Predicting player's movements in CSGO

Or how we *forcefully* applied statistics on a competitive FPS

G07: Dario Loi, Davide Marincione

Bachelor's degree in
Applied Computer Science and Artificial Intelligence
Sapienza, University of Rome

A. Y. 2021 - 2022

The hell's a "CSGO"?

Figure: CSGO's logo.

Why did you do this?

We had our reasons for this choice (of course), some of those being:

- Videogames offer wide data availability.
- Prediction of an erratic phenomena such as movement is impressive *if done right*.
- The process of collecting data with the collaboration of our colleagues was very appealing[1].
- It *seemed* like fun.

[1]You don't get to play games for university *that* often.

The Game Map

The choice for our environment immediately fell on the popular map `de_dust2`, both because we are highly familiar with it and 'cause it is, by a large margin, the most played map of the game, meaning that eventual test subjects would be familiar with playing it and, in the worst case[2] there would be available data from outside sources.

Figure: A *beautiful* shot of Dust2's A site. . .

[2] which did present itself, as we were forced to make use of competitive matches for data due to time constraints.

Hexagons are the best-a-gons

In order to properly process the game data and capture it in a statistical framework, we had to abstract the map's geometry into some neat geometrical primitives, the natural solution would be tiling the map with squares, but the best one turns out to be *hexagons*:

- Definition of *neighbour* is unambiguous.
- Distance between two hexagons' centers is *constant*.
- More sides, leading to finer granularity of movement.
- Best approximation of a circle capable of tiling a plane.
- Looks Fancy.[3]

[3] And reminds us of strategy games.

Heuristics shenanigans

In the discretization of the map we faced two problems:

- Extrapolating height data.
- Create hexagons (cells) for areas of the map that do overlap each other.

For both cases we used heuristics[a] and a bit of eyeballing.

[a] We describe them in the documentation.

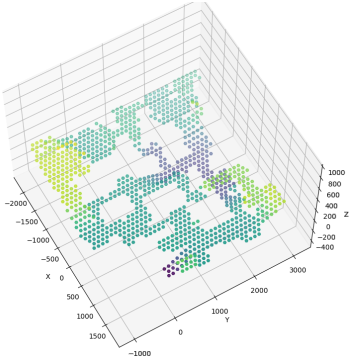


Figure: Our final tiling

A graph. . . Markov Chains?!

At the end of our tiling process, we had obtained a directed graph that represented our map[a], in it, we utilized symmetrical (undirected) edges for *walkable* connections and directed edges for places with a different height, in which it is possible to jump down but not to climb up.

[a] It is here that we realized that Markov Chains would be a proper statistical tool, but too little too late.

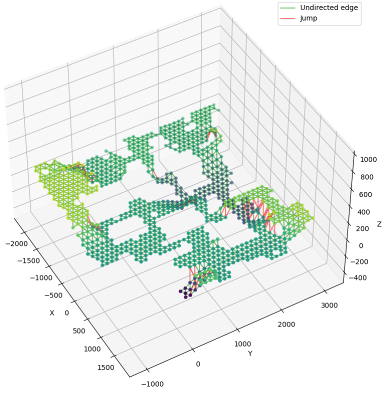


Figure: Dust2's graph, showing edges between our tiles.

Nearsightedness problems

Another information we wanted to provide (that we *hoped* to use) was of the line of sight available to a player, and thus whether one was able, during a frame, to see another... it was a *bit* problematic.

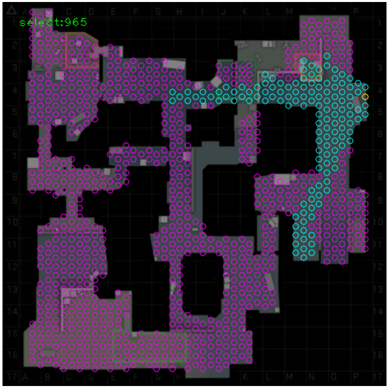


Figure: Painstakingly hand-made line-of-sight info, for *one* cell.

A single step

From the current cell, propose the distribution of the next possible position based on some current information/action.

$$\mathbf{E}\left(\mathbf{P}\left(K_2\right)\middle| K_0 \rightarrow K_1\right) = \begin{cases} \mathbf{P}\left(K_2 = K_1\right) \\ \mathbf{P}\left(K_2 = A\right) \\ \mathbf{P}\left(K_2 = B\right) \\ \dots \end{cases}$$

Figure: Example of the expected distribution for the next position, given the last movement.

Inductive process!

Using (abusing) the law of total expectation, we can derive not only the very next position, but also the ones after that!

$$\mathbf{E}\left(\mathbf{P}\left(K_3\right)\middle|K_0 \to K_1\right) = \\ \sum_{n \in N_{K_1}} \mathbf{E}\left(\mathbf{P}\left(K_3\right)\middle|K_1 \to n\right) \cdot \mathbf{E}\left(\mathrm{P}(K_2 = n)|K_0 \to K_1\right)$$

This may continue for any depth!

Our unused idea

One of our main motivations for the project was the possibility of course mate engagement in the sampling process, for this purpose, we built a small plug-in capable of managing the game's flow according to our experiment.
Unfortunately, time was running out and nobody seemed available for such an activity, so we had to scrap this and gather the data from popular competitive website HLTV.

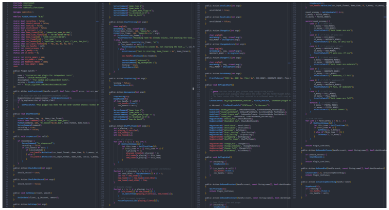


Figure: Our useless plugin, *squeezed* into a single image

Extraction and cleaning

A sample is composed of:

- Next movement (response var).
- Player's team.
- Previous movement.
- Team's center of mass orientation (useless).
- Number of teammates (useless).

The number of training samples in the end was around 540'000, while the test set amounted to 390'000.

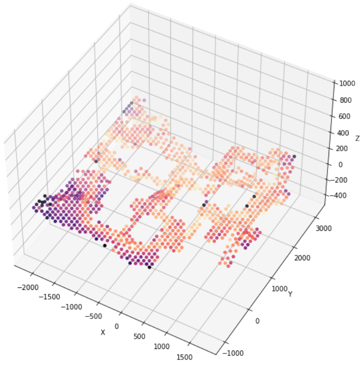


Figure: Overall distribution of our training set

Bagging! Sorta. . .

To regress a model for each cell we need enough data. . . Unfortunately in some cases this luxury wasn't available: thus we made multiple models of different quality.

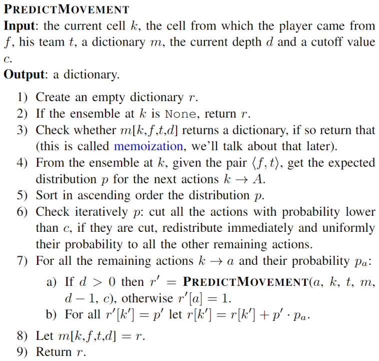
1. `choice ~ 1`
2. `choice ~ team`
3. `choice ~ cell_from`
4. `choice ~ cell_from * team`
5. `choice ~ cell_from * team + dir_team`

Model#	Cells present	$\overline{\rho}^2$ (present)	$\overline{\rho}_a^2$ (present)
1	43	0.00	0.00
2	161	0.11	0.01
3	264	0.25	0.22
4	477	0.23	0.20
Tot	945	0.20	0.17

The algorithm

Our theory may take a *lot* to run if written in a naive way, roughly $O(7^d)$. That's why our algorithm approximates it through two tricks:

- A cut-off.
- Memoization.



```
PREDICTMOVEMENT
Input: the current cell $k$, the cell from which the player came from $f$, his team $t$, a dictionary $m$, the current depth $d$ and a cutoff value $c$.
Output: a dictionary.
1) Create an empty dictionary $r$.
2) If the ensemble at $k$ is None, return $r$.
3) Check whether $m[k,f,t,d]$ returns a dictionary, if so return that (this is called memoization, we'll talk about that later).
4) From the ensemble at $k$, given the pair $\langle f, t \rangle$, get the expected distribution $p$ for the next actions $k \to A$.
5) Sort in ascending order the distribution $p$.
6) Check iteratively $p$: cut all the actions with probability lower than $c$, if they are cut, redistribute immediately and uniformly their probability to all the other remaining actions.
7) For all the remaining actions $k \to a$ and their probability $p_a$:
    a) If $d > 0$ then $r' =$ PREDICTMOVEMENT($a$, $k$, $t$, $m$, $d-1$, $c$), otherwise $r'[a] = 1$.
    b) For all $r'[k'] = p'$ let $r[k'] = r[k'] + p' \cdot p_a$.
8) Let $m[k,f,t,d] = r$.
9) Return $r$.
```

Figure: Pseudo-code!

A matter of opinions

Our model outputs a distribution of the probabilities of presence of a player in a range of given hexagons, in order to tailor the way that this distribution is presented, we propose an hyperparameter, *accumulation* (γ), which acts as a threshold for our tiles, allowing us to present a more *meaningful* distribution.

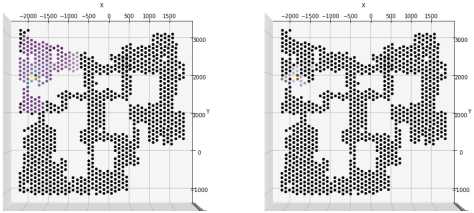


Figure: To the left, one of our model's predictions, to the right, the same prediction that went through the accumulation process, $\gamma = 0.7$.

Conclusions

We believe that, with this project we accomplished:

- A solid model for movement prediction.
- A good set of tools capable of parsing *any* in game map in order to apply our model
- A much more solid understanding of Generalized Linear Models

Therefore, the aims were nothing short of satisfied, given that we both enriched our knowledge and presented a working solution.