# Management Plan

ThunderCats

By Daniel Tompkins & Gregory Kniaziuk

# Table of Contents

Cover Sheet 1

Table of Contents 2

Acronyms 3

Roles & Responsibilities 3

Risk Definitions 4

Risk matrix/mitigration 5

Code repo/proj management 7

Timeline/production 8

Process 9

# 

# 

# Acronyms

GUI - Graphical user interface

TDD - Test-driven development

INI - Initialization, a file type that provides targeting data

Repo - repository

# Roles & Responsibilities

Manager - project planning and prioritization, manage commercial side of development.

Responsible: Daniel Tompkins

Stakeholder - customer, communicates with the team, knows requirements.

Responsible: Brian Lamarche

Code master - manages GitHub repositories.

Responsible: Greg Kniaziuk

Time Management master - helps the team to reach consensus and stay focused.

Responsible: Greg Kniaziuk

Developer - writing tests, implementing features, working with code, commits code.

Responsible: Greg Kniaziuk, Daniel Tompkins

Tester - writes edge cases for testing components. Validates features.

Responsible: Daniel Tompkins

# Risk definitions

Degree of severity

high - Will severely affect project grade

normal - Will affect the project grade

low - Could cause small problems

trivial - Negligible or ignorable

Priority of task

5 - fix first

4 - fix second

3 - fix third

2 - fix fourth

1 - fix last

Resolution Type

Avoid - ignore

Reduce - break it down into smaller problems

Share - outsource or get assistance

Retain - keep the problem and work on it

Likelihood of problem occurring

5 - 80-100%

4 - 60-79%

3 - 40-59%

2 - 20-39%

1 - 0-19%

# Risks matrix/mitigations

What follows is a list of 25 risks that we have evaluated and planned for. The mitigations are defined in more detail following the matrix.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem | Risk | likelihood | priority | mitigation |
| illness | High | 2 | 2 | retain\_01 |
| power outage | High | 1 | 1 | avoid\_01 |
| turret breaks | High | 1 | 1 | avoid\_01 |
| software compatibility | High | 1 | 5 | retain\_02 |
| GIT problem | High | 5 | 5 | retain\_03 |
| finish late | High | 2 | 5 | reduce\_01 |
| can’t target | High | 3 | 4 | retain\_08 |
| integration (code) | High | 2 | 5 | retain\_04 |
| can’t control turret | High | 1 | 5 | retain\_09 |
| hardware failure | High | 1 | 1 | avoid\_01 |
| live feed failure | Moderate | 1 | 1 | retain\_10 |
| bad code | Low | 1 | 3 | reduce\_02 |
| calibration | Moderate | 4 | 2 | retain\_11 |
| Misses target using INI | Low | 2 | 1 | avoid\_02 |
| can’t find range | Low | 2 | 2 | retain\_12 |
| GUI crashes | Moderate | 1 | 3 | retain\_05 |
| integration (hardware) | Moderate | 1 | 1 | avoid\_01 |
| can’t create GUI | Moderate | 2 | 4 | share\_02 |
| can’t play sound | Low | 1 | 2 | retain\_13 |
| turret doesn’t stop | Low | 2 | 2 | retain\_14 |
| no target outlines | Low | 1 | 2 | retain\_15 |
| doesn’t accelerate | Low | 1 | 2 | retain\_16 |
| timer won’t reset | Low | 1 | 2 | retain\_07 |
| won’t fire | High | 1 | 5 | retain\_06 |
| can’t multithread | Low | 2 | 4 | share\_01 |

Mitigations:

avoid\_01 - These are problems we can do nothing about so they will be ignored.

avoid\_02 - The INI is our backup plan for if all else fails. We are not going to worry about a backup plan for the backup plan.

retain\_01 - We will ensure we both have access to all code being developed, and we will update each other on progress several times a week.

retain\_02 - Once we have code written and running, we will get permission to run it on the computer that will be used during the test. We will do this several times along the way to make sure that, as we build, we haven’t inserted anything that won’t run on the final computer.

retain\_03 - One member of the team will be assigned to monitor the repository to make sure that everything is being pushed to the correct place and that any mistakes are reversed before they snowball. In addition, we will develop a plan for how we are going to handle separate branches and when we will merge back to the trunk.

retain\_04 - When we start writing code, each function and program will be given specific inputs and return types. This will help maintain compatibility. We will also both be using the same version of Visual Studio with the same plugins. If one of us finds a new one to implement, we will both install it and keep records to make sure it will then work on the final computer.

retain\_05 - The GUI will just be one interface available for our final project. The program should be runnable in the command line as well with written commands used to start, stop, rotate, etc.

retain\_06 - As soon as we start writing code we will make sure that our fire command will correctly translate to the turret, and we will ensure that the turret we are using is physically able to fire.

retain\_07 - If the timer will not reset automatically during operation, we will have a specific command within the code that will stop and reset the timer. This will allow us to manually halt the time if the automatic stop fails.

retain\_08 - If the turret fails to acquire targets during operation, the first backup plan will be to switch to using INI data to find the targets. To reduce risk, we will get color data on the targets in the room where the final test will be.

retain\_09 - To reduce risk of this we will have the turret controllable through the GUI as well as the command line.

retain\_10 - The primary backup if the video feed fails is using INI data to locate targets. We will test the video feed with our webcam plugged into multiple computers, including the final one.

retain\_11 - Once the video feed it running we will calibrate the turret to fire at a particular location on screen. We will also verify what out zero location is for the purposes of targeting with INI data. We will also place a small piece of tape externally on the turret to show the zero point.

retain\_12 - Our intention is to base range of the size of the target on screen. The size on screen will be compared to data we get by placing the targets at preset distances from the webcam, and it will use the closest. If it cannot find a range to use, it will default to firing at the top of the target.

retain\_13 - The sounds will be set to play automatically at specific points in the program. These will be tested and packaged with the program multiple times throughout development and on the final machine. It will be tested thoroughly, but will not be a priority.

retain\_14 - The turret should stop automatically after firing all shots or two minutes passing. There will be a check within the code to look for either situation. When either is true, the stop command will initiate. This will be tested on the final system. If the turret does not stop, the manual stop can be used on the GUI or on the command line.

retain\_15 - the target outlines should appear on the screen after a target has been identified in the video feed. This will require the correct operation of several plugins. It is a lower priority because of all of the functions and plugins that will have to work first. To make sure it comes up at least for part of the targeting, it will be programmed to appear around the center of the screen after the turret is centered on a target just in case it failed to come up while the turret was moving.

retain\_16 - The turret’s motion will be set to accelerate after the button on the keyboard or GUI has been held for more than a set amount of time. This timer trigger will be similar to the turret auto-stop after two minutes. To reduce risk, it will be tested throughout development and on the final machine.

share\_01 - This topic should be covered in class, so we will keep it a low priority until it is discussed.

share\_02 - The basic GUI is created, but not much has been covered in class, so we will wait to expand the current one until we have discussed it more in class.

reduce\_01 - To avoid finishing late we will use milestones on GitHub for individual tasks. This way we are timing individual tasks in addition to the whole project.

reduce\_02 - To keep our code from being terrible we will look at tabs, variable names, function names, file names, and commenting. Each of these will be discussed individually so that we can come to a consensus on the best format for us. This will be our coding standard.

# Code repo/project management

For this project, we will be controlling our source code with Git. The distributed system will allow us to maintain full copies of the code on our own computers and make changes at will. There files will all be kept on a master branch, with both of us maintaining a separate branch for changes we are making. After finishing a fix or change, we will merge back to the master branch and test the whole system. Git will allow us to step back in the event that we merge faulty code to our main branch.

The central repository we will use is located on GitHub. It is called ThunderCat, like our team, and can be found at <https://github.com/proreco/ThunderCat.git>. This is where we will keep the files for submission, but we will also regularly make copies on our own machines in case GitHub crashes or somehow erases our files.

Since we are already going to be storing our code on GitHub, we will also use it to manage our project. To track tasks we need to complete, we will use issues. This will allow us to prioritize individual tasks, and by assigning issues we can reduce the risk of anything slipping through the cracks and be sure only each student knows what their job is. Milestones will also help us keep the project on time.

For submission, Visual Studio files specified in the assignment will be added to a compressed file and moved to an alternate location for testing. Before submission:

1. Each team member will put the zip file in a fresh location for testing.
2. Each team member will review the requirements to be sure all are implemented.
3. Each team member will run the executable from the command line or file folder after building it in the fresh location.
4. If time allows, the zip file will be given to the instructor early to ensure no build or run issues will occur during grading.
5. Finally the code will be tested for bugs and holes such as invalid files and commands.

# 

# Timeline/production

Our first sprint is as follows:

- become familiar with Visual Studio 2 hours

- select webcam 1 hour

- build basic GUI 3 hours

- verify turret operates correctly 1 hour

- link computers to github repository 1 hour

- set coding standards 2 hours

- get plugins for sound and video 2 hours

Our planning will be tracked on GitHub. We will set individual tasks and issues and larger blocks will be set as milestones. Each milestone will have a deadline and will contain multiple issues. Once we have our tools in place (Visual Studio, GitHub, the turret, and our video package) we will start building the objects that will be used for the project. Then we will fill them in one by one, taking into account which ones depend on other to function correctly.

# Process

TDD is a process that consists of a few stages: first the developer writes an automated test case that defines a particular part of software, then writes a code to pass that test and then refactors the code. To run sets of tests the developer can use testing frameworks, such as xUnit. xUnit are softwares that used for managing the tests.

Using unit testing we can test a particular element (units) of the software by writing code that tests it. The frameworks allow to test units of software, like classes or functions. Unit testing is a convenient tool us to use because xUnit framework provides automated solutions so the same test can be reused many times, without the need to remember the results of these tests.

NUnit is an example of xUnit frameworks made for Microsoft .NET Framework. NUnit can be use to set tests and to proof concepts that will emerge in the future.