
# Cardestia

*Split Chance*

Thomas West

Table of Contents

# Overview

**Team:** Split Chance (Rachel Appleyard, Kieron Duffy, Cian Halpin, Luna Henry, Thomas West)

**Platform:** PC

**Genre:** Strategy

**Store:** thomasthewest.itch.io/cardestia

**Find out more:** splitchance.com

**Description:**

*Cardestia* is a multiplayer strategy card game wherein two players battle head-to-head to restore balance to the heavens. With their heavenly cohort, players use their cards to attack, defend, and manoeuvre against their opponent.

# Inspirations

Apart from my other primary roles in Split Chance, I had a writing support role. Here, I provided some artistic support for the art team. Below are some of my artistic inspirations I used in my work.

## *Mythological and Biblical*

As part of our art team's celestial aesthetic, I sought out cards that evoked those themes, mostly drawing on Abrahamic angels, but also from other religious traditions, ranging from Hellenic to Nordic.

I chose figures that corresponded roughly to the heavenly/celestial atmosphere our art team designed. *Cardestia* is a game about power and battle, so mythical figures such as Brunhilde or the Ancient Roman Furies made for fitting additions, along with the plethora of angels from Abrahamic tradition.

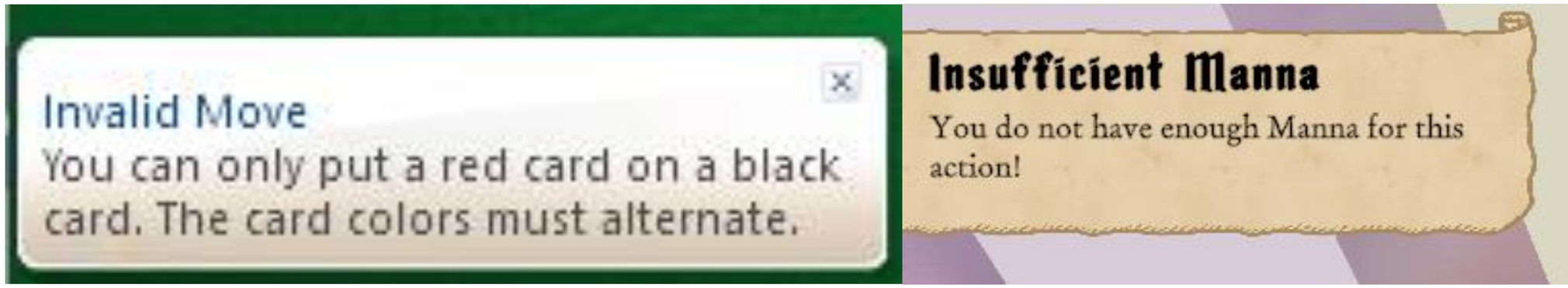

*Promotional graphic featuring many of the cards named by me, designed and illustrated by our art team.*

To further solidify the biblical link, I used the Old Testament spelling of Manna (the player's main resource in *Cardestia*), often otherwise spelled as Mana in other fantasy games.

# *Microsoft Desktop Games*

An unusual and unassuming inspiration that drove a lot of my UX ideas for *Cardestia* were some of Microsoft's desktop games, most notably *Freecell* and *Chess Titans*.

Each of these games had very simple, but very effective, considerations for user experiences. As our game featured a lot of the same player interactions and interfaces, I borrowed a lot as I helped inform and develop our game's UI. *Freecell* gave me a lot of pointers for the card elements of the game, while *Chess Titans* had a lot of pointers for how *Cardestia*'s spaces should behave. I also pushed for *Cardestia*'s free camera view to be more like *Chess Titan*'s, noting how intuitive Microsoft made it.




*Microsoft* Freecell *features pop-up messages that tell players when they've made an erroneous move (left), informing my design of similar popups in* Cardestia (right).

*The simple space highlighting in* Chess Titans (left) *inspired my design of tile space visual effects in* Cardestia (right).

# Laputa: Castle in the Sky

Hayao Miyazaki's *Laputa: Castle in the Sky* also served as a minor visual inspiration for one of my concepts. As our art team pushed for their own 'castle in the sky' look, I took some visual inspiration from this animated feature film for my own concepts (see *Narrative Design* below). *Laputa* was also mentioned by the art team as they expanded their ideas of how the game should look.


*Miyazaki's eponymous Laputa (left) and my concept art of a floating castle (right)*

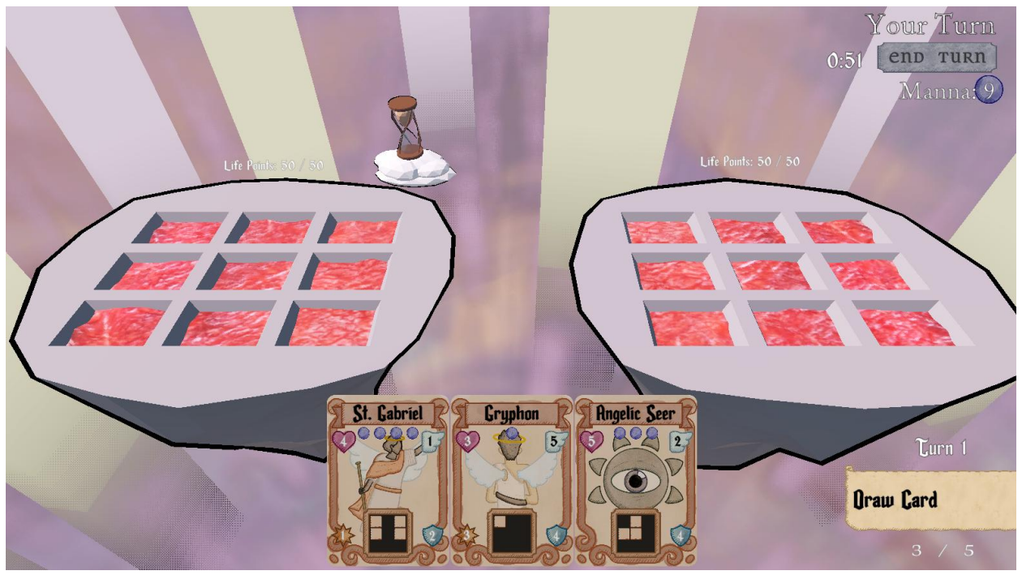
# The Development of Cardestia

My two main roles on the project were producer and technical artist. I oversaw and directed development, while also working on asset and UI implementation.

## *The Concept*

*Cardestia* is a is a multiplayer strategy card game wherein two players battle head-to-head to restore balance to the heavens. With their heavenly cohort, players use their cards to attack, defend, and manoeuvre against their opponent.

The objective of the game is for players to attack the other until they lose all of their Life Points (LP). Each player has their own floating island from where they launch their attacks. The player's island has a 3x3 grid on which to place cards.
The cards themselves each display a pattern of spaces that they perform actions on, either defending spaces on their own board or attacking spaces on their enemy's board. This pattern can be rotated to line up blocks or attacks to synergise with each other. Some cards can also move across the board, either to soak up incoming damage on a space, or to avoid it.



# *Prototyping*

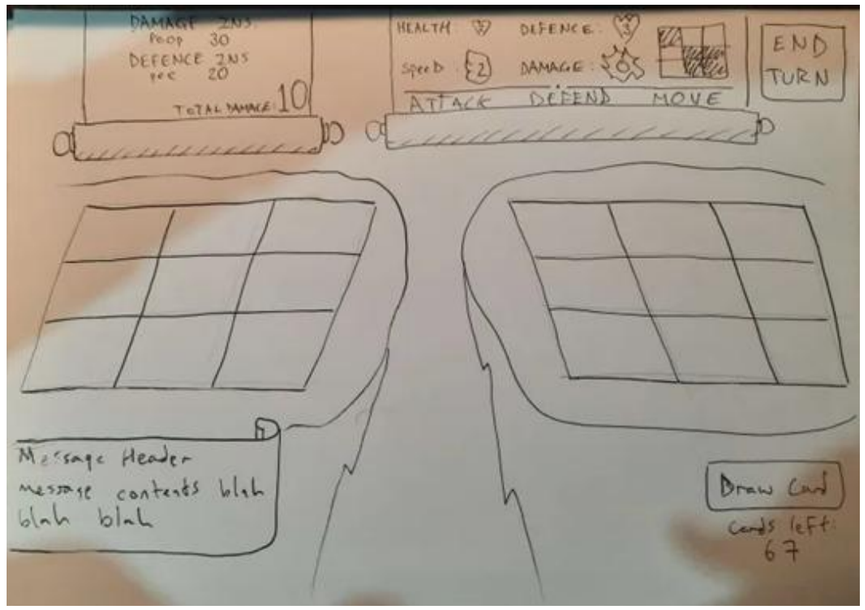
Development for the game was split into two halves: pre-production and full-scale production. Each half was primarily concerned with a different game, with pre-production spent working on a prototype codenamed C&B and our full-scale production working on a reworked design codenamed T&B, later rebranded as *Cardestia*.

C&B's development was largely ad hoc, with it ending up that we worked quickly and identified problems rather than solving them. While there were major feature changes between the two, both games shared the same basic format and took a lot of we learned a lot from our pre-production experiments with C&B into *Cardestia*'s development.

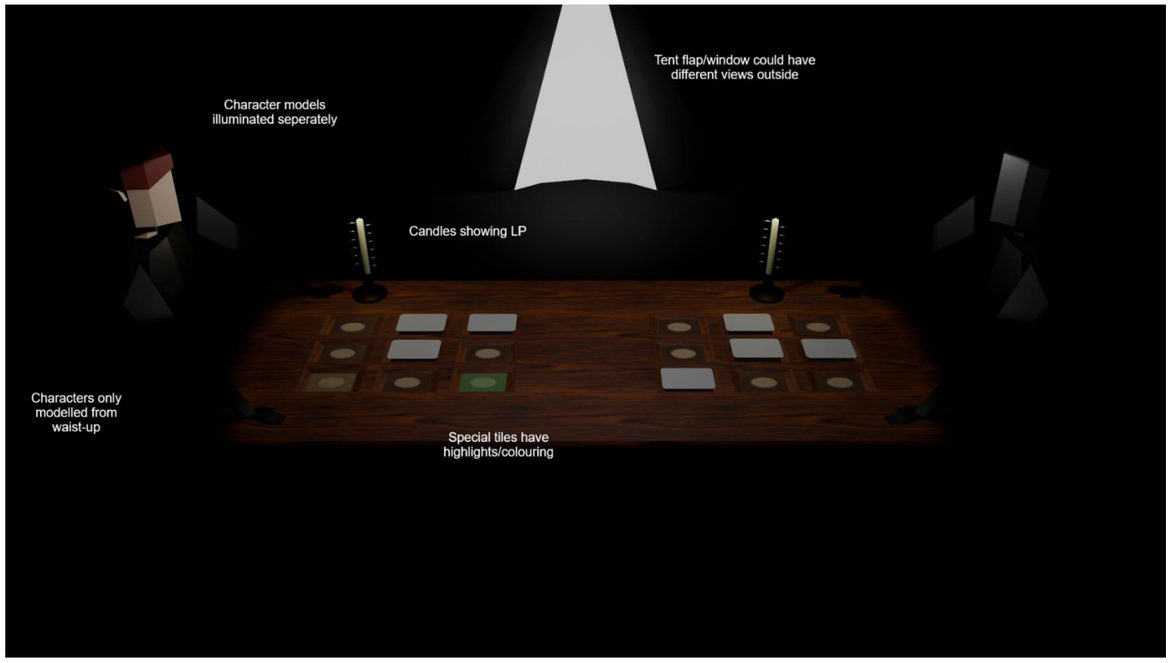
Coming off the heels of C&B's development, I made many adjustments to our development style, pushing for a more iterative Agile workflow. The development for C&B didn't have as much playtesting, and by the time we started putting it in front of players they didn't understand it or find it fun.
Therefore, for T&B we decided to adopt our Agile approach, iterating more on features and performing more playtests to identify and address player frustrations. I prioritised making regular builds and tests all throughout development, making sure new features were proven and tested before moving forward with expanding the game and features. This created a more stable user experience, where we began to understand the game better.

Starting development on T&B in late January, we delivered our first public Alpha build in March, before our first Beta builds were released under the *Cardestia* branding in April.

Apart from digital gameplay prototypes, we also created plenty of paper prototypes and 3D proofs-of-concept.
Designers worked with board game prototypes when exploring new gameplay ideas, and I would create sketches of potential UI mock-ups as I was often involved in UI implementation as technical artist. These mock-ups helped communicate ideas to the art and programming teams of what features and assets would be required, and give us a stronger idea moving forward of how the game would feel.




*An early sketch of mine of how the game's UI elements might be arranged, as well as providing a thematic concept for the paper scrolls elements would be placed on.*




*Before* Cardestia*'s more celestial fantasy style was fully realised, I made some 3D mock-ups in Blender of a low fantasy look for the game.*

# *Production*

As producer, it was my responsibility to organize the studio's manpower and priorities. This included not just managing developer responsibilities and tasks, but also maintaining live documentation to keep development on track.

My main priorities as producer were to reduce risk. Since C&B's development phase wasn't successful in providing a viable prototype, we needed clearer goals going into T&B's development:

- Our previous experience in building C&B in the Godot game engine made us realise we didn't have enough domain expertise for all team members to contribute to the best of their abilities, so I pivoted to getting the team on board with making T&B in Unity.
- To cut down on programming overheads, we would make the game multiplayer, taking pressure off us to design an intelligent agent for a single player to play against. The format is inherently symmetrical as a zero-sum game (i.e. the loser has as much to lose as the winner has to win), where each player has the same abilities. It lends itself naturally to a two-player format. While it was still an involved process to make a networked game, it would be less involved than trying to program an intelligent agent, freeing up space within our tight schedule to have a functional game.
- Apart from the gameplay, artists on the team also talked about exploring new aesthetics for the game, expressing a lack of passion for C&B's more grounded fantasy art style.

## Documentation

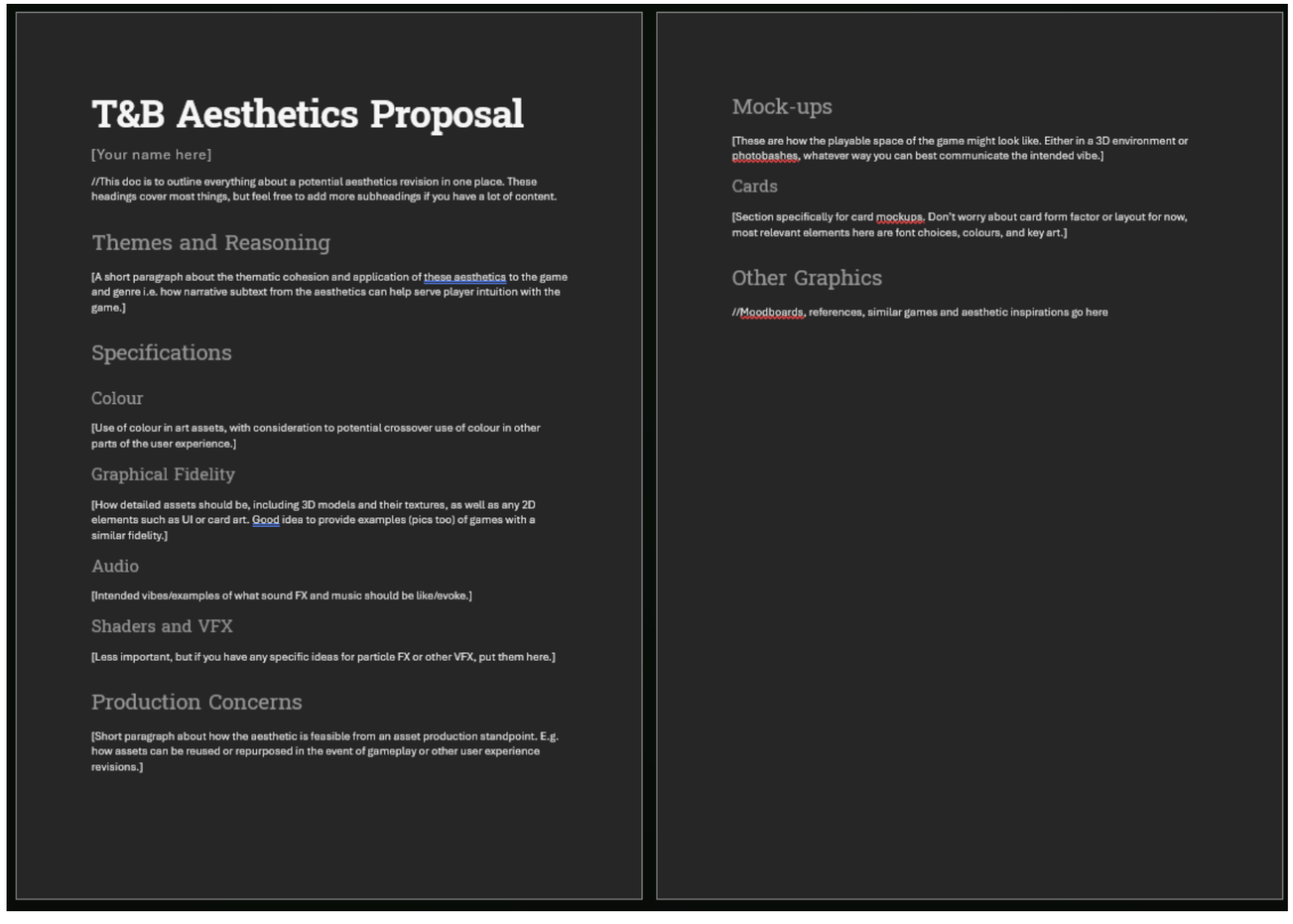
As a reasonably complicated game, our studio needed robust and concise documentation. Across several aspects of development we needed to cover all manner of design, art, production, and development documents. The document formats used all belonged to the Microsoft Office suite, while for live collaboration, we relied on OneDrive.

During our prototyping phase, the team tried maintaining a large design document, including all aspects of the game's development. However, it gradually became too bulky and hard to maintain, so as we moved into T&B's development, I divided the single document into different ones to better focus developers' attention to relevant information.

The game's broad design was moved to a 'rulebook' document, outlining the rules as if it was a board game (which many early prototypes were). We were far less likely to rely on outdated information, and this was better suited to our new Agile development style.

Other design documentation included a developer dictionary, presenting a glossary of game terms. Considering how much the game was evolving, this was useful to make sure the features we discussed and developed were exactly what we were describing; if we agreed to change a term it was updated in the developer dictionary.
As we were making many changes to the game's rules in the course of development, having a central source of definitions kept our documentation robust and modular.

The game's art style was also in flux as we moved away from our earlier prototypes, so I prepared a template document for art developers to articulate their wishes and inspirations.



T&B Aesthetics Proposal

[Your name here]

//This doc is to outline everything about a potential aesthetics revision in one place. These headings cover most things, but feel free to add more subheadings if you have a lot of content.

Themes and Reasoning

[A short paragraph about the thematic cohesion and application of these aesthetics to the game and genre i.e. how narrative subtext from the aesthetics can help serve player intuition with the game.]

Specifications

Colour

[Use of colour in art assets, with consideration to potential crossover use of colour in other parts of the user experience.]

Graphical Fidelity

[How detailed assets should be, including 3D models and their textures, as well as any 2D elements such as UI or card art. Good idea to provide examples (pics too) of games with a similar fidelity.]

Audio

[Intended vibes/examples of what sound FX and music should be like/evoke.]

Shaders and VFX

[Less important, but if you have any specific ideas for particle FX or other VFX, put them here.]

Production Concerns

[Short paragraph about how the aesthetic is feasible from an asset production standpoint. E.g. how assets can be reused or repurposed in the event of gameplay or other user experience revisions.]

Mock-ups

[These are how the playable space of the game might look like. Either in a 3D environment or photobashes, whatever way you can best communicate the intended vibe.]

Cards

[Section specifically for card mockups. Don't worry about card form factor or layout for now, most relevant elements here are font choices, colours, and key art.]

Other Graphics

//Moodboards, references, similar games and aesthetic inspirations go here

There was a conflict between the necessity of an established art style early in the project and permitting the art team to express themselves. There was a risk that allowing the art team to continue to make changes to the style throughout the project could cause significant delay, but on the other hand locking the art style in too early would restrict creativity and might cause the work to become stale and artists to become dispassionated. My strategy was to ensure that the artists bought into the development cycle as they progressed their work.

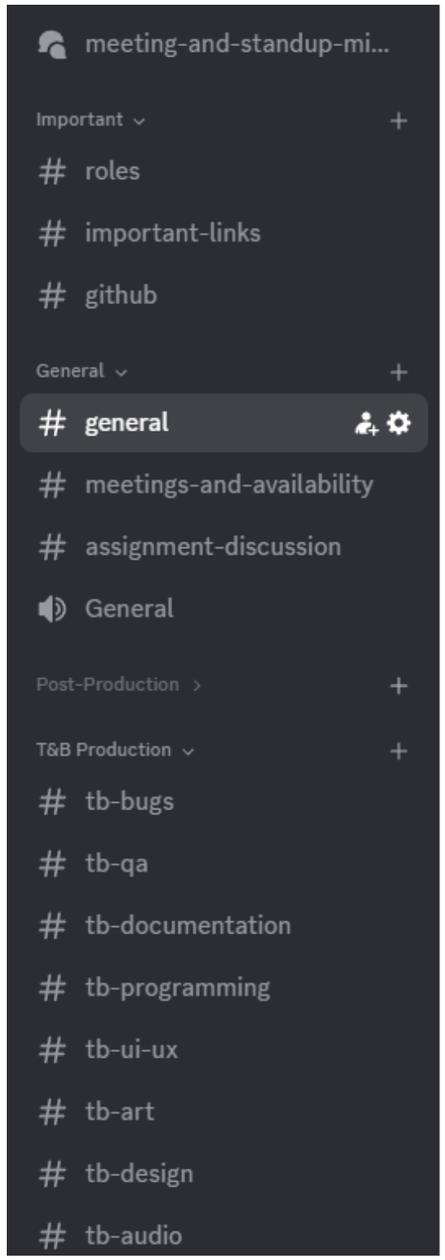
My template had headers and descriptions of what to explain about the art style, including prompts for how artists intended to make the art style sustainable for the entire game (i.e. whether they could produce all of the required assets within the development period). While posited as a proposal template, it would also generally get artists brainstorming about their art style, as a proto art bible.

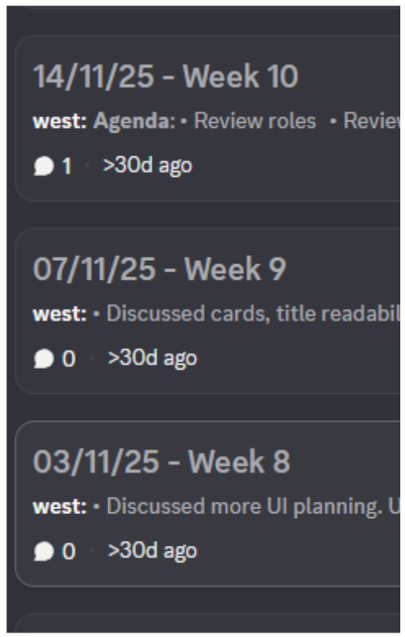
### Communication

I originally wanted to use professional communication tools such as Microsoft Teams, but we instead decided to work via Discord. While not built for developers, Discord came packed with lots of features that made organising text- and voice-based communication easy. The convenience of a tool of which the entire team were familiar with outweighed any features we missed out from professional tools, especially for a small project with few active participants.

This included separate text channels for focused conversations on different aspects of development (Discord's per-channel mute features also made it easy for developers to filter messages not relevant for themselves), and forum features were used to keep meeting minutes. Discord also came with built-in event tools, keeping developers in the loop of meetings, playtests and other meetups both online and in-person.
Important messages and notices could be pinned, letting developers access important tools and information in a single location.

I designed the format for the Discord server's channels, as well as logging events in the server. I also set up update notifications from our GitHub repository to keep developers abreast of the most recent pushes.
I continued to manage the Discord server throughout the entire development cycle, creating new channels to manage links to important documentation and processes.



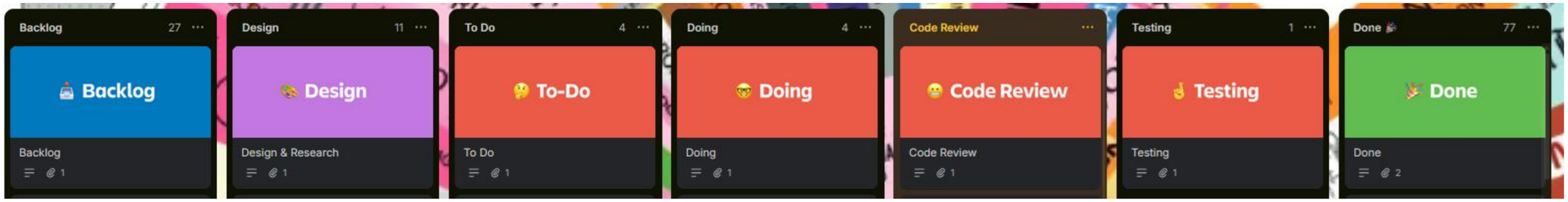

I organised weekly sprint meetings on Mondays to decide tasks and address other development concerns and check in with developers. These Mondays were spent in-person, allowing for collaborative development after we had finished our meeting, keeping the entire team in the loop as to the status and crossover of active tasks.

## Task Tracking

For Agile task tracking, I set up Trello boards. The first was for C&B's development, and the second one made as a clean slate for when we began work on T&B.

As part of an iterative Agile development environment, our board was set up with separate lists that developers would move items into. My intention throughout development was to move product backlog items (PBIs) from the left to the right through sprints and iterative testing. While not strictly a traditional Scrum-styled structure, as producer I did take on a taskmaster role, keeping on top of which developers were doing what and assigning PBIs and leading sprint meetings.

PBIs would have details of a task written in them, sometimes providing a link to a previous conversation or concept that we discussed on Discord. This helped keep a consistent record of what was expected of the feature or asset the PBI described.




- **Backlog:** The product backlog held items we'd have to do at some point, but wasn't something anyone needed to be doing currently i.e. feasible ideas that we talked about, but needed to perform more development and playtests to determine when they could be added as tasks to a sprint. This ensured that we could be certain that important elements would not be forgotten, but that they would not bog down the work with which we were currently engaged.
  I reordered the backlog each week, with more pressing PBIs that we were confident we would implement soon sorted towards the top. These were the PBIs we felt would offer the most value to players.
  As we learned more about the game and adjusted our priorities, I might edit PBI details or remove one altogether. PBIs closer to the bottom of the backlog might have had fewer specific details, and once we started working down the list I would make them more specific or even break them up into smaller PBIs that could be tacked within a sprint.

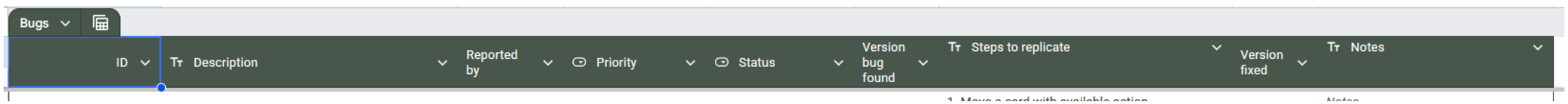
- **Design**: Design items weren't specific tasks, but questions and ongoing concerns. These design questions would be added and we would update and consult them as we looked at what to do each sprint. Items here might have ranged from creating a new art style or creating game design solutions to problems that were cropping up in development i.e. things that didn't have a specific task associated with them.
  We would try to go into sprints to tackle the subject of one of these open design questions, as a guide to help us find out more about our game and figure out what PBIs to tackle first.
- **To-Do**: This was where the week's sprint's tasks were listed. Based on open Design items during our sprint meetings, I would pull PBIs from the backlog and ask developers whether they could finish them in the sprint. Once we had agreed on the PBIs we would implement, they were added to the To-Do list and developers assigned to tasks.
  I would also add time limits to the tasks, keeping developers on track to complete them before the sprint was over. This gave clear ownership for each task, making team responsibilities clear for everyone.
- **Doing**: Once a developer had begun a task from To-Do, they would move it into the Doing list. This let me see what people were working on, but also to see during a sprint meeting what developers might have started and have been unable to finish, letting me identify bottlenecks and adjust where manpower was being allocated for the next sprint. It gave all parties direct feedback as to whether the scheduling was over-optimistic (or over-pessimistic if the To-Do and Doing lists were empty well before the end of the sprint).
- **Code Review**: This was a quick 'sanity check' so that once a developer had finished a task, the item would be added here for a secondary (someone who had a secondary responsibility to a team role) or other available developer to confirm that the item's contents were indeed implemented where it was needed e.g. merged into our repository. This provided plain accountability as developers needed at least one other person to be in the loop on what work was present, and that the work could be found and discerned for later QA.
  Once the item and its assets/features were confirmed to be present, it would be moved along into our QA process.
- **Testing**: PBIs posted here were to be stress-tested by our QA team. After being verified that the content exists within the code repository or other location, QA would verify the integrity of the feature or asset. For assets, this might be making sure it renders correctly in-engine, or for features, make sure they do not fail during play. For some PBIs, this could be a very short process, or a complete feature test of gameplay.
- **Done**: Tasks that were completed and tested were marked as complete and added to the Done list.

Using Trello helped keep a written record of our tasks for developers. The intention was that, during a sprint, we would move the item through production and quality assurance and save us time on bug fixing and prevent us from implementing features that provided little value.

## Bug Tracking

As our prototypes began inheriting more technical complexity, playtests began running into more issues. With a lot of moving parts such as networking, asset implementation, and data storage, we needed a way to log issues related to game performance.

I created a Google Sheet where developers could log a bug per row, with several columns specifying different aspects of the bug.




- **ID**: Each bug had a three-digit numerical ID, starting from 001. This helped sort any topics on the bug alphabetically, and made for useful shorthand on describing certain bugs in conversation.
- **Description**: This was usually a sentence describing the nature of the bug in simple terms.
- **Reported By**: Whoever found and logged the bug would mark their name here, just for accountability. If someone patching the bug had any follow-up questions or wanted to test a fix, this person was their first port of call.
- **Priority**: This described the severity of the bug, with options available in a drop down menu so developers could quickly designate them. As the priority options started with numbers, the list could be sorted by this column, allowing the most pressing issues to filter to the top.
  - **0** – Absolutely critical, this bug was holding up all development.
  - **1** – Game breaking, gameplay could not continue after encountering the bug.
  - **2** – Major, ‘soft blocks’ or other major interruptions/stops to gameplay.
  - **3** – Minor, usually graphical hiccups or misaligned UI elements.
  - **4** – Fixed, the bug was fixed and the item could be ignored.
  - **5** – Couldn’t replicate, whoever attempted to patch the bug couldn’t find the issue. This was a useful designation in case the reporter found more specific instructions on how to replicate the issue and could reopen the item.

- **Status**: The current point in the patching process the bug was at. This helped me oversee any bottlenecks that could hold up other areas of development, especially in relation to more critical bugs.
  - **Fix not started**
  - **Fix in progress**
  - **Testing fix**
  - **Bug patched**
  - **Could not replicate**
- **Version bug found/fixed**: The columns were a simple way of tracking how old issues were to help prioritize bug fixing.
- **Steps to replicate**: Whoever found the bug would describe how they encountered it, helping bug fixers figure out how to diagnose the issue.
- **Notes**: Any other information would be put here.

While most of our documentation was done with Microsoft Officie suite, I decided to use Google Sheets as it was a familiar tool to me and the other developers, and at the point of development it was required, I wanted to reduce overheads finding a Microsoft-based tool to track bugs.

# *Asset Design & Implementation*

Apart from producing the game, I had a smaller, but involved, role as technical artist on the team. My job as tech artist involved implementing front end interfaces and bringing our artists' assets to life.

The following is a selection of various visual fixtures of the game I worked on.

## Hourglass

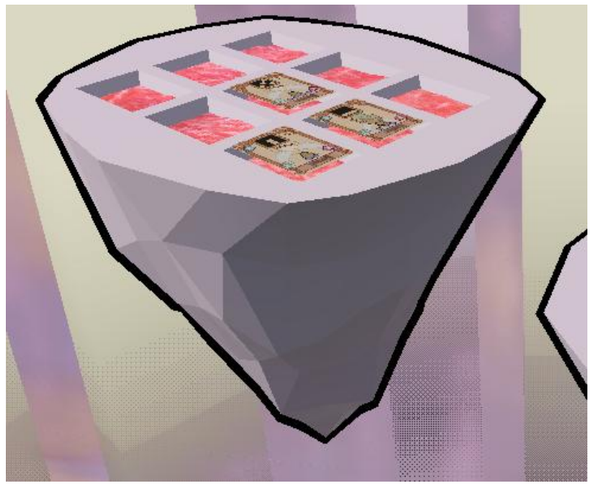
As a turn-based game, we wanted players to intuitively understand the game's format. While there was a text-based UI element with the game's turn timer, we wanted something more noticeable in the scene itself.

An hourglass was an obvious solution. Our art team made the hourglass mesh and I programmed it to jump between the two sides of the board while turning upside down when players ended their turn. By programmatically scaling its different sand segments (the top segment shrinking while the bottom one grows) in time with the turn timer, it creates a convincing illusion and apt visual metaphor.

## Island

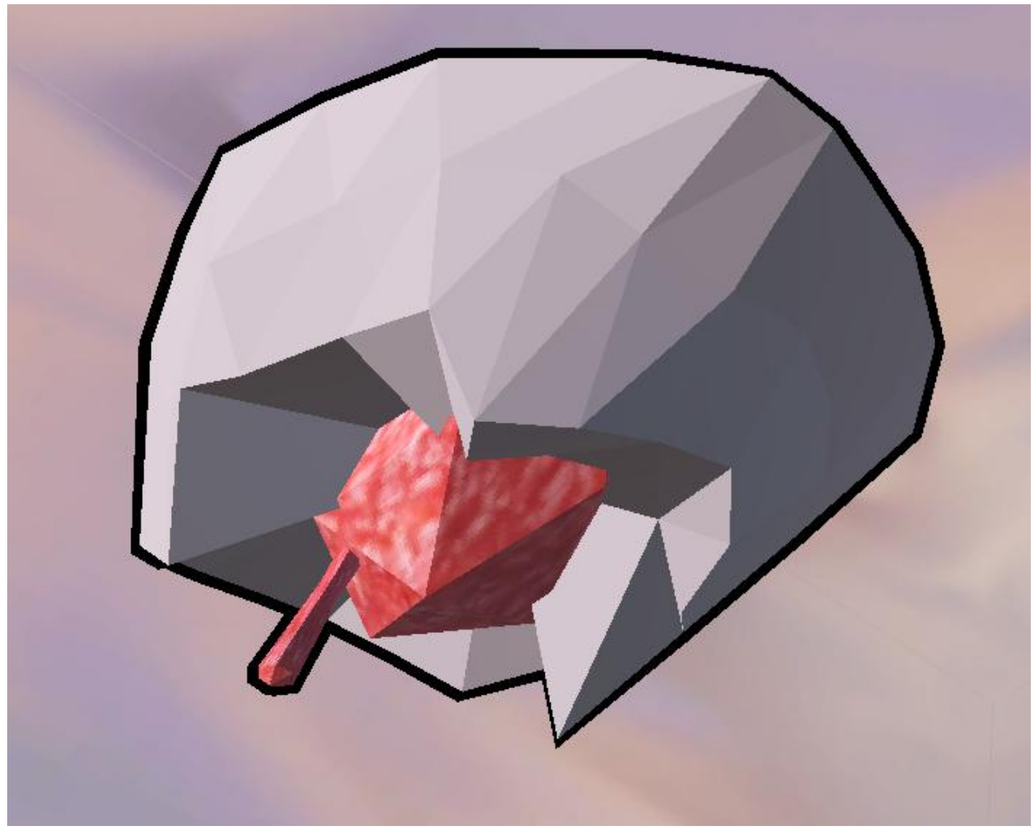
With gameplay happening on floating islands, we needed players to understand that enemy attacks to exposed spaces on their island would deal damage directly to them.

I worked on several facets to bring this idea to life. Using the mesh created by the art team, I made the empty spaces themselves a sort of fleshy mass. As players place their cards on top, they visually block the vulnerable-looking spot, showing players a relationship between the spaces and played cards, indicating what was vulnerable and what was protected.
The mesh for the mass itself was a plane with several subdivisions. Using a C# script, I manipulated the vertices to move up and down in a random pattern to give the impression of pulsating flesh. I also made a bespoke shader that would make the diffuse texture shift in a circle pattern.
The shader also had functionality for a script to make it flash red, called in gameplay scripts whenever the player suffered a direct attack.



Another way to associate gameplay with visuals was to make the island crumble. Splitting the artist's mesh into two halves, I created another C# script with a function that would make the lower half fall, hooked into by gameplay scripts detecting the current condition of the player. When the lower half crumbles away, a fleshy organ sack and vestige that dangles pulsates in the broken cavity, communicating danger and vulnerability. The organ animations were made with simple transform-based animations in-engine.

I was inspired by visual and narrative cues from the Eva units of *Evangelion* and the Daleks of *Doctor Who*, i.e. things with robotic exteriors (in this case, rocky) that hid something nasty and biological.

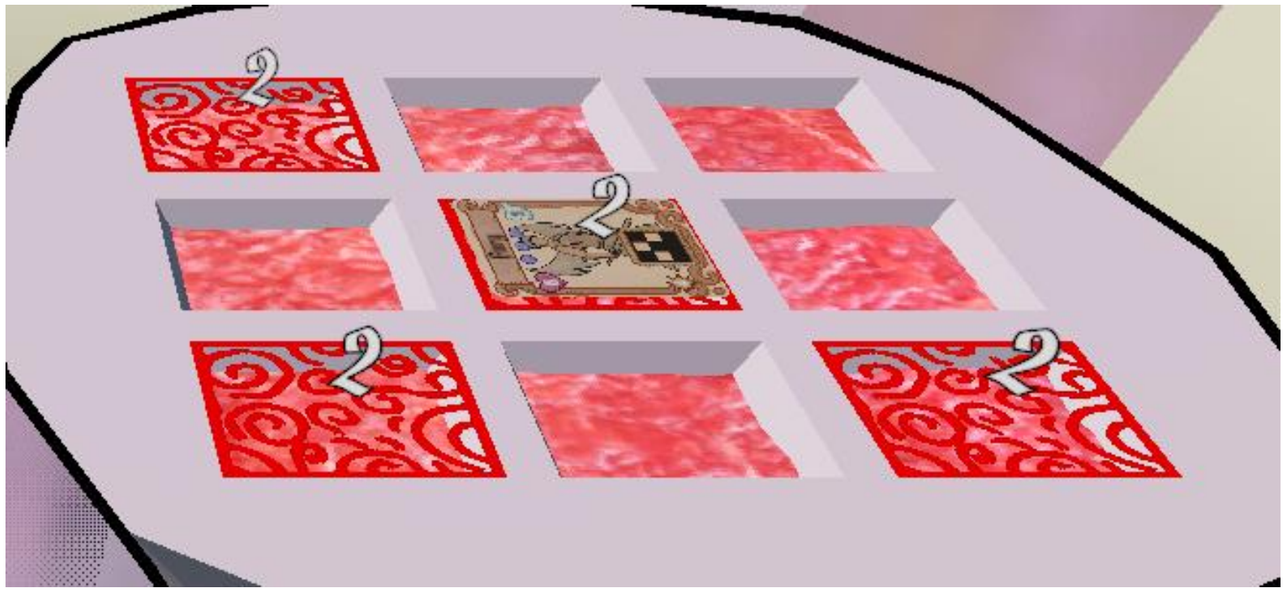


*Underside of the broken island, showing the exposed innards.*

## Status Numbers, Icons, and Colours

As the team tech artist, my job was to create tools and functions that programmers could easily hook into for visual feedback. We had many issues with players understanding what was going on during damage calculations, so I developed several essential visual systems of providing visual feedback.

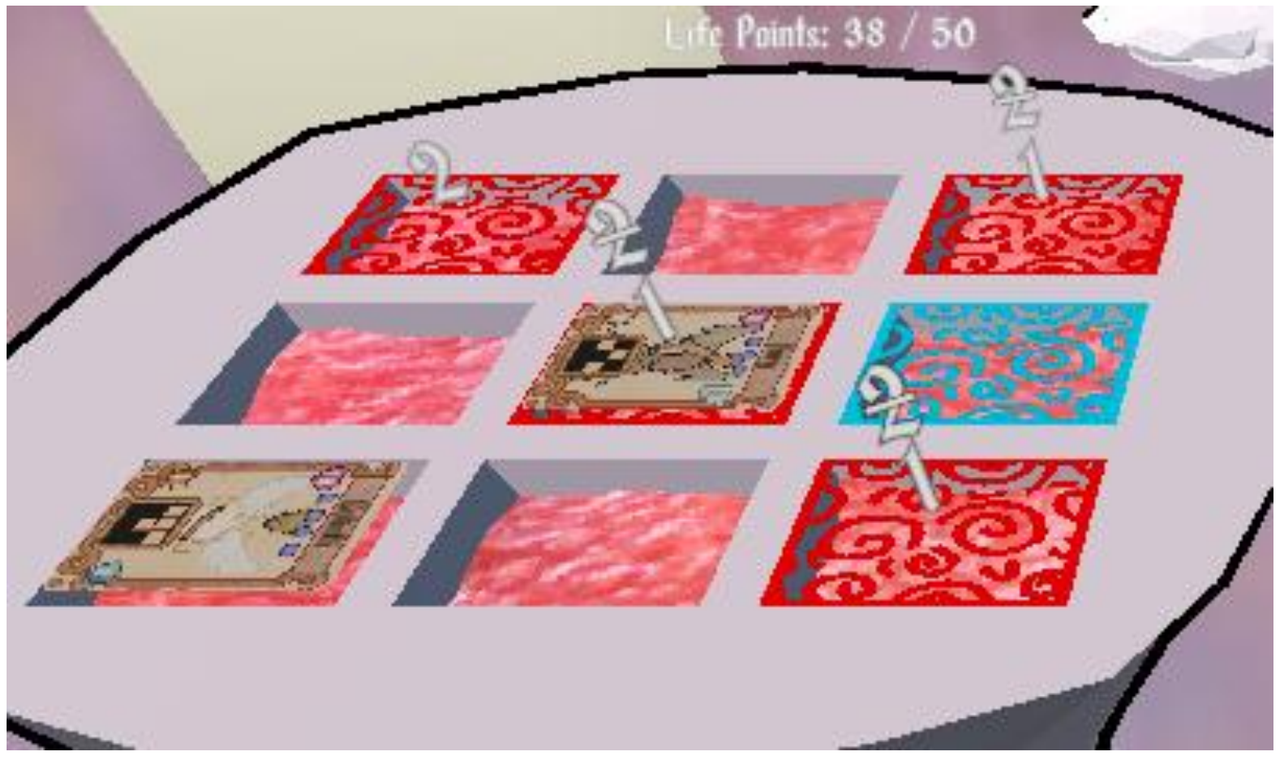
The first was coloured spaces. As cards attack or block spaces, making the spaces red for spaces getting attacked and blue for when they were successfully blocked gave players a simple way to assess their current situation at a glance.
For this, I set up a shader for the spaces that could accept an integer argument to update the shader texture's colour (e.g. 0 for no colour, 1 for red, 2 for blue, ect.). I then made an accompanying C# script that contained a public function to update the space's colour by manipulating this shader. Early on, there were not as many visual effects for the spaces, but over time I added other features to the function, such as a preview mode, or making the space disappear when no effects or statuses were present (to show the aforementioned exposed flesh). Artists also contributed more elaborate graphics to make the effects more visually pleasing.
These changes were made entirely within the space's script and shader, which meant that programmers didn't need to update complex gameplay scripts, only to call my function with the correct arguments. While the visuals were improved over time, programmers didn't need to worry about updating anything.



Another feature implemented early on in our Alpha stages of development were incoming damage numbers. They were intended to show players the total amount of damage an individual space would receive at the end of their turn. While initially made as a stopgap solution, playtesters understood what the numbers intuited so no further changes were needed.
Similarly to the space colours, I wanted to reduce programmer overheads when making this functionality. Using the same space script as before, I added a public function that would receive two arguments: incoming damage and incoming blocks. Our game's

mechanics have a subtraction-based block system e.g. if a space has an incoming attack of 3 damage, and is blocked for 1 damage, the resulting damage to that space is 2 damage.
I gave each space two text elements floating above it, one for the gross incoming damage and one for the net incoming damage. The gross damage element was configured to have a strikethrough in it, to indicate that it wasn't the final value and that players should read the net damage number for the actual incoming value.
In my script, I provided several clauses for what to do in certain situations. For example, if there was only incoming damage (no blocks), the gross damage number would be turned invisible and simply display the incoming damage on the net damage text element.

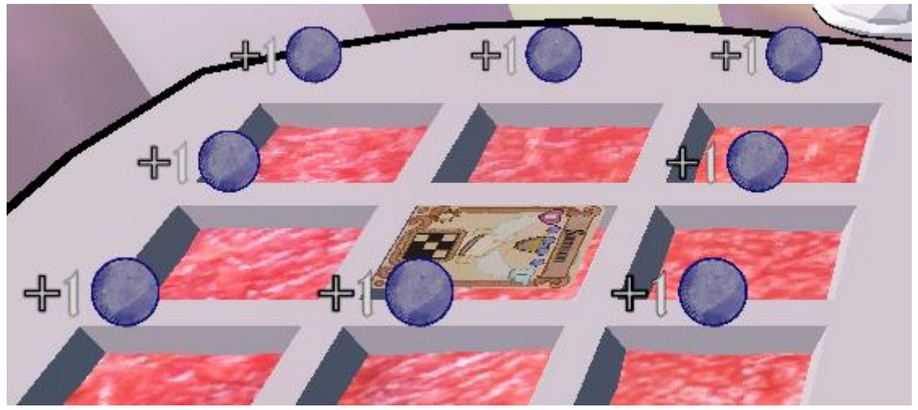
This would also change the colour of the spaces, using the previously mentioned function. All in all, this meant that programmers could feed in the numbers for incoming damage and blocks into my function and the script's configured logic would display the right number graphics and space colours automatically.



Another part of our visual feedback tools were floating damage numbers. Throughout gameplay, different events happen between players swapping turns that effect player health, card status, and player resources. These status changes happen suddenly, and players were often left confused as to what exactly happened.

To this end, I made a popup number prefab and accompanying script. The popups have a text and an image element, with a script that makes them float up and fade away when instantiated.

Similar to my other scripts with public functions, I set it up to take two arguments: the popup type and an integer amount. The image swaps out to the relevant icon depending on what the popup type was set to, and the number updates to the amount. As the icons were stored in an array within the script, we could scale this function up to all sorts of status effects, later adding a card death icon once we realised how useful the script and prefab could be.




*Manna popups appearing over spaces where the player has earned Manna points.*

With all of these visual feedback systems, I had created robust tools for player UX that were easy for programmers to use.

## Card Builder

Another large challenge was displaying each unique card's data. Our programming team set up a system to interpret data from a .json file, so each card would have a unique set of attributes. My job was to create a format to display the card data on the in-game card object. Using a layout made by our design team, my card building logic would interpret the loaded card data and display it on text elements on a 3D sprite object in-scene.
Originally using a single placeholder texture, our artist produced several different textures including icons and different parts to

arrange (see below). I set up the card builder logic and art elements to display the correct values legibly, including the card's stats, name, and grid shape. The card would also display the manna cost value of the card (represented by the blue orbs below the name) and dynamically centre whatever amount of manna was listed in the card's cost data.
Later versions of this setup would also hide certain icons of attributes from cards that didn't have a value in that attribute. The card building process was automatic, and small details like this gave a little bit of clarity to players.


*A card made from the first functional card builder from our Alpha (left), compared to one from our latest build (right).*

## Main Menu and Lobby

The main menu was designed by me, pulling together our game's essential backend functions in a neat and coordinated way for players to navigate.

The first step in making the menu intuitive was to organise all the different possible screens with a UI manager. Creating a C# script, I assigned each screen's elements in their own arrays, before putting them onto an array of arrays. My function to set the screen would take every object in the array of arrays and disable it, before going back to the desired screen's elements and reenabling them.

Earlier prototypes of the main menu were not as robust when trying to change screens, enabling and disabling items individually in code. This meant that if a new element was added, it would need to be updated in several places. Several different scripts were also changing the screens, making for a messy setup. With my new UI manager, any script could use its function to change the displayed screen, which was especially useful for the network manager to display loading screens when connecting to games or return to the main screen when failing to connect.

Other facets I added to the main menu were camera angles that hovered around a decorative island. I took some inspiration from some games such as *Dishonored* which would have a 3D environment in their main menu, and the camera would move throughout it as the player navigated menus.
A minor addition was creating a script that would make the menu buttons (styled as scrolls of parchment) slide into frame. This script would also be used in the main gameplay scene, and also had a function that would make the button slide a tiny bit further when moused over, providing some visual feedback to players.
I reused the script for the popup messages I designed (see *Inspirations* above). These popups would trigger from a public function, taking an error code as an argument, making it easy for programmers to implement in code for when players make erroneous moves.

The main menu also featured a floating castle, designed by me (see *Narrative Design*) and modelled by the art team. I created a simple script that would make these decorative elements orbit around in the background.

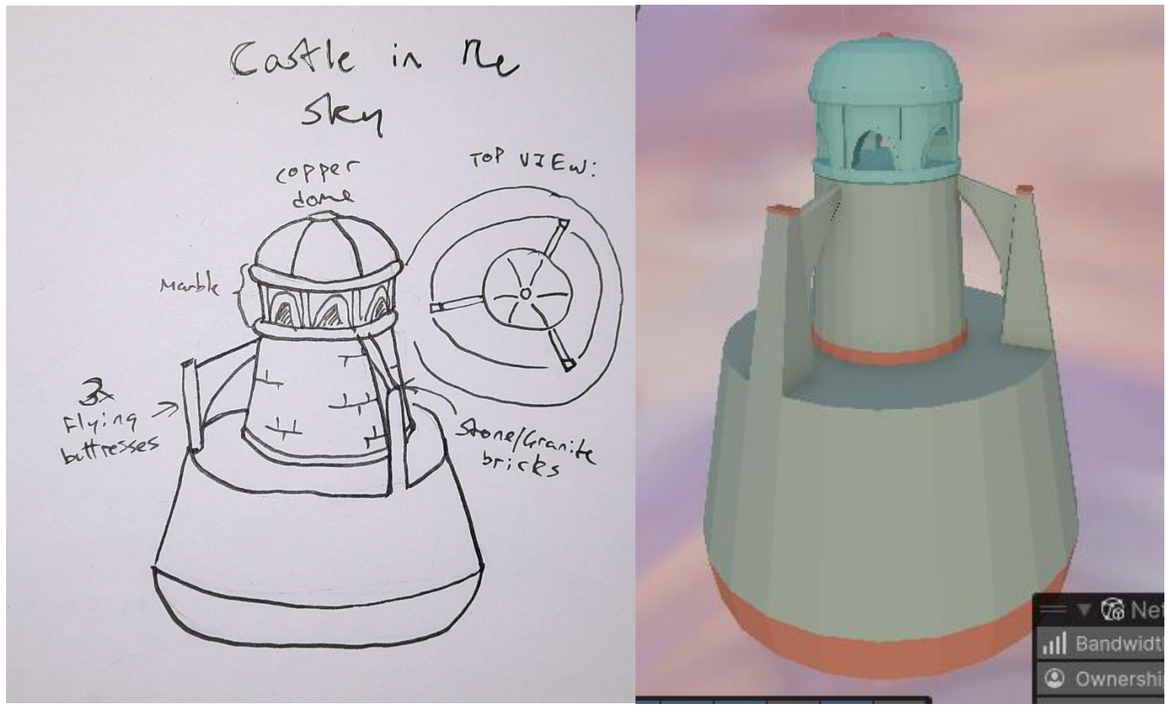
I also configured the environment for the lobby scene to feature the inside of the castle, providing some visual intrigue. Players can see an island float by in the background, and a small detail I programmed in was to make a second island begin to float in the opposite direction when a second player joined the lobby.


*An example lobby, both islands in view.*

# *Narrative Design*

While not a narratively focused game, I was in charge of providing some inspirations and support for the art team. As they explored a new celestial aesthetic, we needed an angle for the game's premise as well as names for the cards.
As mentioned above, I pushed the angelic theming, using biblical, Olympian, Nordic, and Hellenic inspirations for the game's cards. Figures associated with mythology were used, providing our art team with a plethora of inspirations to develop key art for cards.

Apart from the narrative, I also had a small hand in providing some concepts for assets, providing some visual design inspiration. The castle that players can see in the main menu and wait in when first connected to a game was an original concept of mine, passed onto the art team to model. I intended for this castle to be a bit of visual flavour and intrigue, in an otherwise very open environment.




*My original concept (left) and the art team's 3D rendition in-engine (right)*

# Postmortem

Since publishing our Beta builds online and seeing player responses, we've been highly encouraged by our efforts, especially considering the first playable version of game was created from scratch within a 3-month development period.
We're especially happy about how players keep finding new ways to play and strategize; it's very apparent playing with people the different emergent strategies they use.

Given how none of us had a lot of experience with developing this type of strategy game, we covered a lot of ground and made a viable experience. I feel especially vindicated as a producer as a lot of the early tools I set out for developers kept us on track and on schedule. We hit all our Alpha and Beta release goals on time and with sufficient progress at each stage.
With how transparent our communication methods are, we have a clear record of what we've done and learned from our development styles and schedules how we work.
We learned a lot from the development issues of C&B and applied those lessons to *Cardestia* with great success. While C&B was beset with issues of tacking on features and spending time worrying about features we would never implement, *Cardestia*'s workflow focused developers on-task and prioritised playtest feedback and learning from our prototypes above all.

One issue we did find was that we found it was difficult to get good feedback regarding balancing and gameplay early on in development as players had difficulty understanding the game. With more investment into UX development earlier on, it might have allowed us to get to the crux of those gameplay issues. UX was a blind spot early in the semester, and we paid for it by spending most of our development bandwidth trying to implement basic quality of life features and UX considerations for new players, which took away time we could have spent refining and balancing gameplay.

Thomas West MMXXV - MMXXVI