Difficulties I ran into while implementing the game and how I dealt with them

When I embarked on developing this project, it felt like standing at the foot of a mountain, unsure where the trail even began. It was one of the biggest tasks I'd ever undertaken in programming, and my first challenge was figuring out how to start. The project instructions were thorough but dense, and I found myself reading them repeatedly, trying to piece together the bigger picture. Every time I went back to the instructions, I discovered new insights, and slowly, I began to understand the project's flow.

One particular aid that stood out was the flowchart in the project's PDF file. At first, the flowchart looked like a foreign language, and I wasn't quite sure how to interpret its sequences. I spent a lot of time tracing the arrows, studying the relationships between the steps, and mapping them back to the project instructions. It was like solving a puzzle. Eventually, I came to understand not only the flowchart for this project but also how flowcharts work in general. This breakthrough was a small victory that gave me the confidence to move forward.

As I started implementing the project, I quickly ran into another major roadblock: the concept of classes. Before this project, I had little exposure to object-oriented programming. Terms like "class," "attributes," and "methods" felt abstract and overwhelming. To move past this hurdle, I took a step back and did some research, reading tutorials and watching videos that explained classes in simpler terms. I created a few basic practice programs to test out what I had learned. Slowly but surely, the pieces started to click. Understanding how to structure the Player class and connect it to the Game Engine was a turning point in the project—it felt like a light bulb had switched on.

Even with this progress, debugging became the next challenge to tackle. Errors were inevitable, but in the beginning, I struggled to even identify the source of the problems. The error messages seemed cryptic, and I often found myself going in circles, trying to figure out what went wrong. That's when I discovered the importance of breakpoints in Visual Studio Code. By placing breakpoints strategically, I was able to pause the program at specific points and inspect what was happening in real time. This technique became a game-changer, allowing me to pinpoint errors and understand the flow of the program much more clearly. Debugging evolved from being an intimidating task to a rewarding problem-solving exercise.

Along the way, I also encountered smaller but equally challenging moments. For example, I would occasionally misspell variable names or function calls, leading to seemingly mysterious bugs. At first, these errors were incredibly frustrating because I didn't realize the issue was something as simple as a typo. Over time, I learned to double-check my spelling, use meaningful and consistent names, and rely on auto-completion tools to minimize these mistakes.

There were moments of frustration and self-doubt, times when I wondered if I was in over my head. But each challenge, whether it was understanding a flowchart, learning about classes, or debugging code, taught me something valuable. I began to appreciate the process of breaking problems into smaller, manageable pieces and tackling them one at a time. Each small victory, no matter how minor it seemed, built my confidence and reinforced the idea that persistence pays off.

By the time the project was complete, I looked back and realized how much I had grown. What once felt overwhelming had become a rewarding journey of learning and discovery. The difficulties I faced didn't just test my skills—they shaped them. They taught me to be patient with myself, to stay curious, and to embrace the challenges as part of the process. This project wasn't just about creating a game; it was about building the foundation for a future filled with even greater programming adventures.

what I learned while working on the project

One of the first revelations for me came when I started working with classes. At the time, I didn't fully grasp their essence or why they were so important in coding. To me, they were just another term in a sea of jargon. But as I dug deeper, things started to click. Classes, I discovered, are not just technical constructs—they are the backbone of organization and logic in coding. I learned that they represent objects, encapsulating attributes and behaviors in a way that brings structure to chaos. The more I experimented with classes, the clearer their advantages became. They allowed me to break the program into manageable pieces, making it easier to understand, modify, and debug.

What fascinated me even more was the history and philosophy behind classes. I discovered that object-oriented programming was designed to mimic the way we think about the world—by categorizing objects and their behaviors. It was like finding a hidden layer of logic beneath the surface of the code. This realization made me appreciate the elegance of programming and opened up a new way of thinking about problem-solving.

But the lessons didn't stop there. As I pieced together my code, I quickly learned the importance of taking small steps. In my eagerness to complete the project, I initially tried to tackle everything at once, but this approach only led to confusion and errors. I realized that I needed to slow down and approach the project methodically. By focusing on one feature at a time and verifying each step before moving on, I was able to make steady progress without getting overwhelmed. This habit of breaking things down into smaller tasks became a cornerstone of my workflow, and it's something I'll carry with me into future projects.

Another pivotal moment came when I faced the challenge of debugging. Early on, debugging felt like searching for a needle in a haystack. Errors would pop up, and I had no idea where to start or how to fix them. It was frustrating, to say the least. But over time, I learned to approach debugging with patience and strategy. I started using tools like breakpoints to analyze the program line by line, allowing me to pinpoint issues with precision. Instead of viewing errors as setbacks, I began to see them as opportunities to learn and improve. Debugging, I realized, is not just about fixing problems—it's a process of understanding how your code works and why it behaves the way it does.

Perhaps the most valuable lesson I learned from this project was the importance of patience and persistence. Programming is rarely a straight path; it's a journey filled with trial and error, setbacks, and breakthroughs. There were moments when I felt stuck, moments when I doubted my abilities, but I learned to push through those feelings and keep going. Every challenge I faced taught me something new, whether it was a technical skill or a mindset shift. By embracing the process and staying committed, I discovered that the effort was always worth it in the end.

Through this project, I also discovered the power and utility of GitHub, a platform that transformed the way I approached coding and collaboration. Initially, I learned how to use GitHub to push and commit my new code changes. This feature enabled me to save and upload my work to a remote repository, ensuring that I had a backup of my progress and a safe space to store my project. Each time I committed changes, it allowed me to verify the accuracy and functionality of my work incrementally, preventing large-scale errors from piling up unnoticed. This process of systematically pushing and committing changes taught me the importance of version control in programming—it allowed me to document each step of my development journey and track my progress.

What I found particularly fascinating about GitHub was its ability to connect me with a broader community. By uploading my project to a public repository, I realized that others could view my code. Initially, the thought of sharing my work with others was daunting, but I soon recognized its benefits. Sharing my code

meant that I could receive feedback, suggestions, and even advice from other developers, which helped me identify areas for improvement and optimize my work.

Additionally, exploring GitHub opened up a new world of inspiration for me. By browsing similar projects, I was able to gain insights into how others approached problems that were similar to the challenges I faced. I discovered innovative coding techniques, creative problem-solving methods, and alternative ways to structure my code. These explorations not only helped me improve my project but also encouraged me to experiment with new ideas and concepts that I hadn't considered before.

Through this process, I learned that GitHub is much more than just a tool for managing code—it's a collaborative platform where knowledge is shared, creativity flourishes, and learning never stops. It taught me that programming isn't just about working in isolation; it's about engaging with a community, sharing ideas, and collectively striving to create better, more effective solutions. This revelation has completely reshaped the way I view programming and has given me a newfound appreciation for collaboration in coding.

Looking back, this project was not just about creating a program—it was a journey of growth and discovery. It taught me to think critically, work systematically, and approach problems with curiosity and resilience. The lessons I learned will stay with me as I continue to explore the world of coding, and I'm excited to see where the next adventure takes me. This experience has shown me that every line of code is a step forward, not just in programming, but in personal growth as well.

What I might do differently the next time I work on a similar project

If I were to work on a similar project again, there are several things I would do differently to make the process smoother, more efficient, and ultimately more enjoyable. Reflecting on my experience, I've identified key areas where I could improve my approach and workflow to better handle the challenges I faced during this project.

To begin with, one major change I would make is how I utilize external tools like ChatGPT. While it was extremely helpful in providing a general starting point and explaining concepts, I found that the code it generated was often too broad and not tailored to the specifics of my project. This resulted in a lot of bugs that I had to debug without fully understanding the original code. Debugging felt like reverse engineering—breaking apart the existing code and rebuilding it to fit my requirements. The next time, I would approach this differently. Instead of using AI tools for large code chunks, I would focus on asking for smaller, more targeted snippets or clarifications. For example, I could use it to learn how to structure specific methods or understand programming concepts, while keeping the actual implementation more in my control. This would reduce the debugging workload and help me maintain a better understanding of my code.

Another significant challenge I faced was debugging itself. Since I had little prior experience with debugging, it was difficult to identify errors and figure out how to fix them effectively. Debugging seemed overwhelming because I didn't know where to begin. Next time, I would start by learning debugging techniques upfront, such as using breakpoints effectively in Visual Studio Code or employing print statements to trace variable values. Developing these skills early on would save me a lot of time and frustration later in the process. Additionally, I would try to build and test the program incrementally, verifying that each part works before moving on. This "debug as you go" approach would allow me to catch and fix errors immediately, rather than having to sift through an entire program to find what went wrong.

I also realized that the project's size and structure were initially daunting. Having a clear plan and breaking the project into smaller, manageable pieces would have made things significantly easier. Next time, I would start with a detailed flowchart or pseudo-code representation of the project to map out the program's logic and structure before writing any code. By visualizing the relationships between classes, methods, and the overall game flow, I would have a better grasp of how everything fits together. This planning step would help reduce confusion and allow me to approach the project systematically.

Finally, I learned the importance of taking small, deliberate steps in the development process. In my eagerness to complete the project, I initially tried to tackle too much at once, which led to mistakes and unnecessary complexity. Going forward, I would adopt a "slow and steady" mindset, dedicating focused time each day to improving the program little by little. Celebrating small wins along the way would also keep me motivated and build momentum toward completing the project.

By focusing on these areas—using tools more strategically, mastering debugging techniques, planning the structure carefully, and taking small, manageable steps—I'm confident that my next project will be more streamlined and enjoyable. These lessons have not only prepared me for future challenges but also given me the tools to tackle them with greater confidence and efficiency.