Design notes

Javier Alcalde Marchena

javieralcaldemarchena3a@gmail.com

30.05.2024

This file documents the origin of the idea for Elemental Monsters, the reasoning behind some of the design decissions behind the game and the vision for what it could be given enough time and resources. It is not intended to be an actual design doc, but an informal text to provide optional additional information about the game that complements the actual playing experience. Also, this serves as way to record my current thought process for the future, so some parts are going to be more personal and subjective. Of course, feel free to skim over anything if you are not interested or to skip this document altogether.

Setting priorities

When I started the course CS50's Introduction to Gave Development (Feb 2024), I saw all the content from future classes and noticed that the final project would be a game developed from scratch. This would be the first time where I would have total freedom over the design, since the rest of the course consisted of modifying and adding features to already existing games, so it was a great opportunity to create something special. While working in the first classes and getting familiar with the Lua language based LÖVE 2D engine, I started listing the priorities for this final project. It would need to be:

1) SIMPLE, or at least simple enough to be created by 1 person with no game dev experience in a short period of time (initially set to 1 month). Also, too much complexity could be problematic for a first-time project.
2) ORIGINAL, something that playtesters would like to play not because I asked it as a friend, but because it offers some type of experience that does not exist. This also would be important for me to keep motivation and be able to finish the game.
3) EXPANDABLE, that could be reestructured and expanded into a full (but small) complete game, but only if the concept shows enough potential. If I would commit several weeks to this project, is better if it gives me information that could be used in an hypotetical future game.

This meant that the game would need to be mechanics-based, so the focus is in the system design instead on the content creation, aesthetics or game feel (which means that the game is fun even if is somewhat repetitive, ugly or clunky). Also, it would need to be 2D based, since it eliminates the technical difficulty of working with the third dimension and would allow me to use the same engine as the classes from the course (the last 3 introduce the bases of 3D with Unity, but the more I advanced in the course the more confortable I was working with this LÖVE 2D).

Once I had these priorities set, I started listing some optional, nice-to-have requirements:

1) Procedural generation: these are systems that generate content based on parameters and randomization, instead of the hand-crafted content present in most games. Although they need to be carefully implemented, can generate variability and replayability, and allow the player to experience exclusive unique content. I would love to learn more about this since a good procedural system can replace many hours of content creation, and it fits my skills better. Also, I love the concept of epheremal art, in this case playing with something that no other player is going to experience ever. Many games use on procedural generated content during its development or in the game itself, but usually is used to generate levels. I think this tool has unexplored potential.

2) Basic artificial intelligence: neural networks AI is something very relevant and that is being explored by some studios, but here by IA I mean something much more simple. Basic AI patterns have been used in videogames for many years, and they allow the players to feel agency as the game is reacting to their actions, even if only via rudimentary instructions. Personally, I find the idea of autonomous, pseudo-living entities very compelling, and that is one of the reasons I studied Biotech. Usually is used to determine the behavior of enemies, but great experiences could be crafted when combining several layers of AI and group behavior, known as collective intelligence.
3) Action programming: the same principles of AI can be applied to the player actions, so that instead of directly controlling a character they assign . This allows the players to perform different actions simultaneously, reducing the need to micromanage every aspect of the game and testing their planning skills. Games from some genres, particularly 4X, suffer from an excess of micromanagement that creates a high barrier of entry for getting interesting experiences like leading a civilization or a space empire (any Paradox game).
4) Science/biology theme: as I said before, my academic background is in the health science field. Having the theme of the game be about something I am familiar with is not mandatory but may result in a different (and maybe less explored) vision compared to other games, whose devs belong mainly to the computer science field.

Also, I knew from previous experiences with board game design that it is important for the players to have some existing familiarity with the general concept of what the game is about. These board game designs were original and interesting, but this focus on innovation resulted in the games being too abstract and players feeling lost. The general concept for this game needed to be something new, but at the same time something familar, that potential players would understand without much explanation.

With all this in mind several ideas were evaluated, including among others an ecosystem builder, a casual hacker simulator and even a sports management game! But one other idea seemed to met all the requirements.

Inspiration from the classics

The bulk of the course (the first 8 from a total of 11 classes) was about adding features to LÖVE-2D versions of classic/simple games. The first 4 games (Pong, Flappy Birds, Breakout, Match-3) were used to show basic game and coding concepts like showing text, creating objects and allowing input, while the next 4 (Super Mario Bros, Angry Birds, The Legend of Zelda, Pokémon) showed more advanced concepts like physics, animations and menu logic. From the beginning I was biased to the second half, since some of these games have been very important for me when growing up: Super Mario Bros was the first game that I ever played, and both Super Mario 64 and Minish Cap (one of the lesser known Zelda games) introduced me to the concepts of 3D platforming and metroidvania-like progression (explore -> find dungeon -> defeat boss -> get tool or item -> unlock more places to explore) respectively. But it was the third generation of Pokémon games that showed me not only the genre of monster taming, but also served as a gateway to enter the world of roleplaying games, or RPGs.

In videogames, RPGs are a type of game in which you control a group of characters or 'party' that have adventures together and usually is about saving the world. This genre, inspired by the tabletop pen-and-paper game Dungeons and Dragons, allows the player to customize its party (choosing which partners to have in your team, which skills they learn and which equipment they use) and sometimes, to make choices during the game that result in different endings. Some of my favourite videogames are RPGs (Mass Effect, Fallout: New Vegas, Disco Elysium) or have some elements from the genre (Dishonored, Dark Souls), and I decided to work in game development while being a tester for another great RPG, Baldur's Gate III. This type of freedom was relatively new in the

medium, since most classic games tend to be more linear and be more like a puzzle that needs to be solved with skill, patience or both. However, this also means that RPGs tend to be less approachable to casual audiences, and generally considered a niche genre. What Pokémon did was to focus on this core concept of RPGs (the freedom to choose your allies) and streamline the most complex elements: previous games like Megami Tensei or Dragon Quest V allowed the player to capture enemies and use them in fights, but Pokémon made the fights 1 vs 1 and simplified the mechanics so that even young kids would be able to play it.

The success of Pokémon and others like Digimon created a fantasy of training and raising your own monsters, a fantasy that new Pokémon games and others like Temtem or Cassette Beasts try to recreate. These games have been successful, but I feel that they follow too closely the original formula created 30 years ago. Maybe, I thought, it would be possible to create something that captures this fantasy, but also updates some of the playable elements so that the audience of the original games (now adults and with different tastes) can feel again the novelty within the genre? It is possible to take the core of a monster-taming game and streamline some of its elements while innovating in others to create something new and exciting? I think that this core does not necessarily reside in the type-based combat or in the objective of collecting all monsters, but instead in the feeling that the world is huge and full of new, powerful and unique creatures.

Designing a designer

During the course, some of the games provided had basic systems for procedurally generating the levels. In Breakout each level was generated on the spot, and one of the requirements was to tweak the system to add a new type of block that needs a key powerup to be broken. And in Super Mario the level was created using different heights and populated with blocks and enemies, with only minor tweaks needed to make sure that the player is able to reach the end of the level. This allowed me to dabble with simple procedural generation and, when the Angry Birds class came, I decided to add procedural obstacle generation with increasing difficulty, and powerups after each level to scale the player power too (by the way, short showcases of my submissions for all these projects can be seen here: https://www.youtube.com/playlist?list=PLT_P8kp8Q65WqlJcYvmIawa6Je26I1Iv3). But even before this, I was thinking about what type of game would benefit from procedural generation. And then it hit me. What if procedural generation was the piece that monster-taming games are missing? It is even possible to create a system able to generate an infinite number of compelling creatures?

Procedural generation has been used to create different aspects of videogames: levels, terrain, entire worlds and galaxies, and even movement animations. But creature generation is something relatively rare and has proven to be tricky. The reason is that humans are good detecting design flaws in other living organisms, since during milions of years our survival has been dependant on that skill. This is similar to the effect known as uncanny valley, that consists on people rejecting robots with appearance too realistic to humans: when something looks like a person, our brain interprets the lack of facial expressions as something wrong with this 'person', but sees nothing wrong if the robot is clearly not human. A game with an humorous tone like spore can make wonky creatures work within its context, but even if they are more realistic the animal species from No Man's Sky were criticized by some players for not making much biological sense: different types of games mean different player expectations. And Source of Madness compensates the sometimes nonsensical nature of their procedurally generated monsters by setting the game in a lovecraftian universe, in which these physical incongruences make those creatures seem even more alien and otherworldly that if they were hand-crafted.

Between projects for the course, I started researching what makes an interesting creature design (based mainly on Pokémon pixelart sprites) to identify the comon elements and themes, and then creating prototypes for how a procedural creature simulator using these would work. For the first one I tried to create a tool that consisted of a 2.5D environment that would allow to add and rotate

basic shapes (cubes, spheres...) so that they form a base skeleton that could host more parts. This proved too complex, specially due to the problems of working with angles and my own inexperience with the engine, since it was around the second week of the course. The next prototype consisted of a pure 2D cell growth simulator that would periodically receive 'nutrients' and based on some parameters or 'genes' would decide how much to grow and if it should divide or create a new organ, with new cells receiving part of the nutrients of the parent cell and making their own decissions. Although this system generated some cool-looking organs like horns and antennae, the monsters themselves had weird proportions, and the nutrient approach resulted again too complex to materialize.

However, the third prototype was a success. For it, I wanted to simplify everything and rely on a small number of simple sizes. The main inspiration was the visual style from the South Park TV series, since it has a small number of basic sizes and a 2D, mainly frontal perspective. I used the hierarchical cell system from the previous prototype, but streamlined some elements: instead of growing in real time the cells had their size and coordinates relative to the parent fixed. Also, the cell shape was changed from 2 semiellipses separated by a trapeze to just an ellipse. Since cells usually share color with the parent, I added shadows to create an illusion of separation between them. For the snout, two symmetrical Bézier curves made it seem that the monster had a facial expression. But what tied everything together was a reduction of rendering resolution, initially an experiment to make the sprites more similar to the SNES/GBA aesthetic but which also resulted in the monsters feeling more alive and expressive, since arbitrary artifacts generated due to the resolution reduction looked like monster expressions. These sprite designs were, together with the course requirements and other features, implemented in the Pokémon class replacing the default monsters in the original code. And with technical improvements like using canvas for optimization and allowing changes in the sprite scale without modifying its resolution, and with some additions like changing the color of the bodypart when assigning an element, it is basically the same system used for Elemental Monsters.

Going rogue

In collectible card games (CCG) like Magic: The Gathering, players build their deck using cards from their collection and face other players using their own decks. To create a good deck, they have to analize all the cards, identify possible interactions that make these cards work better (or worse) together, and then design a strategy that aims to use these interactions to create value and wins the game. I feel that games in general are similar to these decks, a complex net of interactions in which each card (in this case, each element of the game) has a role in creating the designed experience. Because of this, it we were to add procedurally generated monsters to a monster-taming game but not change any other element it would be catastrophic for the final result, since it would be like swapping the most important card of your deck for other very different card. Instead, for this change to work we would need to identify the role that the hand-crafted monsters fulfill, think of the potential consequences that the change would have, and then alter change other elements (remove the ones that cannot work without hand-crafted monsters, and add others that could not work with them) so that the end result feels cohesive. The key questions here are:

1) What do we lose with the change? First, we lose the collectible aspect, since it makes sense to want to capture every creature if there are 100-200, but not if there is an infinite number of them. Also, we probably lose a bit of identity, since it is inevitable that hand-crafted sprites are more unique that procedural sprites (although we can try to close this gap), which also means losing the possibility of having a 'mascot', since everyone would be playing with different monsters. Lastly, we lose some mechanical uniqueness too, since some monsters are defined by particular characteristics (for example, the pokémon Ditto is the only one that can transform into its opponent), and while this doesn't mean that we couldn't add rare abilities to procedural monsters they are not going to be exclusive to them.

2) What do we win with the change? The obvious advantage of procedurally generating creatures is variety, since each monster is different from the rest (however, it is important that there is enough variance for the monster to not only be different, but feel different). Also, the sprite parameters can grow constantly instead of suddenly like evolutions or transformations, and be altered at any moment (change color, size, shape or animation of the parts of the body), for example, in response to power increase, actions or player choices. And another advantage is that monsters could be fused and/or mate, creating offspring that inherit traits from each progenitor, imitating real life genetics.

It seems obvious that this change creates some conflict with other RPG elements. These games require the player to use the same group of units during many hours and form emotional bonds with them by overcoming obstacles together, so the extra variety is not put to use (also, having less charismatic monsters is a problem when you use the same 3-6 all the time). Being able to change the development of the monster is nice, but if changes are permanent players are missing many other possibilities, while if they are temporal monsters lose even more identity. Having many partners at the same time is also a problem, because they may feel similar if developed in similar ways. And of course, making an RPG means spending more resources in story and setting, which means less focus in the mechanical aspect. All this means that this new mechanic needs a different genre, one that makes the player constantly change his monsters, or to see the same monster grow several times and transform into different possible final forms. Should this new game be a roguelike?

Roguelikes are a type of game based mainly on 2 characteristics: permadeath (when you lose, your character dies and you must start over) and procedural level generation, but this formula can be adapted to be more forgiving to the player letting him keep some progress between sessions or 'runs', named as roguelite: some games make the character stronger after each run, and others unlock new classes and abilites. The most important thing here is that roguelikes rely on resetting progress and starting over (exactly what this game would need) but require a way to create infinite content to justify these replays (which we would already have, in the form of the procedural monster generator). Roguelikes also tend to rely much more on mechanics and less on aesthetics or story (with some exceptions like Hades) which means that we could get away with not using many resources in these areas. And while there could be doubts about if a monster-taming game with almost no story would be interesting for the players, there are living proof of that: fan-made Pokémon roguelike games already exist, with some popular examples being Pokémon Emerald Rogue and PokéRogue.

With all this information, I decided to make my first ever game a prototype of a roguelike about fights between procedural monsters.

Developing the idea

The design started on April 4. During the first brainstorm sessions, I stablished some key aspects that this game would have:

1) A combat system in which the monster figths autonomously: I always felt that the combat in Pokémon games don't mach very well the theme, because if seems more like you are the one who is performing the moves, not the monster. In the Battle Palace combats from Emerald, pokémon chose movements based on their nature, and while this was a bit random I think there is potential in this idea and that monsters could feel more alive if they make their own choices. Also, this could allow me to introduce some concepts for AI and action programming.
2) Instead of a full team, the player would only have 1 monster at a time: since runs in roguelike games typically last 30-45 minutes, having a team of different monsters would dilute the attention that each one gets, so is better if only 1 monster is used during the run.

Also, in other games from the genre players make decissions that shape the experience (like choosing 1 of 3 powerups or cards to add to your deck) and because of this they get attached to the combination of decissions they have made. This will work better if all these decissions are about the same creature.

3) For the run itself, the player would progress through a system of nodes and bonds that regulate if the player has a normal combat encounter, a miniboss combat, a treasure, a store to buy powerups... similar to other games like Slay The Spire or Inscryption. This means that the system could be imported with a relatively low number of changes.

4) An improved creature generation system, with a pool of different monster parts that could be attached during the monster creations (things like wings, horns, cannons, flames...), but also could be added during the game runs depending on the player choices. This means a lot of work, but it is necessary for the game to be engaging in the long-term. For a prototype, only a small sample of parts would be enough.

The first days with the engine consisted on building the bases of the combat system. I opted for a system in which the monster gains different approaches or thoughts depending on the battlestate, and the player could influence with their orders which approach gain, directly influencing the monster decissions. If the monsters make most of the decissions with the players giving orders sparingly, there had to be an element that adds dynamism, so I decided to make a real-time combat with a Stamina bar system inspired by games like Dark Souls and Clash Royale (yes, inspiration can come from very different sources). And to allow players to plan strategies to guide the monster behavior, the movements are chosen from a list in which the player can change the movements and their order outside of combat, which forces the player to combine movements that work well together inspired by action programming games like Opus Magnum.

However, when designing the rest of the elements of the game, I realized that two mechanics would be more important for the player experience but also more resource consuming than I thought at the beginning:

1) It made a lot of sense to change the run structure to a region that you have to explore (like in Pokémon games), with cities and routes being nodes and bonds respectively. This was initially just a cosmetic swap, but the more that I thought about it the more I started seeing the possibilities of this change: what if the map behind the nodes is generated procedurally, to feel that you are exploring a different region each run? And what if the nodes are actually put into the map depending on the geography, and different rutes require items like a boat ticket for sea rutes? (a prototype of the map generator was created using layers of simplex noise and segmented coloring, you can see a demonstration here: https://youtu.be/Ok30IuWb3ts). And what if you can backtrack, and find secrets, fight mythic monsters... Eventually I realised that something really unique could come from this, but I knew I couldn't do justice for it in this prototype.

2) In other similar games you can breed different species of monsters, but there are not many incentives to do it because the offspring are identical to their parents (with maybe some minor stat or movement changes) and start at low level, so cannot go to your party without extensive training. Here the first problem wouldn't matter, since as I mentioned before procedural monsters can be fused and by choosing which new monsters to keep the player is selecting traits that wants to preserve, which has been done in real life with animal and plant species and could be used to fulfill the player fantasy of creating their own monsters. And the second problem also doesn't matter, since the monster would increase in power or level up during the run and then sent back to the starting state, making a new run with untrained monsters equally viable. The system I designed would allow players to spend a currency aquired during runs to capture monsters or create offspring of the ones you already have, which would be the de-facto system for the player to increase the power during runs (since the player would progress from playing with monsters with random traits to having highly

optimized combat machines, with perfect stats and synergistic traits). Usually roguelikes with power increases over time feel cheap because the player knows that they cannot win the run until they get most of the upgrades, but this system would be more subtle and have the benefits of increasing the player power over time without those downsides. Again, this system needed more time I could spare.

For this game my priority was finishing the prototype, since those and other ideas could materialize later, but the more stuff in the game the lesser the chance I would even be able to finish, taking into account that I have no previous game dev experience and my programming skills are inferior to most other devs. This meant that those both systems needed to go, and be replaced with a simple arcade-style game: you need to fight 10 monsters to win, and if you lose you start over with a different monster, which also means no save system and a lot less UI components to create. Another problem was that LÖVE 2D doesn't have a view mode in which you can put elements in the screen, but instead requires that the developer fully programs every aspect of the game. This is very good for learning because you have to think how to solve every problem with only code, but when every UI element must be created manually, and placed in the corresponding coordinates, and then tools like a scrolling bar or dragging windows need to be implemented too... the menus costed much more time that originally planned. And there are other parts that needed their time too, like drawing the icons or creating the sounds, which took more time than planned.

This meant that other desired features needed to be removed, like the upgrade to the monster generation algorithm and the additional parts that would be added with powerups (the old system was used, and for the powerups only the ones that could be represented as simple color changes in the sprite were kept, hence the Elemental Monsters name). Even with those changes, when the 1 month deadline came the game was nowhere to be completed, so I gave me 2 more weeks (which eventually needed to be almost 4).

The 10-minute RPG

But before all the menu and UI elements were created, I needed to decide what exactly is a monster (how many parts, stats and movements it has). For the parts, the system was initially going to be very flexible and support part addition and removal, but due to the simplification of the progression during the run I decided to have a fixed number of 5 parts or organs: head, body, arms, legs and tail. Each part would have 4 different stats associated with different aspects of the game: size (damage), speed (performing movements in sequence), intelligence (negative status effects) and metabolism (positive status effects), with the first drafts making more biological sense (the body would have more size and the head more intelligence, for example) but eventually changed so that any stat could be high or low in any bodypart to create variability. Also, for each stat the power in all organs is added and used to modify combat parameters: size for HP, speed for the movement charge time, intelligence for the approach gain rate and metabolism for the organ cooldown.

Then, after several designs the basic movements (present in all monsters without any special traits) were created. Each part would have 3 movements, 1 for each type (buffs for temporal stat increases, attacks for damage, utility for disruption). From these, 2 would scale with a pair of 2 stats, so that every stat in the part is used for exactly 1 movement. The third movement would scale with the stats of every organ, so every organ has at least 1 usable element even if all the stats for that organ are low. Every movement should have at least a slight shared identity with the others of the same organ (autonomous behavior for head, reckless attacks for body, movement combinations for arms, attack and defend faster for legs, status effects for tail), but to force players to not use only 1 organ, each movement triggers a cooldown for its organ, during which movements from that organ cannot be used.

After implementing the system for gaining approaches (depending on game state) I realized that sometimes monsters spent a lot of time without using any movement, and some approaches

(offensive) were more useful overall. I reworked that system so that the monster gains the approaches that they need for using the first movement of their list of equipped movements that is available (that is, its organ is not in cooldown), and now the monsters were always trying to perform a different movement between the 2-3 first on the list. Other changes for the system were fixing the autonomous movements to 3 maximum, but letting players equip up to 3 additional movements that would only be triggered by its orders (since some movements are more useful if activated in response to enemy actions) and making the defensive approach only be obtained by the monster if the enemy was preparing an attack, to make defensive movements like block or dodge trigger in response to enemy attacks. And since the personality of the monster wouldn't work well with this new system, I reworked it into personality traits that give the monster a specific approach when using a movement of a particular type, which each monster having 2 out of the 15 possible traits.

It was necessary to have an algorithm that identifies which set of 3 movements are better suited for a particular monster, since not only enemies would need to have movements that get the most use of its characteristics, but also players would need a starting list that already works to change and customize once they are prepared. This algorithm calculates the power of each movement (specifically, for the stats that the movement uses to scale its effects) and then applies some modifications to these values after each picked movement, so that for example movements that buff a stat pair with others that use that stat, or making at least 1 attack mandatory, and not allowing to repeat organ to always have at least 1 available movement. The personality is also taken into account, with movements that trigger the personality traits having increased chance of being picked, and the next movement having a higher chance of use the approaches gained, and reactive moves being forbidden if personality gives defensive approaches to not trigger these moves in wrong moments.

The element system works as follows: after some fights (2, 5, 8) the player can choose 1 of 2 from a total of 8 possible elements to add to one of the organs. These elements give the monster a passive that gain counters, which trigger effects in specific conditions (for example, after attacking or being attacked). 4 of the elements are associated with 1 stat, and the other 4 to 2-stat combinations. These elements give the organ a boost for these stats, and the rate of gaining counters scales with the powers of these stats in the organ in which the element is put. Also, each element gives 3 new, element-specific movements (again 1 buff, 1 attack and 1 utility) for the organ that has the element equipped. The monster can use these movements (which also scale with the organ stats associated with the element) but also trigger the cooldown of the organ, so they compete with the basic movements of that organ.

This was designed so that when monsters get elements they always get new powers, even without player intervention or changes in the current strategy that the player is following (for example, a monster with the poison element passively spends charges to poison the enemy and gives a boost to the intelligence stat of that organ), but if players want they can interact more with this element, for example putting it in an organ with an already high power for the corresponding stat (boosting the charge gain rate) or swapping some basic movements with the new ones, that are specifically designed to both increase the passive effects of the element itself, and to create synergies between them. These synergies are hard to pull off because these movemens share organ and thus have a shared cooldown, but if the same element is picked twice in differen organs these movements can be chained. There are also reasons to stack elements in the same organ, since two elements that share a stat can boost themselves and increase the charge gain rate of the other. Of course, changing the organ stats means also changing the power of the basic movements, which in turn can result in reasons to change a basic movement for a different one. And there are reasons for particular element combinations, like plant being able to raise the charge cap for all elements or ice and rock stacking percentual and fixed damage reductions. The idea was to present the players interesting choices when picking element-organ combinations, but with all this complexity hidden until they have learnt how all these systems interact: even if they don't fully understand the implications of this

choice, they should be excited to just add fire or electricity to their monster.

And lastly, the overall aesthetic for the game needed to be decided. I opted for a retro 8-bit style for many reasons:

1) It fits the style of the older pokémon generations (1 and 2), and helps to create a feeling of nostalgia that is otherwise hard to capture.
2) Having other style would clash with the sprites, since the current sprite generator is designed to create low-detail pixelart.
3) Its minimalism allows me to get away with things like most of the game being in black and white, which is used to highlight the most important part of the game, the monsters themselves and the elements.
4) It is very simple and easier to create than other styles. In fact, since everything must be manually coded it is possible that this is the only style viable at all for this project. In the NES era this style was used due to technical limitations, while here I had a workforce limitation instead.

The color palette chosen is made of the original 55 colors that the NES system was able to display, but sometimes this rule needed to be broken to allow for some of the systems to work as intended. A 56th color was added (a different shade of grey) to be able to highlight the grey from the 'tactical' monster approach icon. And while the monsters themselves had their basic color set to a somewhat neutral color within these 55, the shadows are created with semitransparent layers of black shapes, which produces colors that should't be possible in the original hardware. Of course, when a bodypart changes color because of the element, this new color is calculated averaging (with weights) the original monster color with any element in that organ, which again breaks the original palette. However, even with these changes I feel the intended aesthetics are relatively well represented.

Lastly, the sound and musical themes were created using the trial version of FL Studio, particularly the plugin Magical 8bit Plug 2. This plugin allows the user to create sounds reminiscent of the 8bit era by modifying the sondwave shape type and parameters like attack time, delay, oscilation... and while I needed to spend some days to learn the program and the basics of composition, I had a lot of fun creating all the sounds and music, and the game feels more personal than if it had stock audio.

Reaching the goal

On May 26, the game was uploaded to GitHub, and 4 testers were invited to play the game. Usually videogames need many more testers, but since this was only a prototype I just convinced some friends to play it. In general the response was positive, particularly towards the monster design and the overall aesthetics. However, the combat system seemed to be very confusing, since too many new systems were introduced at once and the tutorial cannot explain everything needed (some players even ignored it!) and I decided not to include some UI elements that may have helped because I was running out of time. Also, too many status effects are displayed at the same time in the status bar, which overloads the player's attention. Overall, I feel that for the combat I again changed too many things at once, leaving players confused, and this system would need to be reworked if this project is revisited in the future.

It must also be adressed that the game does not work perfectly. Particularly, there are some problems with the movement asignation during combat that couldn't be solved (sometimes movements should be elegible but aren't when giving instructions), but it didn't seem to matter since players relied on letting the monster fight for itself and not giving it instructions. It was decided not to fix this bug because after some time I couldn't find its source, and the focus during bug fixing went instead to prevent crashes. This proved a good decission, since fortunately no players experienced any crash during play.

In conclusion, this project represents almost 2 months of hard work (maybe more, if one would count the time spent with the procedural sprite generatior during previous projects). Even if is not the perfect game and a lot of features needed to be removed, it still meets the objective of creating a unique experience, even if a bit unaccessible. In retrospective, creating an innovative, real-time battle system based on stats that can change at any moment was a bit out of reach for a first-time developer, but this also served for getting me out of the confort zone and learning at giant steps, particularly to build code with scalability in mind.

If you have played Elemental Monsters, I hope that you had fun doing it. And if you read all this (more than 9 pages!) text, I hope that you learnt something useful.