Test Plan

1. **Introduction**

Test Plan Objectives

* + **To make sure the program complies with the functional system requirements specified in the SFT221 project guidelines, test it. Verify that the functionality and features that have been implemented comply with the given requirements.**
  + **To find any possible problems, faults, or flaws in the programme, perform extensive testing. Send an accurate and timely report of these results to the development team for resolution.**
  + **Check how well the program responds to different user inputs, including legitimate, erroneous, and edge situations. Verify that the programme performs input validation properly and that the necessary error messages are presented.**
  + **Analyse the software's performance under both average and high load scenarios. Make that the system can manage the anticipated volume of concurrent users or transactions and operates within acceptable response times.**
  + **Conduct stress testing, load tests, and endurance tests to confirm the software's stability and dependability. Find any problems with memory leaks, resource use, or system crashes.**

1. **Scope**

**All software functionalities will be tested as part of the test plan. The plan will outline the integration, end-to-end, regression, and unit testing schedules. The strategy will comprise performance and security testing to make sure the program is bug-free as well as testing of all system use case specifications provided in the project instructions.**

1. **Test Strategy**
   1. **All the testes mentioned below may or may not be applicable to testing of every project, and there may be additioinal tests specific to the project that are not mention here.**
      * **System Test: These tests will verify that software works as intended in the integrated system environment. This means that each individual part works properly when put together and gives the desired output. This is done after each module is tested on its own individual level.**
      * **Performance Test: Each program gets different load conditions through their lifetime, this is tested by the Performance testing. It evaluates the responsiveness and how many resources are being used by the program. This also might involve Stress and volume Test in it.**
      * **Security Testing: This type of testing will assess the ability of the software to protect them against security issues like data breaching and unauthorized access. This involves testing for authentication of the user, whether the data encryption is taking place properly or not, and many other security measures to make the software secure.**
      * **Stress and volume test: This involves testing the software under high pressure and extreme conditions such as high user loads or limited system resources. This will test the limits of the program and helps in identifying the weaknesses of the program.**
      * **Beta test: This involves releasing the software to a limited number of real-world end-users(customers) to test how the software performs in the real environment. This helps in getting the user feedback and identifying issues that might not have been identified during testing and validating the software's performance, functionality, and compatibility in diverse operating environments.**
2. **Environment Requirements**

* A Windows machine capable of running Visual Studio Community is necessary. The tests and tools mentioned below are designed to work within the Windows environment.
* Running the Unit Testing framework included with Visual Studio Community requires this software. It emphasises the necessity of a Windows PC for the tests by stating that it is only compatible with Windows.
* Our project management software, JIRA, will be used to plan and oversee the testing procedures. We can handle test creation, execution, and documentation thanks to it. It is necessary to be familiar with JIRA and how to use it.
* The final test documentation, reports, and project source code will all be stored and shared via GitHub at each milestone. It offers the team a platform for communication and version control.
* OneDrive for Scrum report and other plan collaboration.

1. **Execution Strategy**
   1. The execution strategy for the Heavy Chests feature would involve a set of entry and exit criteria to ensure its successful implementation. To begin with, the feature must meet certain entry criteria, including completion of its development, undergoing unit testing and integration testing. Once we are done with the entry criteria the test execution would start. In this we design test cases to validate the functionality covering various points such as adding and removing heavy items, testing capacity limits, and examining error handling mechanisms. Testers would record the results of each test case, documenting any defects encountered during the process. Now the exit criteria for the test execution phase would consist of several conditions that need to be met. First, a predefined percentage of test scripts must pass successfully. For example, as it was given in the instruction 95% of the test scripts pass, the test execution can be considered complete. Moreover, all critical and high severity defects identified during testing must be fixed and verified to ensure the stability and reliability of the feature. The remaining medium and low severity defects can either be addressed immediately or documented for future consideration.
   2. You can describe the severity of defects in this section and break them down into severity levels of:

All the bugs and errors which are not getting resolved can be informed to team by creating an issue on Jira or while team meetings.

* + 1. Critical : This has the potential to stun the development. Most important can affect the working of the system! So they need the most attention and more priority.
    2. High: has a substantial danger of terminating the programme completely. The right, supported user input is one potential remedy, but high-severity problems should be fixed right away.
    3. Medium : Although the danger may not have a significant effect on the system, it is nevertheless important to correct the flaw since it might later cause additional issues.
    4. Low: Unimportant bugs are present. They are extremely uncommon and carry a minimal risk of user discovery.
  1. **Test Reporting**
     + The reports should be produced regularly, and their contents and distribution. For this project it a member would have to submit a report which would include the overview of the testing activities as their status in the report as well as update it on JIRA, report the defect and bugs found during the testing, overall progress and any deviation in the plan, and summary of the whole thing. This report should be submitted to the team lead (JAY VAKIL) and the reporting frequency would depend upon each week's tasks and deadline which the team lead would inform you on a weekly basis after the project is released.
     + The tester and the project manager would need to establish a good communication channel for efficient bug resolution by using collaborative tools such as project management software (JIRA), instant messaging, or email for updates and communication.
  2. This can also be done through establishing communication channels between the two to address any questions or clarifications related to the reported defects. This can be done through various means such as issue tracking systems, or integrated development environments (IDEs) that facilitate communication and tracking of defect resolution progress, email, GitHub or dedicated team communication channels. Regular and open communication helps in resolving queries and ensures smooth collaboration.

1. **Test Schedule**  
   Regarding Scheduling, we aim to complete the testing protocols one day before the submission or, in other words, we aim to complete the requirements and testing in 3 days.
2. **Control Procedures**

This section outlines the processes and practices that will be followed to manage and control the testing activities throughout the software development lifecycle. It also ensures that the testing process is well-organized, standardized, and effectively managed to achieve the desired test objectives within the allocated resources and timeframe. You could include some of these sections:

* Reviews - review of the test plan to ensure it aligns with project objectives, adequately covers test objectives, and follows best practices. Along with that Review test cases to verify their correctness, completeness, and relevance to the requirements.
* Test Environment - set up and configure the test environment, including hardware, software, databases, network configurations, and test data according to this test plan.
* Bug Review Meetings and Progress Monitoring – Weekly bug review meetings will be conducted with the QA team and development team to discuss reported defects, prioritize them based on severity and impact, assign them to developers, and track their progress. Also check kanban board regularly to know what new task have been assigned to the members.
* Defect Reporting – We'll be using a centralized defect tracking system to log, track, and manage reported defects. Defects will be reported following predefined standards, including required information such as steps to reproduce, expected and actual results, severity levels, and categorization. Clear guidelines will be provided to ensure consistent and detailed defect reporting.
* Test Execution Schedule - Progress of test execution will be tracked against the planned schedule. Test status, including completed, in progress, and pending tests, will be regularly updated and shared with stakeholders. Any deviations or delays will be promptly addressed and communicated.
* Documentation - Standard templates will be used for test plans, test cases, test scripts, and test reports. The documentation will follow a consistent format, including sections for test objectives, test steps, expected results, and actual results. A version control system, such as Git, will be utilized to manage test documentation. Proper version control practices will be followed to track changes, maintain document history, and ensure the availability of the latest versions.

1. **Functions To Be Tested**

**These are the few functions which havve to be tested:**

* + struct Map populateMap(): This function populates and returns a structure Map with a predefined map configuration.
  + void printMap(const struct Map\* map, const int base1, const int alphaCols): This function prints the map in a formatted manner, including the symbols representing different elements on the map.
  + struct Map addRoute(const struct Map\* map, const struct Route\* route): This function adds a given route to the provided map and returns a modified map.
  + double distance(const struct Point\* p1, const struct Point\* p2): This function calculates the Euclidean distance between two points.
  + struct Route shortestPath(const struct Map\* map, const struct Point start, const struct Point dest): This function finds the shortest path between the start and destination points on the given map using a modified version of Dijkstra's algorithm.
  + struct Route getPossibleMoves(const struct Map\* map, const struct Point p1, const struct Point backpath): This function returns a route containing all possible moves from a given point on the map, excluding the point provided in the backpath.
  + int getClosestPoint(const struct Route\* route, const struct Point pt): This function finds the index of the closest point to a given point within the route.

1. **Resources and Responsibilities**

|  |  |
| --- | --- |
| Jay Vakil | Manage and distribute tasks, assist with editing, and provide advice. Part of Software development team as well as testing team. Assist the testing team as well as the development team. Every time fresh work is required, one should create issues in Jira and assign themselves tasks there. |
| Mohit Sheth | Acts as a sub-group leader, ensures the proper accuracy of task distribution. Is a part of software developing team |
| Harsh Dugar | Responsible for doing analysis and developing a deep understanding of the solutions built by the software developers in order to develop test cases. |
| Srujal Patel | Responsible for doing analysis and developing a deep understanding of the solutions built by the software developers in order to develop test cases. |
| Hetav Mamtora | Part of a software development team, ensuring proper fulfilment of he project requirements. |
| Viren Vaishnav | Responsible for doing analysis and developing a deep understanding of the solutions built by the software developers in order to develop test cases. |

**All the team members are bond to participate actively in team meetings and both the software and testing teams are responsible for communicating and sharing information and data about the projects.**

1. **Deliverables**

**This section lists the specific documents, artifacts, or outputs that will be produced as part of the testing process. In our case we need the Test Plan, the Traceability Matrix, Test Cases (and test data), the Test Report, and the actual source code for the final project.**

1. **Suspension / Exit Criteria**

**All test cases and scenarios specified in the test plan will be meticulously carried out to decide when testing is complete. The test results will be carefully examined to ensure complete coverage of the system's features, parts, and edge cases. When all test data has been run and at least 90% of the tests have passed successfully, the testing's completion requirements have been fulfilled.**

1. **Resumption Criteria**
   1. **Testing activities may be temporarily suspended under the following circumstances:**
      * Discovery of critical defects that significantly impact the stability or functionality of the system.
      * Changes in project priorities or resource allocations that require a reallocation of testing resources.
      * Unavailability of necessary testing resources, such as hardware, software, or test data.
   2. Authority to Suspend Testing:

The project manager and the assistant project manager would have the authority to make the decision to suspend testing based on the inputs and recommendations of the test lead and other key stakeholders.

* 1. Conditions or Criteria for Resuming Testing:
     + Critical defects impacting system stability or functionality have been addressed and resolved.
     + Adequate testing resources, including hardware, software, and test data, are available.
     + Test environment stability has been restored and validated.
  2. If the resumption criteria cannot be met within a specified timeframe, the following escalation path and responsibilities will be followed:
     + If the test lead determines that the resumption criteria cannot be met, they will escalate the issue to the team lead.
     + The team lead will assess the situation, review the resumption criteria, and consult with other members of the group to determine the next course of action.
     + If necessary, the team lead may convene a meeting or escalate the issue to higher management (in our case the professor) for further guidance or resolution.

1. **Dependencies**

12.1 Personnel Dependencies - The testing team should consist of testers with adequate knowledge of various testing techniques and tools. Along with that Effective communication channels and regular meetings are very important.

12.2 Software Dependencies - The latest version of the software is should be available to the testing team, either through local installations or remote access through Seneca’s MyApps.

12.3 Hardware Dependencies – If the team does not have adequate hardware resources, they can use the College computers which have the proper hardware configurations and are accessible to the testing team to support the execution of tests.

12.3 Test Data & Database - The necessary test data should be available for executing test cases, covering various scenarios and ensuring realistic testing. The required databases should be properly set up, configured, and made accessible for testing purposes.

1. **Risks**

**13.1. Schedule**

Meetings, deadlines, and crucial information may be missed while trying to fulfil strict deadlines and deal with team members' absences. The project's overall quality and completion date may be endangered as a result.

**13.2. Technical**

Team members' ability to contribute successfully may be hampered by technical problems with their hardware or software. Collaboration and fast communication within the team are essential to reducing this risk. Immediately address any challenging project components through group discussions and problem-solving sessions.

**13.3. Management**

Team members' productivity and the accomplishment of the project may be impacted by poor leadership or unclear guidance. To prevent the project's quality from being compromised, accurate time estimation is crucial.

**13.4. Personnel**

Delays and coordination problems can be caused by poor communication and failing to alert team members to unfinished tasks or concerns. In order to handle issues quickly and guarantee smooth progress, maintaining proactive and open communication is essential.

**13.5 Requirements**

Misaligned expectations and probable rework may result from inadequate analysis and understanding of the project's needs and outputs. This risk may be reduced and project success can be guaranteed with careful study and definition of the requirements.

1. **Tools**

**The testing procedure will make use of the Native Unit Testing Module for Visual Studio Community, as well as potent software tools like JIRA, GitHub, and Microsoft Office. These tools offer crucial features for organising the testing procedure, monitoring issues, working together on code repositories, and recording test results. In-depth information on these software tools may be found in the section about the environment.**

1. **Documentation**

**Test Plan: A description of the testing strategy, goals, and timetable.**

**Blackbox and whitebox test results for Milestones 3 and 4 in unit testing matrices.**

**Unit Test Code: Using C and the Native Unit Testing Module, implement the blackbox and whitebox tests.**

**Bug fixes: identifying and documenting fixed problems during development.**

**Test data and execution information for integration testing are shown in the integration testing matrix.**

**Test data and execution details for acceptance testing are shown in the acceptance testing matrix.**

**Final Test Report: A summary of executed tests, issue patches, and the outcomes of the last round of testing.**

1. **Approvals**

**Through a team meeting, everyone will endorse the test results. To make sure the testing was done properly, we will thoroughly review the test findings during the meet-up.**