The Shadow Walker project plan serves as a clear guide for developing a stealth puzzle game. It helps the team move from planning to creating the game step by step. The main goals are to:

* Create smooth player movement using keyboard controls.
* Develop a light and shadow system for puzzles.
* Design innovative enemies that react to light.
* Add switches and objects players can interact with.

The plan also explains:

* Team roles: Who is responsible for what tasks (e.g., coding, art, documents, etc.)?
* Tools needed: Computers, software (C++/SFML), and testing devices.
* Timeline: 6 weeks total, divided into setup, feature development, and final testing.
* Risks: Possible problems (such as delays or technical issues) and their potential solutions.

The team studied past projects to improve their plan. They focused on:

* Make goals easy to understand.
* Explaining risks clearly.
* Organizing tasks in order.
* Use simple formatting so everyone can follow the plan.

Now, a brief introduction of the project. Shadow Walker is a stealth puzzle game where players sneak through a high-tech facility, utilizing shadows to conceal themselves from guards. Players must move carefully and use lights and switches to solve puzzles and avoid being caught.

Key Goals:

* Movement: Smooth controls (WASD/Arrow Keys) on a 2D grid.
* Light/Shadows: A system that changes light in real-time to create puzzles.
* Enemies: Guards that search for players in lit areas.
* Interactions: Switches and objects to change lights and unlock paths.

Challenges:

* Time: Must be completed within 6 weeks.
* Team Availability: Members have other work or study commitments.
* Tech: Built with C++ and SFML; requires optimal performance on older computers.
* Scope: At least one playable level showing core gameplay.

Team Roles:

* Nishant (Manager): Plans schedules, solves problems, and tracks progress.
* José (Documents): Writes plans, notes, and guides to keep everyone informed.
* Tucker (Programming): Designs game rules and logic, including movement and AI.
* César (Graphics): Creates visuals (art, lighting) using SFML tools.

Why This Works:

* Clear roles ensure there is no confusion about who is responsible for what.
* The team shares updates regularly to stay on track.
* Focus on finishing one polished level first.

**Comprehensive Risk Analysis**

Risk analysis is an integral part of project planning that involves identifying potential hazards and taking them into account. For Shadow Walker, we have identified nine major risk categories, each of which is analyzed below with detailed explanations and corresponding mitigation strategies.

1 Cost Risk

Description:  
Cost risk is the possibility of unforeseen expenses that could disrupt the project budget. In this project, the initial decision to use free tools like VS Code and SFML minimizes costs. However, there is always a potential need for advanced tools or software licenses; however, as of today, it does not seem that the project will encounter this issue.

Potential Impacts:

* Budget Overruns: The need for upgraded tools or additional software licenses could lead to unplanned expenses.

Mitigation Strategies:

* Budget Monitoring: Regularly track expenses and maintain a detailed budget log.
* Research Alternatives: Continuously explore free or lower-cost alternatives that could meet advanced technical requirements.

**2 Schedule Risk**

**Description:**  
Schedule risk refers to possible delays that can happen due to unexpected problems, such as difficulties in debugging, team members having other responsibilities, or deadlines from other courses happening at the same time. Since the project must be completed in six weeks, even small delays in one part of the work could cause major setbacks.

**Potential Impacts:**

* **Missed Deadlines:** If important tasks, such as debugging or final testing, take longer than expected, the entire project schedule could be affected.
* **Lower Quality:** Rushing to meet deadlines may lead to less thorough testing or incomplete features, which can compromise the overall quality of the project.

**Mitigation Strategies:**

* **Careful Planning:** Create a schedule that includes extra time for unexpected delays.
* **Regular Check-ins:** Have weekly meetings to track progress, spot potential delays early, and adjust the schedule if needed.
* **Task Prioritization:** Work on the most important features first and follow a "minimum viable product" approach to make sure the main parts of the project are finished on time.

3 Performance Risk

Description:

Performance risk refers to the game's ability to run smoothly, particularly since it relies on real-time lighting and shadow effects. Problems can arise if the game is played on older computers or if complex graphics are not properly optimized.

Potential Impacts:

Lag and Slow Gameplay: If the game is not optimized, it might run slowly, making the experience frustrating for players.

System Crashes: If the game requires too much processing power, it may cause crashes or unexpected errors, especially when advanced effects are used.

Mitigation Strategies:

Early Performance Testing: Start testing performance early to find and fix any issues that slow down the game.

Efficient Optimization: Use SFML’s built-in functions and follow best practices to make the game run smoothly.

Hardware Compatibility Testing: Test the game on different computers to make sure it works well on a variety of systems.

4 Operational Risk

Description:

Operational risk includes challenges that might affect the daily progress of the project, such as accidentally losing data, differences in how team members set up their development environments, or availability issues among team members.

Potential Impacts:

Data Loss: Important files or code might be accidentally deleted, causing major delays.

Inconsistent Development Environments: If team members use different software settings, it can create problems when combining their work.

Work Delays: If one area of the project, such as code integration, gets delayed, it can slow down the entire development process.

Mitigation Strategies:

Version Control: Use a reliable system like GitHub to save different versions of the code and prevent data loss.

Standardized Setups: Make sure all team members use the same development environment and provide clear setup instructions.

Frequent Backups: Regularly save project files in a secure location to prevent data loss.

5 Technology Risk

Description:

Technology risk involves potential problems with the tools and software used for development. Since the team relies on SFML and some members are still learning its advanced features, there could be challenges.

Potential Impacts:

Slow Learning Process: Team members who are not familiar with SFML may take longer to complete tasks.

Integration Issues: Some advanced SFML features may be difficult to use correctly, leading to unexpected errors or compatibility problems.

Mitigation Strategies:

Training Sessions: Set aside time for team members to learn SFML through tutorials and practice.

Early Prototyping: Create small test versions of key features early to make sure they work before adding them to the main project.

Use Online Resources: Rely on forums, tutorials, and official documentation to quickly solve problems.

6 Communication Risk

Description:

Good communication is essential for teamwork. Communication risk occurs when tasks are not clearly assigned, messages are delayed, or important details are not documented.

Potential Impacts:

Confusion and Mistakes: Poor communication can lead to duplicated work or missed deadlines.

Slower Problem-Solving: If team members don’t communicate efficiently, it may take longer to fix issues.

Lack of Coordination: Without regular updates, team members might work separately, making the final project feel disconnected.

Mitigation Strategies:

Use Communication Tools: Platforms like Slack or Microsoft Teams can help keep everyone informed.

Regular Meetings: Schedule short weekly updates and detailed progress meetings to stay on track.

Shared Documentation: Keep a central document with notes from meetings, task assignments, and project guidelines.

7 Scope Creep Risk

Description:

Scope creep happens when a project grows beyond its original plan. In a creative project like Shadow Walker, it is tempting to add more features, improve animations, or expand levels, but this could make it difficult to complete the game on time.

Potential Impacts:

Project Delays: Adding extra features can push back the deadline for completing the core game.

Increased Complexity: A bigger project is harder to manage and test.

Overuse of Resources: Extra work requires more time and effort, which might overwhelm the team.

Mitigation Strategies:

Define Scope Clearly: Decide on the essential features at the start and make sure the team agrees.

Control Changes: Use a formal process to review and approve any new additions.

Focus on Core Features: Prioritize the most important parts of the game and save extra features for later updates.

8 Security Risk

Description:

Security risk involves problems that could affect the game’s stability and reliability. Even though Shadow Walker is a prototype, security issues could lead to crashes, memory leaks, or other failures.

Potential Impacts:

Game Instability: Errors or memory problems could cause crashes during gameplay.

Potential Exploits: If the game were expanded in the future, security flaws could allow unauthorized access or manipulation.

Mitigation Strategies:

Write Secure Code: Follow best practices to prevent vulnerabilities.

Review Code Regularly: Check the code often and use automated tools to find security issues.

Thorough Testing: Perform detailed tests, including stress tests and security checks, to ensure stability.

9 Skills Resource Risk

Description:

Skills resource risk refers to the experience level of the team, especially with new technologies like SFML and advanced C++. If some team members are less familiar with these, development could take longer.

Potential Impacts:

Slower Development: Inexperienced team members might need extra time to complete tasks.

More Debugging Time: Less experience can lead to more errors, which require extra time to fix.

Uneven Work Distribution: If some members struggle with certain tasks, others may need to take on extra work.

Mitigation Strategies:

Encourage Knowledge Sharing: Use pair programming and mentoring to help team members learn from each other.

Dedicated Training Time: Allocate time for learning SFML and advanced C++ techniques.

Use External Learning Resources: Take advantage of online courses, tutorials, and community forums to improve skills.

**6. Hardware and Software Requirements**

**Hardware Requirements**

* **Development Machines:** Each team member will use a personal computer capable of running modern C++ development tools and SFML. A machine with a multi-core processor (quad-core or higher), at least 8 GB of RAM, and a dedicated graphics card is recommended to handle real-time rendering tasks.
* **Testing Devices:** A range of hardware configurations will be utilized for testing purposes to ensure compatibility. This includes older PCs and machines with lower specifications to assess performance on less powerful systems.

**Software and Development Environment**

* **Programming Language:** C++ will be used as the primary language due to its performance efficiency and compatibility with SFML.
* **Graphics Library:** SFML (Simple and Fast Multimedia Library) will serve as the foundation for all graphics, audio, and input functionalities.
* **Development Tools:** Visual Studio Code (VS Code) or a similar IDE will be employed for coding, alongside version control systems such as Git and GitHub for collaborative development and code management.
* **Operating System:** The project is designed to be cross-platform. Development and testing will be performed on Windows, with additional verification on macOS and Linux environments as necessary.

**7. Work Breakdown**

The project is divided into several activities that cover both the implementation and testing of all requirements. Each activity is assigned a specific time estimate and identifies dependencies to ensure logical progression throughout the project lifecycle.

**Key Activities**

1. **Project Planning & Design:**
   * **Duration:** 3 days
   * **Objective:** Define game mechanics, rules, and level progression; set overall project scope.
2. **Set Up C++ & SFML Environment:**
   * **Duration:** 2 days
   * **Objective:** Install SFML, establish project structure, and test a basic window.
3. **Learning Basics of SFML:**
   * **Duration:** 2 days
   * **Objective:** Understand available SFML tools and functionality to prepare for game development.
4. **Design a Basic Layout and Characters:**
   * **Duration:** 4 days
   * **Objective:** Create visual assets and layout designs that will serve as the foundation for the game’s graphics.
5. **Implement Basic Player Movement:**
   * **Duration:** 4 days
   * **Objective:** Develop the logic for player movement, including collision detection and responsive controls.
6. **Develop Light & Shadow System:**
   * **Duration:** 1 week
   * **Objective:** Implement dynamic lighting and shadow rendering to create core gameplay mechanics.
7. **Create Enemy AI (Patrolling Guards):**
   * **Duration:** 1 week
   * **Objective:** Develop guard behavior, including pathfinding and detection based on light exposure.
8. **Implement Light Manipulation Mechanics:**
   * **Duration:** 4 days
   * **Objective:** Integrate interactive elements such as switches and movable objects that allow players to alter the lighting conditions.

**The previous breakdown is an approximation created by our project manager, it may vary a little but it should change that much.**

8. Process Flow Diagrams (Conceptual Overview)

While the detailed UML diagrams have been omitted per the assignment guidelines, a conceptual overview of the process flow is provided below:

* Player Input to Game Response:  
  Illustrates the flow from player actions (via keyboard) to corresponding game responses, including movement updates, collision checks, and triggering of lighting or enemy responses.
* Module Interaction:  
  Details how the various modules—player movement, lighting system, enemy AI, and interactive elements—communicate with each other to produce a coherent gameplay experience.
* Data Flow:  
  Outlines the flow of data between user input, game state updates, and output rendering, ensuring that each step is processed in sequence.

These process flow concepts ensure that all components work in tandem, enabling iterative development and efficient troubleshooting.

9. Project Schedule

The project schedule is organized according to the dependencies identified in the work breakdown. Key milestones are as follows:

* Weeks 1–2:  
  Focus on project planning, environment setup, and learning SFML. Early prototypes for player movement and basic layouts are developed.
* Weeks 2–4:  
  Concentrate on implementing core game mechanics, including dynamic lighting, shadow rendering, and enemy AI. This phase includes the development of interactive elements and initial level design.
* Weeks 4–6:  
  Finalize level designs, integrate UI and sound effects, and conduct thorough testing and debugging. The final review and documentation are completed during this period.

A detailed Gantt chart (not included in this document) will be maintained via project management tools to monitor progress and adjust the schedule as necessary.

10. Monitoring and Reporting Mechanisms

To ensure effective progress tracking and communication throughout the development process, the following mechanisms will be utilized:

* Version Control Systems:  
  Git and GitHub will be used for code management, ensuring that all changes are tracked and that team members can collaborate effectively.
* Project Management Tools:  
  Tools such as Trello or Asana will be employed to assign tasks, track dependencies, and visualize progress across all project phases.
* Regular Meetings:  
  Weekly stand-up meetings and bi-weekly comprehensive reviews will be conducted to discuss progress, identify obstacles, and adjust plans as necessary.
* Centralized Documentation:  
  A shared online repository will house all project documentation, including meeting minutes, technical guides, and change logs, ensuring transparency and ease of access for all team members.
* Progress Reports:  
  Detailed progress reports will be generated periodically to assess milestones, review risk mitigation measures, and adjust the project schedule if delays are detected.

These mechanisms are designed to foster open communication and ensure that any issues are quickly identified and resolved.

11. Appendix: Summary of Activities

The following summary table encapsulates the key activities, estimated durations, and their dependencies. (Note: This table is a textual summary and does not include the risk chart or UML diagrams per assignment guidelines.)

1. Project Planning & Design – 3 days (No dependencies)
2. Set Up C++ & SFML Environment – 2 days (Depends on Activity 1)
3. Learning Basics of SFML – 2 days (Depends on Activity 2)
4. Design Basic Layout and Characters – 4 days (Depends on Activity 3)
5. Implement Basic Player Movement – 4 days (Depends on Activity 3)
6. Develop Light & Shadow System – 1 week (Depends on Activity 4)
7. Create Enemy AI – 1 week (Depends on Activity 5)
8. Implement Light Manipulation Mechanics – 4 days (Depends on Activity 6)
9. Design Levels & Game Flow – 5 days (Depends on Activity 7)
10. Develop UI & Sound Effects – 5 days (Depends on Activity 8)
11. Testing & Debugging – 3 days (Depends on Activity 9)
12. Final Review & Submission – 2 days (Depends on Activity 11)

This summary ensures that every phase of the project is accounted for and that dependencies are clearly articulated.

|  |  |  |  |
| --- | --- | --- | --- |
| **Activity #** | **Activity Description** | **Estimated Time** | **Dependencies** |
| 1 | **Project Planning & Design** – Define game mechanics, rules, and level progression | 3 days | none |
| 2 | **Set Up C++ & SFML Environment** – Install SFML, set up project structure, and test a basic window | 2 days | Activity 1 |
| 3 | **Learning basics of SFML** – getting to know all the tools it offers and how to use them | 2 days | Activity 2 |
| 4 | **Design a basic layout and characters –** this is the SFML part graphics that will be used to play the game | 4 days | Activity 3 |
| 5 | **Implement Basic Player Movement** – Code movement logic (arrow keys), collision detection | 4 days | Activity 3 |
| 6 | **Develop Light & Shadow System (SFML & C++)** – Implement lighting mechanics, shadow rendering | 1 week | Activity 4 |
| 7 | **Create Enemy AI (Patrolling Guards) (SFML & C++)** – Develop guards’ pathfinding and player detection in light | 1 week | Activity 5 |
| 8 | **Implement Light Manipulation Mechanics** – Add switches, movable objects to alter lighting conditions | 4 days | Activity 6 |
| 9 | **Design Levels & Game Flow** – Create at least 3 levels, define difficulty progression | 5 days | Activity 7 |
| 10 | **Develop UI & Sound Effects** – Add HUD, stealth meter, interaction prompts, and sound effects | 5 days | Activity 8 |
| 11 | **Testing & Debugging** – Fix bugs, optimize code, adjust difficulty balance | 3 days | Activity 9 |
| 12 | |  | | --- | |  |  |  | | --- | | **Final Review & Submission** – Prepare documentation, finalize project files | | 2 days | Activity 9 |

This table summarizes the work breakdown in an easier way, and the appendix in a good format.

**13. Conclusion**

The *Shadow Walker* project is an exciting yet well-organized effort to develop a stealth puzzle game using C++ and SFML. This document has outlined a clear project plan, covering key aspects such as risk analysis, task breakdown, scheduling, and progress tracking. By identifying possible risks—such as budget constraints, time limitations, technical challenges, and security concerns—and creating strong solutions to address them, the team is well-prepared to handle obstacles and complete the project successfully.

Our structured planning, careful organization, and proactive risk management are designed to ensure the development of a high-quality and engaging game prototype within the six-week timeframe. The inclusion of dynamic gameplay features, realistic enemy AI, and innovative light and shadow mechanics creates a well-balanced mix of technical skill and creative design. Through continuous development, regular testing, and effective communication, the team is committed to achieving project goals while maintaining high standards of performance and quality.

This document serves as both a guide for the *Shadow Walker* project and an example of the detailed planning needed for real-world IT projects. We hope that this project will not only succeed in its technical execution but also provide meaningful learning experiences that help us grow as developers.

13. References

American Psychological Association. (2020). Publication Manual of the American Psychological Association (7th ed.).