes ces informations.

3 Analyse des fichiers .dem

Les fichiers .dem sont des fichiers contenant toutes les informations dont on pourrait vouloir sur une partie allant de entités présentes sur la carte jusqu'au commentaires audio si disponnibles. Un fichier .dem pour une partie standard (~30 Minutes) pèse ~50Mo.

3.1 Format

Le format global de ces fichiers est assez simple. DotA utilise protobuf, une technologie de sérialization développée par google pour encoder ces fichiers. Le protocole est maintenu à jour par l'équipe SteamRE sur `https://github.com/SteamRE/SteamKit/tree/master/Resources/Protobufs/dota`
Chaque fichier commence par une en-tête, soit "PBDEMS2" pour les replay datant d'après la mise à jour de DotA sur le moteur de jeu Source 2. Soit "PBUFDEM" pour les replay plus anciens. Dans le cadre de ce projet nous nous concentrons uniquement sur les replay d'après Source 2 car ce sont ceux qui sont émis en ce moment.
Une fois ce header lu le fichier contient 8-bytes dont l'utilitée n'est pas connue.

3.1.1 Format global

La suite du fichier est une série d'entrées de ce type :

Nom	Type
ID	VarInt
Tick	VarInt
Taille	VarInt
Données	Bytes

Note : Les varints sont des entiers encodés sur une longeur variable de bits. Cet encocodage est détaillé sur `https://developers.google.com/protocol-buffers/docs/encoding#varints`

L'ID va indiquer le type du message parmis la liste définie dans le protocole :

Messages

```
enum EDemoCommands {
  DEM_Error = -1;
  DEM_Stop = 0;
  DEM_FileHeader = 1;
  DEM_FileInfo = 2;
  DEM_SyncTick = 3;
  DEM_SendTables = 4;
  DEM_ClassInfo = 5;
  DEM_StringTables = 6;
  DEM_Packet = 7;
  DEM_SignonPacket = 8;
  DEM_ConsoleCmd = 9;
  DEM_CustomData = 10;
  DEM_CustomDataCallbacks = 11;
  DEM_UserCmd = 12;
  DEM_FullPacket = 13;
  DEM_SaveGame = 14;
  DEM_SpawnGroups = 15;
  DEM_Max = 16;
  DEM_IsCompressed = 64;
}
```

La seule exception est *DEM_IsCompressed* qui n'est pas un type de message mais qui défini un bit (6ème bit) qui s'il est égal à 1, alors le message doit être décompressé via la librairie snappy avant d'être interpreté.

Le Tick va définir le moment dans la partie ou le message arrive.

Le champ Taille indique la taille en bytes des données de ce message.

Une fois ces informations connues nous pouvons lancer le décodage via protobuf afin d'obtenir le message, du moins c'est le cas pour tout les ID sauf DEM_SignonPacket, DEM_Packet, DEM_FullPacket et DEM_SaveGame. Pour ces messages-ci il va falloir lire les données intégrées.

3.1.2 Données intégrées

Les données intégrées sont basiquement un message encodé dans un autre message. Dans notre cas un DEM_Packet et un DE_FullPacket sont définis comme :

Packets

```
message CDemoPacket {
  optional int32 sequence_in = 1;
  optional int32 sequence_out_ack = 2;
  optional bytes data = 3;
}
```

```
message CDemoFullPacket {
  optional .CDemoStringTables string_table = 1;
  optional .CDemoPacket packet = 2;
}
```

Les DEM_SignonPacket n'ont pas de définition distincte, leur structure est la même qu'un DEM_Packet. Les DEM_FullPacket sont en fait composés d'une *StringTable* et d'un DEM_Packet qu'il faudra interprêter comme les autres DEM_Packet.