Game Design Math

Part 1: business & economics

Aalto University, Game Analysis course

Perttu Hämäläinen 2023

What this is not about

- The usual game and graphics programming math (e.g., linear algebra, transform matrices, quaternions, ballistics, physics simulation...)

Today's goal: Get to know Colab/Jupyter notebooks for game simulations and analytics

- Jupyter notebooks are an easy to use browser-based Python programming environment. Colab is Google's Jupyter service with some UI improvements.

- Python and the Numpy & Scipy packages are perhaps the most popular data science and analytics tools

- No need to install anything, the notebooks can be run on Google's Colab cloud service

- Alternative: Excel / Google sheets. However, many tasks are easier in Python, and Python notebooks are increasingly replacing spreadsheets

- Materials here: https://github.com/PerttuHamalainen/GameAnalysis

Dynamics, again

Motivation

- Competition is fierce. 1000+ new apps every day.

- Game developers must think statistically and optimize their chances to monetize and stay alive

- Think of chains of random events, e.g., the acquisition and monetization funnel (the *P* denotes probability):

$$P(\text{revenue}) \qquad = P(\text{ad eyeballs AND install AND engage AND pay})$$

Motivation

- Competition is fierce. 1000+ new apps every day.

- Game developers must think statistically and optimize their chances to monetize and stay alive

- Think of chains of random events, e.g., the acquisition and monetization funnel:

$$P(\text{revenue}) = P(\text{ad eyeballs AND install AND engage AND pay})$$

$$= P(\text{ad eyeballs})\, P(\text{install})\, P(\text{engage})\, P(\text{pay})$$

(Simplification, see slide notes)

Motivation

- Competition is fierce. 1000+ new apps every day.
- Game developers must think statistically and optimize their chances to monetize and stay alive
- Think of chains of random events, e.g., the acquisition and monetization funnel:

$$P(\text{revenue}) = P(\text{eyeballs AND install AND engage AND pay})$$

$$\approx P(\text{eyeballs})\ P(\text{install})\ P(\text{engage})\ P(\text{pay})$$

At every step of the funnel, you lose a portion of players. For example, optimizing App Store screenshots and icon affects p(install). If you achieve a 10% increase in retention at every step, above, total number of paying customers increases by 46%.

Motivation

- Competition is fierce. 1000+ new apps every day.

- Game developers must think statistically and optimize their chances to monetize and stay alive

- Think of chains of random events, e.g., the acquisition and monetization funnel:

$$P(\text{revenue}) = P(\text{eyeballs}) * P(\text{install}) * P(\text{engage}) * P(\text{pay})$$

(Simplified, see slide notes)

- Total probability is low if even one factor is low. At every step, you lose a portion of players. For example, optimizing App Store screenshots and icon affects p(install). If you achieve a 10% increase in retention at every step, above, total number of paying customers increases by 46%.

Key concept: The power of recursion

- Due to recursive relationships (feedback connections), small changes in attributes (e.g. retention) can have huge impacts (e.g., on long-term revenue)

- Recursion leads to exponential behavior (decay or growth, forms of emergent dynamics)

Exponential decay

F2P Retention

Tracking a user's in game activity every day since first install.

Most users will only play for a day or two

Some users will play every day for a very long time
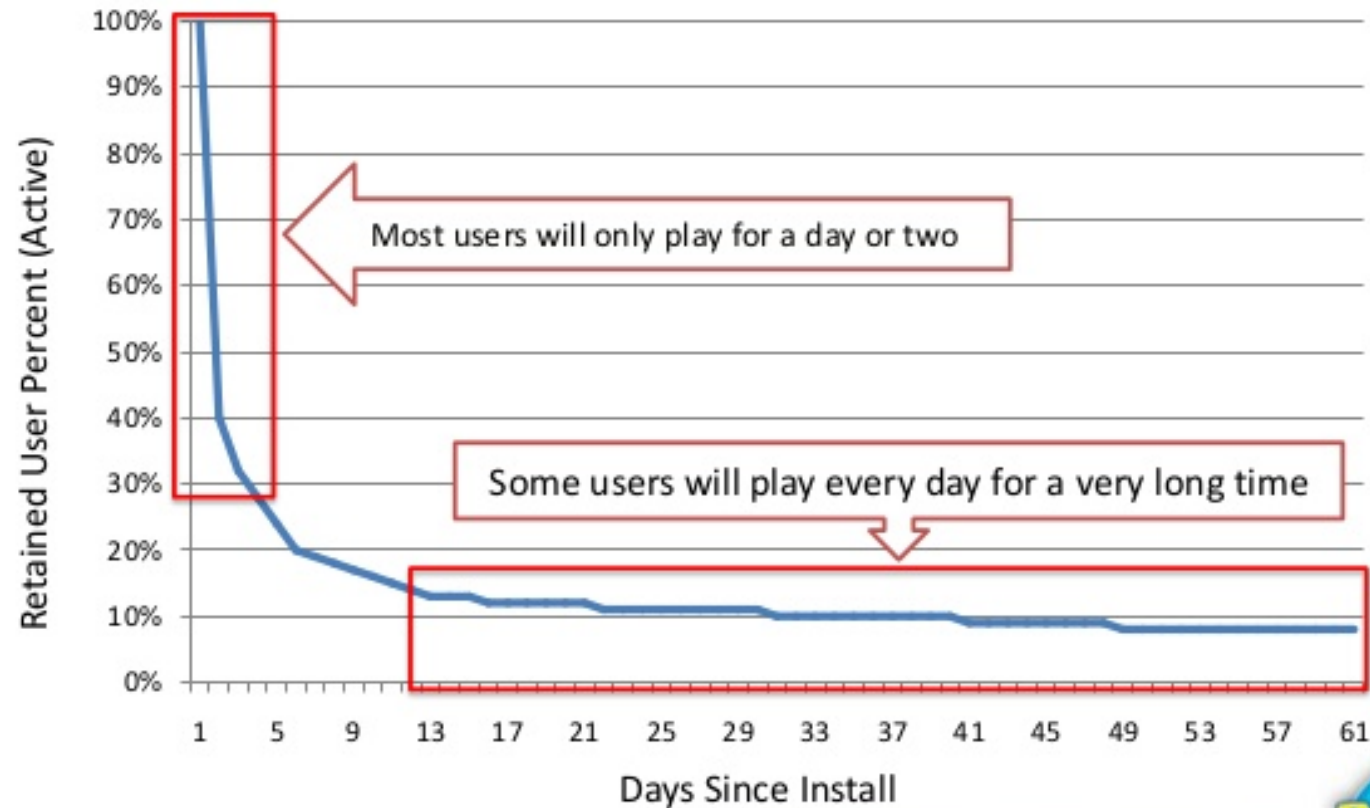
Exponential decay: $u(t+1) = u(t) - p_{churn}\, u(t)$
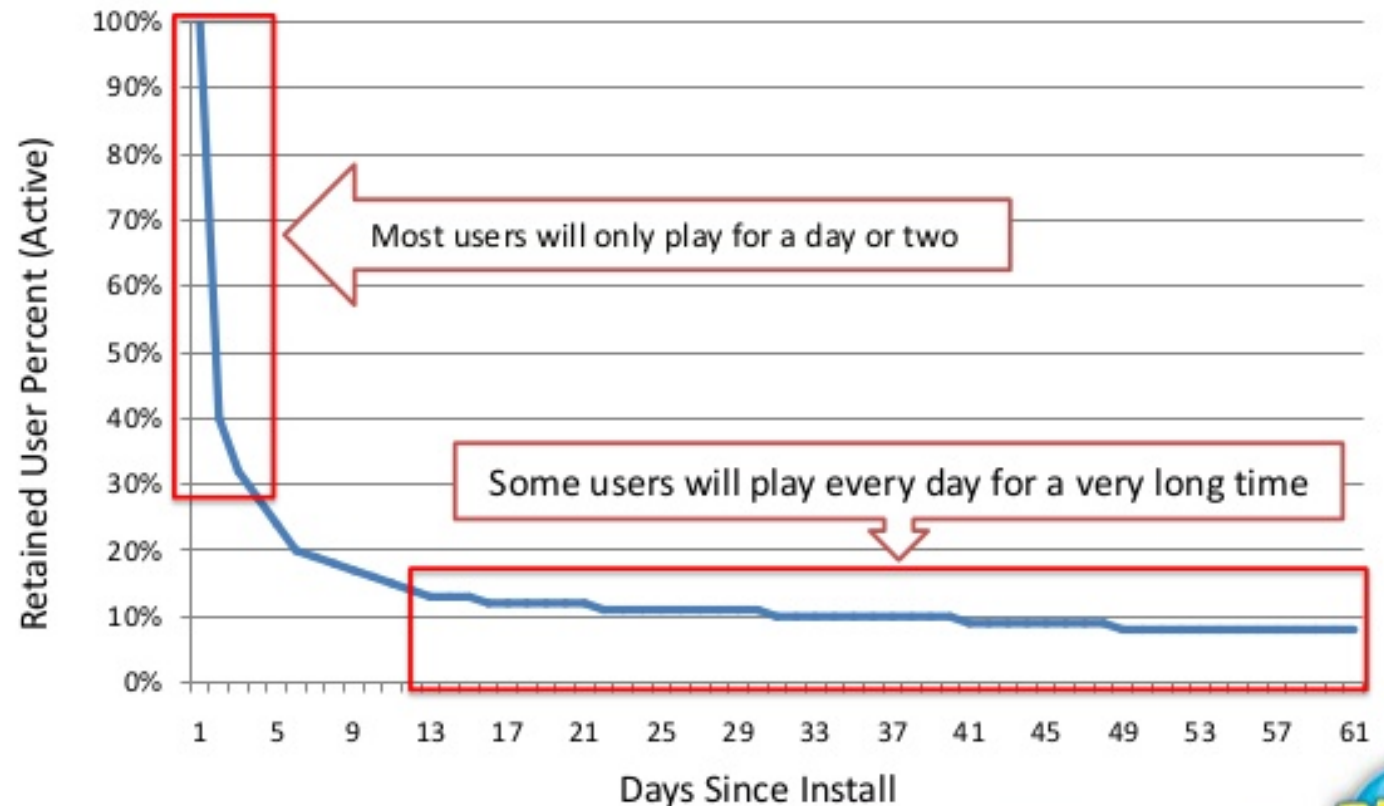
Notation:

$u(t)$ — Number of players at time t

p_{churn} — Probability of player quitting (percentage of churned players divided by 100)

F2P Retention

Tracking a user's in game activity every day since first install.

Most users will only play for a day or two

Some users will play every day for a very long time

Retained User Percent (Active)
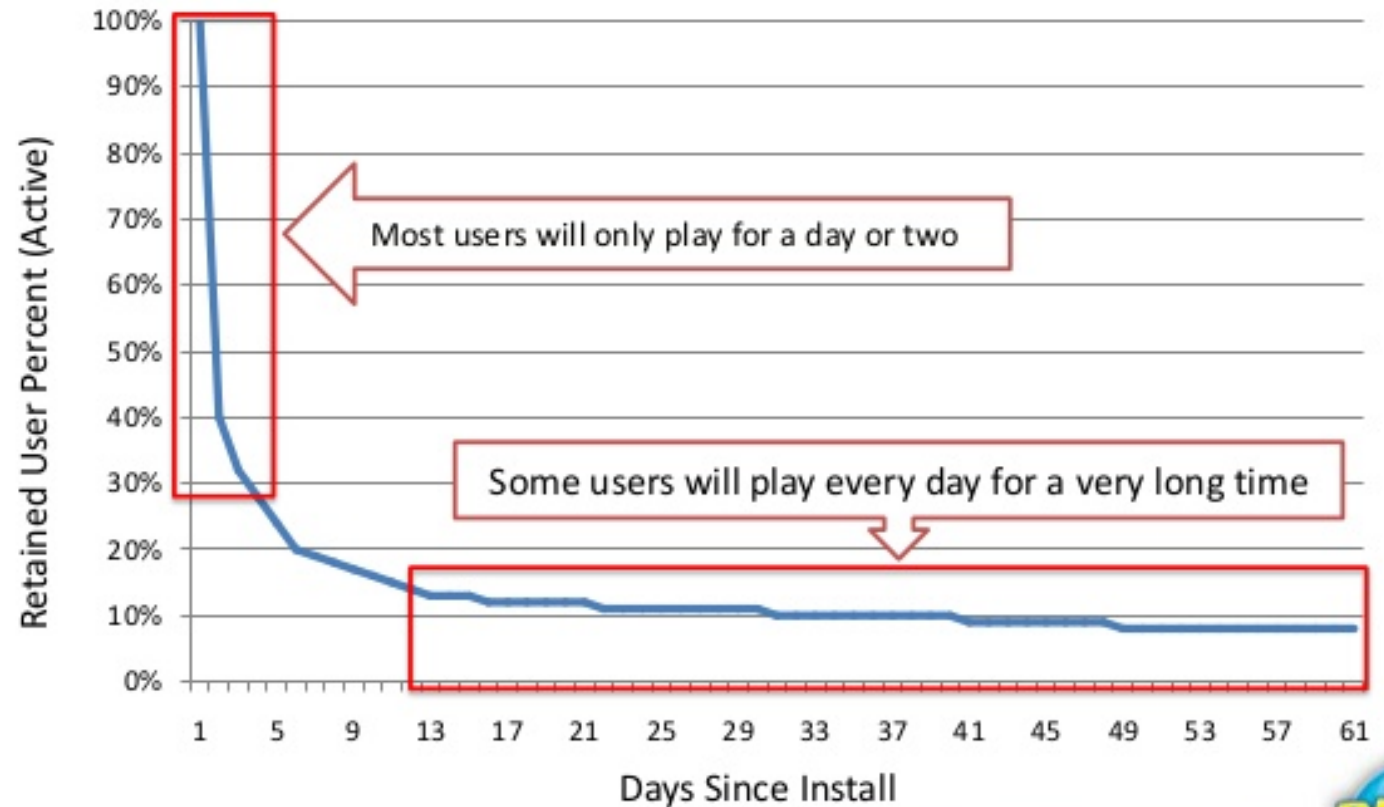
Days Since Install

Exponential decay: $u(t+1)=u(t) - p_{churn} u(t)$

Alternatively:

$u(t+1)=p_{retention} u(t)$,

$p_{retention}=1-p_{churn}$

F2P Retention

Tracking a user's in game activity every day since first install.

Most users will only play for a day or two

Some users will play every day for a very long time

Exponential decay: $u(t+1)=u(t) - p_{churn} u(t)$

Alternatively:

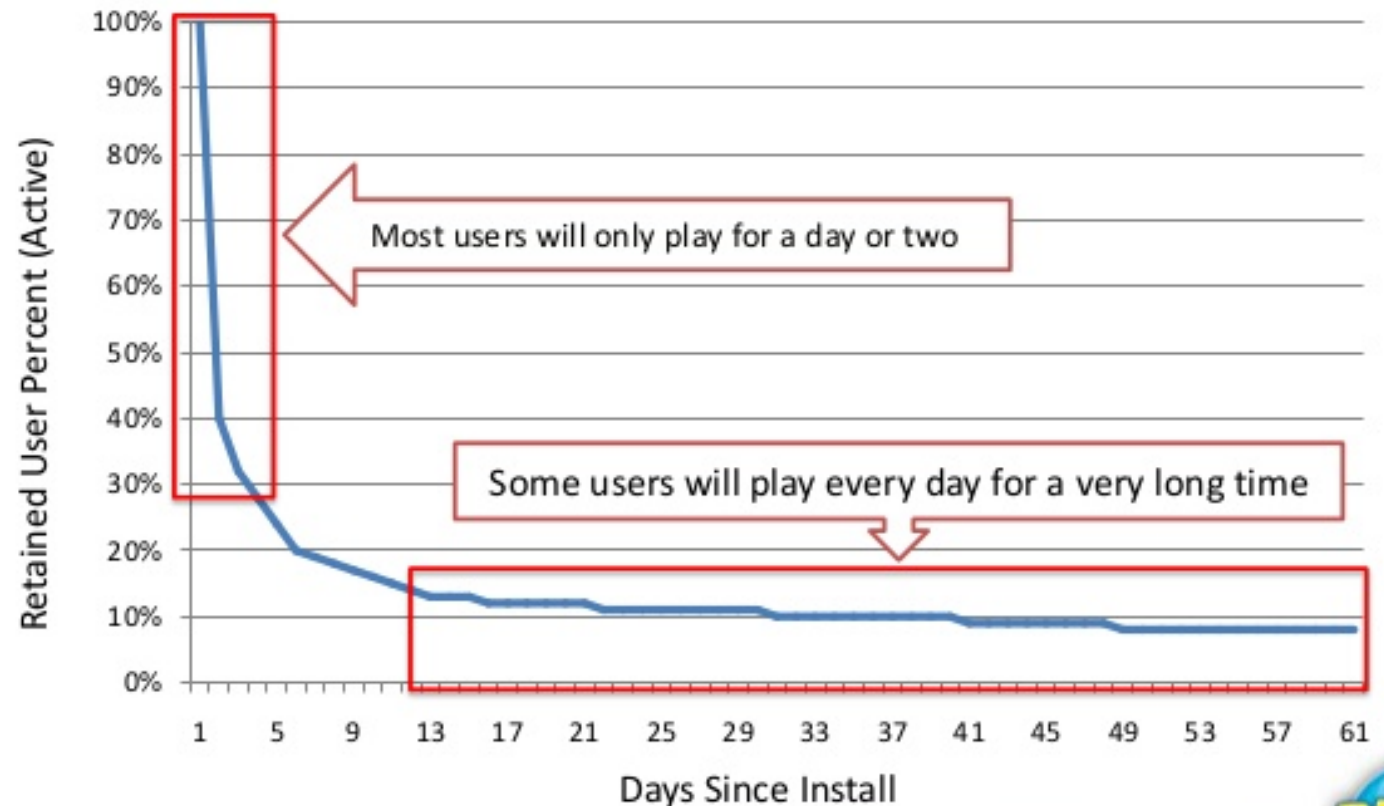$u(t+1)=p_{retention} u(t)$,

$p_{retention}=1-p_{churn}$

Nonlinear because the number of players depends on itself recursively and *multiplicatively*, instead of, say, $u(t+1)=u(t)-1$

F2P Retention

Tracking a user's in game activity every day since first install.

Most users will only play for a day or two

Some users will play every day for a very long time

Retained User Percent (Active)

Days Since Install

Exponential decay model's accuracy?

- In real games, churn is often higher for the first day

- Bad tutorial/onboarding can easily make you lose most of your players

- On the other hand, the more time one has already invested in a game, the more committed one might be to continue, provided that the game provides meaningful progress

- If you use this kind of model for making predictions based on your initial player data, measure the churn rate over multiple days, maybe without taking the first few days into account.

Retention outside free-to-play

F2P player churn is dramatic: With zero initial investment, players can easily switch to another game.
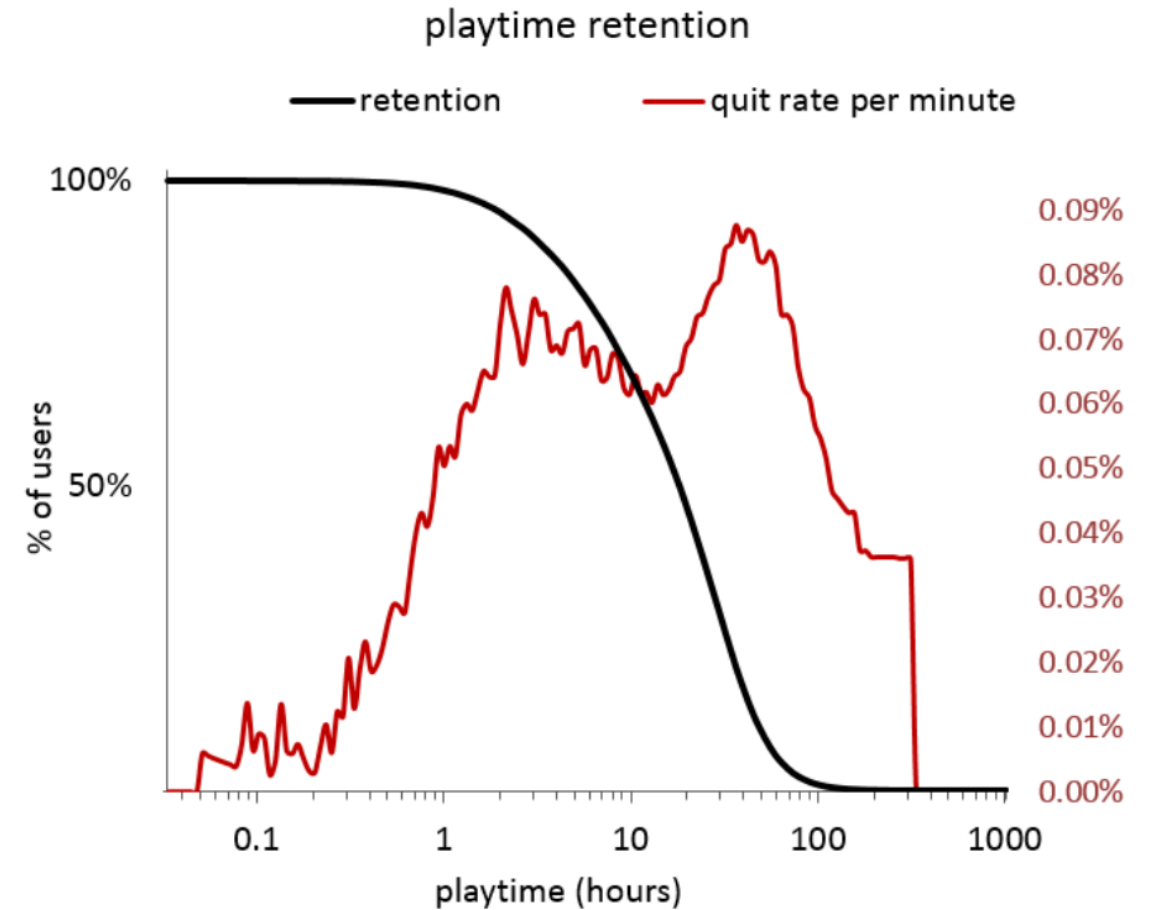
Data from paid games can be quite different.

Fig. 4. Far Cry 4 retention in black and quit rate in red.

https://hal.archives-ouvertes.fr/hal-02436692/document

Notable churn/retention research

- Viljanen, Markus, et al. "Modelling user retention in mobile games." *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016.

- Allart, Thibault, et al. "Design influence on player retention: A method based on time varying survival analysis." *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016.

- Periáñez, África, et al. "Churn prediction in mobile social games: Towards a complete assessment using survival ensembles." *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016.

- Demediuk, Simon, et al. "Player retention in league of legends: a study using survival analysis." *Proceedings of the Australasian Computer Science Week Multiconference*. 2018.

- Roohi, Shaghayegh, et al. "Predicting Game Difficulty and Churn Without Players." *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. 2020.
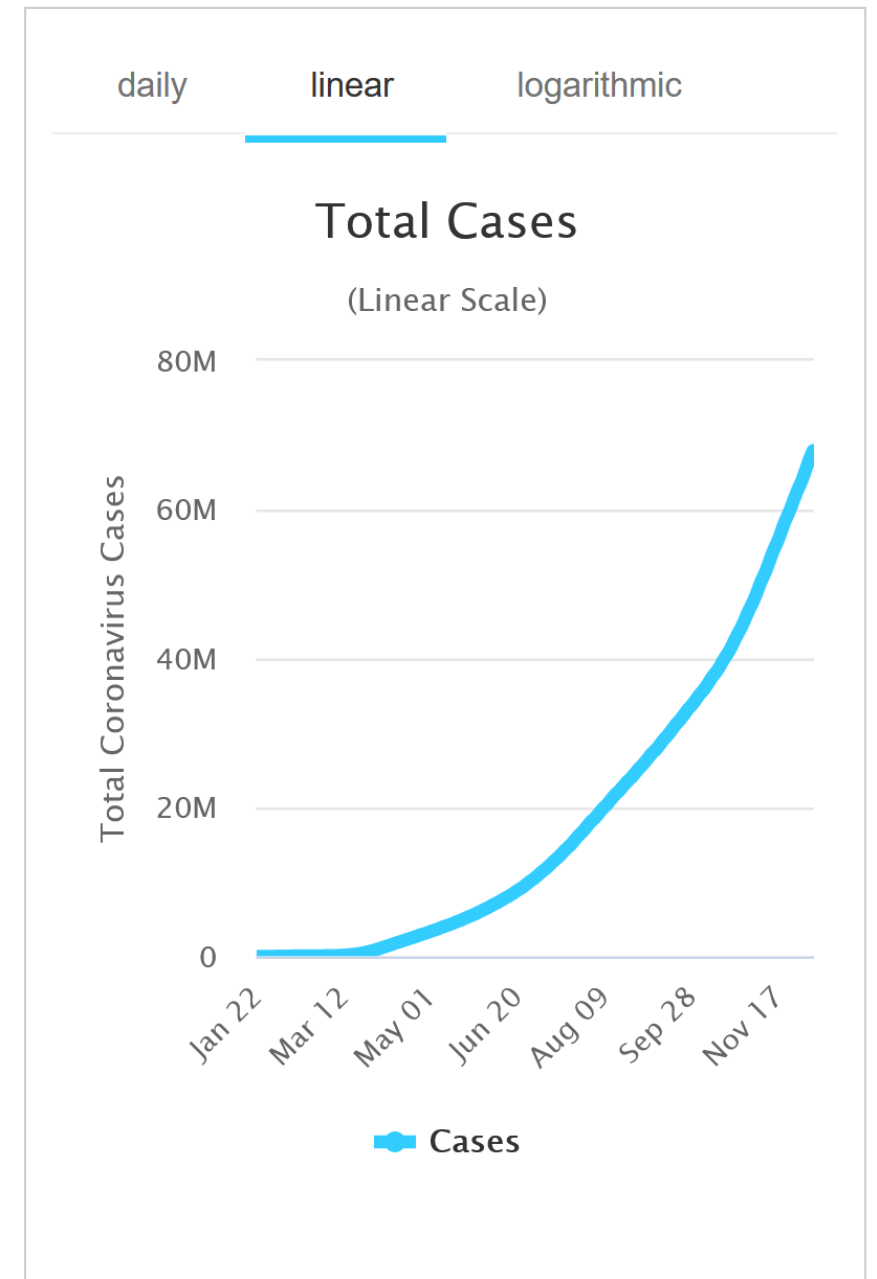
Exponential (viral) growth

- Every coronavirus case infects a number of new cases

- Organic growth of games: Portion of players share on social media or show a game to their friends at school etc.

Equation:

$$u(t+1) = u(t) + vu(t),$$

where v is "virality", i.e., how many new players each player "infects" per day

Source:
https://www.worldometers.info/coronavirus/?

Another source of exponential growth in games as a service

- Achieving high virality is hard.
- Combining the following can also produce exponential growth:
  1. Achive good retention and monetization: At least some players keep playing for a long time and provide a steady revenue stream (e.g., season passes)
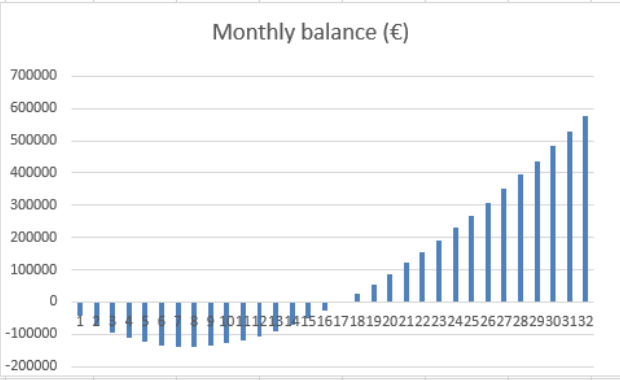 2. Invest a portion of the revenue in acquiring new players

Why does the math matter?

- You can measure parameters like p_{churn} from your player data
- Using the parameters, you can make predictions about your playerbase and company cashflow
- Soft launch: A business practice where you launch the game on a limited market to measure, make predictions, and iterate so that your parameters improve.
- Successful soft launch: Your data and predictions indicate good ROI (Return On Investment) for global launch (marketing, servers, support staff...)
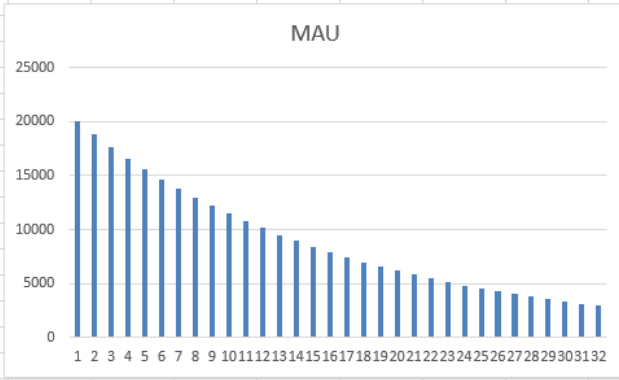
Exercise: Cashflow prediction

|	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Revenue forecast of hypothetical game																		
2																			
3	User acquisition cost		1																
4	Daily churn percentage		10			Day 1 retention		90											
5						Day 5 retention		59,049											
6						Day 30 retention		4,239116											
7	Average daily IAP purchases/user		0,01																
8	Initial marketing budget		10000			Note! Does not take into account the effect of getting to top 10													
9	Launch viral installs		10000																
10	Viral shares per user		0,1																
11	Viral share conversion rate		0,1																
12																			
13	Employees	5																	
14	Salary per employee	3000																	
15	Auxiliary cost	1,55																	
16	Overhead	1,7																	
17	Burn per employee	7905																	
18	Total burn	39525																	
19	Percentage of revenue used for UA	10																	
20		Q1/1					Q2/1					Q3/1				Q4/1				Q1/2
21	MAU	20000	18800	17672	16611,68	15614,98	14678,08	13797,4	12969,55	12191,38	11459,9	10772,3	10125,96	9518,406	8947,302	8410,464	7905,836	7431,48
22	Revenue	6000	5640	5301,6	4983,504	4684,494	4403,424	4139,219	3890,866	3657,414	3437,969	3231,691	3037,789	2855,522	2684,191	2523,139	2371,751	2229,44
23	Cumulative revenue	6000	11640	16941,6	21925,1	26609,6	31013,02	35152,24	39043,11	42700,52	46138,49	49370,18	52407,97	55263,49	57947,68	60470,82	62842,57	65072,0
24	Balance	-43525	-71410	-93993,4	-111593	-124509	-133021	-137393	-137875	-134700	-128086	-118241	-105358	-89619,7	-71197	-50251,2	-26933,6	-1386,5

Monthly balance (€)

MAU

Tweak the values in the cells highlighted in yellow. Try to complete the exercises at the bottom of the sheet.

Spreadsheets vs. Python notebooks

- Spreadsheet formulas be very hard to understand for someone who didn't create it. Python code uses human-readable variable names.

- Simple things might be easier in Excel, but Python scales better to complex simulations

fx `=$I$6*D22/100+D24/$D$3+$D$10*$D$11*D22`

	A	B	C	D	E
	MAU		20000		
	Revenue		30000		
	UA spend		21000		
	UA new users		21000		
	Cumulative revenue		30000		
	Cumulative burn		46575		
	Balance		-16575		
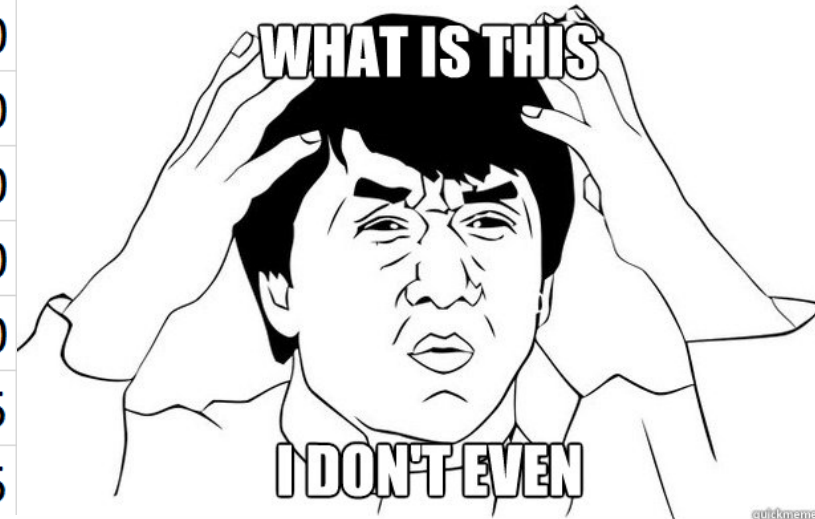
WHAT IS THIS

I DON'T EVEN

Same in Python

Basic recursion

First, let's write a very simple simulation of how each player might recruit new players virally with some probability, e.g., by sharing on social media

```python
players=1000
viralityProbability=0.1
for i in range(20):
    players=players + viralityProbability*players
    print(players)
```

```
1100.0
1210.0
1331.0
1464.1
1610.51
```

https://colab.research.google.com/github/PerttuHamalainen/GameAnalysis/blob/master/RetentionAndVirality.ipynb

This is the first Python simulation in the RetentionAndVirality notebook. Let's investigate it to learn Python syntax, assuming that you're only familiar with Unity C#...

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

\# means that the rest of the line is comments that explain the code. The computer running the code ignores these.

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

This introduces a variable called "players" and sets its value to 1000.
Python does not need type specifiers like C#'s "int players=1000"

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

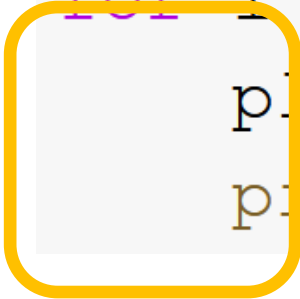In Python, you don't need to add the ";" at the end of a line.

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

This declares a loop: All code inside the loop is run for 20 times.
In C#, this would be "for (int i=0; i<20; i++)"

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

For loop declaration ends with ":"

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

Everything after the ":" that is intended belongs inside the loop. In C#, the loop contents would be inside curly brackets.

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

This assigns the variable "players" a new value based on its old value and the virality parameter. Mathematical expressions follow standard rules: Multiplication is executed before addition.

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

The print function prints things into the Colab output. In Python, print usually works no matter the variable type.

```python
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

Combining spreadsheets and Python/Colab

- Many data scientists do some quick visualization and manipulation of data in Excel

- More detailed analyses in Python or R.

- The Pandas Python package allows one to load, manipulate, and save spreadsheets (we use this in the Clash Royale notebook)

- Python can also directly manipulate Google Sheets using the Google Sheets API:
  https://developers.google.com/sheets/api/quickstart/python

Rest of today: Work on the Colab notebook

- Everyone goes through the virality and retention Colab in their own pace.

- Go for a break when you feel like it

- If you complete the notebook, you can already proceed to the Clash royale balancing one

- Note: Kaggle is another notebook service that currently provides more free compute than Colab (including high-memory GPU:s for AI training), but this course's notebooks haven't been tested in the Kaggle environment.

# Lesson learned: To make it with a small initial budget, one needs to…

- Minimize burn

- Invest revenue in user acquisition

- Maximize odds of virality and free exposure => create something unique

- For example: Is your game unique and innovative enough to, e.g., get to GDC Experimental Gameplay Workshop or shortlisted for Independent Games Festival? Check the games featured there in previous years.

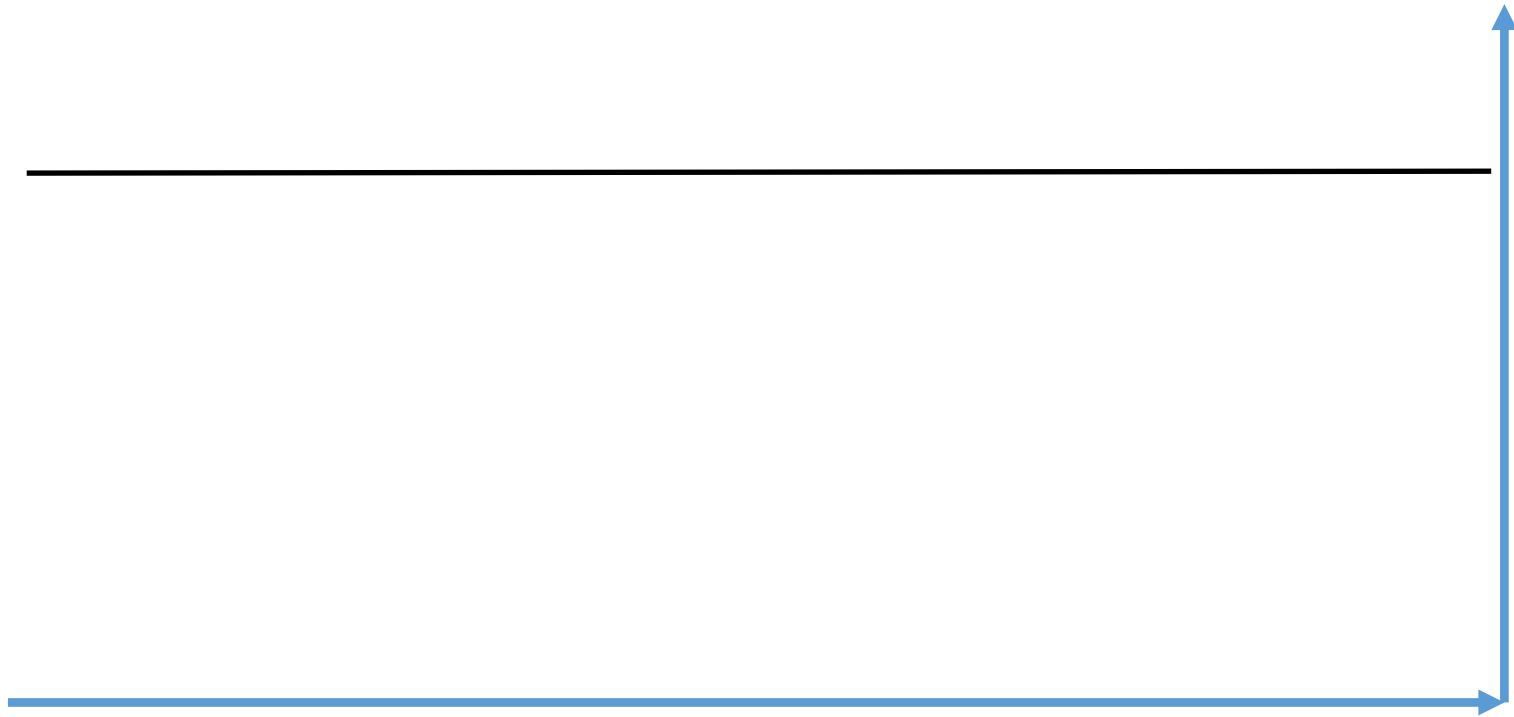# More on "power law" relations

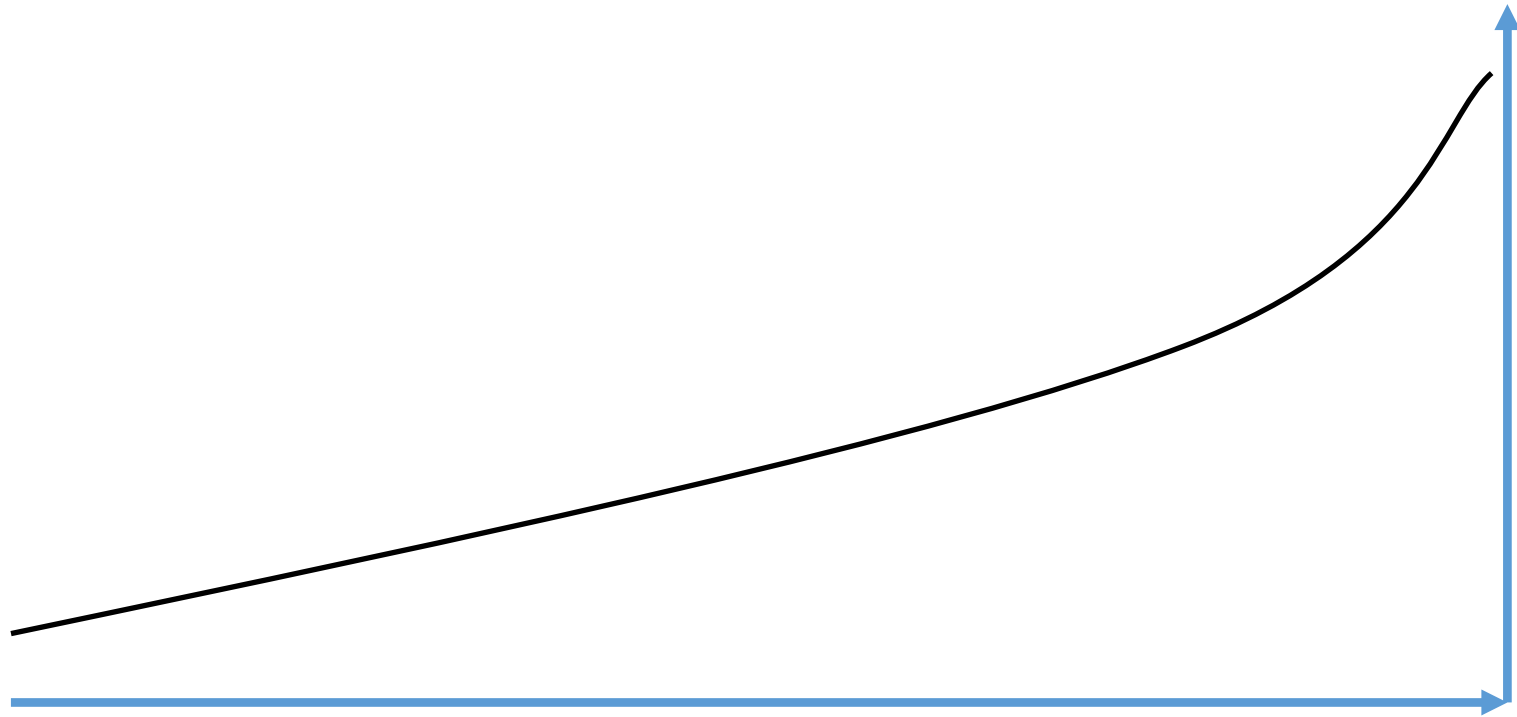# Income distribution looks like a power law. Is there a recursion?
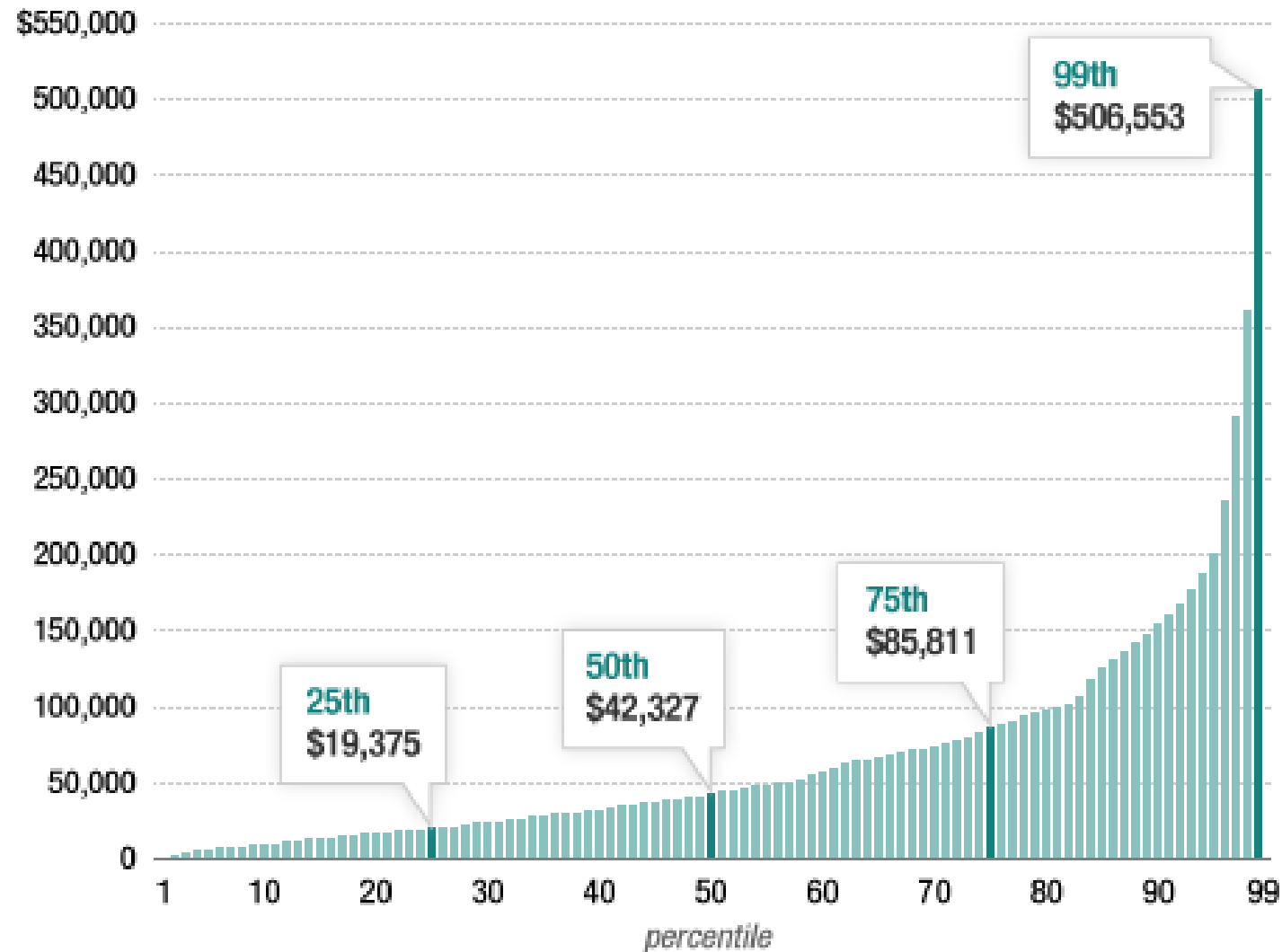
Distribution of wealth through recursion in Monopoly (game start)
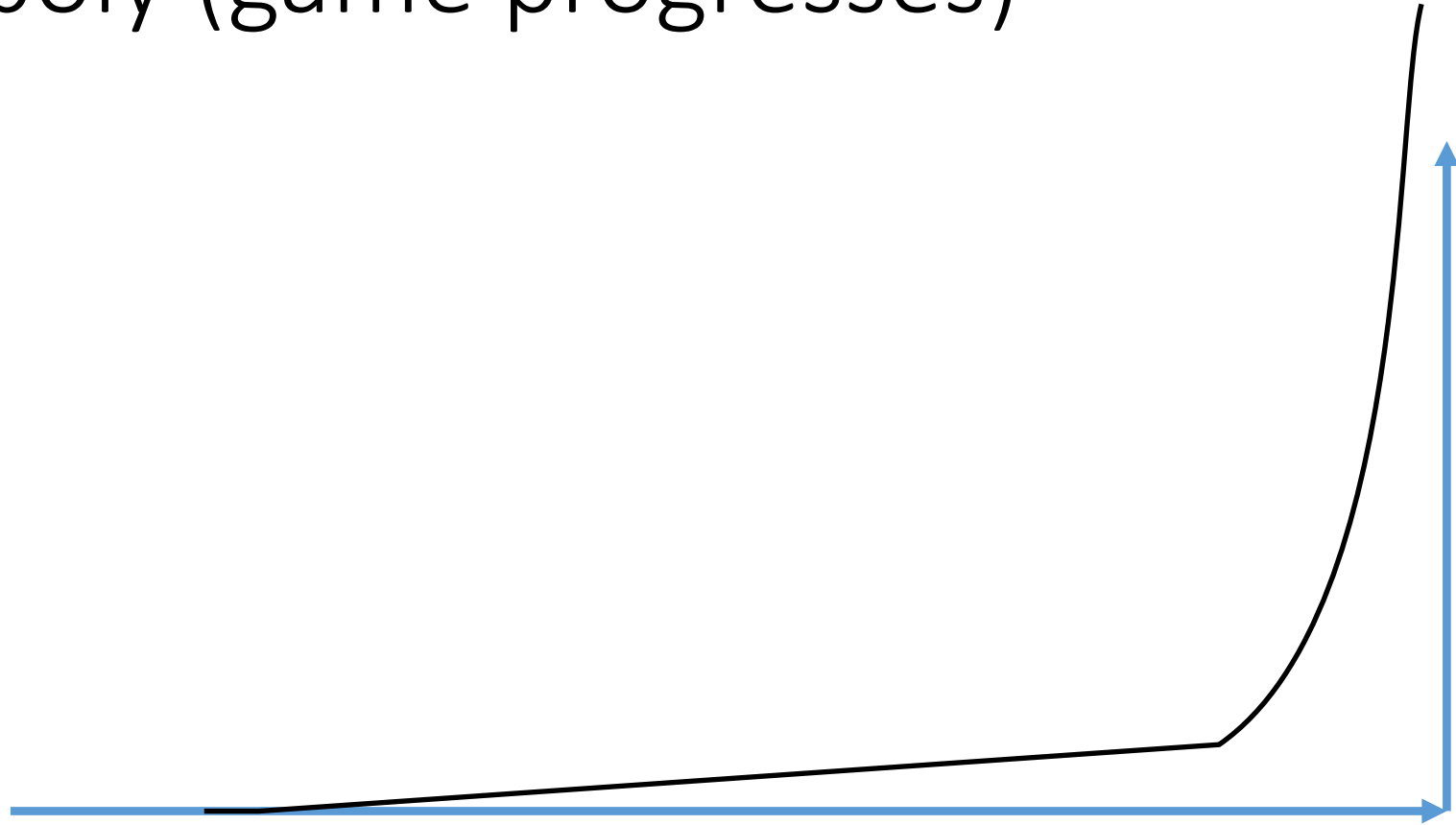
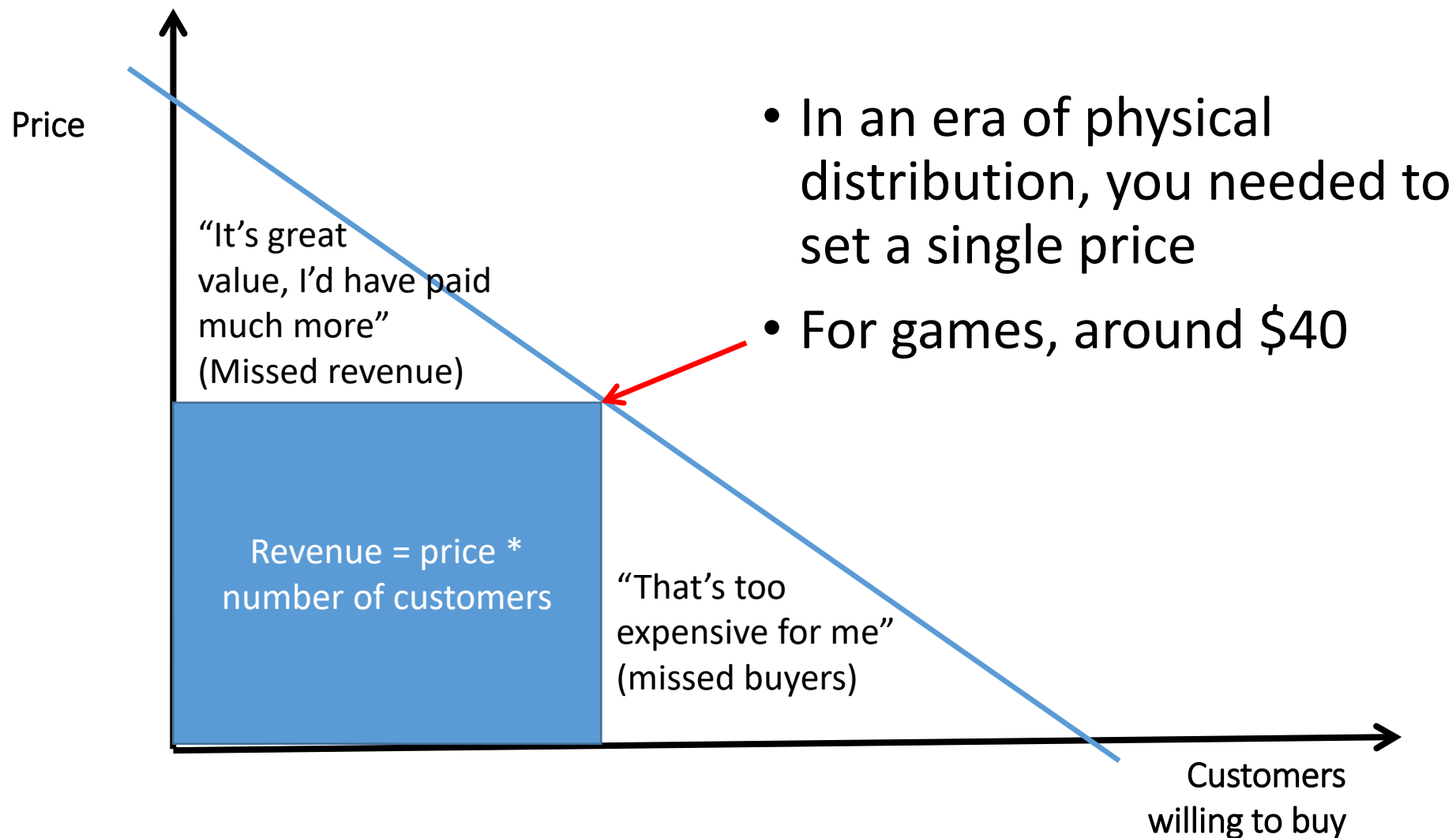# Distribution of wealth through recursion in Monopoly (game progresses)

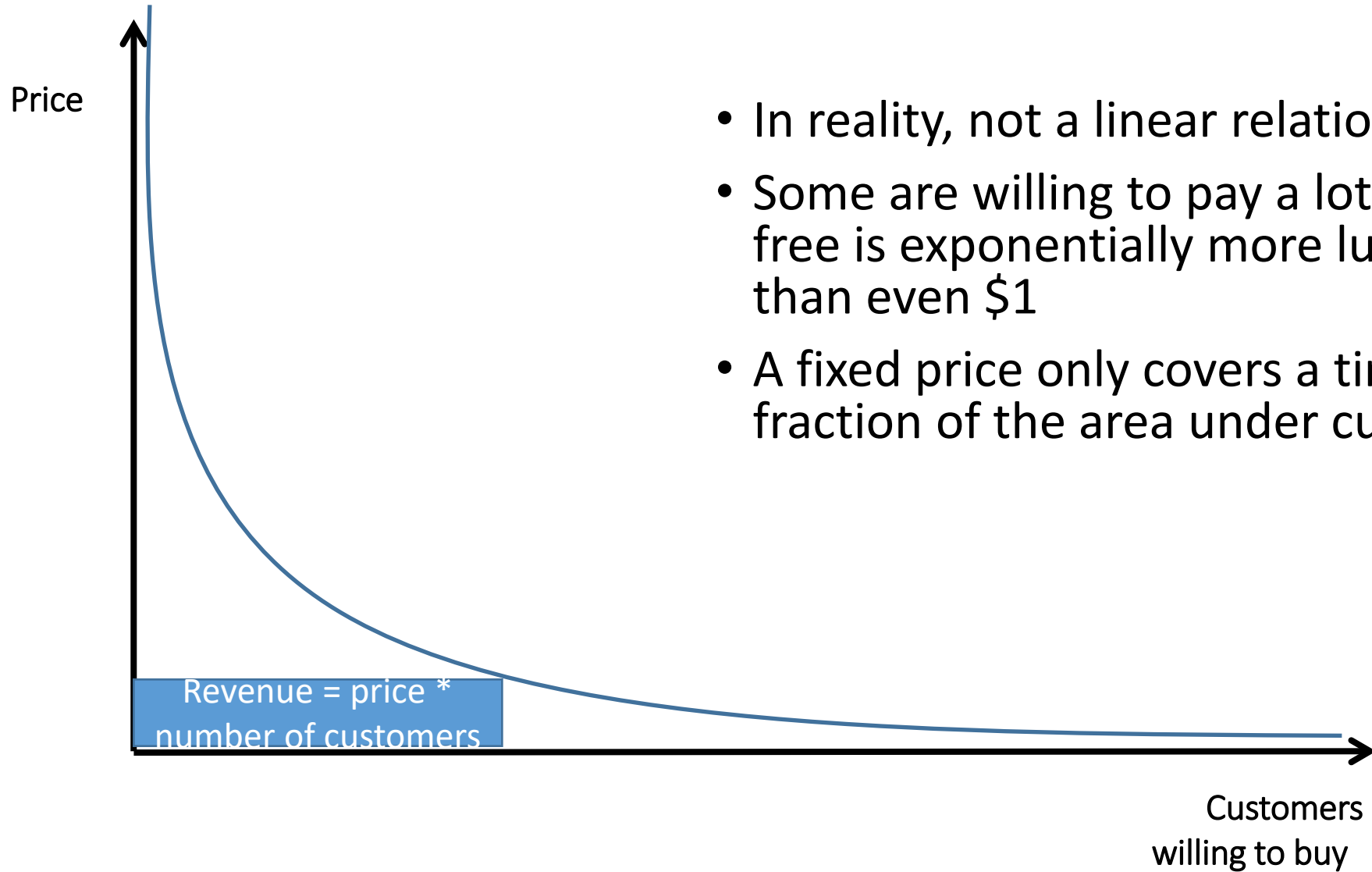# Distribution of wealth through recursion in Monopoly (game progresses)

The connection to game monetization

The price/demand curve

Price

"It's great value, I'd have paid much more" (Missed revenue)

Revenue = price * number of customers

"That's too expensive for me" (missed buyers)

Customers willing to buy

- In an era of physical distribution, you needed to set a single price
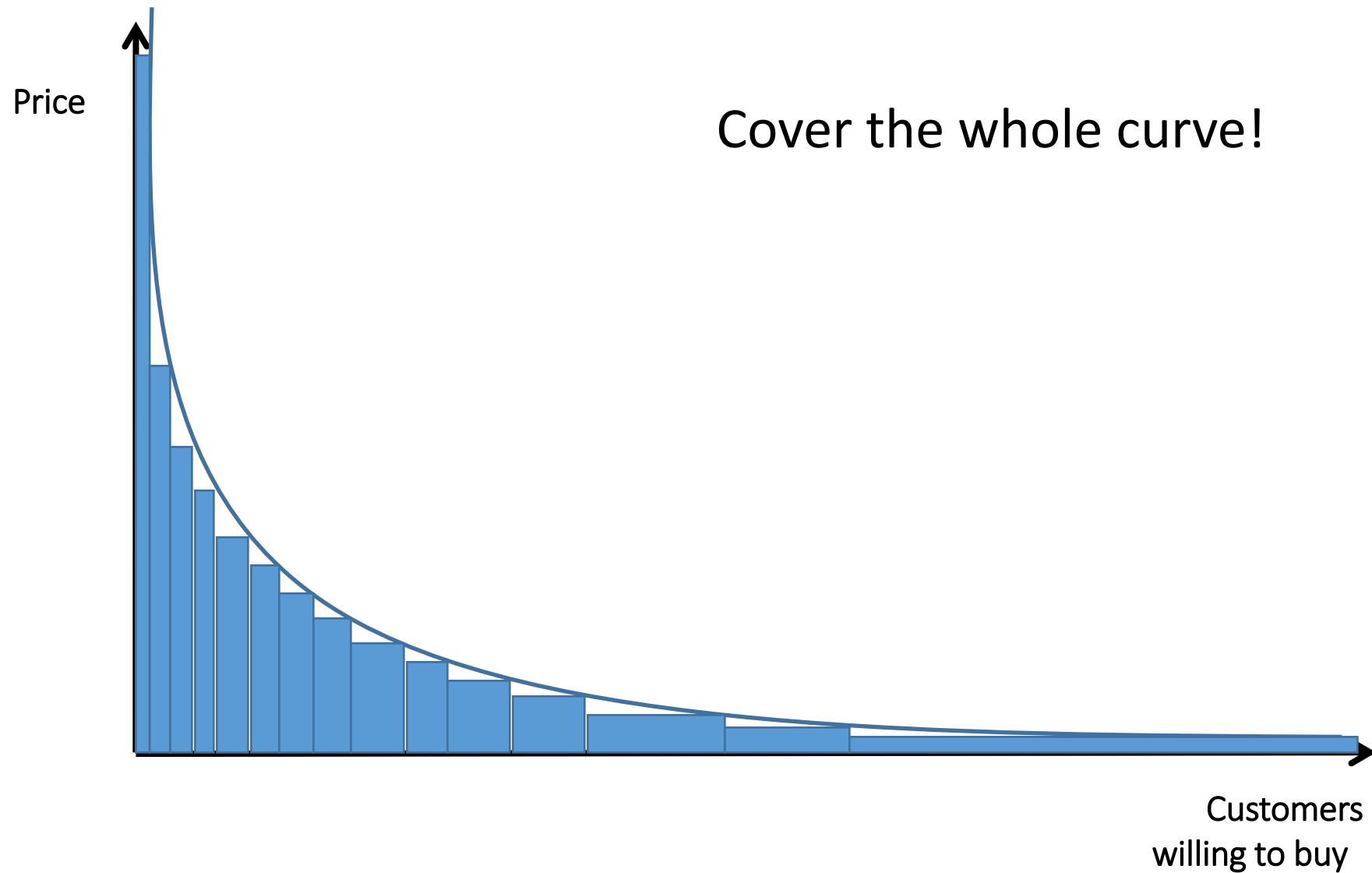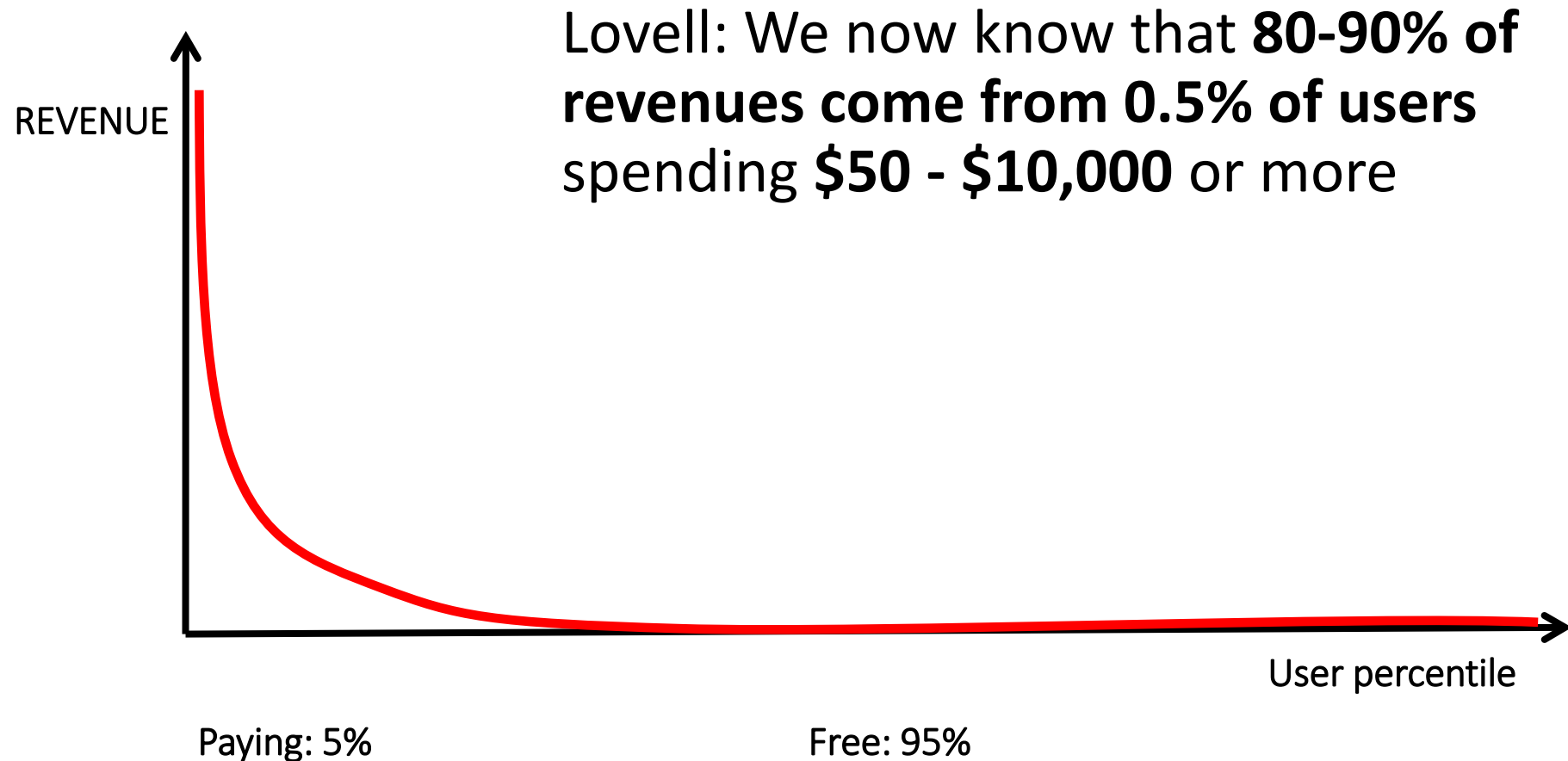- For games, around $40

The price/demand curve

- In reality, not a linear relationship.
- Some are willing to pay a lot, and free is exponentially more lucrative than even $1
- A fixed price only covers a tiny fraction of the area under curve

Price

Revenue = price * number of customers

Customers willing to buy

F2P: Allowing users to choose how much to pay

Price

Cover the whole curve!

Customers willing to buy

F2P: Allowing users to choose how little to pay

Lovell: We now know that **80-90% of revenues come from 0.5% of users** spending **$50 - $10,000** or more

REVENUE

User percentile

Paying: 5%   Free: 95%

Huge differences in players is not news

- 90% of gamers don't finish a game
- Only <10% of purchasers participate in online discussion forums

Other explanation: addiction

- Free-to-play games utilize principles that lead to gambling addiction with some people (more of that in upcoming weeks)
- Combining ethics and economics: Limit player spending, just don't set the limit too low.

Summary

- Power laws and exponential growth or decay caused by recursion are common in game economics

- Tipping points: sometimes small changes have huge effects, e.g., change behavior from exponential decay to exponential growth

- Retention, virality (organic growth), and investing revenue for continuous user acquisition are crucial