1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. How many of your requirements were inspired by speaking to your client(s) or their proxies (e.g. your peers, stakeholders, potential users)?

4. Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project.

5. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

6. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

**ANSWERS:**

Team:

3: While we did not conduct any direct requirements elicitation with our stakeholders, we did consider each other as gamers who have useful insights into what makes a good game. This inspired requirements such as the need for auto-save, since that is a common feature in modern games, appealing to players who do not want to have to constantly manually save their progress.

Felix Hurst:

1. Dividing the work went well, since the Meyer's template allowed us to commit to individual Asciidoctor files without causing merge conflicts in our GitHub repository.

2. There were some problems with team members failing to follow proper Asciidoctor syntax, as well as failing to follow the template as intended. This led to delays in submitting our work. Additionally, the team member assigned to review these incomplete parts failed to see the errors. This will be resolved by requiring more reviewers for each major pull request, to ensure mistakes are caught and not overlooked.

3. See team answer above

4. Courses I have taken in the past that will help us be successful in this capstone project include:
   -SFWRENG 3RA3 - Software Requirements and Security Considerations
   -SFWRENG 2OP3 - Object-Oriented Programming
   -SFWRENG 3BB4 - Software Design II: Concurrent System Design
   -SFWRENG 3S03 - Software Testing
   All of these courses are highly relevant to developing our project; our game. It will involve concurrent systems (graphics, audio, controller, etc.), object-oriented programming (game objects such as pixels of broken terrain), requirements (this very deliverable), and testing.

5. One skill that I will personally need to learn to become a more effective developer on this team, will be the Unity game engine. Since this is the engine upon which we are building our game, I will need to get familiar with it.

6. These are two approaches to learning Unity:
   -Find and watch a series of tutorial videos on YouTube or from official Unity sources.
   -Open Unity and practice with it by testing all of the options and tools one by one, only reading documentation for help when necessary.
   I will choose the first approach, as I believe that I learn better by watching demonstrations first before attempting to test tools on my own.

Andy Liang:

1. I started by outlining the SRS and assigning sections, which let us write and review in parallel without stepping on each other's toes. I set up requirement IDs and a lightweight traceability path from goals to use cases to requirements and test ideas, so every statement had a home. Standardizing the voice to "The system shall. . . " with measurable acceptance criteria kept the writing consistent and reduced ambiguity. Using the Meyer's template with modular Asciidoctor includes made merges clean and iteration fast. Most importantly, I translated our core game vision (slime-mould growth, trails, spores, readability of state) into concrete player-facing use cases and quality attributes, so the document stayed grounded in the experience we want to deliver.

2. Early drafts tended to be broad design wishes rather than testable statements; I fixed this by adding a concise glossary for key terms, rewriting requirements to be atomic and measurable, and attaching acceptance criteria. Non-functional targets were vague at first, so I set explicit baselines ($\geq$60 FPS on target hardware, ¡1 s save/load, ¡100 ms input latency) and captured them as NFRs. Scope creep showed up as "nice-to-haves" becoming mandatory; I introduced a Non-Goals section and MoSCoW prioritization to log ideas without derailing scope. We also hit occasional broken anchors and cross-references during merges; I added a local doc-

build check to the PR template and required a second reviewer for major edits to catch these before merging.

3. See team answer above

4. Software Requirements & Security (SFWRENG 3RA3) directly informed elicitation structure, validation, and traceability in the SRS. Object-Oriented Programming (SFWRENG 2OP3) underpins our C#/Unity component design. Software Design II: Concurrent Systems (SFWRENG 3BB4) maps to our cooperating subsystems—update loop, input, audio, and rendering. Software Testing (SFWRENG 3S03) helps convert acceptance criteria into practical test ideas and an automation strategy. The current capstone course provides the project-management scaffolding—milestones, integration cadence, and stakeholder communication—that keeps the technical work on track.

5. I will learn to design and implement a Physarum-inspired, agent-based "slime mould" algorithm in Python—and systematically encode a queer design lens into its rules and evaluation. Concretely, I'll prototype chemoattractant fields, diffusion/decay on grids, and agent sensing/steering with NumPy/Numba, then "queer" the mechanics by privileging multiplicity over single-path optimality (e.g., branching, reversible flows, fluid goal states, identity-switching agents, and non-hierarchical resource sharing). I'll visualize behaviors in notebooks, define metrics beyond shortest-path (e.g., diversity of routes, resilience, and co-existence), and export fields/paths as CSV/PNG for Unity integration so the Python studies directly inform game feel and level mechanics.

6. Approach A (replicate → extend): Reproduce a baseline Physarum model from the literature in Python (grid diffusion + agent sensors and deposition), validate it against known behaviors, then iteratively extend it with "queer" rules (multi-source attractors, stochastic identity shifts, anti-teleological objectives) while running parameter sweeps and profiling (Numba) to keep it performant. Approach B (project-first vertical slices): Start from our specific game moments (e.g., trail growth, spore exchange, merging/splitting bodies) and build focused Python notebooks that embody a queer mechanic per slice, with quick visual metrics and export pipelines to Unity; run short playtests to judge readability/feel, then refactor into a small internal Python package. I'm choosing Approach B because it yields immediate, game-relevant artifacts, lets us evaluate "queerness" through playable behavior (not just theory), and creates a practical export path the team can iterate on right away.

BoWen Liu:

1. Our team was able to effectively solidifying and expanding our core concepts in the concrete planning of our game design. The Meyer's template provided a solid basis in our planning and design development

2. A major pain point during the deliverable was deciding on the logical flow of the components in the component diagram as well as the user interactions which required thorough reference of current existing games.

3. See team answer above

4. 2AA4 and 4HC3 were positive contributing factors in the continued success of our developing capstone project because it introduced and guided us in the iterative development and planning process as well as general criteria to look for in terms of a successful product.

5. For this project, I will be focussing on the area of knowledge and skills in developing the destructive environment as that is my area of responsibility. This is in terms of how to structure the methods and components as well as how the overall logic and flow should be.

6. One approach would be to learn exactly how a preexisting game of the same 2d pixel genre handles destructive environments and use their methodology in our game. Another approach would be to evaluate our games requirements and plan and build the destructive system around that. I chose the second option due to us having different requirements in terms of gameplay, timeframe, and scale compared to the games that could be referenced. Developing our own unique destructive system could be more time effective and aligns better with our game requirements

Marcos Hernandez-Rivero:

1. Dividing the work was easy. The template we are using had different asciidoctor files for each section, from there, it was easy to simply assign parts (files) to individuals, which made getting started on it a lot easier and helped us start thinking about the rest of the document.

2. For a long time, we were all doing our own thing until it was time for review. This meant that we all used different standards for how to write stuff such as dates, times, how to reference objects, how to insert pictures, etc. We resolved this by simply standardizing all of this information, so now every date and name (used in the control.adoc tables) should be standardized, and every image insert should be done the same way. We also struggled due to poor quality reviewing. We had assigned each indiviudal to review one other individual and be reviewed by one other individual. Sometimes, some errors got through when the reviewers were not meticulous enough in their reviewing. We plan to resolve this by assigning two reviewers (or more) for each section of work that we do from now on.

3. See team answer above

4. Some courses I've taken in the past that might help are:
-SFWRENG 3RA3 - Software Requirements and Security Considerations,

as understanding requirements is ESSENTIAL for ANY project, and especially in one so user-facing
-SFWRENG 2OP3 - Object-Oriented Programming, as we plan to use an object-oriented architecture to develop the game and its systems, and understanding object oriented design helps a lot with visualizing and understanding how such a system might be designed
-SFWRENG 3BB4 - Software Design II: Concurrent System Design, we have many loops and many systems working in parallel, and so this course helps understand how to execute and manage concurrent tasks, which is crucial in a game, as we will have many systems running at the same time
-SFWRENG 3S03 - Software Testing, a bit obvious, but testing is vital for any software engineering project, especially games.
-SFWRENG 4HC3 - Human-Computer Interfaces, since this is a game, it will be extremely user-facing, and users will interact directly with the interfaces we develop the entire time. Understanding good interaction and design principles will help us develop a game that is more enjoyable, and easier to learn.

5. I definitely need to learn more about using the Unity game engine, as that is what we will be working with for our project. I have used other game engines in the past, but Unity offers some features that other game engines don't, which will be the main focus of what I want to learn. I am already somewhat familiar with scripting and object generation from other game engines, so I also need to investigate the differences between doing that in those other engines and in Unity.

6. I think the easiest way for me to learn this information is by trying to recreate a project (or parts of it) I made on a different game engine, in Unity, as that will let me work with familiar concepts, but learn the specific skills necessary to Unity development. Additionally, I will watch videos on the Unity game engine, learn the new features associated with Unity, and try incorporating them into the test project. This helps because I am working off a foundation that I am already familiar with, so the learning curve isn't as steep and I am also someone who learns best by actually doing and seeing, so this will help more than just reading a guide or purely watching a video without any work from my end.