❯ export SESSION_NAME="post-reflection"

❯ whoami

Philippe Hebert (27013728) <philippe@human.space>

❯ groups philippe | grep CART

CART 253-2252-A

❯ getent group "CART 253-2252-A"

Dr Pippin Barr

❯ date

Wed Dec 3 23:10:22 EST 2025

❯ URL=$(which ‘post-reflection’)

https://github.com/philippefutureboy/cu-cart253/tree/main/projects/reflective-essay/Post-Reflection.pdf

❯ echo “$(curl $URL | wc) words”

2064 words

❯ cd /Users/philippe/GitHub/cart-253 && \

  git branch | cat

main

2025/cart-253/art-jam

2025/cart-253/art-jam-response

2025/cart-253/mod-jam

2025/cart-253/variation-jam

2026/onwards

 # Extracting state prior to CART 253

❯ git switch main && git log -1 --before="2025-09-04" | cat

commit 8d86f0e98808ca770d86a79b63e3a1941a9f734e

Author: Philippe Hebert <philippe@human.space>

Date: Before: Thu Sep 4 13:15:00 2025 -0400

> Data Engineer. Founder & CTO.
> Previously in SOEN (2013-2016).
> 10 years experience programming.
>
> Extensive experience in building complex, end-to-end systems in microservice architecture in the cloud, and with many technologies, including Python 3, Javascript, NodeJS, SQL, Docker, Kubernetes, etc.
>
> Little-to-no prior experience in creative programming. Complete prior understanding of technical concepts taught in CART 253.
>
> Did not take equivalence for CART 253 because wanted to take the opportunity of being ahead technically to focus on the creative & playful aspects of programming & making games.

```
# Extracting insights from the Art Jam project
# Squashing insights into a single commit
❯ git switch 2025/cart-253/art-jam && \
  git merge –squash –summarize 2025/cart-253 && \
  git switch 2025/cart-253 && \
  git log -1 | cat

commit 5ac8c377f6790ea466f43a61283cdeb755c1a5a6
Author: Philippe Hebert <philippe@human.space>
Date:   Thu Sep 11 13:15:00 2025 -0400 -thru- Thu Oct 2 13:15:00 2025 -0400

    The combined demands of work & the Art Jam were a lot to juggle. I was forced to see the limitations of
    my programming practice.
    I ended up spending hours building a ReactJS + P5 framework, to barely use it
    I built it because I thought I might need it. And that's a recurring pattern in my professional life as well.

    YAGNI, you might say.
    You Aren't Gonna Need It.
    John Carmack wrote "It is hard for less experienced developers to appreciate how rarely architecting for
    future requirements / applications turns out net-positive."[1]

    I had been struggling with this issue for months;
    The Art Jam was really the straw that broke the camel's back.
    IAGNI. I Ain't Gonna Need It.
```

[1] Wikipedia, The Free Encyclopedia, *You aren't gonna need it*

```
# Extracting insights from the Art Jam response
# Squashing insights into a single commit
❯ git switch 2025/cart-253/art-jam-response && \
  git merge –squash –summarize 2025/cart-253 && \
  git switch 2025/cart-253 && \
  git log -1 | cat
commit 03c85a667bd2d22e6e4291229e5640ab147d5421
Author: Philippe Hebert <philippe@human.space>
Date:   Thu Oct 9 13:15:00 2025 -0400 -thru- Thu Oct 16 13:15:00 2025 -0400

    The other students' submissions for the Art Jam really surprised and inspired me, mainly because a lot of them were able to create engaging, creative, and unique experiences with little programming knowledge.

    Reviewing other students' submissions really showed me that great things can be achieved with minimal technical complexity. It highlighted that sometimes, oftentimes even, less is more, and that an emotionally engaging/impactful experience does not necessarily imply technical complexity.
```

```
# Extracting insights from the Mod Jam project
# Squashing insights into a single commit
❯ git switch 2025/cart-253/mod-jam && \
  git merge –squash –summarize 2025/cart-253 && \
  git switch 2025/cart-253 && \
  git log -1 | cat
commit eea1484cab9a89ccf6df980f504f38fee43ed5a4
Author: Philippe Hebert <philippe@human.space>
Date:   Thu Oct 9 13:15:00 2025 -0400 -thru- Wed Nov 5 13:15:00 2025 -0400

    Developing a Game Design Philosophy
    Through Pippin's presentations, I learnt that design decisions can affect the emotional tone, and philosophical implications of the game.
    This changed my philosophy on game design. I realized that simplicity wasn't a detriment to the quality of an experience. The quality of an experience is determined by how effectively it impacts & engages the player
    By making emotional impact and engagement the guiding principle of my game design practice, I will ensure that every design, technical, and artistic decision reinforces that principle, resulting in stronger, more memorable, and more enjoyable experiences for my audience.


    Serendipitous Game Ideation
    Pippin's presentations and our discussions made me realize that sometimes the best ideas happen serendipitously, through bugs even.

    A technical analysis of P5 as a medium
    Learning P5, it became clear that P5 was a great tool for creative programming, but also has its limitations when used in isolation.

    P5 is effective at producing procedurally-generated visuals, especially given its extensive math & utils set of functions – p5.random , p5.sin , p5.lerp , p5.Vector , etc. For that specific purpose, it's a great prototyping tool: reactive, fast feedback, well-tooled.

    P5 also provides an easy-to-reason-about mental model – setup, draw (loop), input handling. These entry points clearly separate concerns; this being said, the P5 model is very coarse for more complex experience. Suffice to look at the need to support scenes & scene transitions to see that P5 provides only a high level shell/container, and a set of low-level primitives/utilities. This narrows its utility space, the negative space being fulfilled by more fully fledged engines.
```

```
# Extracting insights from the Variation Jam project
# Squashing insights into a single commit
❯ git switch 2025/cart-253/variation-jam && \
  git merge –squash –summarize 2025/cart-253 && \
  git switch 2025/cart-253 && \
  git log -1 | cat
commit 42366bd7ff539a23993e28233febd3b0cb52562b
Author: Philippe Hebert <philippe@human.space>
Date:   Thu Oct 9 13:15:00 2025 -0400 -thru- Wed Nov 5 13:15:00 2025 -0400
```

Working simplicity & variation

In the same continuation as previous jams, the Variation Jam led me further down the path of designing with simplicity in mind. I settled on the game of tag – a universal game, played by both humans and animals alike. Given its simplicity, the tag game allows for a lot of variations:

- *Control based variations:* Control scheme changes, direction inversion, movement speed, acceleration
- Rules variations: Tag count, rock-paper-scissors dynamic,
- *Time based variations:* Game duration (static, variable),
- *Environment based variations:* Obstacles, labyrinth, different movement mediums, safe zone, obscurity
- *Philosophical variations:* What constitutes “tagging” someone?

The simplicity of the idea of tag means that there are less constraints to adhere by, thus allowing for a wider spectrum of experiences.
This realization further cemented my understanding that simplicity and divergent thinking can lead to engaging experiences with different emotional tones.

Working NPC AIs

As simple as the tag game may be, it is a multiplayer game; This forced me into a dilemma - Do I want to host a websocket multiplayer server or do I want to build a game AI? I decided implement an AI system.

I could not have foreseen how complex this would get. While this was time consuming, I’m grateful because it taught me a lot about AAA game structure.

```
# Committing state;
# Reflecting on what's on & over the horizon
❯ git switch 2025/cart-253 && \
  git switch -c 2026/onwards
  git commit
commit 42366bd7ff539a23993e28233febd3b0cb52562b
Author: Philippe Hebert <philippe@human.space>
Date:   Thu Dec 4 17:30:00 2025 -0400
```

Musing on programming primitives

Loops

Loops are a control flow structure that allows the recursive repetition of a set of instructions over time.
Loops are not only a control flow structure; they are also an aesthetic logic, a way to shape an experience.
Loops are a temporal structure, in that each iteration is tied to a time interval (C/GPU cycles or actual time).
Loops can be:

- explicit – such as the `while` loop
- implicit – such as a webserver socket-listening loop
- abstracted – such as the repeated application of the `draw` function in P5
- conceptual – such as the concept of game loop – or the kind of input/reward cycle that keeps player engagement

But at the fundamental level, loops are tools for learning.
This is because their primary utility is to repeat a cycle while state changes – whether it is internal state (state changed by the logic within the loop itself), or external state (state received from an external buffer/input source,).
The most fundamental, and archetypal loop, is the OODA Loop: Observe, Orient, Decide Act, a decision-making model developed by United States Air Force Colonel John Boyd. The OODA loop is in effect the template from which all loops arise. This is because all loops arguably contain the same fundamental steps:

- Observation: Observation (sampling) of internal/external state; minimal state requirement is time delta (CPU cycle or physical time)
- Orientation: Processing of internal/external state
- Decision: Application of internal logic based on pre-configured logic (said logic may be based on prior observations integrated in the model) to determine action to apply
- Action: Application of the Decision taken.

The application of this iterative/recursive structure has many applications, especially when designing experiences:

- Play: Play is an OODA loop; play provides a space with limited stakes in which the player can explore, fail/succeed, and learn from said interaction.
- Rhythm: Loops provide rhythm – through combat (attack, block, evade, recover – often with telegraphed enemy sequences), narrative loops (Campbell Hero's Journey, story "beats"), game loops, and more.
- Progress: Game loops can provide feedback to the player about their progress. Example: resource gathering (XP or otherwise), claiming reward / unlocking new levels, and gathering again. Alternatively a loop can also minimize/eliminate progress if the state is partially/completely reset after every loop – think Sisyphus & Prometheus unending ordeals, Majora's Mask 72h cycle, The Stanley Parable's reset, Roguelike games where every game is a new game. Repetition can be transformative: learning is done by repeating the same thing over and over again while learning. In a game, the machine iterates its state, the human iterates their skill.

The power of loops power lies beyond repetition. Loops establish the tempo of computation and the rhythm of experience. In games, they become the heartbeat of interaction, shaping what feels alive and what feels static. Loops are not just technical devices—they are experiential architectures. They reveal that in programmed worlds (as in life), repetition is not the opposite of change, but its engine.

Conditionals

Conditionals are control flow structures that evaluate a state and makes a branching decision based on it.
Where loops give software a rhythm, conditionals give it a direction.
Conditionals encode structure, restraining the possibility space of a program.
Conditional encode choices, and the consequences of said choices.

In *Technology Matter: Questions to Live With*, Nye argues that technologies are composed of three components:

- An intention: Purpose or imagined sequence of action – a story, a situation, a need, an imagined future state, and an intention to change conditions
- An artifact: The physical or logical (in the case of software) material
- An actual practice/use: The skills necessary to use the artifact, but also the positionality of the user and circumstances of usage of the artifact[2].

[2] David E. Nye, *Technology Matters: Questions to Live With*

Conditionals are part of the artifact, encoding the intention of the author. Conditionals limit the choice possibilities afforded to the player. Said choices are influenced by the intention (positionality, creative desire) of the author – whether experiential, moral, cultural or otherwise.

Conditionals provide player agency through the options provided. Sometimes these options are sufficient, but in some cases, these options can be restrictive. For instance, the game Catherine (Classic/Fullbody) limits the choices to impose their moralistic view of romance and companionship. Catherine is a dating-sim game with a puzzle component. Falling into a dichotomy between good (staying committed to an impending marriage), and bad (cheating, choosing freedom), the game simplifies the nature of relationships through their infantilizing and moralizing lens. This often left me with a distasteful sense of lack of agency as a player. Catherine is a clear example of how developers applied their own positionality when designing the narrative of the game. As a player, I am forced to make choices that I would not take in real life, only to follow the flow control structure embedded in the game.

More generally, conditionals force the player to adhere to certain sets of rules and experiences. If for instance a game like Red Dead Redemption did not allow shooting civilians, players would live a very different experience from what currently exists. On the opposite side of the spectrum, procedural games like No Man's Sky use the concept of conditionals to lazy generate/load additional experiences, thereby providing the player with additional experiences, expanding the player's agency. Conditionals can also be used to shape narrative structure – one choice can lead to a new dialogue branch/ending/quest opening up. More generally, conditionals encode consequences of actions done by the player.

In summary, conditionals are decision points in the program/around the boundary of the program. They encode design decisions, leading to constraining possibilities, encoding consequences, responding to player action, shaping narrative flow. They most often end up expressing the positionality of the authors, or make the produced media have a positionality of its own.

Classes & OOP Primitives

OOP in game design is effectively ontology management.
OOP essentially maintains what is possible, what type entities exist, how they relate to and interact with each other, what traits they exhibit, etc. OOP is worldbuilding at the software level.

Coming back to Nye's theory of technology, we could argue that OOP structures embed design beliefs:

- What counts as similar
- What counts as an archetype
- What behaviours belong together
- What variation looks like

A class hierarchy & composition is not neutral; it reflects the designer's positionality & thinking paradigms. An interesting question would be: How does a person's positionality – culturally, ethnographically, linguistic, philosophical, spiritual affect how they structure their thought in programming?

Working with Generative AI

I was really grateful to be granted the permission to use Generative AI for the scope of this class. Without Generative AI, many of the more complex features of my projects would not have been possible –

- Art Jam: SVG + Video Overlay
- Mod Jam: Physically-Simulated Frog tongue
- Variation Jam: NPC behaviour engine

This being said, I did feel a little bit lesser as a programmer for using Generative AI.
GenAI really helped push the envelope within my limited time (and energy) constraints;
In the end however, I am disappointed that I didn't have the time to work out these problems unassisted.
Additionally, I did feel like using AI lifted a constraint that would otherwise maybe force me to be more creative – after all, constraints breed creativity.

On the future of my role as a creative programmer

The many learnings I have accumulated through CART 253 will undoubtedly make me a better creative programmer in the future.

The focus on simplicity, engagement, and emotional tone will help me craft precise, intentional designs. The exercise in divergent ideation & welcoming serendipitous ideas will help me free my creativity & practice – both as a programmer and as an artist, no matter the media.

The limitations imposed by my limited time and creativity taught me to be leaner and to focus on what brings the project to fruition/what brings values to the user.

My encounter with P5 forced me to learn how to build a partial game engine on my own, but also renewed my understanding that not everything is a nail, and not every tool is a hammer.

This musing about the control flow structures in programming made me more aware of their use and the possibilities they unlock.

I look forward to applying these newly acquired skills and concepts into my career as a creative coder. I am especially excited about biologically/procedurally inspired algorithms, and how to make visualizations & sounds that create happy accidents. I'm excited to marvel at the little curiosities and their consequences, and let my creations take a life of their own.

BIBLIOGRAPHY

Nye, David E. 2007. Technology Matters : Questions to Live With. *Cambridge, Mass. ; London: Mit.*

Wikipedia, The Free Encyclopedia, s.v. "You aren't gonna need it," (accessed Dec 4, 2025), https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it