Design Notes from 5-Min Civ

Total dev time: ~ 4 days

Engine used: Löve 2D

Idea development

The theme for the jam was “Collective intelligence vs. collective madness”. Here are some starting ideas that were evaluated:

1. A strategy RTS in which players can only control 'generals', other units follow orders down the command line
2. A narrative game similar to *We Become What We Behold*, in which players are led to indirectly and unknowingly kill other people by spreading disinformation and hate
3. A strategy game similar to *Plague Inc*, but in which madness and paranoia are transmitted via social networks

It was decided to go with the first idea, but with a focus on resource management instead of combat (for more ease of development). The game could be a strategy game about managing your own city/civilization, but in very short playtime (5 min) and automating most of the process. These games are usually very complex because the player needs to manually manage the movement of all units. Some games opt of allowing the automation of some parts, so that the player focuses on the decisions at macro level. Fully automating the game reduces the agency the player has, but also makes the game extremely simple. The key is allowing the player to still decide the ways the civilization evolves, but with limited input options.

Sliders

Sliders would be the perfect tool for allowing player actions: they are a visual indicator of a numeric parameter, and can be physically moved. In the finished game, the sliders allow changing the weights for each individual job, and then the jobs are calculated and assigned (mainly, if a change in a slider creates a disbalance, a worker from the job with lower demand is assigned to the job with higher demand). Since the sliders are the only action the player can take during the actual game, it felt natural that this would be extended to the menus and other screens (instead of using buttons as any other game), which has an additional benefit, because to arrive at the play screen the players have already moved 2 sliders, so they are aware that they can be moved horizontally within limits. It would have been nice to add a visual indicator to stop players from potentially getting stuck in the main screen, but time was short, so the absence of other interactible elements should do the trick. Another advantage is that using sliders to transition between scenes is more original than the classical button, which makes it more memorable (that game with sliders in the menus!).

Resources and buildings

There are 4 resources: wheat, wood, stone and iron. Wheat is used to grow new workers (when the city reaches a number of wheat units equal to 1.5 x current population, uses 1 x current population to create a worker), wood to create new buildings, and stone and iron to upgrade current buildings to get a 3rd or 4th worker position, and are unlocked after the 1st or 2nd main building upgrade respectively. The main building acts as a storage area, and needs to be upgraded 5 times using the best available construction material (wood, and then stone) to unlock not only the new material, but also new buildings and jobs: stonemason and medic/hospital, miner and blacksmith/forge. When a building is finished, it is decided which of the artificial buildings (so, excluding forest, quarry and

mine) is built, depending on the demands of the current worker distribution: the job with more workers without space receives the building. Also, the position for the next construction site is decided at random, but as close to the starting building as possible.

Workers and AI

Another important aspect of the game development was creating the AI that governs the behavior of the workers, which was something relatively new for me. For the resource gatherers, they search for the closest available work position of the resource corresponding to their type and asign that position to them so stops beinig available to other workers, then go there, and then start to work until they get the resource, which then they transport to the storage. Builders get assigned a random position, then go to the storage to search the needed resource, then carry it to the assigned position and they work until they finish and the resource is spent. Soldiers fight any enemy in sight, or get armor from blacksmiths, or stay at the barracks training. Lastly, medics and blacksmiths heal or give armor to soldiers respectively, both from their corresponding buildings. Also, some behaviors are independant of job (for example, going to heal if less than 50% HP). The hardest challenge was to allow transitioning between different jobs, since every worker at every moment needs to be available to transition at any time.

Tiles

The game terrain is made of roughtly squared tiles forming a vertical staggered bond pattern (each column is displaced vertically half the size of a tile). Each tile can host up to 1 building, and each building can host up to 4 worker positions, starting with 2. The center tile is assigned as the starting tile and receives the starting building. Other tiles are assigned 3 tiers: tier 1 are adjacent to the starting tile, tier 2 are adjacent to tier 1 tiles and tier 3 are adjacent to tier 2. 3 tiles from each tier 2 and 3 receive the 3 buildings that produce building materials: forest for wood, quarry for stone, mine for iron: this way there is always 1 position closer than the other, which rewards players for keeping a work distribution balance until they upgrade these buildings to allow additional workers.

Tutorial

Learning from previous experiences, players learn to play better if: a) there is little information needed at the beginning of the game, and b) is presented in a visually appealing manner. Because of this, the tutorial is condensed in only 1 screen, reduced to the absolute necessary and with visual aid (icons, colors) for the jobs themselves. Also, some information that is not necessary early (like the advanced resources and jobs, or how exactly the buildings are build) is ommited to not overwhelm the players, but also to reward them for being curious and discover how these mechanics work on their own. Some examples are:

- ➢ What are the advanced jobs and their buildings
- ➢ How exactly combat works
- ➢ Formulas for wheat use and worker generation
- ➢ How each building is decided upon construction
- ➢ How exactly the work-changing algorithm works
- ➢ What each advanced resource do
- ➢ How many upgrades/resources needs the main building
- ➢ Soldiers train in barracks if not in combat (getting more damage but also speed, which is useful also for the resource gatherers)

Art

As in other previous projects, the style was simple, small pixelart. The sprites were quickly created in GIMP to serve as a placeholder for future better replacements, but with time I felt they fit nicely enough (and again, there was no time to change anything). I think the reason is that the game benefits the lack of realism that simple icons provide, and that sprites with more quality would make the lack of particles or animations stand out.

Music and sound

The two songs used and most of the sound effects were downloaded asses. The rest of the sound effects were taken from previous projects. Originally the idea was to create original sound and music for this project to develop my music composing skills, but due to time constrains it was decided to download assets from the Internet and reuse sound effects from previous projects, since the game would not necessarily benefit from original sounds and music (in fact, it is likely that the originally composed music would be of inferior quality compared to the used themes).

Bugs

Some bugs were detected but not fixed due to time constrains:

- ➢ After upgrading the main building, sometimes two or more workers of the same type can use the same space, and some work positions can be displayed without transparency, looking like a worker.
- ➢ Healers can heal from distance after touching a worker, when they are supposed to heal only upon contact.

Ultimately, these bugs appear when the game is relatively advanced, and at this point the player is too distracted interacting with the main mechanics to notice these. And since they don't significatively impact the game in any way (since this is a short, single-player game balance is less relevant) it was decided that the time and resources needed to fix these bugs would improve the game more if spent in other areas.

Long-term viability (Is this idea viable for a succesful complete game?)

Overall, I think the idea of a simpler, more casual-friendly RTS in which most of the tasks are automated can succesfully work in the current market, where many players are deterred by the high complexity of the genre, which in turn creates a void for the empire management-experience for the majority of the players (for example, the only game from Paradox I have been able to play is Stellaris, while others are too complicated for me to properly enjoy). However, this would require a way to create depth: the problem with the concept is that the complexity that caracterizes these games is also its main appeal, and limiting it is fine if the game is for a jam but to keep the player engaged for hours this micromanaging complexity needs a substitute. A good example would be having different civilizations, each with their own twist on the main unique mechanic (be it sliders-only management, or a replacement). Facing other AI or player-controled cities/empires would be necessary to create more variability, which in turn would require adding different kinds of military units. Procedural level generation, already present in many similar games, can also provide more replayability and reward players who use the game's unique mechanic to its fullest. Fixed automation should be evaded to maintain a unique identity and avoid games like Factorio or Mini Settlers.

Lessons learnt

This was my first jam, so there is much to learn. I feel that this observations/tips can be applied in general, so if someone else is reading this I hope they are useful to you:

What did not work? (things to improve)

- Keep the concept simple. Less time programming = more time with design / art / polish. Also, easier to play and to communicate information to the player.
- Do not underestimate development time. Double your already doubled previsions.
- Develop tools before the jam: Tools like a tilemap generator or a basic AI framework will save time to improve the game (in this case, would have if I had made them). The time is limited and better spent working in the concept and the game-specific mechanics instead.
- Check the uploaded build before AND after uploading. In this particular case, a game-crashing bug was fixed and the build updated, but unfortunately it wasn't saved and the final build presented for the jam has this bug.

What did work? (things to maintain)

- Explore many different ideas at the beginning, but once the game has been decided fully commit to that. However...
- Always be flexible and open to changes: some of the game characteristics like unlocking worker positions or resources until upgrading the main building were not planned at the beginning, but were added later because it reduced complexity in the first minute and allowed to see the resource-producing buildings.
- Get the most out of the jam's theme: think about a concept that doesn't exist and that really goes into the theme. In this case, the starting idea was about generating a collective intelligence made from all the AI of each worker, but the actual gameplay lead to some chaos as the player tries to keep the job proportions balanced, which perfectly fits the other half of the theme (collective madness) and pushed the development in that direction.
- Keep your expectations low: even if you put your soul in your game, there is always going to be another one made by a more experience dev, or with a bigger team made of specialists. And even if your game is more mechanically innovative or deep than others, the main factor to attract players is beautiful visuals and polish, so your game is going to be mostly ignored. Remember that you are making this to hone your skills, not to impress anyone.