Work In Progress

The work presented in this document is still ongoing.
Page and text formatting, grammar and spelling checks have not been made yet. Some parts are still missing.

VALORANT Data Analysis Report

Edouard MURAT - 2023

Table of content

Introduction

About myself

My name is Edouard MURAT. I'm a 26-year-old French student. I have just finished my 5-year studies in game design at RUBIKA SupInfoGame. After my high school diploma, I went to higher school preparatory classes in mathematics. There, I discovered Python programming that I enjoyed doing. This positive experience leads me toward a two-year technological bachelor in IT in which I have sharpened my skills in various programming languages and started learning how to manage SQL databases. After that diploma and an Erasmus in Finland, I chose to continue my studies in game design to apply this knowledge to a practical subject : video games that I loved playing since I'm a child.

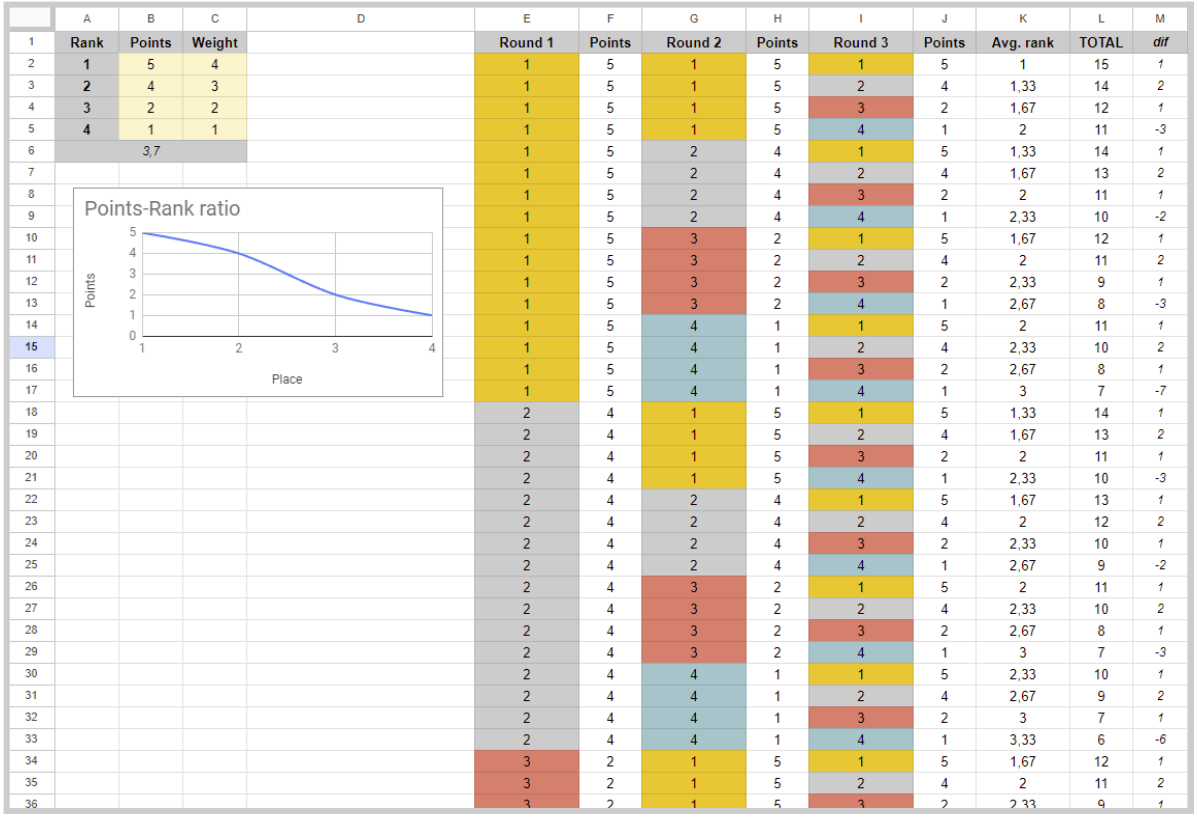
Game projects

During my 5-year at RUBIKA, I had the opportunity to work on various game projects in teams including designers, artists and programmers. Everyone was coming with their own career objectives, thus giving birth to innovant projects, shaped accordingly to match the professional goal of all team members.

Among the plenty of projects I contribute to, some are relevant to my professional objectives, here is a quick overview of them and works I have accomplished for.

In my first year at RUBIKA, I had to make a board game from scratch in a team of 6 people. We came out with "Habemus Papam", a 4-player card game inspired by the conclave of 1492. Players embody one of the 4 favorite cardinals in this election. The goal is to be elected as the new pope, using cheat and manipulation to control the other electing cardinals. Overall, the game is complex enough thanks to its many intertwined mechanics, giving unique game situations every time you play it. The game relies on resource

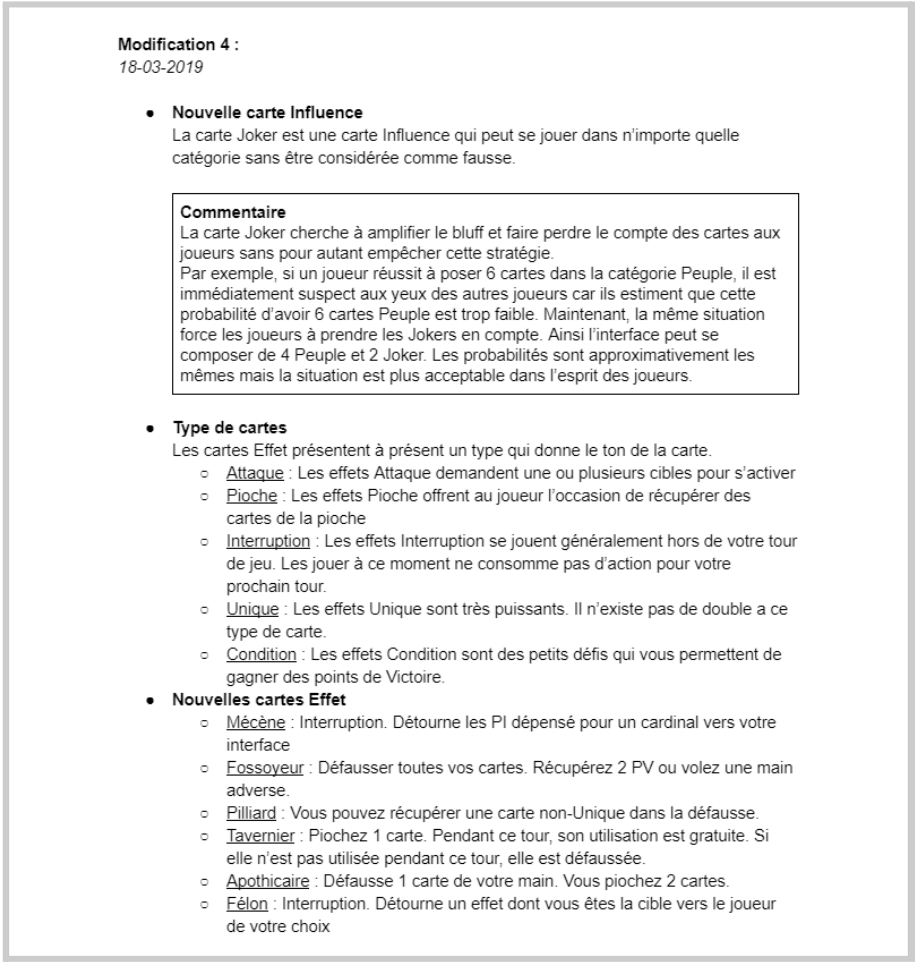
management, bluff and hidden information to hold players' suspicion in a competitive environment.
At the end, "Habemus Papam" was a game with many resource types that had to come together to drive players in one smooth direction. To achieve that, a particular attention was given to the resources flows and game balancing. As the "math guy" of my team, I was in charge of this part. I did several spreadsheets to list the resources of our game and find a good balance between all trades and conversions. By using formulas and graphs, those spreadsheets were useful for the team as anyone could modify some base values to see the effect on resource flows. I also did some documentation at each balancing iteration to list changes made on the game resource economy, why they have been made and what are the conclusions after testing it.



Rank	Points	Weight
1	5	4
2	4	3
3	2	2
4	1	1
3,7		

Round 1	Points	Round 2	Points	Round 3	Points	Avg. rank	TOTAL	dif
1	5	1	5	1	5	1	15	1
1	5	1	5	2	4	1,33	14	2
1	5	1	5	3	2	1,67	12	1
1	5	1	5	4	1	2	11	-3
1	5	2	4	1	5	1,33	14	1
1	5	2	4	2	4	1,67	13	2
1	5	2	4	3	2	2	11	1
1	5	2	4	4	1	2,33	10	-2
1	5	3	2	1	5	1,67	12	1
1	5	3	2	2	4	2	11	2
1	5	3	2	3	2	2,33	9	1
1	5	3	2	4	1	2,67	8	-3
1	5	4	1	1	5	2	11	1
1	5	4	1	2	4	2,33	10	2
1	5	4	1	3	2	2,67	8	1
1	5	4	1	4	1	3	7	-7
2	4	1	5	1	5	1,33	14	1
2	4	1	5	2	4	1,67	13	2
2	4	1	5	3	2	2	11	1
2	4	1	5	4	1	2,33	10	-3
2	4	2	4	1	5	1,67	13	1
2	4	2	4	2	4	2	12	2
2	4	2	4	3	2	2,33	10	1
2	4	2	4	4	1	2,67	9	-2
2	4	3	2	1	5	2	11	1
2	4	3	2	2	4	2,33	10	2
2	4	3	2	3	2	2,67	8	1
2	4	3	2	4	1	3	7	-3
2	4	4	1	1	5	2,33	10	1
2	4	4	1	2	4	2,67	9	2
2	4	4	1	3	2	3	7	1
2	4	4	1	4	1	3,33	6	-6
3	2	1	5	1	5	1,67	12	1
3	2	1	5	2	4	2	11	2
3	2	1	5	3	2	2,33	9	1

Spreadsheet of Victory point distribution that shows every possible outcome of a player. Users can change values in light yellow cells to see the effect of the change.



Modification 4 :
18-03-2019

- **Nouvelle carte Influence**
  La carte Joker est une carte Influence qui peut se jouer dans n'importe quelle catégorie sans être considérée comme fausse.

  Commentaire
  La carte Joker cherche à amplifier le bluff et faire perdre le compte des cartes aux joueurs sans pour autant empêcher cette stratégie.
  Par exemple, si un joueur réussit à poser 6 cartes dans la catégorie Peuple, il est immédiatement suspect aux yeux des autres joueurs car ils estiment que cette probabilité d'avoir 6 cartes Peuple est trop faible. Maintenant, la même situation force les joueurs à prendre les Jokers en compte. Ainsi l'interface peut se composer de 4 Peuple et 2 Joker. Les probabilités sont approximativement les mêmes mais la situation est plus acceptable dans l'esprit des joueurs.

- **Type de cartes**
  Les cartes Effet présentent à présent un type qui donne le ton de la carte.
  - Attaque : Les effets Attaque demandent une ou plusieurs cibles pour s'activer
  - Pioche : Les effets Pioche offrent au joueur l'occasion de récupérer des cartes de la pioche
  - Interruption : Les effets Interruption se jouent généralement hors de votre tour de jeu. Les jouer à ce moment ne consomme pas d'action pour votre prochain tour.
  - Unique : Les effets Unique sont très puissants. Il n'existe pas de double a ce type de carte.
  - Condition : Les effets Condition sont des petits défis qui vous permettent de gagner des points de Victoire.
- **Nouvelles cartes Effet**
  - Mécène : Interruption. Détourne les PI dépensé pour un cardinal vers votre interface
  - Fossoyeur : Défausser toutes vos cartes. Récupérez 2 PV ou volez une main adverse.
  - Pilliard : Vous pouvez récupérer une carte non-Unique dans la défausse.
  - Tavernier : Piochez 1 carte. Pendant ce tour, son utilisation est gratuite. Si elle n'est pas utilisée pendant ce tour, elle est défaussée.
  - Apothicaire : Défausse 1 carte de votre main. Vous piochez 2 cartes.
  - Félon : Interruption. Détourne un effet dont vous êtes la cible vers le joueur de votre choix

Excerpt of a balancing change document.

This project taught me what impact can have useful tools, made for everyone who has questions in mind about the balancing of a game. Spreadsheets are a great support for that because many people are familiar enough with it to understand the purpose of a sheet. Communicating analysis and balance ideas was also tough at the start, but I found that writing documents that can be read quickly, while bringing my thought process with it, was helpful to share my vision with the team.

Another project that brought me more knowledge took place during my last year at RUBIKA. I joined a team of 9 people to make our final student game project that will conclude our game development studies. I worked on “Seance”, a cooperative card game for 3-player fighting against a game master in a similar fashion as Inscryption. Players embody a role playing party playing a new game.

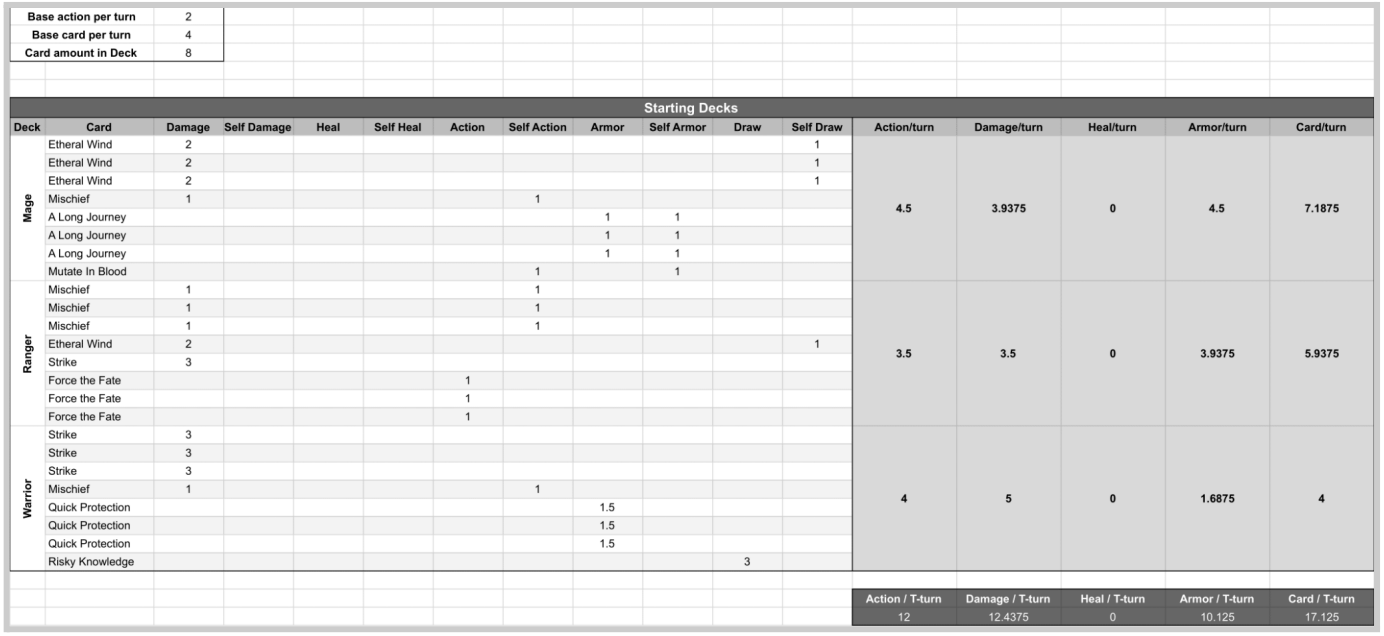
The more they play it, the more they realize they are trapped in it. As the game goes on, the game master is slowly possessed by the Wayfarer, the demon that lies in this cursed board game. The goal is to defeat it at its own game to break the curse, by winning successive fights, using their deck of cards. However, the game master will play by his rules. Players will need to outsmart the Wayfarer and cheat at its rules to have a chance to be freed from the curse.

In the development of “Seance”, I programmed many game features in Unity but I also produced balancing documents and elaborate a complete data workflow for this project. I made an analytics system that could be integrated into game builds and documented it with a tag plan. I developed and monitored an API to allow data to be sent from the game into a database that I also had to manage. With this setup, I was able to extract the players data after each playtests to produce a report about indicators we want to keep an eye on. For instance, during one of our iterations, we wanted to give the game a better pacing, with fights harder and harder. But we noticed some were too long and some were too short. Thanks to the data gathered, it was easy to pinpoint the exact ones that were causing an issue to the pacing. And so, it was easier to adjust the balance of the fights to match the rhythm we aimed for.

Event Name	KPI	Trigger	Event definition	Property name (Unique: 36)	Property definition	Data type	Sample values	Implemented	Notes
Any meaningful action a user takes within yours product	*Quantifiable metric associated to the event being tracked*	*When / Where the event should be triggered in the code*	*Description of the event being performed (for any given end user to understand)*	*Any rich meta data associated to the event, either describing the event itself or the user performing that event*	*Description of the property associated with the event (for ant given end user to understand), if necessary*	*Type of data that the property will take*	*Sample values of the property*	*Wheter the event/property has been implemented or not*	*Any additionnal informations*
GLOBAL									
				event_uuid	UUID of the event	BINARY(16)			All events have to fill these fields
				event_time	Timestamp of the event	TIMESTAMP(3)			
				game_time	Time elapsed since game start	TIME(3)			
				user_uuid	UUID of the user sending event	BINARY(16)			
USER DEVICE									
				device_id	Hash ID of the user device	VARCHAR(40)			User device info are stored in a separate table. The join bewteen device table and event table can be done by using `session_start` event.
				device_model	Model of the user device	VARCHAR(128)			
				device_name	Name of the user device	VARCHAR(128)			
				operating_system	OS of the user device	VARCHAR(128)			
				graphics_name	Name of the user graphics card	VARCHAR(128)			
				graphics_version	Version of the user graphics card	VARCHAR(128)			
				graphics_memory	Approximate amount of graphics memory in megabytes	INT			
				processor_type	Name of the user processor	VARCHAR(128)			
				processor_count	Number of logical processors of the user device	INT			
				processor_frequency	Frequency of user device processor in megahertz	INT			
				memory_size	Amount of system memory present on user device	INT			
				screen_width	Width of the user screen device	INT			
				screen_height	Height of the user screen device	INT			
session_start	- Average session duration	When the application started	A user has started game application					Yes	
				device_id	ID of the user device	CHAR(40)	e0c6e939690a8d7a78b4f22c97e0b02b4afba2a3	Yes	
				game_version	Version of the game	VARCHAR(8)	8Wm04f	Yes	
				dev_build	Build in developpar mode ?	BOOLEAN	TRUE	Yes	
session_end	- Average session duration	When the application closed	A user has closed game application					Yes	
								Yes	
lobby_create	- Game start success rate	When the lobby is created	A user has created a lobby					Yes	
				ip	IP address assigned to this lobby	VARCHAR(15)	127.0.0.1	Yes	
				port	Port assigned to this lobby	UNSIGNED SMALLINT	7767	Yes	
				username	Username used in the lobby	VARCHAR(32)	xXxPsEudOxXx	Yes	
lobby_join	- Game start success rate	When a player join an existing lobby	A player has join a lobby					Yes	
				ip	IP address assigned to this lobby	VARCHAR(15)	127.0.0.1	Yes	
				port	Port assigned to this lobby	UNSIGNED SMALLINT	7767	Yes	
				username	Username used in the lobby	VARCHAR(32)	xXxPsEudOxXx	Yes	
game_start	- Average game duration - Games per sessions	When lobby owner starts game	The game has been started by lobby owner					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
game_end	- Average game duration - Games per sessions	When one of the players in a lobby quit the game	The game has been stopped					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
chapter_reveal	- Average chapter duration	When a chapter is shown on player screens	A new chapter has begun					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				chapter_name	Name of the chapter	VARCHAR(128)	The Last Guard	Yes	
chapter_resolve	- Average chapter duration	When a chapter is ended	A chapter has ended					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				chapter_name	Name of the chapter	VARCHAR(128)	The Last Guard	Yes	
reward_pick	- Reward popularity per class	When a reward is selected by a player	A reward has been picked					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the reward card	VARCHAR(128)	Dagger Throw	Yes	
				player_class	Name of the player class	VARCHAR(32)	Warrior	Yes	
external_pick	- Card usage - Cheat usage	When an external card is pick up by a player from the environnement	An external card has been picked					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the reward card	VARCHAR(128)	Dagger Throw	Yes	
player_begin_turn	- Average armor gain outside player turn - Average action gain outside player turn	When a player begin his turn during a chapter	A player has taken his turn					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				player_class	Name of the player class	VARCHAR(32)	Warrior	Yes	
				health_value	Value of player health	UNSIGNED TINYINT	10	Yes	
				armor_value	Value of player armor	UNSIGNED TINYINT	1	Yes	
				action_count	Amount of player action	UNSIGNED TINYINT	2	Yes	
player_finish_turn	- Average armor gain during player turn - Average action gain during player turn - Average action usage per turn	When a player finish his turn during a chapter	A player has finished his turn					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				player_class	Name of the player class	VARCHAR(32)	Warrior	Yes	
				health_value	Value of player health	UNSIGNED TINYINT	10	Yes	
				armor_value	Value of player armor	UNSIGNED TINYINT	3	Yes	
				action_count	Amount of player action	UNSIGNED TINYINT	0	Yes	
				action_used	Amont of action used during turn	UNSIGNED TINYINT	3	Yes	
book_open	- Player book usage	When a player opens the plyer book on the table	A player has clicked on the player book					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
book_close	- Player book usage	When a player closes the plyer book on the table	A player has clicked the 'close' button in the book					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
enter_death_door	- Failure rate by chapter	When one of player health value hits 0, and enter in death door state	A player has no more health point. He is in Death Door state					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				chapter_name	Name of the chapter	VARCHAR(128)	The Last Guard	Yes	
exit_death_door	- Failure rate by chapter	When one of player leave Death Door state	A player in Death Door has regain health points or current chapter has ended					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				chapter_name	Name of the chapter	VARCHAR(128)	The Last Guard	Yes	
player_die	- Failure rate by chapter	When a player in death door take damage	A player has died. Game is over.					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				chapter_name	Name of the chapter	VARCHAR(128)	The Last Guard	Yes	
card_drew	- Card playrates - Playrate per chapter - Playrate per class	When a card is drew	A card has been drawn					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the card	VARCHAR(128)	Dagger Throw	Yes	
card_play	- Card playrates - Playrate per chapter - Playrate per class	When a card is played	A card has been played					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the card	VARCHAR(128)	Dagger Throw	Yes	
card_discard	- Card playrates - Playrate per chapter - Playrate per class	When a card is discarded	A card has been discarded by player					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the card	VARCHAR(128)	Dagger Throw	Yes	
card_dump	- Card playrates - Playrate per chapter - Playrate per class	When a card is dumped	A card has been dumped at player end turn					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the card	VARCHAR(128)	Dagger Throw	Yes	
cheat_health_dice	- Cheat usage ratio - Cheat usage at low health - Cheat usage at high health	When a health dice value change outside normal game rules	A player has cheated on his health dice value					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				initial_health_value	Value of health dice before cheat	UNSIGNED TINYINT	6	Yes	
								Yes	
cheat_armor_dice	- Cheat usage ratio	When a armor dice value change outside normal game rules	A player has cheated on his armor dice value					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				initial_armor_value	Value of armor dice before cheat	UNSIGNED TINYINT	0	Yes	
								Yes	
cheat_draw	- Cheat usage ratio	When a card is drawn outside normal game rules	A player has cheated by drawing another card					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				initial_card_amount	Amount of card in hand before cheat	UNSIGNED TINYINT	4	Yes	
								Yes	
cheat_discard	- Cheat usage ratio	When a card is discarded outside normal game rules	A player has cheated by discarding one of his card					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				initial_card_amount	Amount of card in hand before cheat	UNSIGNED TINYINT	5	Yes	
								Yes	
cheat_pass	- Cheat usage ratio	When a player pass a card to another player	A card has been passed from a player to another					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the card	VARCHAR(128)	Low Blow	Yes	
cheat_receive	- Cheat usage ratio	When a player receive a card from another player	A card has been received from a player					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the card	VARCHAR(128)	Low Blow	Yes	
cheat_goop	- Cheat usage ratio	When a player mask a card effect with game master corruption	A card effect has been intentionnaly masked by a player					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of the card	VARCHAR(128)	Low Blow	Yes	
				effect_id	ID of the effect (effect ordering on card)	INT	1	Yes	
cheat_hide	- Cheat usage ratio - Cheat usage per card	When a card is hidden by the player	A player has cheated by hidding a card					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of card hidden	VARCHAR(128)	Low Blow	Yes	
cheat_unhide	- Cheat usage ratio - Cheat usage per card	When a card is pick up by the player from a card stash	A player has cheated by unhidding a card					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				card_name	Name of card hidden	VARCHAR(128)	Low Blow	Yes	
cheat_detect	- Success rate of cheats	When the game master detect a player cheat	The game master has detected a player cheat					Yes	
				game_uuid	UUID of the game	BINARY(16)	deb33bd0-82de-4b30-be31-5ef73aaeb4cd	Yes	
				cheat_type	Type of the detected cheat	VARCHAR(128)	DrawCard	Yes	

Tag plan of Seance

For the card balancing, I had the experience of similar games with previous projects. I made some spreadsheets to model the game systems at a low level. For cards, I reduce their effects to simple categories (Damage, Heal, Armor, Self Armor... etc). This simplification allows me to calculate the esperance of each one depending on the others. I also evaluate every category by “table turn”. This helps me to balance the fights as I can know the average damage players can deal and the average amount of damage they can neglect with armor per table turn, with a given deck of cards.



Base action per turn	2
Base card per turn	4
Card amount in Deck	8

Starting Decks

Deck	Card	Damage	Self Damage	Heal	Self Heal	Action	Self Action	Armor	Self Armor	Draw	Self Draw	Action/turn	Damage/turn	Heal/turn	Armor/turn	Card/turn
Mage	Etheral Wind	2									1	4.5	3.9375	0	4.5	7.1875
	Etheral Wind	2									1					
	Etheral Wind	2									1					
	Mischief	1					1									
	A Long Journey							1	1							
	A Long Journey							1	1							
	A Long Journey							1	1							
	Mutate In Blood						1		1							
Ranger	Mischief	1					1					3.5	3.5	0	3.9375	5.9375
	Mischief	1					1									
	Mischief	1					1									
	Etheral Wind	2									1					
	Strike	3														
	Force the Fate					1										
	Force the Fate					1										
	Force the Fate					1										
Warrior	Strike	3										4	5	0	1.6875	4
	Strike	3														
	Strike	3														
	Mischief	1					1									
	Quick Protection							1.5								
	Quick Protection							1.5								
	Quick Protection							1.5								
	Risky Knowledge									3						

Action / T-turn	Damage / T-turn	Heal / T-turn	Armor / T-turn	Card / T-turn
12	12.4375	0	10.125	17.125

Spreadsheet of the balance of the starting decks

The game master was also part of the balancing. As players can cheat, we wanted the game master to be able to detect them and punish players that weren't discreet enough while cheating. We came up with a “suspicion system” that relies on players actions to vary a “suspicion score” that will impact the game master behavior towards players. I prototyped this idea using a spreadsheet to model the backbone of the system step-by-step. The model abstracted some game elements and mimicked player behaviors with randomness, to keep the main part we wanted to test. By using a variable table, it was easy to change a parameter and see the result on the model, so anyone could use it to quickly prototype a change on the game master AI and check if that met their expectations.

Step	Game Turn	Num. Chapter	GM Turn	P1 Turn	P2 Turn	P3 Turn	P1 Deck Power	P2 Deck Power	P3 Deck Power	Global Power	P1 Sus act count	P2 Sus act count	P3 Sus Act count	P1 Cheat count	P2 Cheat count	P3 Cheat count	P1 ISS	P2 ISS	P3 ISS	Room Suspicion	GSS	EISS
1	1	1	☑	☐	☐	☐	8	13	8	29	0	1	0	1	1	0	5	6	0	1	12	
2	1	1	☐	☑	☐	☐	8	13	8	29	0	1	0	1	1	0	5	6	0	1	12	18
3	1	1	☐	☐	☑	☐	8	13	8	29	0	1	0	1	1	0	5	6	0	1	12	19
4	1	1	☐	☐	☐	☑	8	13	8	29	0	1	0	1	2	0	5	11	0	1	17	18
5	2	2	☑	☐	☐	☐	13	17	8	38	0	1	0	1	2	1	5	11	5	1	22	
6	2	2	☐	☑	☐	☐	13	17	8	38	0	1	0	1	2	1	5	11	5	1	22	28
7	2	2	☐	☐	☑	☐	13	17	8	38	0	1	0	1	2	1	5	11	5	1	22	34
8	2	2	☐	☐	☐	☑	13	17	8	38	0	1	0	1	2	1	5	11	5	1	22	28
9	3	2	☑	☐	☐	☐	13	17	8	38	0	1	0	1	3	1	5	16	5	1	27	
10	3	2	☐	☑	☐	☐	13	17	8	38	0	1	1	1	3	1	5	16	6	1	28	34
11	3	2	☐	☐	☑	☐	13	17	8	38	0	1	1	1	3	1	5	16	6	1	28	45
12	3	2	☐	☐	☐	☑	13	17	8	38	0	1	1	1	3	1	5	16	6	1	28	35
13	4	3	☑	☐	☐	☐	13	17	8	38	0	1	1	1	3	1	5	16	6	2	29	
14	4	3	☐	☑	☐	☐	13	17	8	38	0	1	1	1	3	1	5	16	6	2	29	36
15	4	3	☐	☐	☑	☐	13	17	8	38	0	1	2	2	3	2	10	16	12	2	40	58
16	4	3	☐	☐	☐	☑	13	17	8	38	0	1	2	2	3	2	10	16	12	2	40	54
17	5	3	☑	☐	☐	☐	13	17	8	38	0	1	2	2	3	2	10	16	12	2	40	
18	5	3	☐	☑	☐	☐	13	17	8	38	0	1	2	2	3	2	10	16	12	2	40	52
19	5	3	☐	☐	☑	☐	13	17	8	38	0	1	2	3	3	2	15	16	12	2	45	63
20	5	3	☐	☐	☐	☑	13	17	8	38	0	1	2	3	4	2	15	21	12	2	50	64
21	6	3	☑	☐	☐	☐	13	17	8	38	0	1	2	3	4	2	15	21	12	2	50	
22	6	3	☐	☑	☐	☐	13	17	8	38	0	1	2	3	4	2	15	21	12	2	50	67
23	6	3	☐	☐	☑	☐	13	17	8	38	0	1	2	3	4	3	15	21	17	2	55	78
24	6	3	☐	☐	☐	☑	13	17	8	38	0	1	2	4	5	3	20	26	17	2	65	84
25	7	4	☑	☐	☐	☐	14	22	8	44	0	1	2	4	5	3	20	26	17	5	68	
26	7	4	☐	☑	☐	☐	14	22	8	44	0	1	2	4	5	3	20	26	17	5	68	93
27	7	4	☐	☐	☑	☐	14	22	8	44	0	1	2	4	5	3	20	26	17	5	68	99

Model of the "suspicion system" used by the game master AI. One step represent a player turn
Yellow cells show value that changed over the previous step.
(ISS = Individual Suspicion Score ; GSS = Global Suspicion Score ; EISS = Effective Individual Suspicion Score)

Name	Description	Default	Minimum (incl.)	Maximum (excl.)	Set Random	Value
ChapterSuccess	Probability of completing a chapter during a game turn	0.4			☐	0.4
DeckPowerImprove	Probability of improving a player deck after a chapter	0.8			☐	0.8
SuspiciousAction	Probability that a player does a suspicious act outside of thier turn	0.95			☐	0.95
CheatAction	Probability that a player cheats during their turn	0.8			☐	0.8
SuspiciousScoreWeight	Weight of a suspicious action in the Individual Suspicion Score	1			☐	1
CheatScoreWeight	Weight of a cheating action in the Individual Suspicion Score	5			☐	5
BaseNormRoomSus	Base value for suspicion score for a room	1			☐	1
FactorNormRoomSus	Factor value to applied to Normal distribution of suspicion score for a room	25			☐	25
MeanNormRoomSus	Parameter of the logistic distribution that determines Room Suspicion Score	40			☐	40
DerivationNormRoomSus	Parameter of the logistic distribution that determines Room Suspicion Score	20			☐	20
SpreadNormRoomSus	Parameter that determines Room Suspicion Score spread		1	5	☑	2

Variable table used by the “suspicion system” model

Interest about data

So how does an apprentice game designer end up with data as his career path ?
During the coronavirus pandemic, I was in my third year of study and I found myself a bit lost with my future. I didn’t see myself doing game design for my entire career. I wasn’t even enjoying it that much at school. Still, it is great to work with video games, working with people that share the same interest with the media but there was something else I needed to do to fulfill my career path. Looking at all I’ve done, all my past studies, things that I have pleasure to do, it was obvious I was missing something somewhere.

After days and nights thinking about this question, I found that a job in data could resolve the issue and link everything together to form a whole. After some research about some jobs in data, I read some job offers about data analysts in the video game industry. And surprisingly, it matches with things I like and I can do ! Working with math, using spreadsheet, Python programming, database management, analysis skills, writing reports for people ? I can do all of that. All I have to do is to practice more of those skills and fill the gaps I have in my knowledge. A plus of a data analyst is it can be valuable anywhere, not only in video games. A blessing for me who fears being bored of the industry someday.

Knowing what to aim for, I now have another issue to face. RUBIKA, my game design school will not help me to develop the skill I need for this position. There are too few courses about data, balancing or analysis to bring me to a higher level. I have to work on my own, find my shortcomings and fill them alone, while finishing my studies at RUBIKA. That is the tough part.

During my learning, I found that I could participate in an online data analysis course. I started data analysis training on OpenClassroom that should teach me everything I need to know. However, keeping up with two schoolings at once was not manageable, especially because RUBIKA is a school which is calling for big results and can put huge pressure on students to accomplish that goal. I choose to put my online course aside, to focus on the master diploma that should bring me more value over time at the cost of being late on my objective. One work around I did was to work on a data analysis personal project that I can work on when I have time. The goal with this project is to go through as many skills as I can to be able to find a data analyst position after my RUBIKA schooling. I found that this solution fit me way better than any other, as I am the kind of person who loves learning by himself with concrete projects that could be the starting point of a portfolio.

Project overview

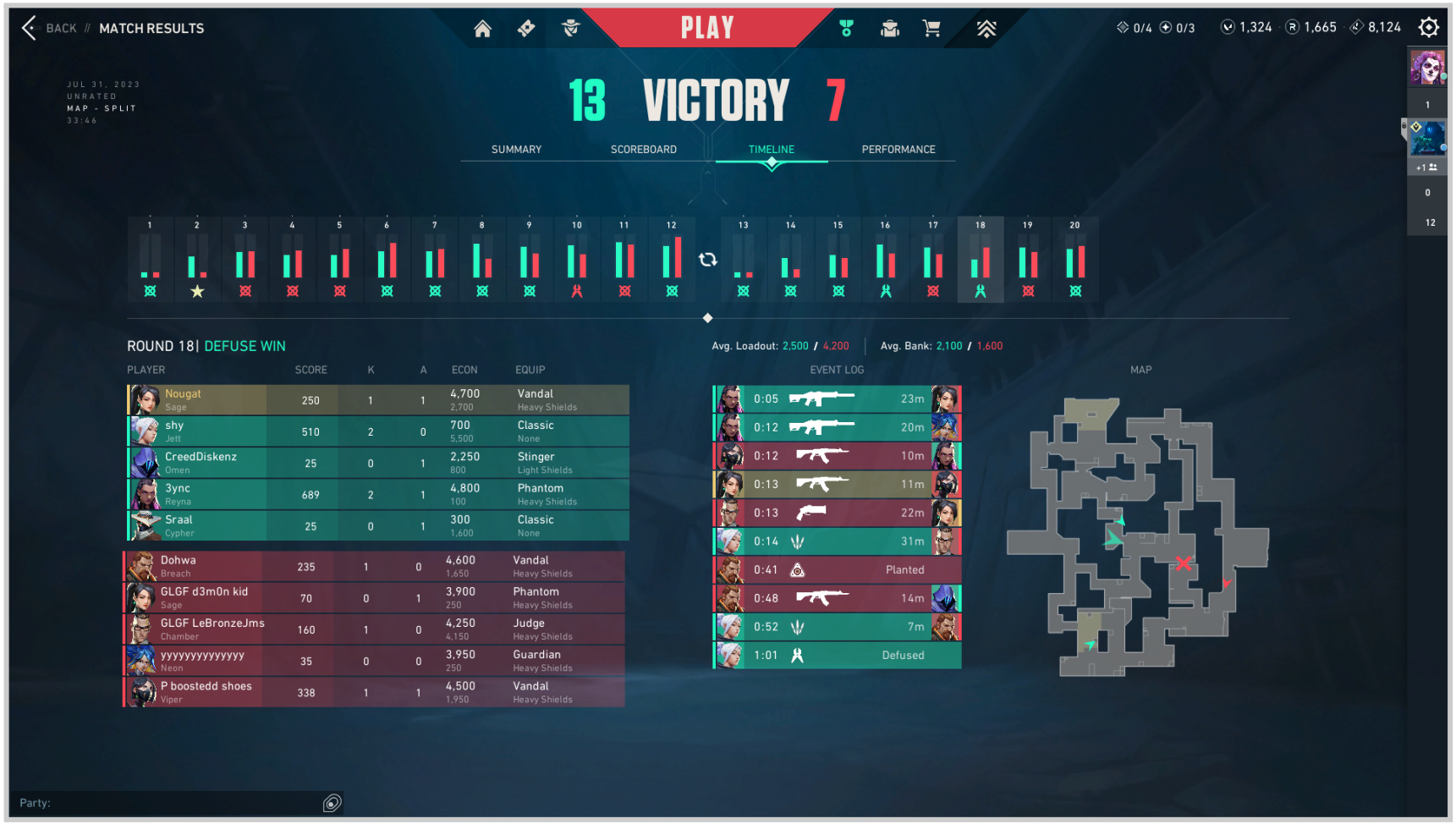
To find a good project to work on, I looked at ideas that will check some requirements. Firstly, I would like to work with game player data. I could find this by looking at some public game API available online, but not every API offers convenient player data. They often give game data which is more static, like a list of items, cards, equipment names and descriptions and other similar data.
I will also need to fetch the data from the API and store it into a database to work with more efficiently. Python will work fine for a fetching algorithm. I already have a good mastery of this language and as one of the standard programming languages in data science, it looks like the best choice to improve myself. I also have a good mastery at managing databases as I already know SQL language. For the database management system, I choose MySQL as it is popular and it will be easier to manage on my computer.
Then comes the analysis tool, where many choices appear to me. I could go with SQL extract than I could import into a spreadsheet like Excel. I have the possibility to use Python again with some data science libraries like Pandas, NumPy, SciPy and Matplotlib. But I chose another option to let myself learn something completely new : Tableau Software. During my research about data analyst positions in the video game industry, I noticed a trending skill in most of the job offers : the capacity to use a data visualization tool, Tableau and PowerBI being the most requested. I felt better to go with Tableau as I seem more popular and I found enough learning resources to start without a hitch.

With all this pipeline in mind, I now have to find my study subject. Looking toward competitive multiplayer games will be my best option as they depend a lot on player data to do their balancing and monitor game health state, which is the main condition to maintain competitiveness and fairness for all players. This is a key point. If these products fail at keeping a good state, they attract less players, or even lose active players. As most of those games work on a game-as-a-service model, a lack of players means a loss of income. That is why the data has a crucial role in those games and might be the most interesting to work with.

My attention was driven towards VALORANT, a first-person tactical shooter game, developed by Riot Games, the company behind one of the colossuses of the video game landscape : League of Legends. The game came out in mid-2020, during the COVID pandemic that carried it to be one of the most popular games at its launch.

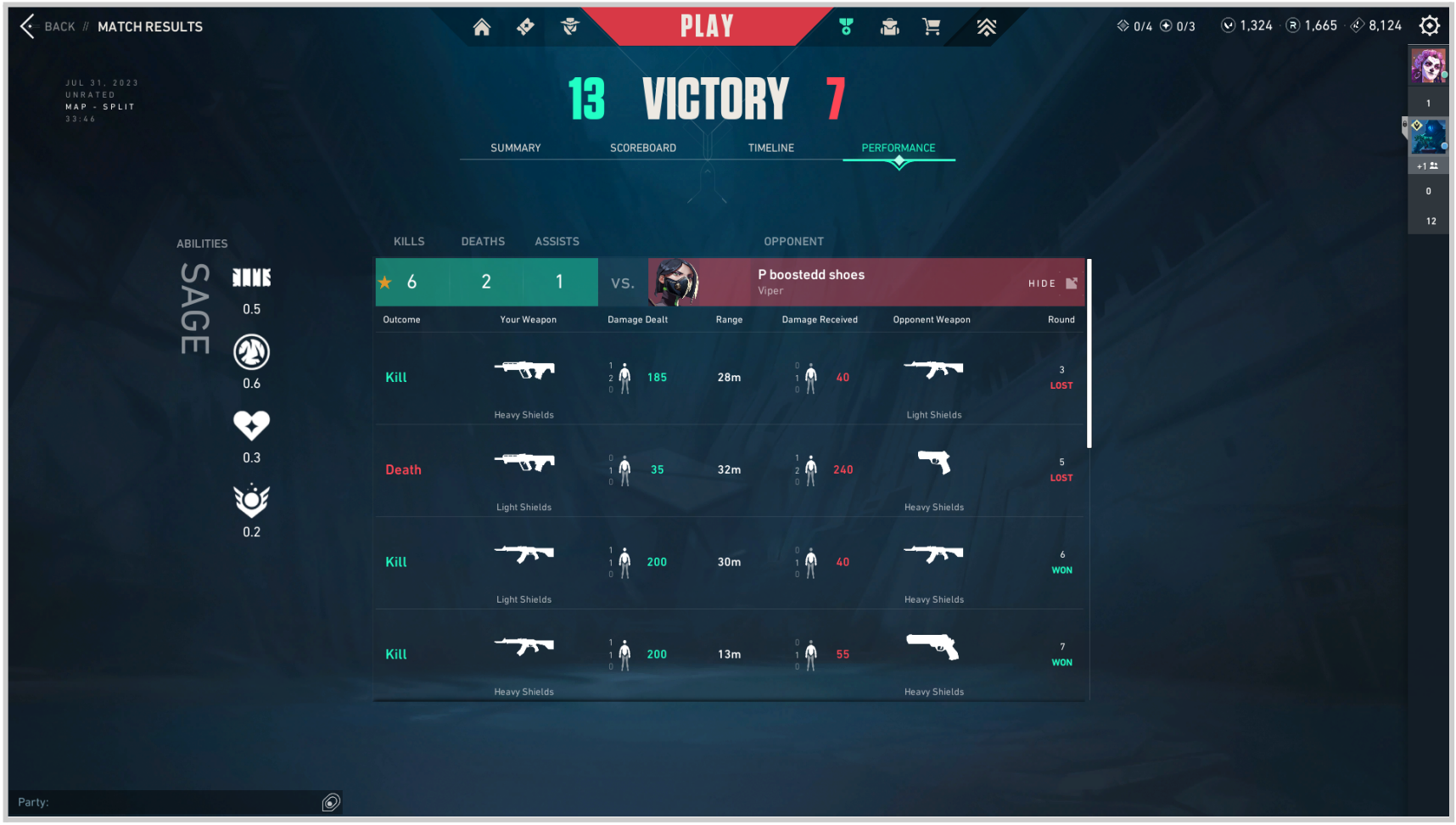
Players face each other in a 5v5 match that consists of multiple similar rounds. The first team who wins 13 rounds, wins the match. One team plays the Attackers, the other one plays the Defenders. After 12 rounds, they will swap sides. To win a round, a team has to eliminate the opposing team, or complete their objective : planting or defusing an exploding device, depending on their side. All players embody an Agent, having their own abilities that could create tactical opportunities along the round. Rounds are linked one after another by an economic cycle. A round gives more or less economic resources to players, depending on the result of the previous round. With this resource, players can obtain various weapons, each one with a particular gunplay, strengths and weaknesses.

VALORANT is a game that I love to play. That is a benefit for this project, knowing the game will definitely help me to have a better understanding of my data. Moreover, I have been fascinated by the game since the first day I played it. The main reason is a bit silly, but it definitely represents me well. At the end of each game, a game report is made and shown to the player. Here is an example of what could be displayed in those.



Timeline screen

It shows round outcomes, player performances by round, round events and players positions during each event.



Performance screen.
It shows every fight the player has been caught in and how it ended, giving details about each fight.

Those reports are very detailed and give an overall view of player performance during the game, each round, each event that happened, each kill, each damage event inflicted or received. Most of the game is translated on those screens and players can see their previous match reports if they want. That means all this data should be stored somewhere to let the player rewatch it later. And since I saw all those numbers I got obsessed with being able to access it, from outside the game.

Another point that pushes my idea to study VALORANT is the work Riot Games already did with their data. By way of developer blog posts, Riot Games devs share some insights about the making of VALORANT. Some articles were about the Insight department and how

they use data to maintain the ecosystem of the game. These articles are deeply inspirational and convince me even more to dedicate this project to VALORANT.

Here is some link to the dev blog, and some of the articles they wrote concerning balancing and data studies :

- **VALORANT Developer Blog** : https://playvalorant.com/en-us/news/dev/
  - *How we balance VALORANT* :
    https://playvalorant.com/en-us/news/dev/how-we-balance-valorant/
  - *Trust the balance process: Data and Insights* :
    https://playvalorant.com/en-us/news/dev/trust-the-balance-process-data-and-insights/
  - *VALORANT Data Breach: 9-3 curse* :
    https://playvalorant.com/en-us/news/dev/valorant-data-breach-9-3-curse/
  - *VALORANT Data Drop: Phantom vs. Vandal* :
    https://playvalorant.com/en-us/news/dev/valorant-data-drop-phantom-vs-vandal/

Project presentation

Data source

To realize the data analysis, I require a large amount of recent player data. I have several options to obtain the data I need.

The first option is to get access to the official API. Riot Games offers resources for developers to create third-party software around their games. This option is the “safest” as all the data should be completely raw, allowing me to explore many ideas for my analysis. However, this could be the perfect solution for a League of Legend or a TeamFight Tactics project as all Riot Games products benefit from a free personal access to the API. The only exception is VALORANT that requires a strong project to obtain a production key to use the API. Unfortunately, this process could be long as project submission reviews can be very slow. So by lack of time, this data source wasn’t exploitable.

The second option is to scrap the data from third-party websites, like tracker.gg or blizz.gg, which use an official Riot Games API access. I could scrap the data using the Selenium Python library. However, the data might have a limited usage, those websites don’t publish raw data from the API. The data is aggregated in the backend and stays inaccessible to the standard website user. Knowing all the data lost between the API and the website, I keep looking for a better way to access the data.

I searched on the web for some datasets that could contain enough data for me to use. During this research, I found an interesting GitHub repository about an “Unofficial VALORANT API”. After a closer look, this unofficial API might be what I need. There were many useful endpoints and the data seems to be as raw as the original API. To be sure, I wonder where that data came from, it could not be a simple redirection of the official API, that would lead to many security issues. After a complete inspection of the project documentation, I discovered a small developer community around VALORANT that uses the in-game API. Given that the VALORANT game client has to access live data to show players leaderboards or match reports, for instance, that means the game has access to its own API to request and receive the live data.

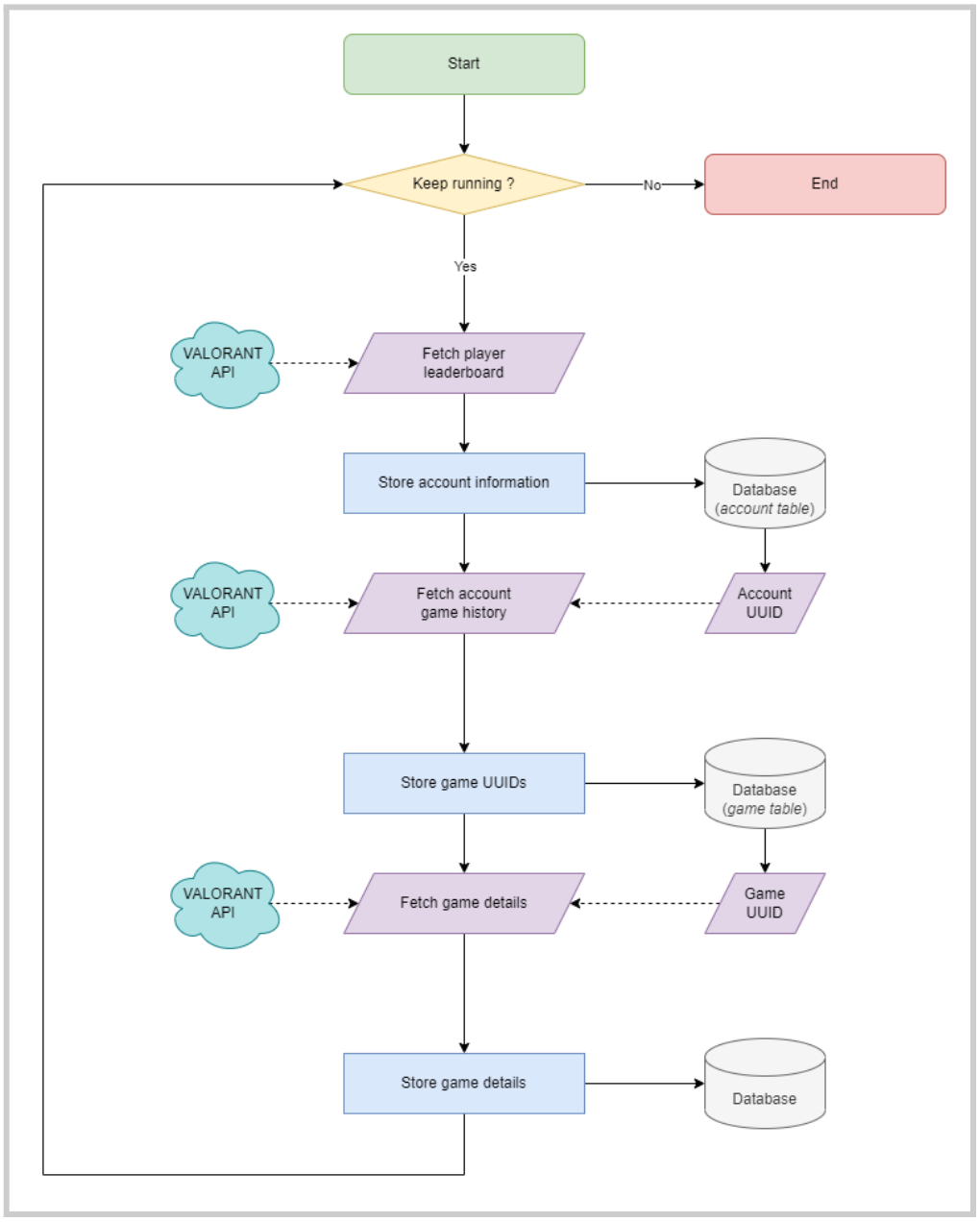
To avoid spamming the unofficial VALORANT API which is a small project I don’t want to cause issues with, I choose to follow the path of its developper and use the in-game API, which is similar to the official API without the official support of it. However, there is a major caveat : This method is legally in a gray area. It is not reprehensible, as to access the endpoints I have to authenticate officially to Riot Game services, but still, it might not be the way that Riot Games wants developers to exploit to get some data. Despite this issue, this option was the best way to fetch enough raw data to lead my analysis project. By taking extra precautions during the data collection process, I should be able to extract enough data without causing problems to Riot services or myself.

Data collection

Data flow

Now I have found my data source, I have to find my way through the available data to fetch those I have interest in. After reading the documentation made by the community, I did some requests on some endpoints of the API that pulled my attention. The most significant was the endpoint that sent me back match details data of a specific game, identified by an UUID. The typical response is a long JSON containing participating players, round details, kill descriptions and every other aspect of the game. I was able to conclude that this endpoint was the one used by the game to show those match report screens I described earlier.

However, this endpoint requires a valid game UUID beforehand. Now, I have to find a source of game UUID to feed this endpoint. I came up with the following dataflow :



Flowchart of the data collection process main loop

To be able to extract recent games, I made a 3-step flowchart.

In the first step, I want active player account UUIDs. To obtain a good amount of account UUID, I used the leaderboard endpoint. This endpoint sends me back the account basic information of players present in the leaderboard (Username and UUID). VALORANT ranking system works in a way this leaderboard is reset and emptied every 2 months approximately, guarrenting me to only get active players.

The next step is to fetch the match history of those accounts with a second endpoint that simply sends back a list of game UUID in which the account has participated. I choose to only get games played in Competitive mode. Game UUID are removed from the list after a period of time which lets me only see the most recent games of each account.

The last step is to complete the process by using those game UUID to the match details endpoints. Match details are closing the loop, they bring new unknown accounts participating to games, those accounts are saved for the next algorithm cycle.

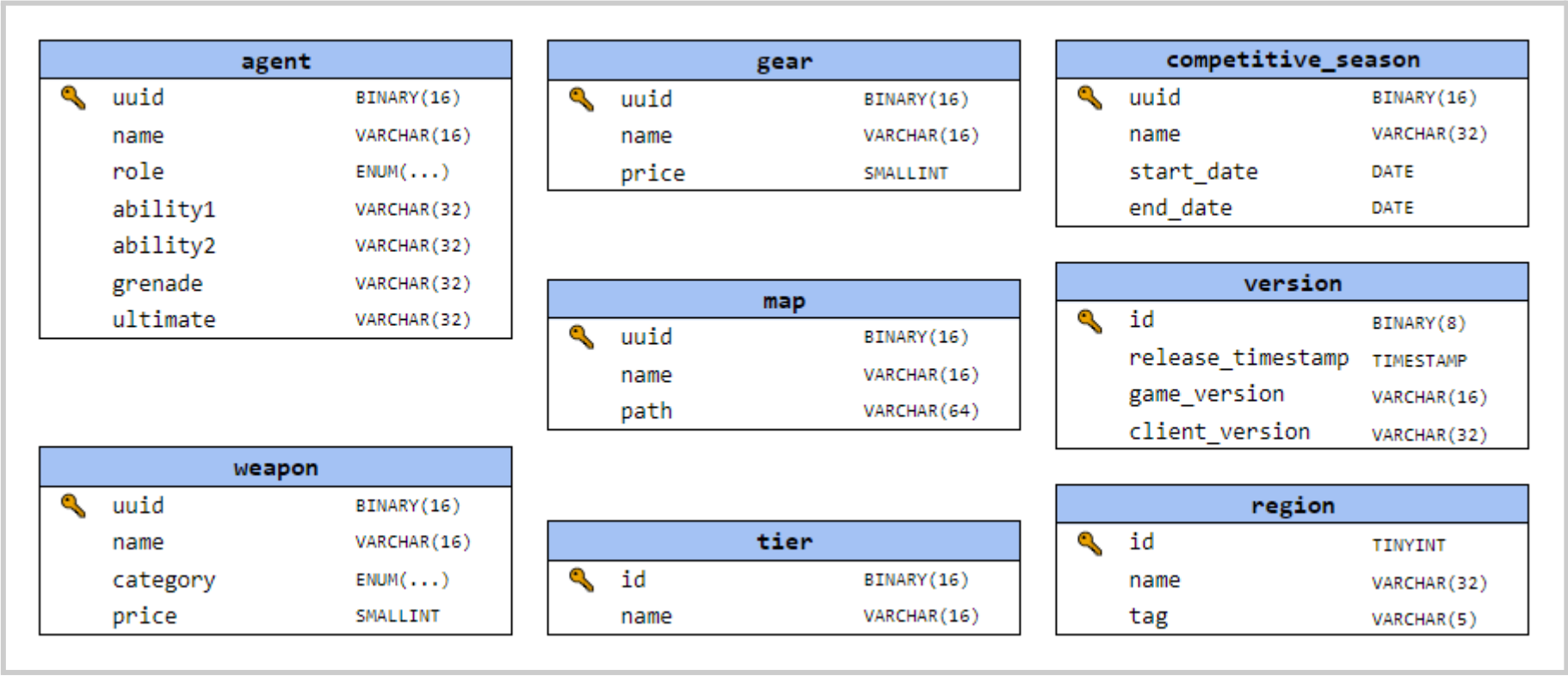
Note that all the data received during the whole process is stored in the database. This allows me to keep track of which requests have already been done to avoid sending twice the same one. The database also serves as a save and backup solution. If the process needs to be interrupted for any reason, it can be resumed later without any data loss.

An idea I had during the development was to increase and diversify the accounts to counter a major flaw of this system. The player leaderboard gives only a list of top players of a region (In EU servers, 15,000 players on average, approximately 1% of the total rated player base), all the accounts will be high-rated accounts. If I want to study data of lower rated players, this system will not be reliable. One possible solution is to add a fourth step in the flow that will fetch games played in Deathmatch mode. This game mode matches 12 players to play a free-for-all game. Matchmaking conditions for this mode are much more permissive than in Competitive mode. By fetching Deathmatch games of known accounts, it is possible to get plenty of new accounts by looking at participating players that have a high chance to not be in the leaderboard. This idea didn’t make it in the end because of a lack of time, but it is a great improvement that could be added later on.

Database

The database will be the heart of this project. It will receive the data and it will have to return it for a later usage. As previously said, I choose MySQL to operate the database, and I will run it locally to limit my financial cost. This implies that my personal hard drive will hold this data. To keep a good quality in my analysis, I will need a decent amount of games in the database but with my limited space to store the data, I need to be conscientious with my database management.

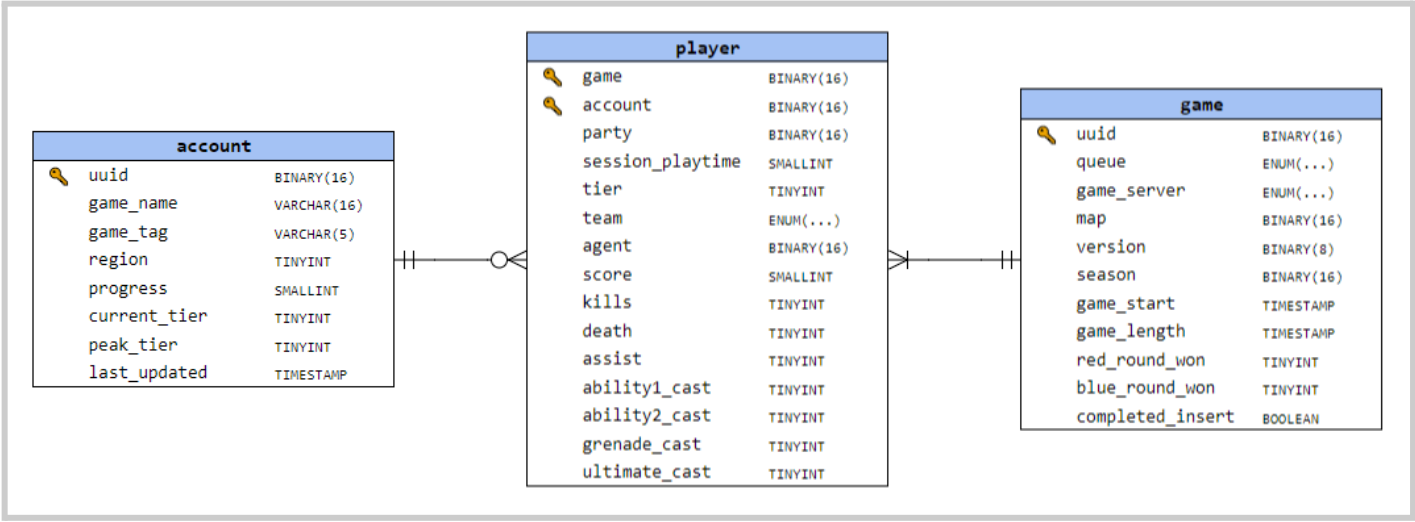
My tables need to be well organized to optimize my available space. Firstly, I have some tables that contain data that don't frequently change overtime like Agents, Maps, or Weapons. I called them "static tables". Most of them only have one field as primary key that correspond to an UUID or an ID



Static tables of the database

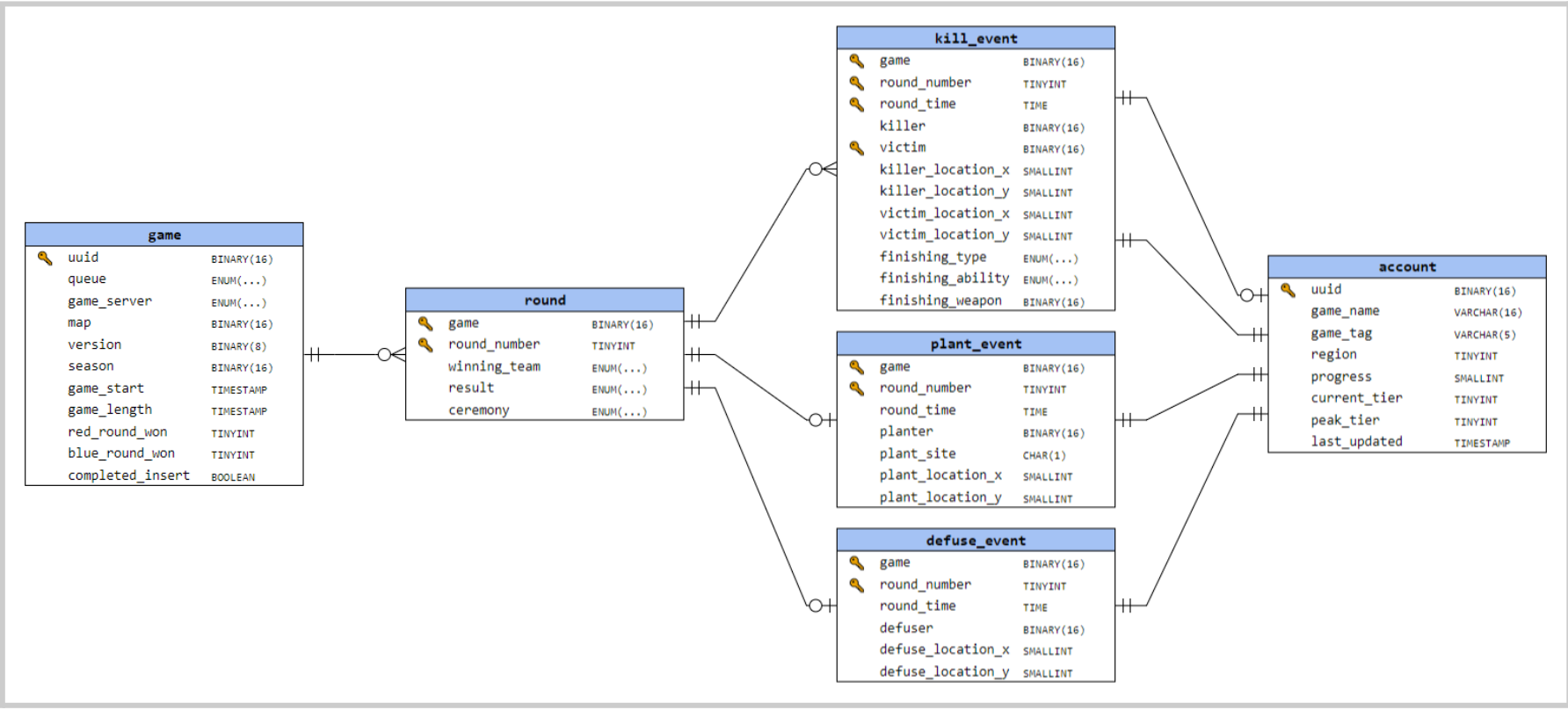
Those are the first tables I made and they were important as they are the base of VALORANT game data and the database structure.
By exploring the response of the VALORANT API, I noticed that most of the references between game elements are made through UUIDs. To limit their impact on the storage space, I convert all of them to BINARY objects. That way, each UUID takes only 16 bytes of space instead of a potential 128 bytes if stored as CHAR(32) with UTF-8 encoding. I occasionally use ENUM type that only takes 1 byte to store. Numerals are stored into INT types that match their potential maximum size. Same with string values, stored into VARCHAR that are limited in size. Time values are stored into DATE, TIME or TIMESTAMP types, depending on the need, taking 3 and 4 bytes of storage respectively. I'm avoiding the DATETIME format that takes 8 bytes. These typing rules will be applied to all tables in the database.

Next we have the table holding the player data. They are built around one or many relations. We have the Account table and the Game table that hold data acquired from the main process of data collection. Those two tables are linked together by a third table, the Player table, representing an account participation to a game.



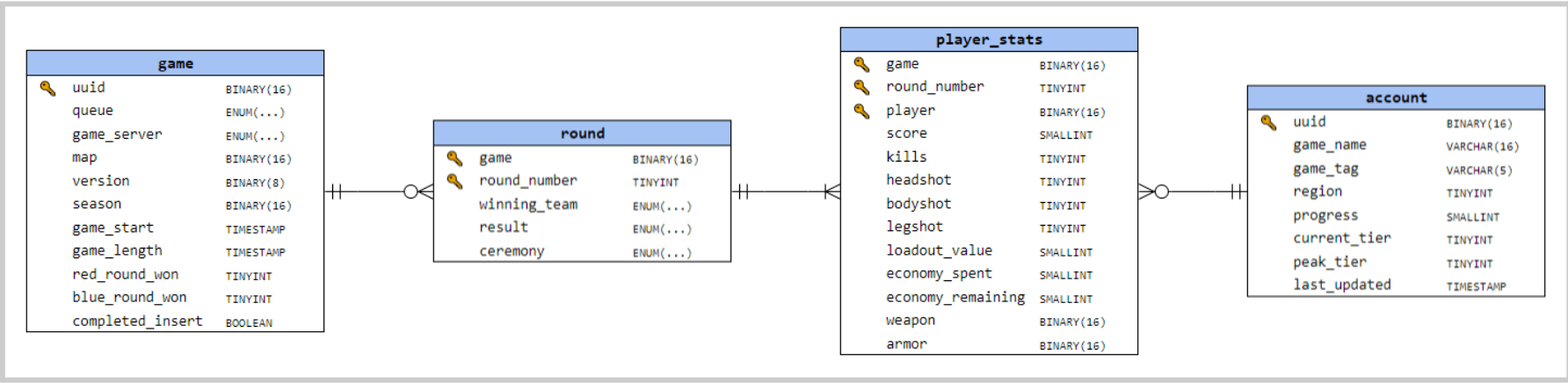
Account, Player, and Game table relations

To store game details, the API response is splitted in lighter parts like Rounds, Plant/Defuse events and Kill events. Those tables depend on Account table and Game table.



Round, Plant event, Defuse event and Kill event table relations

Then, to store player stats, round-by-round, I have a Player stats table that has similar relations as round event tables.



Player stats table relations

Together, those tables can describe any game that has been fetched. The table structure and relations will also help for the analysis part. The database is now ready to receive tables. After I wrote the table creation script in SQL, the base was ready for the next step. I also create some views that will be helpful to quickly display the data, understand it and work with it efficiently.

Python

Now we have a ready database, I have to fill it with precious data needed for the analysis. To do so, I have to create an algorithm that sends requests to the API, parse API responses and build SQL query from parsed data to send to the database. As I said earlier, I will use Python as I'm familiar with the language. The program should be made using asynchronous programming. Async programming is almost mandatory when you are dealing with web requests. That allows it to work on multiple tasks while the asynchronous ones are pending. Asyncio and Aiohttp libraries will be

I started by making a Requester module that will manage requests waiting to be resolved. Its job is to dispatch waiting requests to available request handlers. Handlers can be added by the user to work with proxies. Handlers are independent and can suspend themselves if a problem occurs, like receiving a 429 Too Many Request or 403 Forbidden errors from the API.

Above that, I made a Valorant module which is technically a wrapper of the VALORANT API that uses the Requester module. That module consists of simple functions like `pull_competitive_leaderboard(season: uuid)` or `pull_game_detail(game: uuid)`. Each function builds the right request and orders the Requester module to resolve it. In the same way, I made a Database module which consists of similar simple functions to store the data into the database like `push_account(account_data: dict)` or

`push_round(round_data: dict)`. Both modules handle connections and authentication to create the data input and output of the program.

Then in the middle I have the core of the data collection. Like a conductor, this part of the program, made of many other modules, is the one that will dictate which requests should be made, with which parameters. In an Operation module, I have an Operation Executor which can perform any possible operation. An operation consisting of one pull from the Valorant module, the parsing of the data and one or many pushes to the Database module. Operations to do are determined by a Strategy module that picks a strategy depending on the number of cycles completed, the previous strategy and the amount of available data still unused in the database. The strategy changes at each program cycle.

Finally, I made some convenient modules to help me during the development and the program execution. First, I added a logging feature for each module to have a good overview of the program during a run. I also made an administration system using the Python socket library and an administration terminal that I can connect to the main program, send commands and have some feedback about the current execution. This helps me a lot to have some extra control on the execution flow and it is a precious tool for debugging.

When I felt right with my program, without major bugs, and able to run in complete autonomy, I started a collect session. The program ran for several days. During this time, I kept monitoring the program and the database to avoid blocking behaviors or any database overflow. When all was doing great, I started to explore the data.

Exploratory analysis

Data description

At the end of the data collection process, I had inserted in my database :

Tables	Entries	Entries *(valid games)*
Account	565 025	n/a
Game	5 421 180	109 002
Player	1 127 329	1 090 020
Round	2 379 147	2 299 478

Plant event	1 461 793	1 412 796
Defuse event	391 959	378 785
Kill event	17 350 980	16 798 110
Player stats	23 771 893	22 994 775

This table presents the amount of entries for each table. However, due to possible collection issues like manual interruptions, some games didn't complete their insertion into the database. Those are tagged as invalid. We will keep only valid games for the analysis. Database entries related to valid games are specified in the third column.

The program has fetched over 5 million game UUID. Only 2% of them have been detailed and are valid, providing data about their players, rounds and round events. These 100 000 games are shared out between 2.3 million rounds. On average, one game goes on for 21 rounds (average score: 13-8).

The Spike (object that attackers have to plant, and defenders have to defuse if planted) has been planted on 61% of the rounds. Only 27% of those planted Spike have been defused.

16.8 million kills have been registered. On average, one player makes 15.4 kills per game and 7.3 kills happen per round.

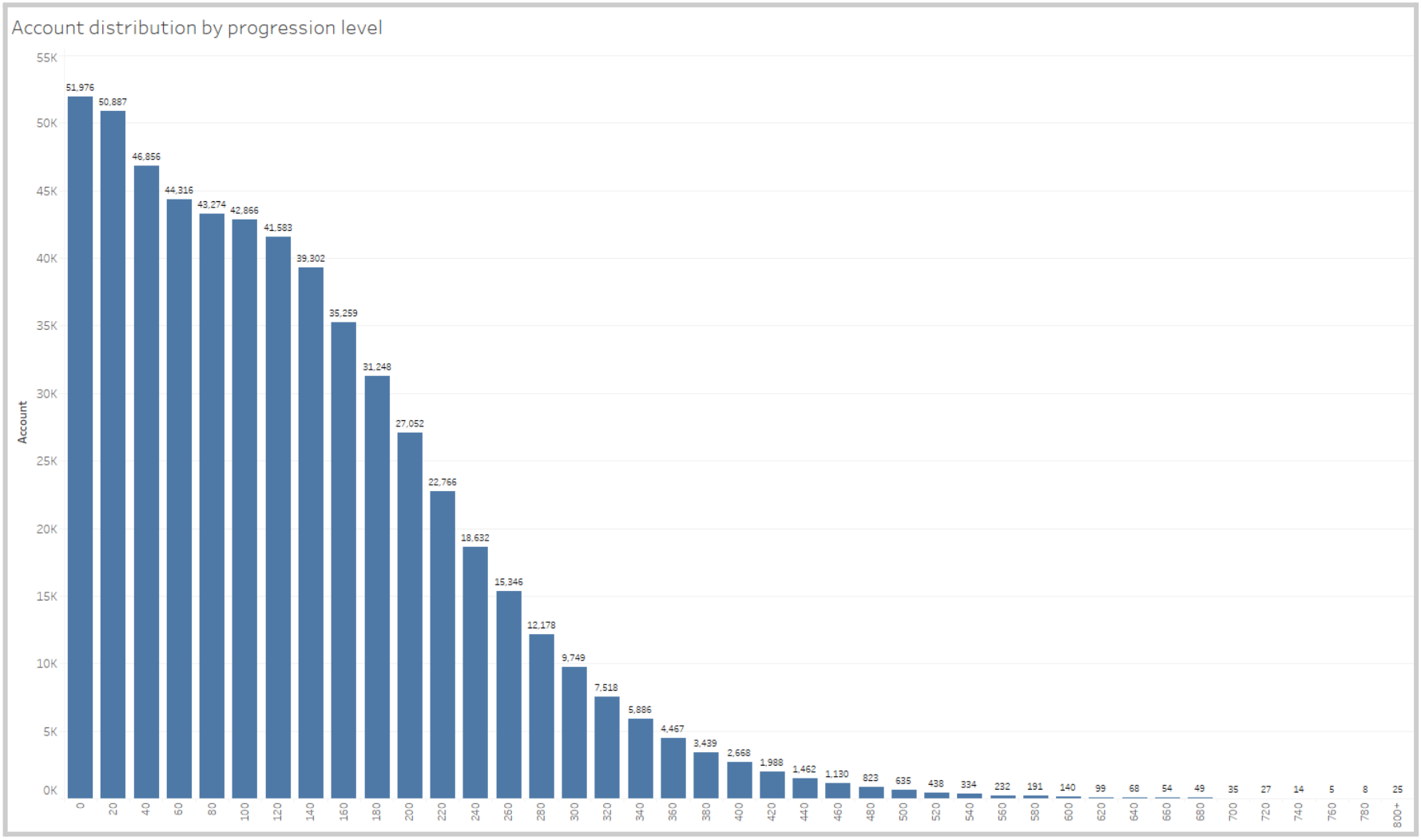
You can notice that some values are correlated. As there are 10 players in a game, the amount of Player entries equals 10 times the amount of Game entries. We should have the same reasoning with Player Stats. One entry of Player Stats is recorded for each player and for each round. The expected result is equal to 10 times the amount of rounds : 22 994 780. However, we got a lower value, with 5 missing Player Stats entries. After some research, I found that those 5 entries were missing from 5 different games last round. Those weird missing entries are present a lot in the data sent by VALORANT in-game API, due to game bugs, or flaws in the VALORANT data system. I fixed the most obvious one, but the rarest issues could have gone unnoticed like this one. Those gaps in the data should not be an issue and can be ignored.

To go deeper in the exploration, I made some graphs via Tableau. From now on, I will have to learn new skills to continue my analysis. I never used Tableau Software before. Making some simple graphs would help me to have a better understanding of this tool.

First, I wanted to explore accounts. For them, values I have interest on are :

- **Progression level** : The higher the level is, the more play time there is on this account.
- **Peak tier** : A high tier means the user of the account is skilled at the game. Peak tier is the maximum known ranking tier the player reached

- **Current tier** : Unused here, current tier of an account may be underestimated. We will prefer using Peak tier which is more reliable.

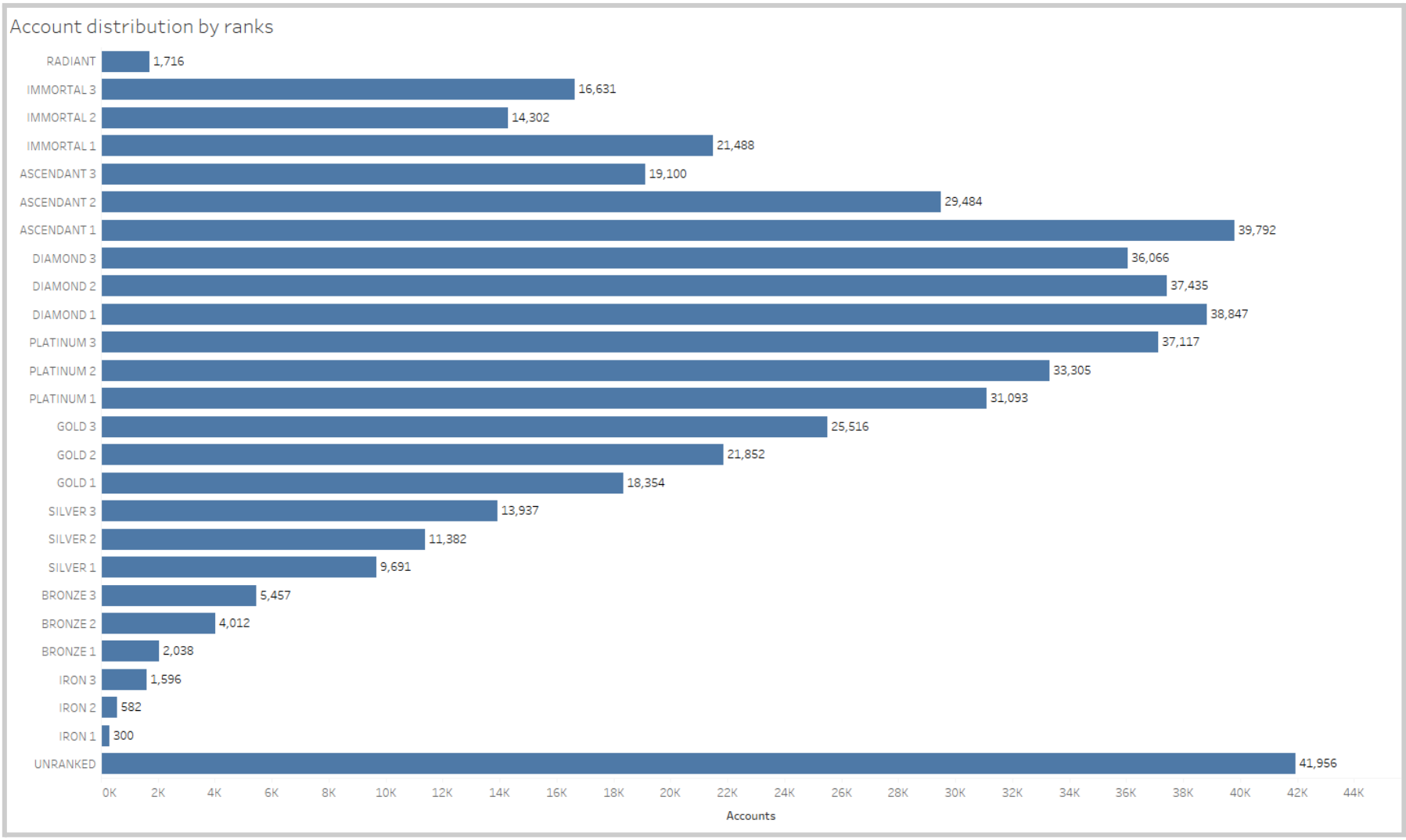


Distribution of accounts by their progression level, in 20-level slices

Account progression levels follow what seems to be a logistic distribution. Most of them have a level under 140. After that level, the number of accounts starts to decrease rapidly slice after slice. This result was expected, but I am still intrigued by the strange

transition in the 4 first slices that I can’t explain.

Progression level Statistic summary	
Mean	136.6
Std. derivation	102.9
Minimum	0
Q1	56
Median	121
Q3	197
Maximum	1 194



Distribution of accounts by their peak rank, rank are ordered from higher to lower

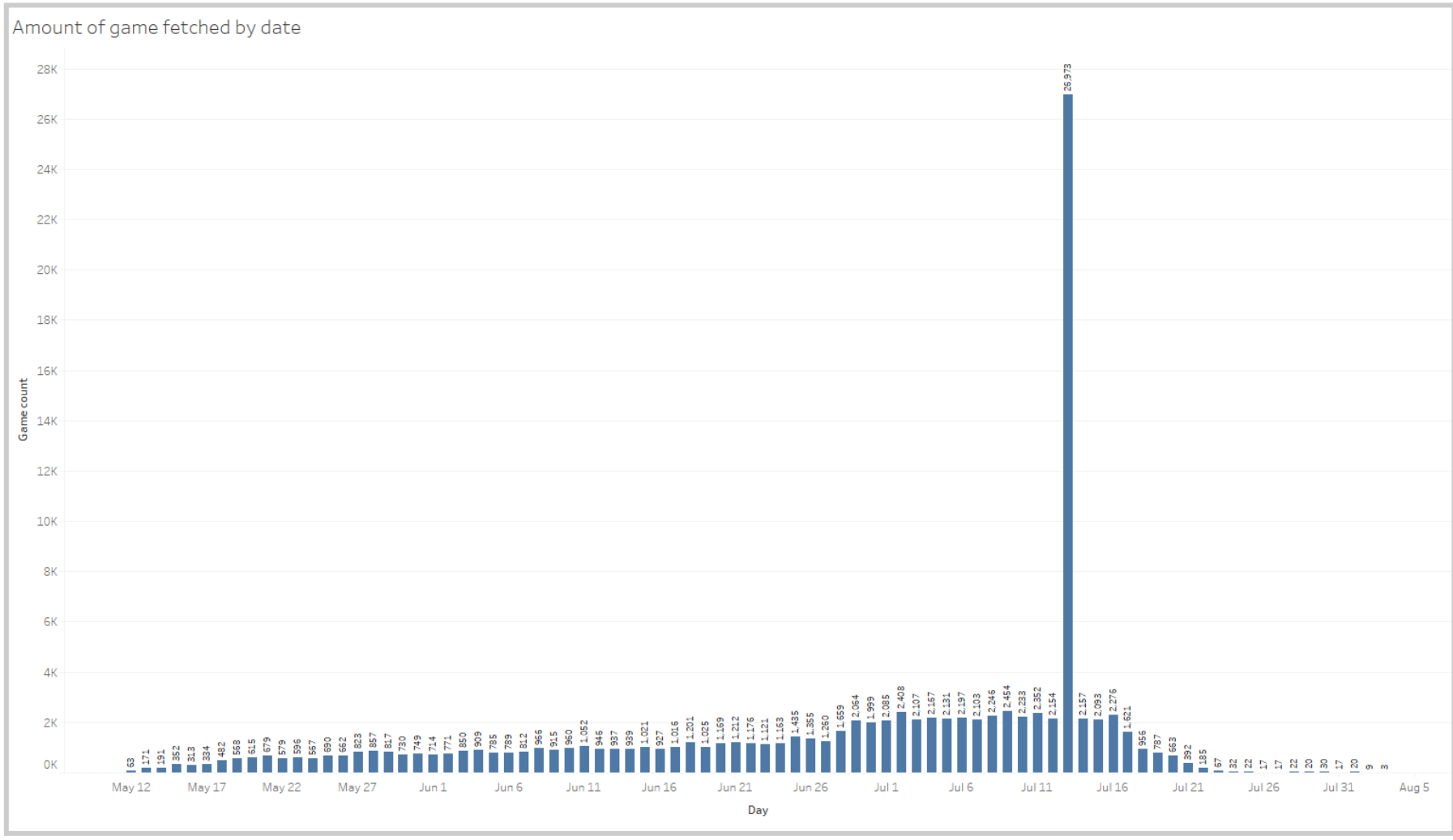
Outside of Unranked accounts, the rank distribution follows a normal law. However, the spread is messier around Immortal and Ascendant ranks. This is caused by the fetching of the leaderboards. To increase the amount of accounts in the database, I choose to get all previous leaderboards available. However, in 2022, Riot Games made a change to the rank system. Due to the large number of

Immortal accounts, they add an intermediate rank between Diamond and Immortal : Ascendant, shifting many Immortal account ranks. In my data, old Immortal 1 accounts are considered as Ascendant 1 and so on.

This exploration around account ranks points out one of the limitations of my data collection process. This is not a big deal as to analyze game balancing I will focus on the players rank at the moment of the game, which does not depend on the account peak rank.

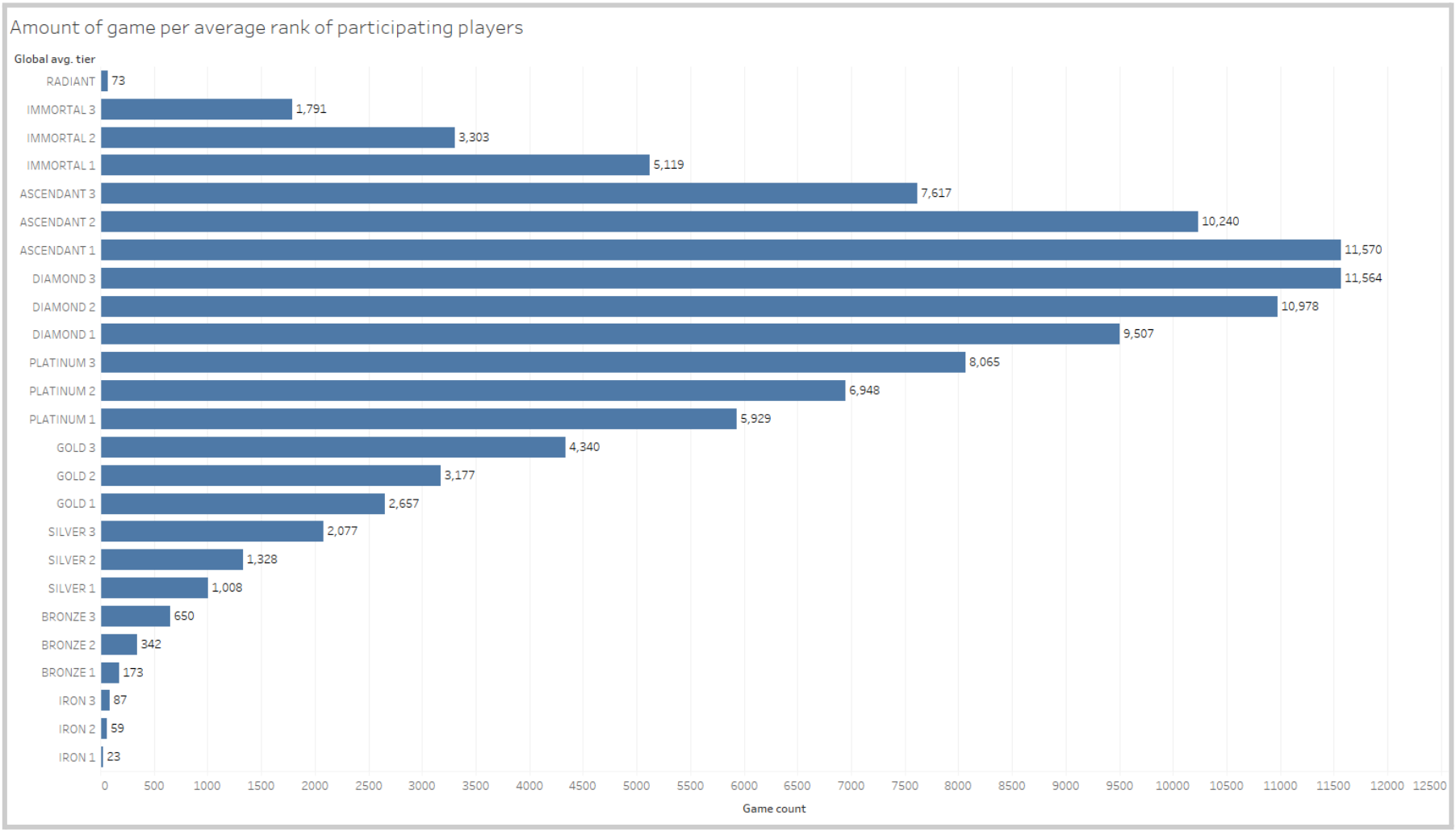
Next element I want to explore is games. There are a lot of things to see in it :

- **Date** : This will give us some context and the game balancing state at the moment of the game.
- **Average rank** : This will add a layer of context for the game. A balancing choice might not have the same impact on a low-rank game than on a high-rank one.
- **Map** : The map is a main segmentation factor to analyze games, each map has to be treated independently.
- **Final score** : Final score of a game can give us a way to evaluate the balance of a game.



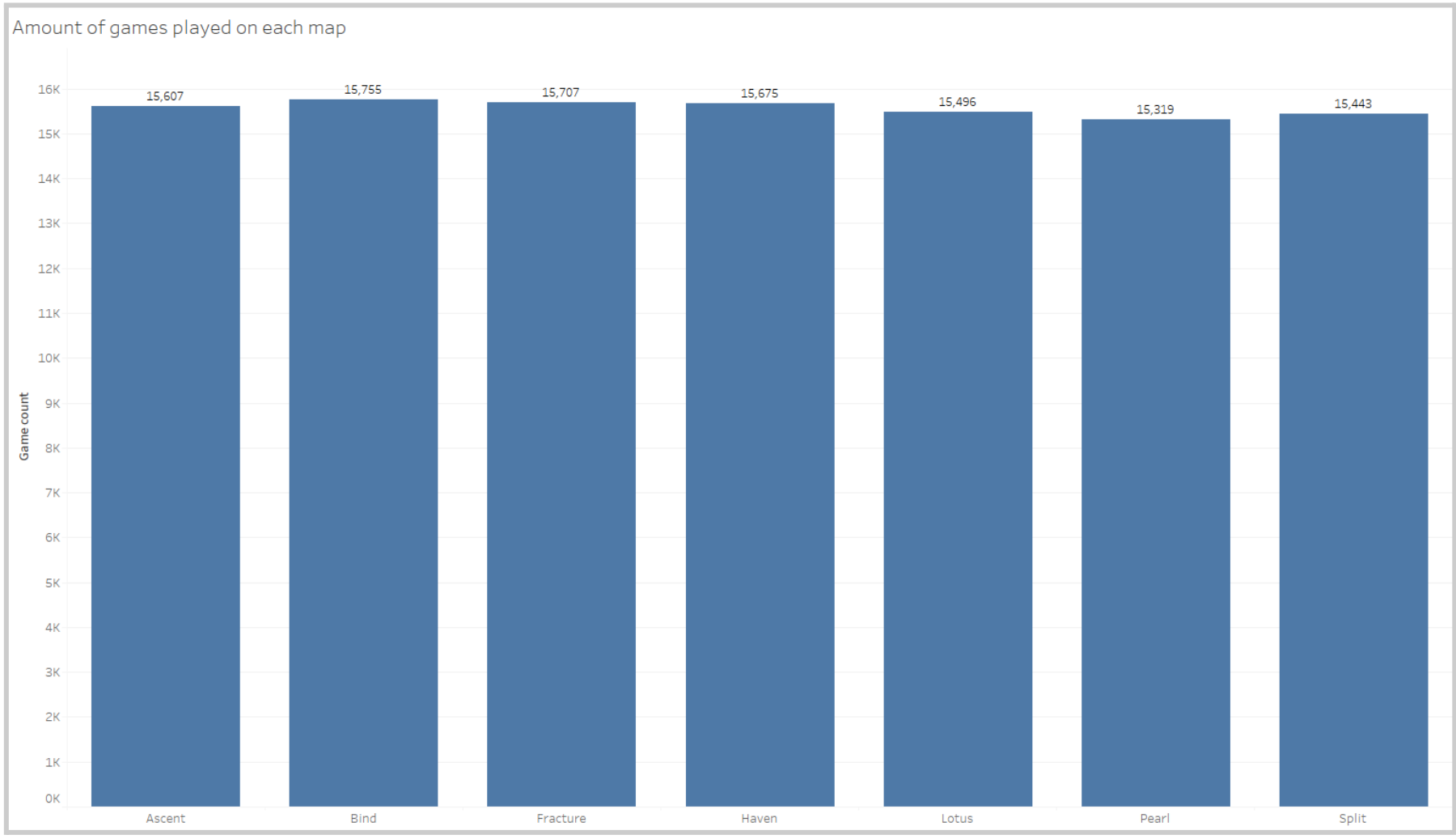
Distribution of games by the date they were played

Games in the database were played between May 12th 2023 and August 3rd 2023. A quarter of the games in the database were played on July 13th. This is due to a change I made on the collection algorithm on this date. Originally, after collecting as many game UUID as I could, I fetch game details starting with the most recent ones. I decided to change the game fetching process to a random selection of game UUID, but I kept the 20 000 games already gathered.



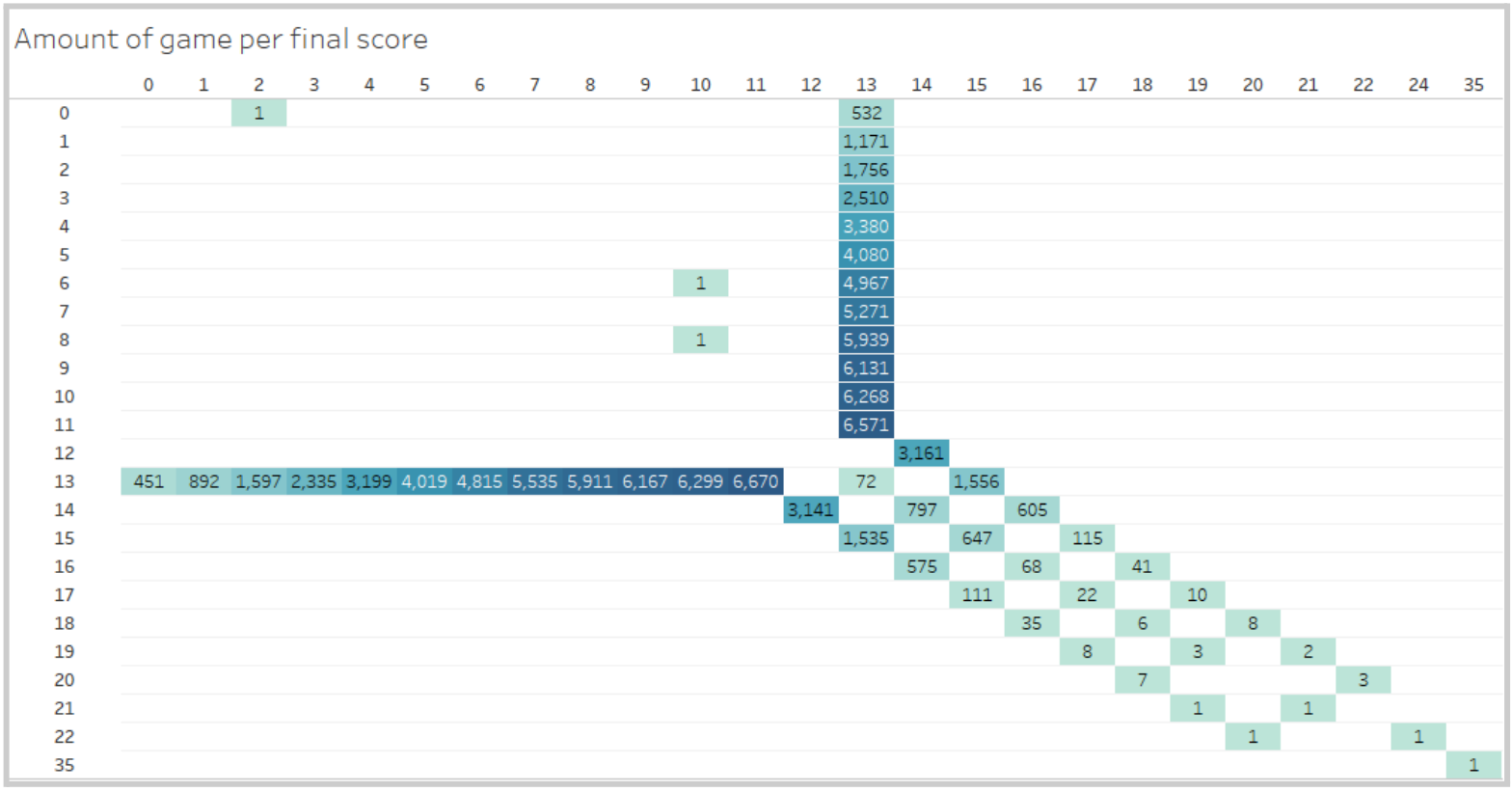
Distribution of games by the average rank of all participating players at the time of the game

The average game rank follows a normal distribution which is much cleaner than the account rank distribution. Games in base were played on every rank, but the peak is between Diamond 3 and Ascendant 1. However, on complete VALORANT data, I would have expected a peak around Gold ranks. Once again, this bias is mostly due to the collection flow. As I start with the high-rated players, I easily fetch their high-rated games. If I let the collection run, the peak would have slowly decreased toward Gold ranks.



Distribution of game by map played

For maps, I expected to have close values between each map of the 7 maps that make the map pool at this time. For each game played, the map is randomly selected. The result I got is convincing and looks like what I expected.



Distribution of games by their final score
Note: there is gaps is the axes (23 and 25-35 range are hidden)

The final score exploration gave me many insights about how games ended. I expected the main Y-shape of this table. However, it seems there are some games that don't fit this form.

First, the two main branches of the Y-shape are where most of the game ends. Those branches represent games that ended on a regular score. The winning team got 13, defeating the other team by at least 2 points before the 24th round. The closer we are to the cross, the more games there are, showing that in Competitive mode, games tend to be tight.

Then, the tail of the Y-shape are all the games that have gone over regular score and start overtime. In VALORANT, when the 24th round brings both teams at 12 to 12, an overtime is played. It consists of 2 successive rounds, one as Attackers, one as Defenders. To win, a team needs to win them both. If they draw again at 13-13. Two options are possible : continue and play another overtime or draw the game. Overtimes could technically go forever.

The tail shows both options. The middle diagonals are drawn games and the outside ones are games ended by a team winning both overtime rounds. The more overtimes are played, the less games there are.

On this table, 4 games stand out. 3 of them don't fit the Y-shape, and the last has a crazy final score of 35 to 35. I keep those 4 games for the next part about outliers.

Finally, to end my data exploration, I looked at Agent play. I roughly output a table showing basic statistics for each Agent, ordered by count of selection in games.

Agent	Picks	Pickrate	Avg. Score	Avg. Kill	Avg. Death	Avg. Assist	Max. Score	Max. Kill	Max. Death	Max. Assist
Jett	143 563	63.67%	4 973.38	17.29	16.08	3.49	43 713	175	37	16
Sage	109 305	48.48%	3 968.95	13.66	14.97	6.94	12 558	44	38	27
Reyna	105 235	46.67%	5 015.28	17.41	15.71	4.21	16 687	58	36	18
Omen	88 324	39.17%	4 262.99	14.77	15.30	7.46	15 726	51	36	29
Raze	77 942	34.57%	4 976.19	16.51	16.03	4.70	17 533	56	44	19
Brimstone	76 979	34.14%	4 262.83	14.39	14.99	8.59	14 694	53	35	33
Killjoy	61 193	27.14%	4 190.27	14.53	14.53	3.51	12 174	44	39	14
Skye	60 733	26.94%	4 135.26	14.30	15.25	7.44	12 457	44	35	57
Cypher	53 067	23.54%	4 164.63	14.69	14.71	4.54	13 499	46	31	19
Chamber	47 174	20.92%	4 571.32	16.26	15.14	2.71	14 683	48	34	15
Breach	45 038	19.98%	4 119.11	14.20	15.51	7.08	12 709	45	34	26
Sova	38 154	16.92%	4 269.12	14.54	14.77	5.74	13 724	48	35	25
Phoenix	33 585	14.90%	4 877.31	16.76	15.54	4.49	18 344	63	35	19

Deadlock	28 492	12.64%	4 192.26	14.84	15.20	3.69	12 337	44	31	18
Gekko	27 421	12.16%	3 985.88	13.97	15.20	4.62	12 171	41	32	18
Fade	25 995	11.53%	4 130.21	14.29	15.20	6.31	11 540	43	34	24
KAY/O	24 891	11.04%	4 206.31	14.14	15.64	8.80	12 077	43	34	27
Viper	23 812	10.56%	4 271.86	14.66	14.84	5.74	12 470	46	29	21
Astra	21 029	9.33%	4 282.05	15.02	14.84	6.71	13 065	47	29	23
Neon	15 243	6.76%	4 596.67	15.72	16.42	4.02	15 130	57	33	15
Yoru	12 722	5.64%	4 501.44	15.63	15.76	4.26	12 691	46	30	16
Harbor	7 432	3.30%	3 993.31	13.86	15.35	6.03	10 727	38	29	21

Like in all competitive games with a character selection, we can see that players have some preferred and put aside characters. With a deeper analysis, we could be able to understand why. Player feedback is also a great tool to find out game balancing issues. It could bring many starting points to look at.

One curious point about this table is the maximum kills that a Jett player has managed to make. 175 is a huge amount. So huge, that it is theoretically impossible to get. We will see this case in the next part. Many outliers here have a lot in common.

Outliers

Once I had finished my exploration, I had some points to check. I started with the one that most intrigued me, the 175-kill Jett player. From the player entry, I go back up to the game UUID and then I query the 10 participating players of this game. The game was played on Madrid server, version 06.11.00, on June 22nd 2023. The final score was a 35-35 draw. And yes, this is the 35-35 game I found during the exploration of game data. Now that I linked together two outliers, let's see the player results :

Party column represents the 5-character left substring of the UUID to avoid display long unreadable values

Account name	Party	Team	Agent	Kills	Deaths	Assists

tyh #7855	F5448	Red	Jett	175	35	0
bnh #9434	F5448	Red	Brimstone	0	35	0
nbh #3401	F5448	Red	Phoenix	0	35	0
hyu #8744	F5448	Red	Sage	0	35	0
mji #6432	F5448	Red	Sova	0	35	0
ghy #4303	6C531	Blue	Jett	175	35	0
bnh #5947	6C531	Blue	Brimstone	0	35	0
loi #9452	6C531	Blue	Phoenix	0	35	0
vbg #6827	6C531	Blue	Sage	0	35	0
dsf #1867	6C531	Blue	Sova	0	35	0

Now, we can understand how making 175 kills is technically possible. As far as I know, this match looks like a fixed game. Party IDs show that each team was grouped together. Each team consists of the same Agents that finished the game with the same score, except from Jett who made all the kills. Finally, and a good one at that, the account name looks pretty strange, not the kind of username you usually see on VALORANT. To conclude with this outlier, I am convinced this game was made with a single man behind it that made some bots to farm ranked games. His objective was probably to sell those accounts.

Unfortunately, those accounts have no more games in my database, so I am unable to go further. However, this outlier could be a good starting point for a new analysis to show the impact of fixed games, selling accounts, smurfs, and bot accounts on the VALORANT ecosystem which is a very interesting point that players often complain about.

The next outlier I would like to explore deeply are the 3 games that ended before a team reached 13 points. Those unconventional scores are 10-6, 10-8 and 2-0. I would like to know what has caused this behavior in the data. My two hypothesis for this one are :

- The server handling the game has encountered a fatal error that has led to a game crash.
- The 5 players of a team have been disconnected and the game has a system to conclude the game if this happens.

I think the first one is wrong. As a VALORANT player, I know that when a server is closed before a game is ended, the match results of the closed game are never displayed in the VALORANT game client or anywhere else. I am almost certain that game data is recorded at

the end of the game. Meaning a server crash should not be able to record the game data as the game has not finished yet. So I will test the second hypothesis.

I do not know exactly the cause of a player disconnection. But let's imagine a scenario. 5 friends want to play VALORANT, but to spice it up, they want to play it, together in the same room, maybe in a cybercafe, in the friend's garage or at a LAN event for instance. This leads the 5 friends to use the same Internet connection. If it drops, everyone would be disconnected at the same time. This scenario is probable. Now, what are the traces that could prove it ?
In this scenario, I would expect the 5 players to play in the same party, so looking at their party UUID could be a good start. To do so, I query my database to show me the participating parties, grouped by game and team.

Game and Party columns represent the 5-character left substring of the UUID to avoid display long unreadable values

Game	Party	Team
3AA89	BDB81	Red
3AA89	7FE1D	Blue
C48D4	742CC	Red
C48D4	19821	Blue
D878B	DEC95	Red
D878B	81D66	Blue

As I expected, for the 3 games, each team consists of a single party of 5 players. That starts to confirm my hypothesis. Next, I looked at round, kill events and player stats data for those games. I found some clues that confirm the 5-player disconnection. The 2-0 game showed evidence that the involved team never managed to connect to the game. VALORANT seems to prevent a game closing itself before the start of the third round. The last rounds of the two other games (10-6 and 10-8) show traces of player inactivity (no economy spent, spawn kill location, no enemy killed, no damage dealt).

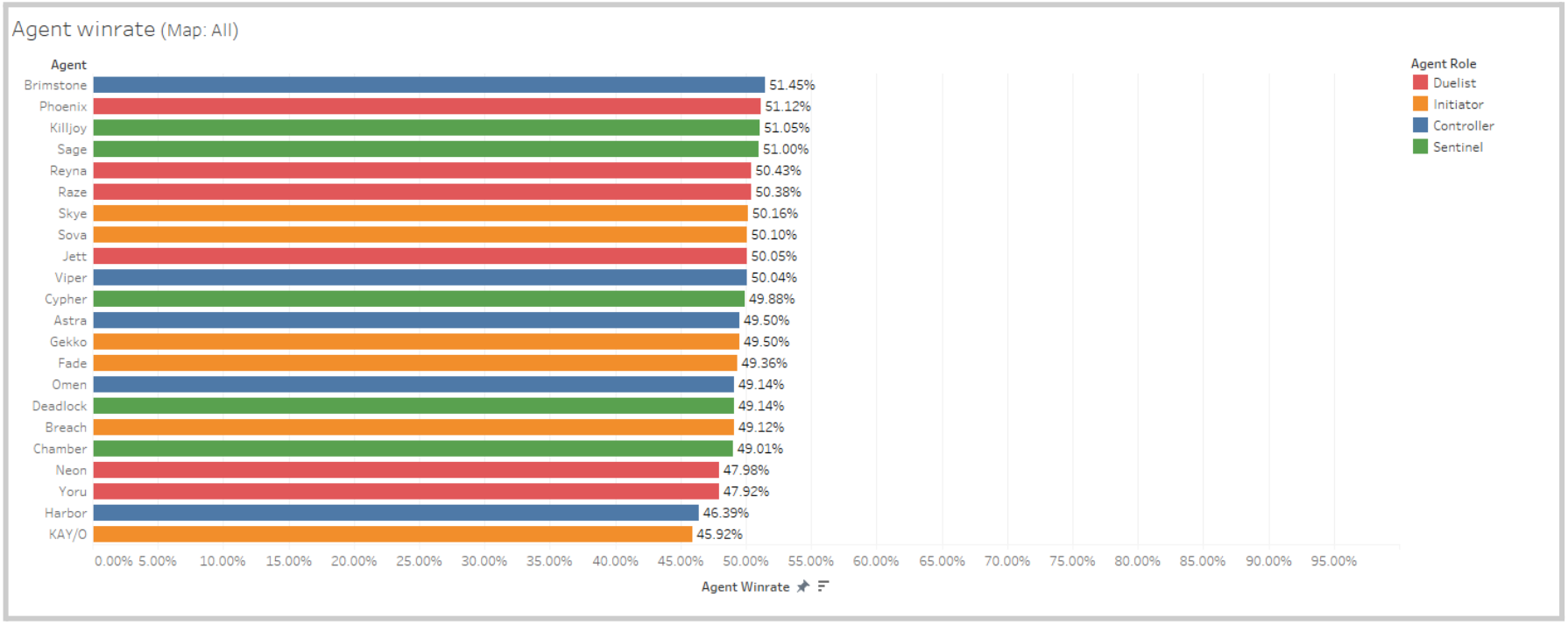
I am now confident in my hypothesis of a group of players using the same Internet connection that unfortunately got away at the wrong moment.

Game balancing

Agents

Time has come to analyze VALORANT game balancing, starting with Agents balance. Agent balancing has always been a huge subject of player conversations. Choices made here have a strong impact on the whole ecosystem. Driven by the meta (short for “most effective tactic available”), standard players as well as professional players keep an eye on every balance choice.

First thing I want to analyze is the win rate of every Agent.

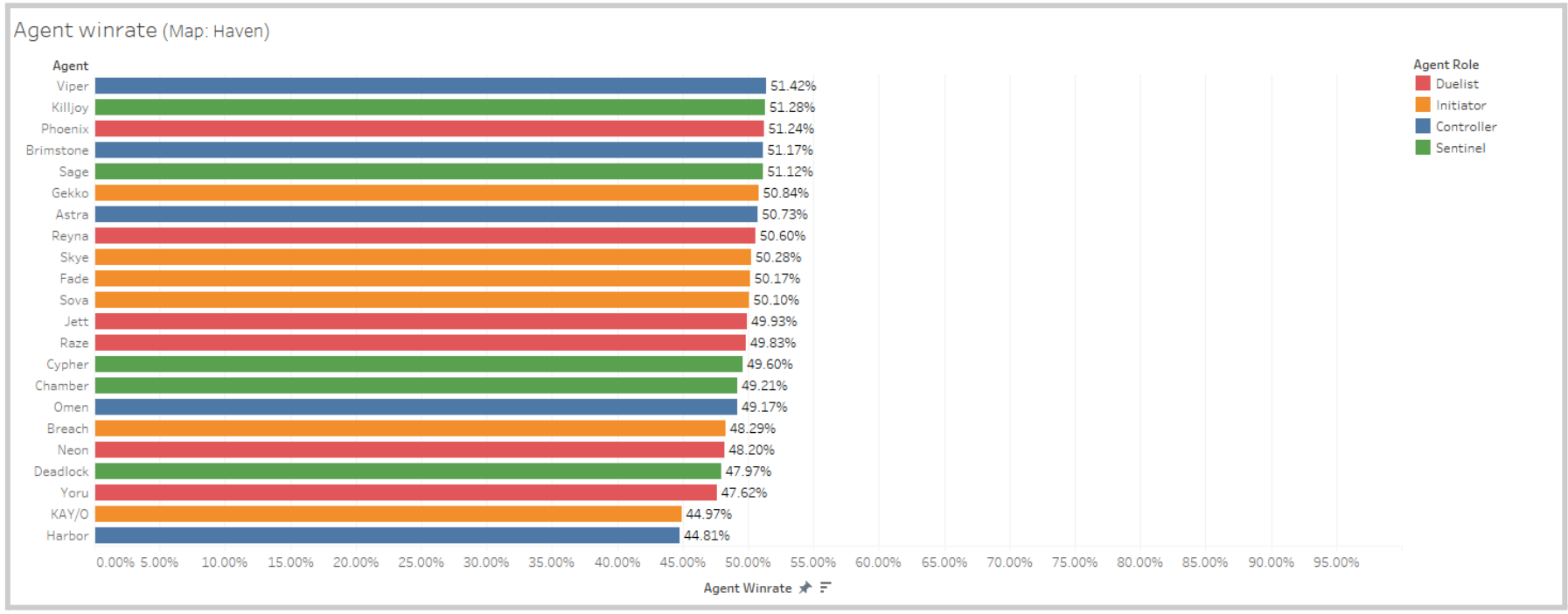


Agent win rate for all map

Those win rates are calculated on all maps. Draws are not counted at all. Four agents stand out from the others, Brimstone, Phoenix, Killjoy and Sage are the Agents that win the most games over all maps. However, four agents stay at the bottom, Neon, Yoru, Harbor and KAY/O. Those last four are more concerning, as their win rates are very low compared to other Agents. Thanks to the exploratory

analysis, we also know those four Agents are among the least played agents.
The Initiator role, in orange, seems to be the role with the lower win rate over all Agents in this category.

Thanks to Tableau, it is easy to use filters to see Agent win rate per map. We can have various results depending on the map selected.



Agent win rate on Haven

On Haven, win rates change a lot compared to the overall win rates. Viper has found her way from a middle win rate to a top pick for this map. Same thing with Gekko that seems to find some good value on this map. However, the bottom-four agents stay at the bottom of this list.

Weapons

Maps

Conclusion