# **What dismotivates you from the current C++ project building with Qt?**

* Qt offers complex libraries, which I can’t fully make use of without understanding C++ in depth.
* Code is unorganized.
* I heavily rely on chatgpt for code generation/starting point and can barely write my own code.

# **Different approaches to consider?**

* Build an in-depth knowledge in C++ through another fundamental project which covers C++ core concepts.
* For C++ core concepts, check out the C++ tour book and gpt.
* Brainstorm a project that is feasible to finish within 2 weeks.

# **To-do list**

## **C++ core concepts:**

* Object-oriented
* Low-level memory manipulation with pointers, references, ‘new’, and ‘delete’ keywords. This is crucial in performance optimization.
* Template and generic programming with STL (standard template library), a collection of template-based algo, containers, and iterators.
* Understanding what is a compiled language and why it efficiently contributes to the performance.
* What is inline function?
* Operator overloading which allows operators to work with developer-defined data types.
* Exception Handling
* RAII (Resource Acqusition Is Initialization).
* Concurrency with multithreading.
* What makes C++ a portable language?

## **Project ideas with pros and cons:**

**Findings after some GPT consulting:**

* It is almost impossible to develop a GUI application without relying on an external GUI framework in 2 weeks.
* C++ power could be exploited and learned through non-GUI ideas. Think about when you learn JavaScript or Python, no interface needed to learn these languages. Same to C++, focus on understanding the concepts through non-GUI programs, the ability to utilize GUI frameworks will naturally come.

**Ideas:**

* A very simple game that implements very fundamental/simple game engine components. **-> Final selection.**
* A simple physic engine.
* Create a compiler

**Pre-development questions**

1. How this project fits into the scope of core concepts you want to cover?
   * Object-oriented
     + each entity/component could be one class.
   * Low-level memory manipulation with pointers, references, ‘new’, and ‘delete’ keywords. This is crucial in performance optimization.
     + Be mindful when passing arguments to functions.
   * Template and generic programming with STL (standard template library), a collection of template-based algo, containers, and iterators.
     + Use case?
   * Understanding what is a compiled language and why it efficiently contributes to the performance.
   * What is inline function?
   * Operator overloading which allows operators to work with developer-defined data types.
     + Use case?
   * Exception Handling
     + Handle user’s invalid inputs.
   * RAII (Resource Acqusition Is Initialization).
     + Use case?
   * Concurrency with multithreading.
     + Use case?
   * What makes C++ a portable language?
2. Is it feasible to build within 2 weeks?

* A main function holding the game loop (2 days)
* An Entity Component System (ECS) to manage game entities and their components. (2 days)
* A Rendering System with ASCII. (2 days)
* An Input Handling System. (1 day)
* Game Logic and Rules (2 days)
* AI System (2 days)
* Resource Management with file I/O for game state. (1 day)
* Optimization (1 day)
* Testing and final touches. (1 day)

1. Is it scable into a GUI app?

Yes, with game engine updates.

**Development question**

What is considered an entity, component?

* Components:
  + data structures which hold specific attributes or data about an entity.
  + Different entities can share the same type of component.
  + Do not contain logic or behavior.
  + Examples: HealthComponent, PositionComponent.
* Entity:
  + An identifier representing anything within the game world.
  + Holders for components.
  + Have no behavior or properties. They’re only meaningful when associated with components.
  + Examples: a character player, an enemy, a tree.
* Systems:
  + Logic or processes that operate on entities with specific components.
  + Example: RenderSystem draws all entities with a SpriteComponent, a PhysicSystem updates all entities with PositionComponent and VelocityComponent.

Further research on ECS (Entity-Component System) design pattern.

What to be included in the game?

* Game loop
  1. Input handling
  2. Updating game state by updating systems and entities.
  3. Rendering current game state on screen. For this project, this could include drawing the game grid with updated positions of the entities.
* Scene or World
* GameObject
* Prefab
* Tick

Any conventional codebase structure? Done

What to be included in a GameSettings?

* Difficulty level
* Map dimension