Part VI

Python Project 2025–2026: MedievAIl BAIttle GenerAIl

`pyproj2526_generail.tex` **WORK IN PROGRESS !**

Short version: *implement a medieval battle simulator capable (hopefully) of sophisticated tactics.*

The context in a nutshell: *Age of Empires (2)* (AOE) is a medieval-themed RTS (cf. Sec. 74[p399]: "What is an RTS / AoE?"). I'm rather fond of it, both as a game and as an bottomless well of ideas for Python projects. Although 2021 and 2024's projects[bt] were also inspired by AOE, all three have different focuses and present very different challenges. You won't gain much from reusing code from last year.

68 Scope of the project

Whereas previous projects covered both economic and military aspects, with and without user interaction, with the goal of imitating a small subset of AOE, this year's goal is to focus exclusively on AI battles, without any user interaction, base building, economy, or unit production, and to *exceed* the performance of AOE in that regard. Basically you will write artificial generals, give them command of equal armies, and may the best AI win.

The AOE AI is terrible in this regard.

The principle of AOE is to balance the player's attention between building an economy (villagers gathering resources and erecting buildings), training an army (units produced from specific buildings, at the cost of time and resources), and managing the armies in battle.

The AI in AOE is simultaneously absolutely godlike and utterly pathetic.

It is **godlike**, because it always pays attention to everything at the same time.

A human player is doing one thing at a time. When he is micromanaging a battle, he is not reassigning his villagers towards new resources *and* rebuilding his base that just got destroyed *and* building a new military complex to prepare for the switch to a different kind of unit that counters the enemy's.

The AI is always micromanaging each of its (up to 200) units at all time. If you give the AI infinite resources and instant construction and conscription time, it can and will turn three

[bt]Sec. VIII[p399]: "Archived Python Project 2024–2025: AIge of EmpAIres", Sec. XI[p424]: "Archived Python Project 2021–2022: Age of Cheap Empires"

villagers into a heavily fortified base covering half the map and spewing a constant stream of hundreds varied units *in twenty seconds flat*. And while it's doing all that, if you fire an arrow at some random villager, it will casually move the villager to sidestep the arrow.

It is also programmed to implement the optimised openings developed by the expert players of AOE2 over 25 years.

It is **pathetic**, because the massive armies which its economy can sustain are usually streamed as a constant diarrhoea of units spewed in utter disorder at whichever point it's decided to attack, with complete disregard for common sense and survival, let alone tactics.

Combat in AOE is a game of *counters*; almost of rock-paper-scissors.

Cavalry counters archers, because it's fast, tanky, and hits hard, so it charges, gets up close and massacres the squishy archers. Infantry counters cavalry, because they're cheaper, so you have more of them. Archers counter infantry, because they aren't too tanky and are quite slow and take a lot of damage before getting close.

It gets more complicated, because there are many types of units within those categories, each with different HP, armor (melee or pierce), and *bonus damage*.

For instance, *pikemen* are a very weak infantry unit generally, but they do *massive* extra damage against cavalry.

Skirmishers are very weak archer-like units, but they have high pierce armor (defending against arrows) and do high bonus damage against all archers (and some against pikemen).

Add to that *micro*-management of units. Do pikemen counter cavalry archers?

If the player with the cav archers is not controlling them at all, he'll be wiped out in seconds.

If he is paying attention and just uses hit and run tactics, all pikemen will die without landing a single hit. Having 1000 pikemen against one cav archer won't change that. It just takes longer. That's the entire point of having cav archers, whether in the game or in real life.

Add in all there considerations, and an AOE battle involves many moment to moment tactical considerations, especially with regards to battle formations and positioning generally. The goal is to manoeuvre the battle in such a way as to have the right type of soldier dealing with the right type of enemy, even if you sometimes have to sacrifice some of your army to move counter-units out of position.

The AI does not do any of that; it will send a skirmisher alone against a castle, to die instantly without doing any damage; and the next one 15 seconds later; and the next ones indefinitely. It will not use battle formations, though that is an AOE feature.

You will do better.

69 Requirements

(1) The game takes place on a **map**, which is a grid of size N × M tiles. The absolute minimum size you must be able to handle is 120 × 120, which corresponds to a "tiny" map in AoE2.

(2) Battle scenarios.

Your battle engine will take as input a battle scenario, which is to say a map with military units and defensive structures placed on it for each of the two players.

I have little interest in asymmetric battles. I want to see the clash of different *tactics*, not of different *armies*.

In fact I strongly advise you to define your scenarios by **populating half the map**, and mirroring that for the other player.

I have no interest in scaling up to N players in this project.

You should be able to handle armies in the 100–200 units range; plus a few buildings if you implement those.

Scenarios should play out very fast (2min max at human-compatible speed); don't position the armies very far from each other; but still far enough that the general has time to get his army in whichever formation he wants before the engagement begins.

(3) Programmable Scenarios and Lanchester's Laws.

You should make your battle scenarios by programming them. Do not make a visual map editor. A map/scenario is defined by a script that places units on the grid. It should take parameters, for instance, the number of soldiers.

You will make (at the very least!) a `Lanchester(type, N)` scenario for the purpose of testing Lanchester's Laws[bu].

`type` is the type of units to create (melee or archer type)
`N` is the base number of units: you will pit `N` soldiers against `2*N` of the same soldiers, within range of each other.

By varying `N`, you'll see (and graph, more in the CLI section) how the losses correlate with `N` for both types of units. If Lanchester's laws apply, you should get very different curves depending on unit type.

(4) Generals.

[bu]https://en.wikipedia.org/wiki/Lanchester's_laws
https://cgsc.contentdm.oclc.org/digital/api/collection/p4013coll3/id/1786/download

Your program will also take as input a type of AI (a "**general**"), who commands its army.

You will have *at least* two generals.

Victory = destroy target building (wonder) first OR kill all enemy units.

a. Captain BRAINDEAD.

The good captain has just come back from a successful lobotomy. He is in no state to give any order.

Units on the field will still attack take individual action, attacking units within their line of sight and fighting back against an attack, but they will not seek out a fight if left alone.

Basically, zero tactical AI.

b. Major DAFT.

His name stands for... itself..., and Dumb-As-Fuck-Tactician.

Major DAFT will command all of his army to attack the nearest enemy with absolutely no further consideration.

The difference with Cpt. BRAINDEAD is that units will actively seek out targets and get back into the fight.

Needless to say, those exist just to make anyone else look good.

Impressively, the AOE AI manages to consistently lose to *both of them* in my little test scenario. I will demonstrate this to you during the project's introductory lecture.

You will do better. I hope. You will program other generals, ideally capable of using unit formations, exploiting counters, using hit-and-run, interceptions, feints, and sacrifices. I'm less interested in dodging micro (arrows, onagers shots), but it's also an option.

Concretely, your final version should probably have one central AI program, and each of your generals should be defined by a few parameters, activating some features (e.g. micro) or changing the weight of different incentives. The alternative risks huge code duplication.

That said, during development you should also be able to pit successive versions of your general against each other, so make sure to structure in such a way that this can be done.

No general will play "keepaway" indefinitely. Even if he sees no path to victory, he will send his last men on a suicide attack.

(5) Automated battle tournament.

You should have a way to automatically pit your generals against each other and score their relative strengths.

You select the participating generals (by default all available functional ones) and the scenarios (by default a selection of pertinent scenarios), and launch the tournament.

Once the tournament is launched, it is fully automatic. For each scenario, and each pair of generals, it runs the scenario a set number of time (alternating competitors' positions / player number by default) and records victories (draws if takes too long, which should be see as a problem and fixed!).

The tournament will generate a document (HTML, PDF,...) listing outcome and stats, with score matrices:

a. General score for each general across all opponents and scenarios (in percent of victories)

b. General vs General, across all scenarios.

c. One version of above for each scenario

d. General vs Scenario, across all opponents (would be interesting if a general had inconsistent performance across scenarios)

Do not exclude reflexive matchups (X vs X): if you don't get roughly 50% everywhere (with enough matches), when not alternating positions, it means you have a bug that favours one "player" over the other.

See also the section on CLI syntax.

(6) Mechanics, Units, and Structures.

This is not a game design course, so we will just reuse the mechanics and exact stats from AOE. The AOE wikis and other websites will be your sources.

For instance, you will in `https://ageofempires.fandom.com/wiki/Attack`

the general damage formula, with $k_{elev} \in \{.75, 1.25\}$ as appropriate (see the next point on *elevation*)

$$\mathrm{Damage} = \max(1, k_{elev} \times \sum_i \max(0, \mathrm{Attack}_i - \mathrm{Armor}_i))$$

The is also nice: `https://www.aoe2database.com/damage_formula/en`, `https://www.aoe2database.com/unit/25/-1/en`.

For units, I also like `https://aoe2techtree.net/#Magyars` as a synthetic source.

For instance I can quickly extract the relevant info for Pikemen:

```
Pikeman
```

```
Stats
HP: 55, Attack: 4, Armor: 0, Pierce armor: 0, Range: 0,
Line of Sight: 4, Speed: 1, Build Time: 22s, Reload Time: 3s

Advanced Stats
Attacks
1 (Shock Infantry), 1 (Standard Buildings), 25 (Elephants),
4 (Base Melee), 22 (Mounted Units), 16 (Ships), 18 (Camels),
7 (Mamelukes), 16 (Fishing Ships), 0 (All Archers)

Armours
0 (Spear Units), 0 (Infantry), 0 (Base Melee),
0 (Base Pierce), 0 (Obsolete)
```

You will use the exact value from the game — HP, Armors and armor classes, bonus damage, reload times, damage values, line of sight, etc — for each unit and building you implement.

We will ignore all civilisation bonuses and pretend we're in Dark Age, meaning no upgrades whatsoever. The units themselves will be mostly from Castle Age, with some from the Imperial Age. That means you can take their raw stats as listed in the sources, no +1 or +2 or anything.

The way archery is handled in AOE2 is complicated: Ballistics, Precision, Missed Shot Damage, and Hitboxes are some of the concepts involved. I'm not asking you to reproduce all that. Just pick a system that works and uses the same damage and fire rate calculations as AOE2. We copy AOE2 behaviour where and to the extent that it's easy, and don't sweat the details where it isn't, so long as it's similar enough that I understand at a glance the balance of the forces in play.

The first three units below are absolutely mandatory. Implement the rest as you can, preferably roughly in that order.

a. Knight

b. Pikeman

c. Crossbowman

d. Long Swordsman

e. Elite Skirmisher

f. Cavalry Archer

g. Onager

h. Light Cavalry

i. Scorpion

j. Capped Ram

k. Castle (defence and target to destroy)

l. Trebuchet

m. Elite War Elephant (trample damage)

n. Monk (healing and conversion: counter to elephants :-)

o. Wonder (target to destroy: victory condition)

You will use clearly recognisable sprites (crib those from AOE) so that I know at a glance exactly what the units on screen are.

(7) Agile implementation.

Do not try and implement all units and buildings in one go.

Begins with just Knights and BRAINDEAD and DAFT. Then add Pikemen and Crossbowmen.

Then make a general that uses counters to beat the pants off of DAFT, every time.

Then spend time improving him. You can (and must) automate tournaments across different generals / versions / scenarios to quickly test whether your ideas are paying off in practice.

Don't optimise first. Develop your tactics using formations with smaller armies first, then optimise and scale up.

Don't add new units if you don't do anything with them in the end. If you "have" Elite Skirmishers but your AI uses them exactly as it does Crossbowmen, then it's pointless. Do not claim to have fully implemented them, then. You'll show a yellow *partial implementation* checkmark for that unit in the checklist during the project defence.

A unit is fully implemented when it has all the right values/armor classes etc AND a specific behaviour in the hands of at least one of your generals, which plays to that unit's strengths.

You'll preferably has a small scenario ready to show for each unit that illustrates that behaviour in 30s flat, outmanoeuvring DAFT through superior tactics / positioning.

FISA: You have much less time for this project than your FISE comrades. Trying to pointlessly multiply units would be an even greater trap for you. Focus on a few, and make them work as intelligently as you can. Using one's time efficiently is always important, but the less time you have, the more critical it becomes.

(8) No main manu: CLI

Do not spend much time implementing a sophisticated game menu with a pretty background and music etc. I could hardly care less.

I care about functionality. Let's use a CLI. You will use `argparse`.

You should have something like that

```
battle run <scenario> <AI1> <AI2> [-t] [-d DATAFILE]
battle load <savefile>
battle tourney [-G AI1 AI2...] [-S SCENARIO1 SCENARIO2] [-N=10] [-na=False]
battle plot <AI> <plotter> <scenario(arg1,...)> <range for arg1> ..... [-N=10]
```

To launch a battle / load a save / run an automatic tournament / plot the outcomes of a scenario with parameters.

`-t` for launching terminal view instead of 2.5D.

`-d` for specifying whether / where to write data from that battle.

`-N` number of rounds for each matchup.

`-na` for *not* alternating player position across `N` matches.

Tell me if you think of important options, I may want to put them here so every group uses the same syntax.

More explanation for `plot`, by example:

```
battle plot DAFT PlotLanchester Lanchester [Knight,Crossbow] range(1,100)
```

Recalling the signature of the scenario: `Lanchester(type, N)`, this does roughly (pseudo code):

```
for type in [Knight,Crossbow]:
  for N in range(1,100):
    data[type,N] = Lanchester(type, N).run() # also, repeat N times etc...
PlotLanchester(data)
```

where `data[type,N]` contains at least the number of casualties / units remaining / total HP damage taken on each side, and `PlotLanchester` produces pertinent graphs to visualise that data. In that specific case, I expect the larger side to win every time, and what's interesting is to have a graph with two curves corresponding to `types`, with `x=N` and `y=`casualties sustained by winning side.

Of course, a scenario may have several pertinent `Plotters`, and you may want to produce the data once and iterate on the plotters, so it's probably best to separate the code for the scenarios and the plotters.

For the CLI arguments, do not hesitate to use `eval()` it will simplify things, allowing you to use Python code with you own class names etc directly in the arguments, rather than waste time writing a front-facing translation layers between them. (What are the security implications of using `eval()` in that case?).

For `tourney` and `plot`, note that it should be possible, even easy, to have these battles in parallel, on powerful CPUs at least. Two reasons:

(1) If you have properly separated the logical engine from the graphical interface, you can run battles without displaying them graphically, and, without no need to slow down for the benefit of humans, as fast as the CPU can compute the moves (headless mode). **(2)** You don't need a very sophisticated architecture to run battles in parallel. Especially in headless mode. `tourney` can just call plenty of instances of `run` as batches of external processes and wait for them to finish to collate the data / call a new batch.

Thus a very good implementation will allow you to gather data from hundreds or thousands of battles in few minutes.

Of course, if you can't do that, a naive implementation of `tourney` and `plot`, running at human speed is better than no implementation at all :-)

(9) **Map visualisation: terminal**

Just like the game of chess is independent of the gameboard you use — wooden or glass pieces? on the table or on a computer or purely via chess notation through email or snail mail or . . . — the logic of your game must be independent of its graphical visualisation.

This is extremely important to understand, with far-ranging consequences on development time, quality and reliability of code, etc.

To force you to separate the logic of the game (the rules, its state in the abstract) from how it is visualised, I ask you to provide two visualisations.

The first one is a terminal view. It does not need to show everything, but should still be sufficient to get a general idea of what's going on for small games on very small maps.

It will serve as a failsafe should you fail to develop the 2.5D view, so you have something to show on the day of the evaluation.

You will use the letters given for each building / tile to represent the map. For instance

```
CCCCCCCC
CCCCCCCCC


    PPPPP
     PPP
      P
    kkkk
      k
```

represents a formation of Pikemen (P) converging towards enemy knights (k). I use uppercase/lowercase to distinguish players here, but you should use colours.

One must be able to pause the game with P, to scroll the map using ZQSD and directional keys (+Maj to go fast).

Pressing TAB will pause the game and open a webpage (generated HTML file) listing all units in the game and their stats (HP, position, etc) and current tasks, as well as any relevant data on the states of player AIs. (This is purely a snapshot of the current state; it need not be regenerated as the game progresses!)

For instance, one could learn that the soldiers have lost HP.

You will not spend too much time making the page look *pretty*, but do give some thought to making it readable and searchable (collapsible sections) etc.

(10) Map visualisation: 2.5D

Separately from the terminal visualisation, you shall provide a graphical, top-down 2.5D (isometric, sprites-based) visualisation of the game map, in the style of AoE.

That means you can start or load a game either in terminal of GUI mode, or even switch between them on the fly, using the F9 button.

You may the use the sprites of AoE or other games, if you can extract or download them. It's a programming project, not an art class.

You will need a graphical framework for this task. Various possibilities include:

- PyGame, `https://www.pygame.org` is the most common choice among students for this type of projets.
- the Arcade Library `https://api.arcade.academy` `https://learn.arcade.academy`.
  Very fresh, but active; a few groups used it and had a good experience.
- TkInter,
- PySimpleGUI (with TkInter backend only; simpler to begin with)
- PyQt5 or PyQt6 (more powerful, more complex, external requirements[bv])
- wxPython, `https://www.wxpython.org`
  Bindings to wxWidgets, similar to PyQt.
- PursuedPyBear, `https://ppb.dev`
  This one seems very fresh out of the oven, and not documented.
- Kivy, `https://kivy.org`

[bv]`https://pypi.org/project/PyQt5/`; cf. `http://doc.qt.io/qt-5/examples-graphicsview.html` pour de la documentation C++. C'est à adapter à la version Python, car PyQt5 est juste une bibliothèque de liens (bindings) vers Qt5.

◇ or anything that works with Python, really, I'm not picky, what matters is the result.

Test the different possibilities, and choose wisely.

Do not attempt to do this in full 3D.

You should also be able to pause in this view. Variable speed would be great as well.

(11) Minimap (2.5D):

The 2.5D view should include a minimap for fast navigation — this will be of great use during the demonstration.

This can take the form of an actual minimap always present in the corner as in AoE, or be a global, zoomed-out view activated through the M key. You can also have both, if time allows.

(12) Army visualisation:

One should be able to see the numbers and types of units for both players at all times.

If all this obscures too much of the screen, use F1 through F4 to activate / deactivate some or all of the displayed information.

(13) Save and load.

You should be able to save the game state whenever you want, and load it with minimal loss of information.

Ideally, that includes what the AI's state of mind (planning an attack, game plan, etc).

You must be able to handle an arbitrary number of saves, manipulating them as standard files.

F11 = quick save
F12 = quick load

Since your scenarios should only last a couple of minutes at most, it's less important than for past years' projects, but still nice to have, and it's another thing that forces you to have a decent architecture behind the scene, in which case it can be done in a few lines via serialisation.

(14) Elevation. (*"I have the high ground, Anakin!"*)

In AOE, a map tile exists on a level of elevation (0-16). A unit on "the high ground" relative to another, that is to say on a tile with a higher level, inflicts 1.25 times the damage, and takes .75 times the damage. This applies to all units, buildings, and damage types, not only archers.

This adds a very interesting tactical consideration, and I'd like you to implement that. It is not a great priority, however.

It can be tough to represent graphically, but topological lines and a colour gradient will amply suffice; plus perhaps moving up slowing units down a bit.

(15) Obstructions. VERY OPTIONAL

We'll begin with entirely open maps with no obstructions besides units and (maybe) a few building.

If you've pretty much covered all the rest, you can add obstructions to your map: impassable tiles.

The completely changes how how the forces interact, more complicated pathfinding algorithms become necessary. . . you get the idea.

Part VII

Python Projet: Practical Modalities

70 Groups: size and composition

This project is done in groups of 5 or 6.

Groups will be determined "randomly", not chosen by students. The aim is both to save time and avoid reproducing the usual cliques.

It is recommended that each group designate a "project secretary", whose tasks include keeping track of who does what; he should have a global vision of the state of the project, and be able to inform me of it efficiently. He will probably be the main writer of the final report, so pick somebody who likes to write (French or English, I don't care).

None of this should take much time, so only a *slight* reduction in overall programming or design tasks is acceptable for the group secretary.

Nor is he automatically the taskmaster, bossing others around. If that's what you want, why not, but how you organise yourselves in the group is entirely up to you.

71 Evaluation

At the very end of the semester, each group will:

◇ Hand over a **short report**, a couple of days before the defence, summarising which requirements have been met, the individual contributions of each member, and the individual score of each member, agreed upon by group consensus.

◇ Present their work (15 minutes). This is referred to as "the **defence**" / "la soutenance". It is mostly a live demonstration of the work, with the help of a few *very specific* slides.

◇ Hand over the **git history** of the whole project (all source code and assets). Immediately before the defence.

◇ Hand over the **slides** used for the defence. Immediately after.

An individual mark shall be given to each student on the basis of all that.

Neither the report not the defence are marked in and of themselves. They are tools to communicate and assess the scope and quality of your work, and *that* is what we endeavour to evaluate.

72 Short report

The report must be a single PDF file, not exceeding 5Mb, titled "<group number> python report.pdf". Only *one* must be uploaded on Celene for the group.

It must contain the following things:

(1) **A *recent* photograph of each member of the group**, with the corresponding names, arranged so that all fits on one A4 page, vertical / portrait.

(2) **A screenshot of the 2.5D view** of your project, showing as many things as possible. Again, must fit on a single A4 portrait page.

(3) **A synthetic list of requirements**:

The project has a number of requirements, numbered in the document, each with a title in bold.

For each requirement, in order and using the same numbering scheme as me, and using the same titles in bold as me, you will state whether it is fully met (green), partially met (yellow), or not met (red), with a few lines of text explaining, where applicable, how your implementation deviates from the requirements.

(4) **A detailed description of the contributions of each.** It must be clear enough, in conjunction with the presentation, to enable the jury to mark the work of each student.

Each member must write a paragraph listing their useful contributions to the project.

The whole group must read every such paragraph, and a consensus must be reached that they are accurate. If a consensus cannot be reached — which will reflect poorly upon the whole group — a dissenting opinion must be written in a paragraph *below* the offending paragraph.

For instance, suppose that X claims to have done all the GUI, and Y and Z think they have meaningful contributions to it, and the rest of the group has not followed what happened in that part of the code.

Y and Z protest X's claims in his contributions paragraph when the group reads it. X refuses to modify his paragraph. Then Y and Z should add a dissenting opinion under X's paragraph, explaining what they disagree about. X cannot modify their dissenting opinion, anymore than Y and Z can modify X's contributions paragraph.

The final report must of course bear the imprimatur of all group members, but this is especially vital for those paragraphs.

(5) **A zero-sum scoring of the contributions of each group member.**

Il vous est demandé de pondérer la quantité de travail (utile, justifiable) de chacun par consensus du groupe. Par exemple: tout le monde a fourni le même travail, sauf X qui a travaillé 2 fois plus (fourni deux fois plus d'"utilité", pas seulement "remué deux fois plus vent") que les autres. Ces pondérations affecteront la note individuelle.

Qu'entend-on par travail utile, justifiable ?

Le plus facile à évaluer est la quantité de fonctionalités conçues et implémentées, pondérées par leur importance pour le projet.

Des aspects plus indirects ou flous de l'ordre du managérial ou "aide à la cohésion du groupe" sont à prendre en compte avec modération et beaucoup de prudence. Ne donnez pas un poids élevé à "ce gars a maintenu notre moral en faisant des blagues hénaurmes toutes les 5min" ou même au plus sérieux "il a joué le Boss et fait les diagrammes de Gantt de tout le monde" – sauf si c'était vraiment très utile, finement détaillé techniquement, et a vraiment eu une influence forte sur le groupe. Mais même dans ce cas, c'est un travail d'ingénieur qu'on note, pas de manager. S'il a fait les deux c'est un bonus, mais s'il n'a fait que le manager le score doit être faible, car ce n'est pas ce qui est demandé.

On note les "résultats", pas juste le temps passé. Quelqu'un qui bosse jour et nuit mais fait surtout des bêtises ou dessine 50 versions des icônes dont personne n'a besoin pendant qu'il reste des bugs urgents doit avoir un bas score. Quelqu'un qui fait ça alors que le groupe *insiste* pour qu'il fasse autre chose, mais est ignoré, doit avoir un très bas score.

Notons que "résultat" n'implique pas que, si ça n'apparaît pas dans le produit fini, ça ne compte pas. Le débuggage d'un bug complexe est un travail à valoriser dans le score, même si au final la partie du code qui a été debuggée a fini par être retirée du projet pour d'autres raisons.

La question est "au moment où le travail a été entrepris, était-il pertinent étant donné les connaissances du groupe à ce stade".

Par exemple, un travail de recherche en profondeur est tout à fait valorisable, même si le résultat final n'est pas à la hauteur des espérances – mais évidemment c'est toujours beaucoup mieux s'il l'est !

De même, aider un camarade est aussi valorisable – là aussi dans la mesure du raisonnable.

Le rapport doit obligatoirement fournir la pondération de la manière suivante, obtenue par *consensus*[(bw)] au sein du groupe:

Chaque membre du groupe i est assigné un score / une pondération $p_i \in \mathbb{N}$, de manière à ce que le ratio ${}^{p_i}/_{p_j}$ reflète bien la proportion de travail utile fourni par i par rapport à j. **Vous devez donc CONCRETEMENT rendre une liste d'association "membre du groupe $\mapsto$ score (nombre entier)".**

(Note: ***use your full name, of the form "FAMILY Given-name"*** for this, not just your first name. My lists are sorted by family name, and I don't know by heart who "Bob" is. Bonus points (morally, at least) if you sort by family name.)

Par exemple, si l'on a $p_{Basil} = 2$, $p_{Cunégonde} = 1$, et $p_{Quetzalcoatl} = 8$ cela signifie que Basil a en gros été deux fois plus productif que Cunégonde, mais bon globalement Quetzalcoatl est un dieu et a bien porté le groupe, ayant fourni

$$\frac{p_{Quetzalcoatl}}{p_{Basil} + p_{Cunégonde} + p_{Quetzalcoatl}} = \frac{8}{11} = 73\%$$

du travail total – en supposant un groupe de trois.

Pour discuter des scores de chacun, il peut être utile d'utiliser des nombres de points "intuitifs".

Par exemple, si $\sum_i p_i = 100$, i.e. on a 100 points au total à distribuer entre tous les membres, alors p_i représente la proportion du projet (produit fini ou travail de recherche valide) attribuable au travail de i, exprimée en pourcents.

On peut aussi partir de $p_i = 100$ pour chacun (tout le monde est égal est moyen) et ajuster en rajoutant des points aux membres moteur (par exemple, Machin est à 120% par rapport au membre moyen, donc $p_{Machin} = 120$) et en enlevant aux membres qui ont été plus tirés par le groupe $p_{Truc} = 80$, en essayant de maintenir l'invariant $\sum_i p_i = 600$ (pour un groupe de 6), afin de préserver l'idée que 100 représente le score du membre moyen du groupe. (Même si ça fait chaud au cœur de dire "tout le monde est au dessus de la moyenne du groupe", mathématiquement ça ne marche pas. L'utilisation d'un score "zero sum" évite ce biais.)

[(bw)] pas *majorité*; ce n'est pas un vote. On en discute ensemble jusqu'à ce que tout le monde tombe d'accord. Voir plus bas si l'on n'y arrive pas.

Having $\sum_i p_i$ be a nice, round number is not strictly necessary, but it helps me check that I have copied the numbers correctly on my spreadsheet. In any case, tell me what $\sum_i p_i$ is supposed to be, so that it can serve as a checksum of sorts.

Note: ce score *ne doit pas être ajusté par le groupe pour prendre en compte des excuses, valides ou non*. Si A et B ont objectivement moitié moins avancé que la moyenne (notée à 100) alors tous deux doivent avoir un score de 50. Le fait que A a passé la moitié du semestre à jouer à Minecraft alors que B a passé la moitié du semestre à l'hôpital suite à une attaque de Vélociraptor (non-provoquée) ne doit pas intervenir. Les excuses valides d'ordre médical ou autres sont prises en compte *par le corps enseignant* à divers niveaux; en ce qui concerne l'auto-évaluation, ce n'est en aucun cas votre problème.

Note: this score must must be computed *with respect to the whole group*, not wrt. subgroups. For instance, one group was broken into 3 pairs with different tasks, and each pair was given an equal number of points to distribute between them. This is *not valid*, as it bypasses the hard work of evaluating the value and difficulty of each task.

Note: Consensus ≠ Vote:

A way some groups have "achieved consensus" in previous years is by averaging or summing scores given (sometimes anonymously !) by all members to each member. This has a chance of being a meaningful metric *only if* everybody is very well-informed about every other member's contributions. Otherwise it tends to produce noise, which tends to yield poorly differentiated scores. You may use such techniques if you wish, but it must be a mere *starting point* that is then *discussed* by the group until nobody is shocked by any mark.

I consider vote-based methods a bit of a "cheat code" when it comes to achieving consensus. Votes are a conflict-breaking tool, not a truth-finding tool. The only consensus truly achieved by taking a vote about X is the meta statement "the outcome of the vote about X, whatever it is, holds value." This actually says nothing about X or whether that outcome is correct.

That's fine if the question is "what colour should the bike shed be", because to the extent that there might be a right answer here, it's probably "whichever colour most people want" anyway. Even if many (or even all[bx]) people dislike the result, the most important thing is not to waste more time on the issue, and to avoid fighting over it. Let's just pick a colour and move on.

The situation is quite different when matters of fact must be decided, with real stakes and decidedly right (correct, accurate,. . .) or wrong answers. The scientific method does not proceed through votes.

Votes should only be used when there is a conflict to break, not before, and this only if there is a pressing need to coalesce to a decision. Whenever possible, a consensus

[bx]If you average the result of a vote on colour, you'll probably get a vomit-inducing khaki nobody wanted :-)

obtained through rational discussion of all available evidence must be preferred.

Do not use voting as a clever tool to circumvent rational, possibly heated, discussion, and avoid having to actually formulate, defend, and change your opinions.

I want a consensus on the quality of the work of each, obtained through thorough discussion — and I know it can be hard — not *a consensus on a hack to avoid really having said discussion.*

The last thing I want is for a group (I take an extreme theoretical case) where everybody wants *all* the points to average anonymous votes and come up with the same score for everybody. . .

Si la pondération donnée par le groupe est manifestement fausse, c'est tout le groupe qui sera pénalisé.

Par exemple si le groupe dit "ben tout le monde a travaillé pareil, 100 à chacun", alors que pendant la soutenance on voit bien qu'il y a une personne qui sait répondre à toutes les questions, que ce soit sur la vue d'ensemble ou sur le fonctionnement du code, et une autre qui découvre la sujet et le logiciel le jour de la soutenance, ça va mal se passer. Soit tout le monde était tellement à la masse pendant le projet que personne n'a remarqué les grosses différences entre membres, soit le groupe est trop disfonctionnel pour avoir une conversation bilan honnête entre ses membres.

Les étudiants se plaignent souvent – et à raison ! – que les notes des travaux de groupe sont injustes; c'est l'occasion de rectifier le tir et, les enseignants n'ayant pas le budget pour une boule de cristal ou de chevaux de Troie dans vos ordinateurs, vous êtes encore les mieux placés pour le faire.

Si un consensus ne peut pas être atteint au sein du groupe (essayez, quand-même, parce que ça n'amusera personne de gérer ça et risque de pénaliser le groupe globalement) proposez plusieurs pondérations (e.g. celle soutenue par A, B, D, et E, selon laquelle C et F sont des glandeurs, et celle soutenue par C et F, selon laquelle ils ont tout fait) et nous en discuterons calmement.

Si le groupe atteint tant bien que mal un consensus mais qu'un (ou plusieurs) membre (ou sous-groupe) n'est pas satisfait, mais pas tout à fait au point de refuser entièrement le consensus (i.e. "J'accepte, mais pas content !", versus "Je refuse ! Révolution !"), ce membre peut joindre au rapport, sous le consensus, une *opinion dissidente* expliquant ce qui le chiffonne un peu dans le consensus tel qu'il est.

Le rapport peut également mentionner si le consensus a été obtenu facilement ou s'il a été difficile à négocier.

FAQ : comment la pondération donnée par le groupe affectera-t-elle la note individuelle, exactement ? Y a-t-il une formule ?

Nous noterons au mieux, dans un monde imparfait, avec les informations dont nous disposons.

There is indeed a formula that is being (somewhat) systematically applied. Following Goodhart's law[by], I will not share it.

I would just note that in 2018–2019, the maximal difference between the worst and best marks within a group was 9 points out of 20. (The minimal intra-group difference was 0.1 points. The average intra-group difference was 4 points.)

The upshot is that you should not expect to get a good mark merely because other people are working and the end product is good. You have to contribute to it.

Conversely, if you are unlucky and end up in a disfunctional group, this does not automatically mean your mark will be terrible, so long as you can show meaningful contributions.

Overall, this system, while imperfect, proved much better, meaning much *fairer*, than handing out the same mark to everybody in each group, as was the case previously.

The cost is to force the group to confront and to *evaluate* the very real differences of skill and investment within the group, and confront one's autoevaluation to that of the group, which are very socially difficult exercises, without a doubt, but necessary ones.

73 The defence

Il y aura une journée de soutenances à la fin du semestre, où chaque groupe présentera très rapidement ses travaux, en fera une démonstration, et répondra à des questions.

Les modalités exactes sont comme suit:

73.0.1 Horaires de passage

Le planning sera en ligne sur Celene.

Deux salles seront réservées pendant les soutenances: c'est le jury qui passe d'une salle à l'autre.

De cette manière chaque groupe peut s'installer tranquillement pendant que le groupe précédent soutient dans l'autre salle.

73.0.2 Timing

Each group will have a 30min slot. You will have 15min to talk without interruption. The remainder of the time will be for questions / remarks.

Your talk will spend

[by] *When a measure becomes a target, it ceases to be a good measure.*

- ◇ 2 minutes max on slides — In the next paragraph I'll tell you exactly what your slides are going to be. You will open with this.
- ◇ 13 min of live demonstration.

Be extremely strict on time. I shall interrupt you mercilessly the instant you go overboard either on the slides of the global duration.

You will be ready to switch back and forth between the slides and the demonstration during the questions.

73.0.3 The slides

Each slide will have your group number somewhere.

There will be no animation or slide transition, or generally anything that does not perfectly translate into a two page PDF.

You will have *exactly* the following slides:

(1) A synthetic table of requirements met (green/yellow/red), like in the short report, but with fewer details so that it all fits on a single slide.

   You will speak for about 1m30 **maximum** to summarise the important points, especially to make us understand the scope of the limitations regarding the yellow/red requirements.

(2) A single slide with a *recent* photograph of each member, their name, their individual score (as in the report — the scores must of course be the same as in the report !), and a few keywords as to the nature of their contributions.

   About 30s maximum will be spent on this; the aim is the have a rough idea how uniformly (or not) the work was allocated before beginning.

We may use the slides as support during the questions phase.

73.0.4 The live demonstration

The demonstration must convince the jury that every requirement you claim to have met is indeed met.

Prepare it and rehearse it well in advance, like a theatre play, using functionalities of your project to help you, such as save and load. In your discourse, use the same keywords as in the requirements list, and quickly state the corresponding requirement number, to make it clear what requirement(s) you are demonstrating.

The demonstration **must be live**, we will not accept prerecorded videos or screenshot slide shows.

During the questions phase, you must be ready to let the members of the jury interact directly with your project.

While every member of the group must be present and answer direct questions from the jury, not every member needs to talk equally (or indeed at all) during your 15min. Again there is no mark for the defence itself; the aim is to convey the scope and quality of your work. Apportion the presentation time among your group in order to maximise the clarity of the defence.

Les démonstrations peuvent se dérouler soit sur l'ordinateur de la salle (celui connecté au vidéo projecteur), soit sur votre ordinateur portable personnel.

Dans tous les cas, évitez les temps morts dus à des contraintes techniques. Do everything on a single computer rather than spending time fighting the video projector each time you switch machines.

Vérifiez aussi la connexion de l'ordinateur servant à la démo avec le vidéo-projecteur avant le jour J.