CSDemoParser

CS:GO Demos Retrievement and Parsing Project

Overview del lavoro svolto

- Studio di Fattibilità per il download delle partite
- Scelta del parser: demoinfogo

- Codifica dello script di download
- Codifica dello script di parsing automatico
- Codifica dello script di validazione automatica

- Creazione di una lista di eventi disponibili completa
- Creazione di demo di test per il ritrovamento delle features
- Comprensione del funzionamento del parser tramite analisi dell'output
- Modifiche al parser e suo adattamento allo scopo

- Processo di Download, Parsing e Validazione

Dove sono salvate le demo? Come sono condivise?

Ufficialmente, le demo sono salvate nei Server di Valve e sono scaricabili tramite link:

```
replay131.valve.net/730/xxxxxxxxxxxxxxxxxxxxx_xxxxxxxxx.dem.bz2
```

dove con «x» si intende un codice composto da MatchID, OutcomeID, TokenID.

Questi parametri non sono pubblici. Infatti, solo il giocatore stesso può creare uno share-code, per ovvi motivi di privacy.

Steam API e node-csgo

- Valve rende disponibile un'API per siti di terze parti in grado di dare loro accesso alla lista di partite di un giocatore.
- Purtroppo, viene richiesto che il giocatore stesso fornisca un Game Authentication Code, assieme allo share-code della partita.

Sarebbe possibile trovare il link tramite node-csgo:

- dallo share-code: `CSGO.SharecodeDecoder(string code).decode()`ritorna MatchID, OutcomeID e Token ID, da cui si può ricreare il link. Impossibile, come già visto, perchè non disponiamo degli share-code.
- Da `requestRecentGames()`: ritorna la lista di partite (con link) dell'account loggato. Non è fattibile chiedere a 50 giocatore di loggarsi.

Usare l'API di Valve o node-csgo porta quindi ad un vicolo cieco.

csgostats.gg

Tentativo con siti esterni: csgostats.gg

- Usa node-csgo per trovare le partite dei suoi utenti.
- Demo scaricabili all'interno di CS:GO tramite Steam Bootstrapper e Steam Protocol.
- Automazione impossibile: presenti reCAPTCHA v3, molto difficili da aggirare.

La nostra scelta: HLTV.org

HLTV è un famoso sito che raccoglie tutte le partite di giocatori professionisti di CS:GO. La lista di partite per giocatore è decisamente più vasta rispetto a csgostats.gg.

Pro:

- Download diretto, senza Steam Protocol o API.
- Nessun CAPTCHA per bloccare l'automazione

Script: autodownload.py

Abbiamo creato uno script Python, usando Selenium, in grado di scaricare autonomamente tutte le partite. In particolare lo script:

- Dalla lista delle partite di un giocatore, ne sceglie una e naviga nelle varie pagine fino a premere il pulsante «GOTV Demo».
- Attende fino al completamento del download.
- Estrae i file all'interno del file compresso.
- Rinomina i file con il nome del giocatore e il numero della partita [1-100]

```
↳ https://www.hltv.org/stats/players/matches/317/pashabiceps
 ↳ https://www.hltv.org/stats/matches/mapstatsid/90233/heretics-vs-youngsters?contextIds=317&contextTypes=player
  ↳ https://www.hltv.org/matches/2335421/youngsters-vs-heretics-lootbet-season-3
   ↳ https://www.hltv.org/download/demo/51659
```

Il Parser: demoinfogo

Il Parser scelto è demoinfogo, ovvero il parser open-source ufficiale di CS:GO, sviluppato da Valve stessa e scritto in C/C++.

Pro:

- Ufficiale
- Potenzialmente affidabile

Contro:

- Nessuna documentazione sul suo funzionamento interno

Il Parser: demoinfogo

Per comprendere il suo funzionamento generale, abbiamo:

- Reperito, nei descriptors, una lista eventi
- Creato delle demo di test ad hoc (movimenti/azioni).
- Analizzato l'output dopo aver reindirizzato lo stdout su file tramite `freopen()`.
- Individuato l'origine dell'output tramite apposite flag vicino ai `printf()` originali.
- Letto un research paper (Charles University, Praga) sul Data Preprocessing delle demo di CS:GO, specialmente per comprendere il funzionamento dei Delta Update.

Entità

In particolare, dalla tabella `DT_CSPlayer` abbiamo trovato le seguenti features, che abbiamo deciso di salvare in variabili globali:

- MouseX, MouseY
- PlayerPositionX, PlayerPositionY, PlayerPositionZ
- PlayerVelocityX, PlayerVelocityY, PlayerVelocityZ
- CrouchState

Si tiene conto anche del tick del server per scandire il tempo.

Entità

Si noti che:

- MouseX e MouseY sono particolarmente interessanti perchè rappresentano i movimenti del mouse, usato per mirare.
- La mira è uno dei parametri più significativi per riconoscere un giocatore, in quanto i peculiari micro-movimenti della mano sono pattern memorizzati e ripetuti grazie alla cosiddetta “memoria muscolare” e sono caratteristici per ognuno.
- Le armi in CS:GO hanno un proprio pattern di rinculo, che va contrastato con un movimento “specchiato” del mouse (spray control) . L’abilità nell’eseguire questo movimento e il pattern usato è chiaramente un parametro da tenere in considerazione.

Eventi

Dai descriptors delle demo abbiamo ottenuto la lista di eventi completa, da cui abbiamo estratto 33 eventi finali.

In particolare, si notano alcuni eventi interessanti per l'identificazione di particolari abitudini e pattern dei giocatori:

- `weapon_fire`: one-taps vs spray, a seconda della preferenza.
- `player_jump`: Se combinato con MouseX, MouseY, si può riconoscere il pattern specifico di un giocatore nella pratica del «bunny-hopping».
- `item_equip / item_pickup`: armi particolari in base ai ruoli dei giocatori e preferenze.
- `player_blind`: unito a MouseX, MouseY, il valore `blind_duration` indica se il giocatore ha i riflessi pronti per evitare una flashbang.

Modifiche al Parser

- Set degli argomenti del parser predeterminati
- Rimozione delle `printf()` originali
- Sistema di individuazione del player target tramite SteamID, EntityID, UserID e gestione di disconnessioni e riconnessioni del giocatore.
- Creazione di `GlobalPlayerInfo.h`, contentente informazioni riguardo alle features riguardanti il giocatore, dinamicamente aggiornate.
- Sistema di lettura e filtraggio della tabella `DT_CSPlayer`.
- Reimplementazione di una funzione di decodifica delle prop interessanti dell'entità giocatore.

Modifiche al Parser

- Estrapolazione del tick del server dal messaggio protobuf `CNETMsg_Tick`.
- Filtraggio degli eventi scelti e modifica della gestione di alcuni eventi (e.g. `player_death` -> `_k / _d / _a`).
- Interpretazione di prop di `DT_CSPlayer` per la creazione di un possibile `crouch_event` ad hoc, successivamente inserito come valore costantemente aggiornato in Entity.
- Formattazione dell'output dei log come richiesto, tramite specifiche `printf()`.

Script: autoparse.py

Vista la quantità di partite da sottoporre a parsing, abbiamo creato un apposito script:

- Utilizza il dizionario `.json` esterno proveniente da autodownload.py per iterare tra i file `.dem` nella cartella.
- Per ogni demo, cerca lo SteamID del giocatore corrispondente in un dizionario interno.
- Chiama il parser tramite subprocess passandogli i parametri corretti.
- Presta attenzione agli exit code per gestire eventuali errori autonomamente o avvertirci se il processo di parsing è fallito in maniera irrecuperabile.

Script: validatedataset.py

E’ buona prassi assicurarsi che il dataset non contenga errori. Questo script valida il dataset controllando:

- Presenza di Entity e Action nei log.
- Assenza di contaminazione del log da errori del parser.
- Assenza di log con output costantemente nulli.
- Assenza di log con troppe poche righe.
- Presenza di 50 giocatori e 100 partite ognuno (totale 5000 partite).
- Assenza di match duplicati.