|  |
| --- |
|  |
| Project 2: Write-up |
| ThunderCat |
| Daniel Tompkins & Gregory Kniaziuk |

|  |
| --- |
|  |

Table of Contents

Description 2

GUI Storyboards 3

User Narratives 7

Formal Use Cases 8

UML Modeling Diagrams 12

Class Diagram 12

Use Case Diagram 14

Sequence Diagram 14

Design patterns used 15

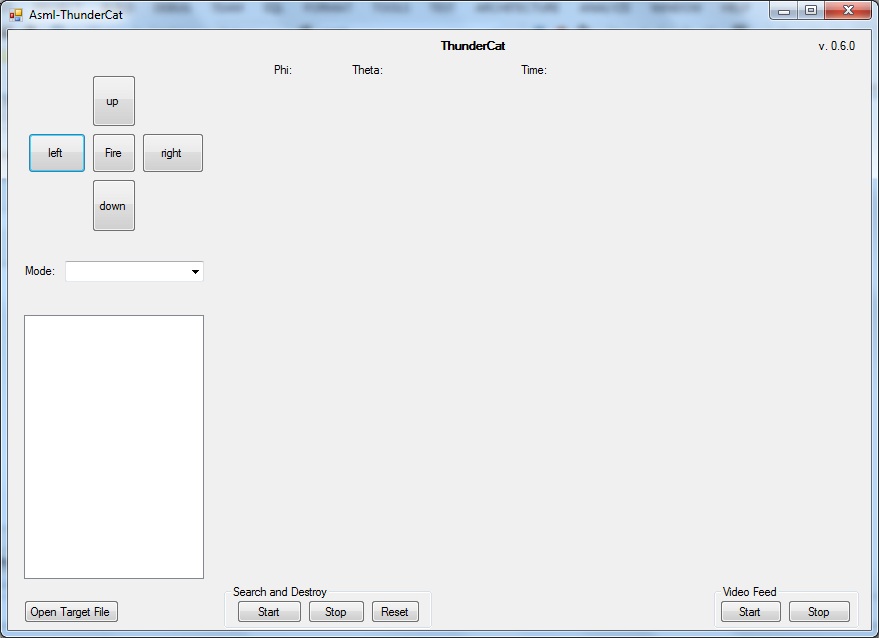
Design considerations 15

Issues 16

Description

This document contains the details surrounding our Graphical User Interface (GUI) for controlling the turret that we will use in the final project for CPTS 323. The information here was used in the creation of the GUI for this project, and will continue to be used as more parts are added. The GUI storyboards will cover the possible states of the interface to make sure button function makes sense and doesn’t allow the user to end up in an invalid state. The User Narratives and Use Cases are being used for testing and design. These will cover the normal and potential use of the program so that we can be sure to include the correct functionality and test for potential problems. UML diagrams and design patterns will describe the objects within the program and how they are working together. The design patters will focus on the general uses that we apply, and the diagrams will show how they were implemented in the code. Design Considerations will be an elaboration on the previous section that discusses other considerations we will be making as we continue on the project. The final section, issues, covers our issue tracking during the making of this project.

GUI Storyboards

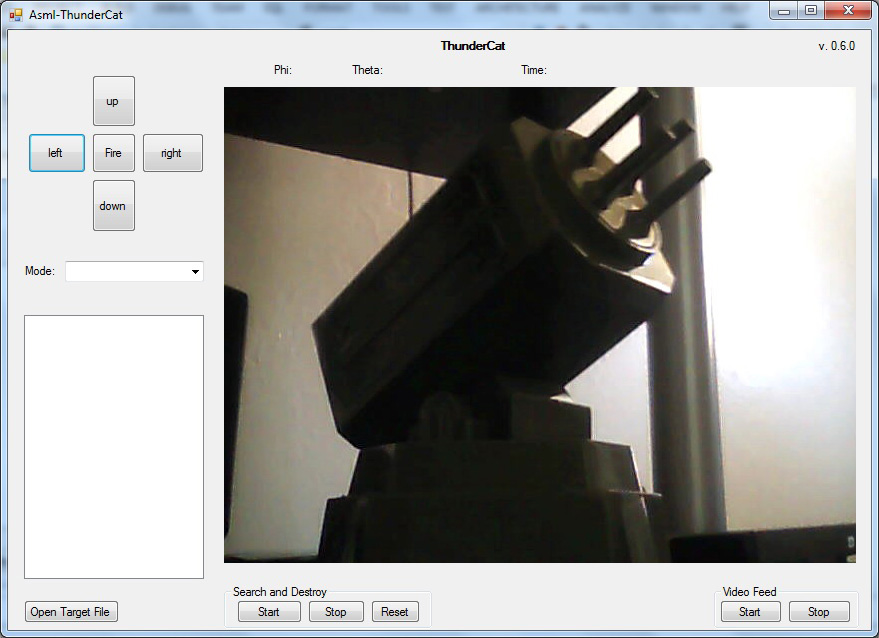


Group name is displayed in the top-center and the version is displayed in the top-right corner of the GUI. In the top portion of the GUI Phi and Theta coordinates are displayed, as well as the timer.

On the left side of the GUI and from top to button are manual controls of the turret, the mode selector, Start button, Stop button, and Reset button, the Target list box and Open Target File button. Directional controls are left, right, up, and down, when one of the buttons is held, the turret moves in that direction and phi and theta angles change accordingly. Button in the center of directional controls is the fire button, when pressed the turret should fire a missile in the current direction. The mode selector allows a user to choose the mode.

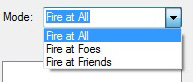
On the right side of the GUI there is a window for the live video feed and two buttons under the feed are start and stop buttons for the feed.

On the bottom of the GUI controls for the Video Feed and the Search and Destroy are located. The Search and Destroy: Start button starts the Search and Destroy mode. Stop button should stop the Search and Destroy mode. Reset button calibrates the position of the turret by setting the gun at 0, 0.

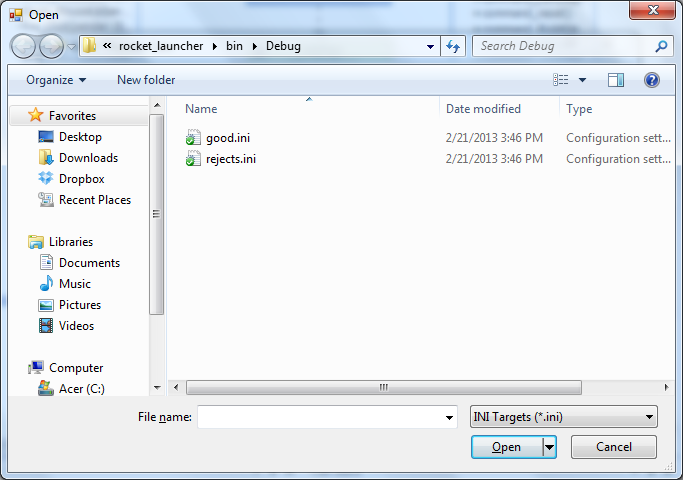
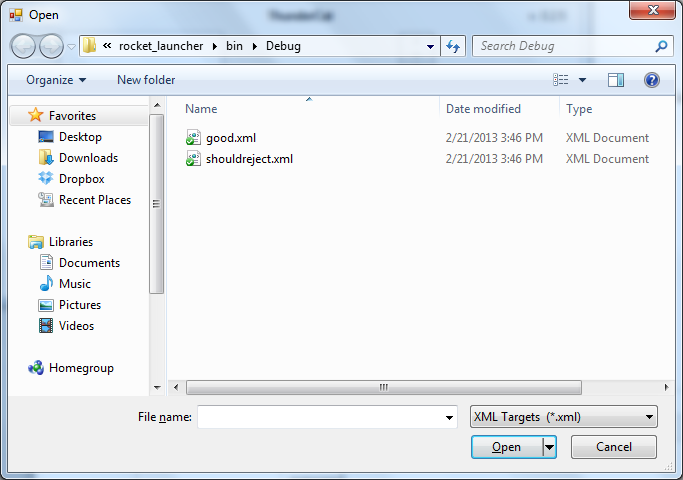
When the start button for the video feed is pressed, live video feed is displayed:

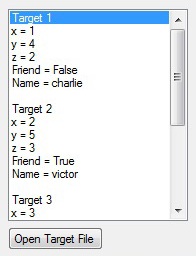
When the stop button is pressed the feed stops.

The mode selector can be used to choose one of the modes, fire at all, fire at foes or fire at friends:

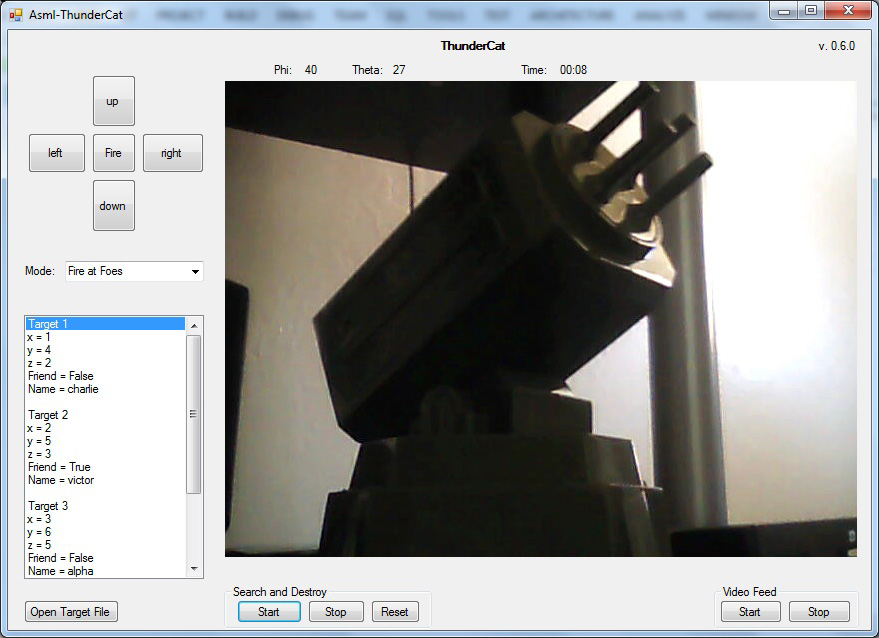


When Open Target File button is pressed, file explorer window opens and lets you choose a target file of one of the formats, xml or ini. Examples for both formats are below:

After file is chosen, it gets red by the program and the acquired targets are displayed in the Target list box:



When the Start button for the Search and Destroy is pressed the turret goes in to the Search and Destroy mode, it moves from one target to another and fires at them depending on the mode, the coordinates and the timer are displayed.



User Narratives

There are two basic users for this program: builders and judges.

Judges will run the code for final testing. They are concerned with the overall requirements. Potential operations include starting the program from explorer or command line, manually rotating and firing the turret, loading valid and invalid target files, and running search and destroy with various numbers of targets. The builders will do these as well as attempting to find holes in the code. They will run search and destroy with 0 – 5 targets, load in target files that include coordinates outside of the physical capabilities of the turret, and test individual portions of code using Visual Studio.

Formal Use Cases

|  |  |
| --- | --- |
| **ID:** | **UC-1** |
| **Title:** | Fire Missiles from idle |
| **Description:** | The AMSL should fire missiles when the fire button is pressed. |
| **Primary Actor:** | Judge |
| **Preconditions:** | Software has started in idle mode |
| **Postconditions:** | Missile is fired |
| **Main**  **Success Scenario:** | 1. User clicks fire button  2. Turret fires missile in current direction  3. Program makes a firing noise |
| **Extensions:** | 1a. Nothing happens  1b. Turret fires more than one missile |
| **Frequency of Use:** | For testing |
| **Owner:** | Daniel Tompkins |
| **Priority:** | 5 – High |
| **Risk** | Retain\_06 Retain\_13 |

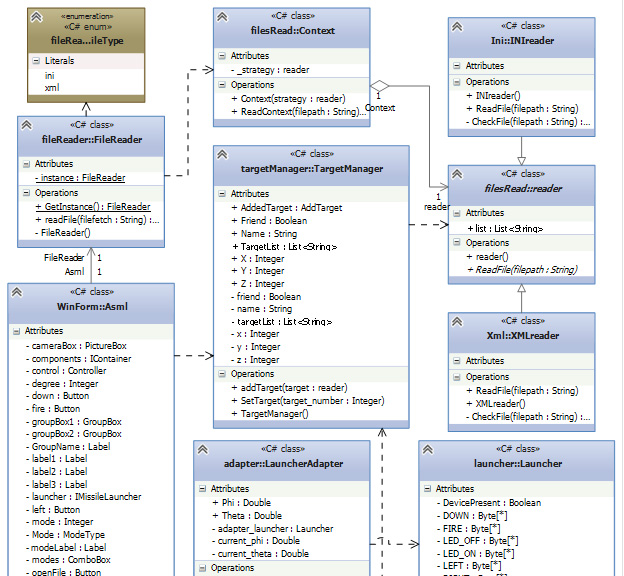
|  |  |
| --- | --- |
| **ID:** | **UC-2** |
| **Title:** | Fire Missiles from Search and Destroy |
| **Description:** | The AMSL should fire missiles when a target is acquired. |
| **Primary Actor:** | Judge |
| **Preconditions:** | Software has started in Search and Destroy mode |
| **Postconditions:** | Missile is fired |
| **Main**  **Success Scenario:** | 1. Program identifies a target  2. Turret rotates to point directly as target  3. Turret fires a missile |
| **Extensions:** | 1a. Mis-identifies Friend as Enemy  1b. Wrong coordinates given  2a. Turret position not calibrated correctly  3a. Missile misses target  3b. Missile hits a friendly target |
| **Frequency of Use:** | For testing and final competition |
| **Owner:** | Daniel Tompkins |
| **Priority:** | 5 – High |
| **Risk** | Retain\_06 Retain\_11 |

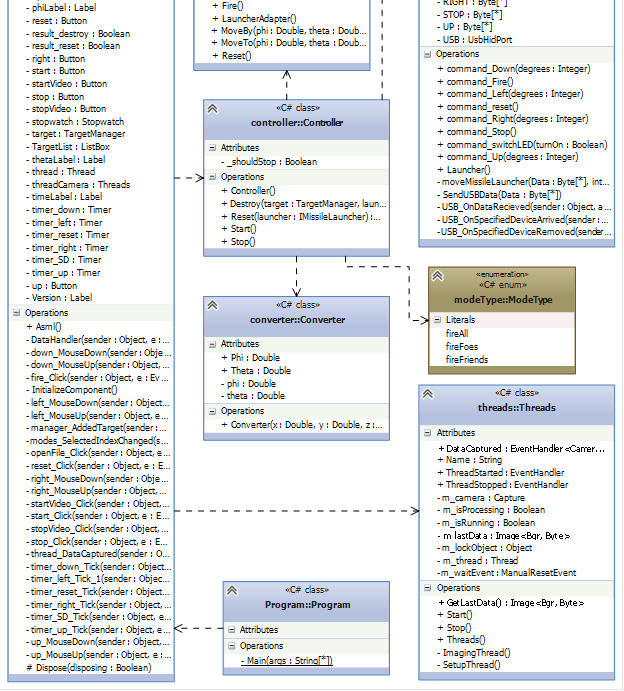
|  |  |
| --- | --- |
| **ID:** | **UC-3** |
| **Title:** | Search and Destroy |
| **Description:** | The program should start the timer, fire at targets, depending on the mode, and stop the timer |
| **Primary Actor:** | Judge |
| **Preconditions:** | Software has started in idle mode |
| **Postconditions:** | All missiles fired, timer stopped, software in idle |
| **Main**  **Success Scenario:** | 1. User clicks on start 2. Timer starts   3. After the turret fired at all targets timer stops and programs goes into idle mode |
| **Extensions:** | 1a. Program fails to start Search and Destroy  2a. Timer doesn’t start  3a. Timer fails to stop  3b. Timer does not reset |
| **Frequency of Use:** | For testing and final competition |
| **Owner:** | Daniel Tompkins |
| **Priority:** | 5 – High |
| **Risk** | Retain\_14 Retain\_07 |
| **ID:** | UC-4 |
| **Title:** | Identifying Targets from files |
| **Description:** | The program should import target information from an .ini or .xml file |
| **Primary Actor:** | Judge |
| **Preconditions:** | Software has started in idle mode |
| **Postconditions:** | Target information stored and ready for Search and Destroy |
| **Main**  **Success Scenario:** | 1. User clicks open target file and selects a target file 2. Program extracts target information |
| **Extensions:** | 1a. File does not exist  1b. File does not have .ini or .xml extension  2a. File is not a valid .ini or .xml  2b. Files does not contain any targets |
| **Frequency of Use:** | For testing and final competition |
| **Owner:** | Daniel Tompkins |
| **Priority:** | 5 – High |
| **Risk** | Retain\_06 |

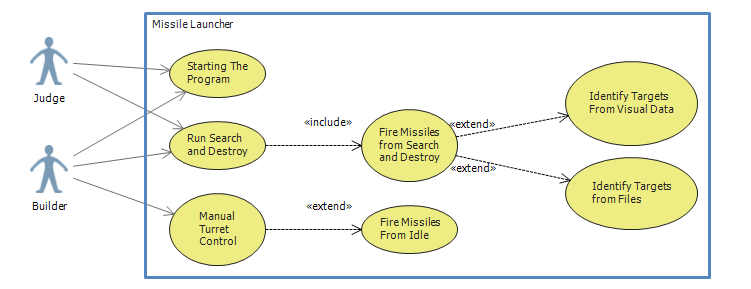
|  |  |
| --- | --- |
| **ID:** | **UC-6** |
| **Title:** | Starting the program |
| **Description:** | When the program is run, the GUI should launch, allowing the user to control the turret |
| **Primary Actor:** | Judge |
| **Preconditions:** | The program is built on the computer |
| **Postconditions:** | GUI is loaded and user can control turret |
| **Main**  **Success Scenario:** | 1. User runs the executable 2. GUI loads |
| **Extensions:** | 2a. GUI fails to load |
| **Frequency of Use:** | For testing and final competition |
| **Owner:** | Daniel Tompkins |
| **Priority:** | 2 – Low |
| **Risk** | Retain\_05 |

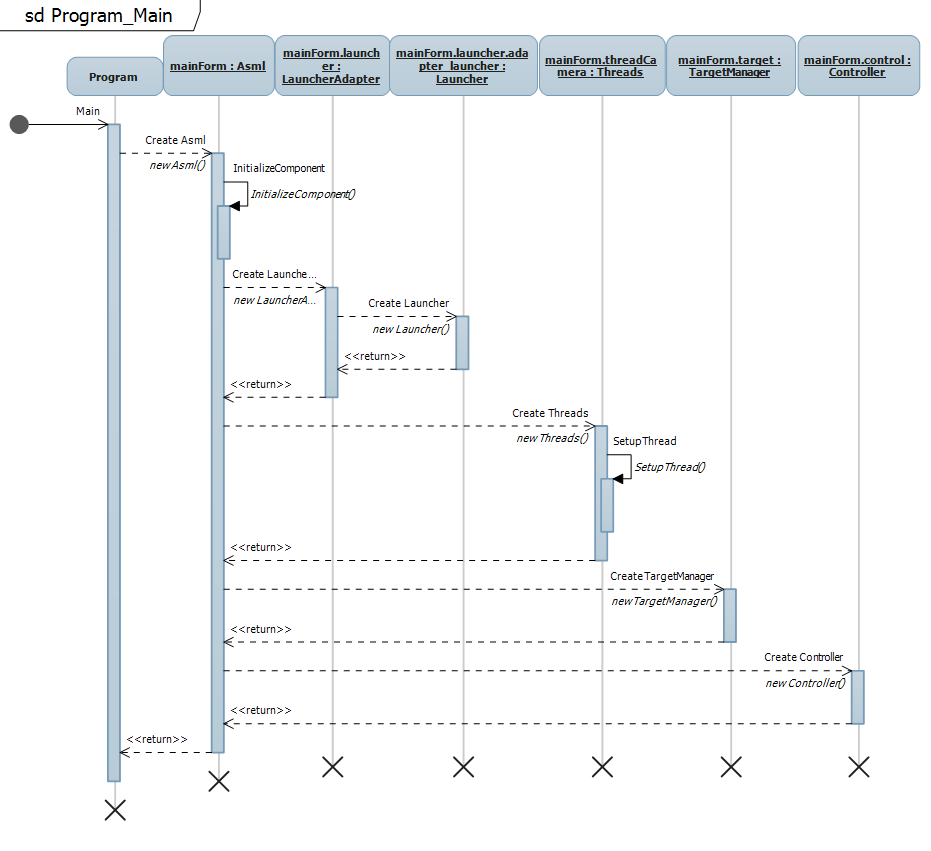
|  |  |
| --- | --- |
| **ID:** | **UC-7** |
| **Title:** | Manual turret control |
| **Description:** | The turret should be able to be rotated horizontally and vertically using manual controls |
| **Primary Actor:** | Judge |
| **Preconditions:** | The program is in Idle |
| **Postconditions:** | Turret is rotated from initial position |
| **Main**  **Success Scenario:** | 1. User clicks left, right, up, or down and turret rotates 2. User holds left, right, up, or down and turret rotates at increasing speed |
| **Extensions:** | 1a. Turret doesn’t rotate  1b. Turret doesn’t stop rotating  2a. Turret rotation speed does not increase  2b. Turret doesn’t continue rotating while button held |
| **Frequency of Use:** | For testing and final competition |
| **Owner:** | Daniel Tompkins |
| **Priority:** | 2 – Low |
| **Risk** | Retain\_16 Retain\_09 |

UML Modeling Diagrams









Design Patterns Used

For this project we combined the Singleton and Factory patterns so that we had one reader pulling in target data. This made sure that with multiple calls to the reader, we don’t end up with a big stack of them using up memory. We also implemented an adapter for our launcher. This translates positions of phi and psi into the launcher’s native coordinate system. This ensure that none of our targeting needs to be concerned with the turret’s native system and can instead use a system we are more familiar with.

The Singleton pattern will allow simpler execution. The program doesn’t need to worry about how many times the class will be called and it reduces the concern on how the function is called. This way, if the class has already been instantiated, then another attempt to do so will simply return a reference to the one that already exists. The only downside is that if you wanted to make more than one, for some reason, you would have to alter the class to allow for this.

The factory used here is allowing the program to access multiple file types without concern. This could be applied to other sources of target information as well. For instance, we could have it read target data from a class that searches visual input for target locations. The downside is that any subclass the factory might make will need to be fairly similar to the other subclasses in the data it contains.

Finally our adapter provides similar functionality to the factory. It allows classes to be unconcerned about the format of their data. When they pass it to the adapter, it will modify it in a way that makes it readable by the class it will in turn pass the data onto.

Observer design pattern was used as an event handling system. The object maintains its observers and updates them of any changes. In our program after reading targets from target files, it adds targets’ data to the target manager list.

The other pattern that was used in our project is the Mediator design pattern. The mediator defines an object that encapsulates other object and interactions between these objects. The design pattern helps to reduce coupling, because objects communicate through the mediator, instead of communicating directly with each other. In our case the mediator is used as a controller for the search and destroy mode.

The use of Strategy design pattern allows us to define a family of algorithms and to encapsulate them interchangeably. In the project strategy pattern is used to vary the algorithm of our readers, depending on a type of a file that needs to be parsed.

Design Considerations

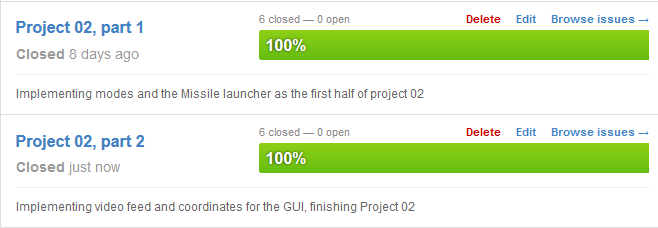
The final parts we are going to implement after this are the visual target acquisition, audio to be played during different actions, and the box that will appear on the audio feed to show targets.

To implement the visual targeting we may put in a Target manager. This will pull target locations from the reader and visual targeting. It will be a Singleton to ensure all checks for a target go to it, and it will feed location information into the launcher adapter.

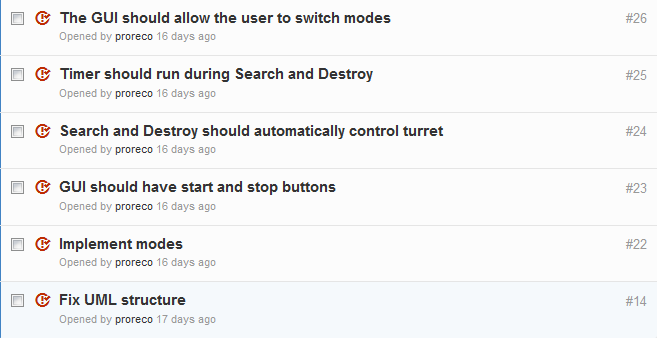
The audio will be triggered during actions like startup, beginning search and destroy, firing, ending search and destroy, and possibly when positive confirmation of a hit is received. There will be one audio manager that is called to play audio at the appropriate times. This will be directed through the speakers on the computer running the program, so we will ensure we select a system with adequate speakers.

The box will appear on the visual feed to indicate targets that have been identified. We have not settled on a specific algorithm for taking x,y, and z coordinates and turning them into a screen location. For targets acquired visually, the image will simply be added so that its center is where the system says the target is located.

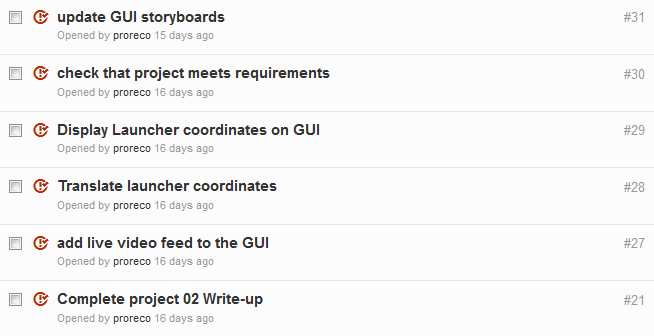
Issues



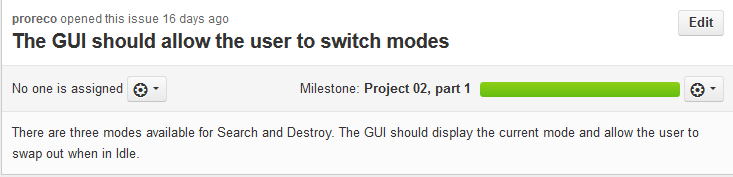
https://github.com/proreco/ThunderCat/issues/milestones?state=closed&with\_issues=yes



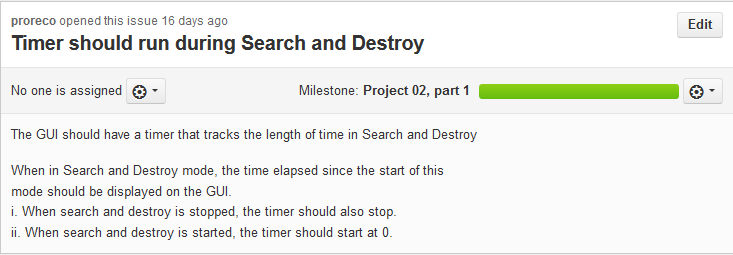
https://github.com/proreco/ThunderCat/issues?milestone=4&state=closed



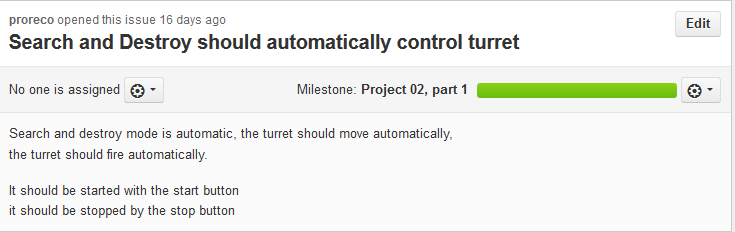
https://github.com/proreco/ThunderCat/issues?milestone=3&state=closed



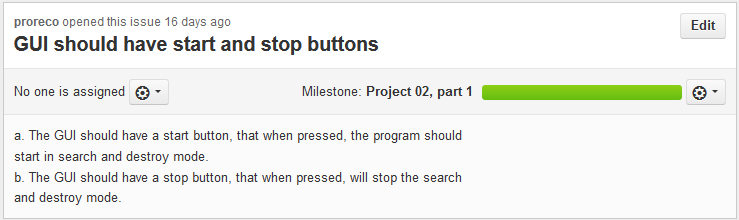
https://github.com/proreco/ThunderCat/issues/26



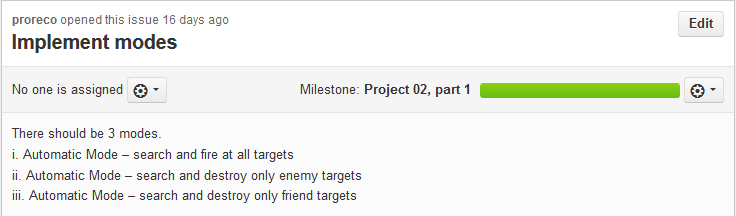
https://github.com/proreco/ThunderCat/issues/25



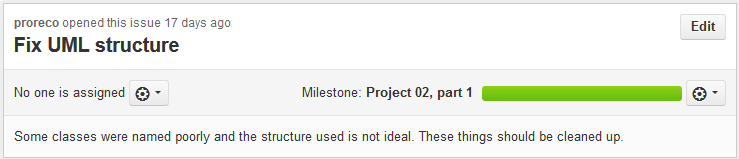
https://github.com/proreco/ThunderCat/issues/24



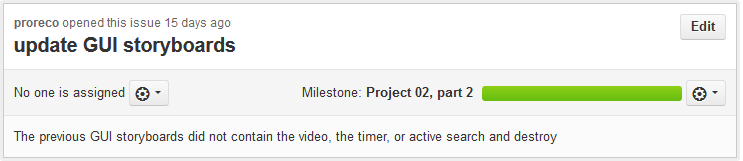
https://github.com/proreco/ThunderCat/issues/23



https://github.com/proreco/ThunderCat/issues/22



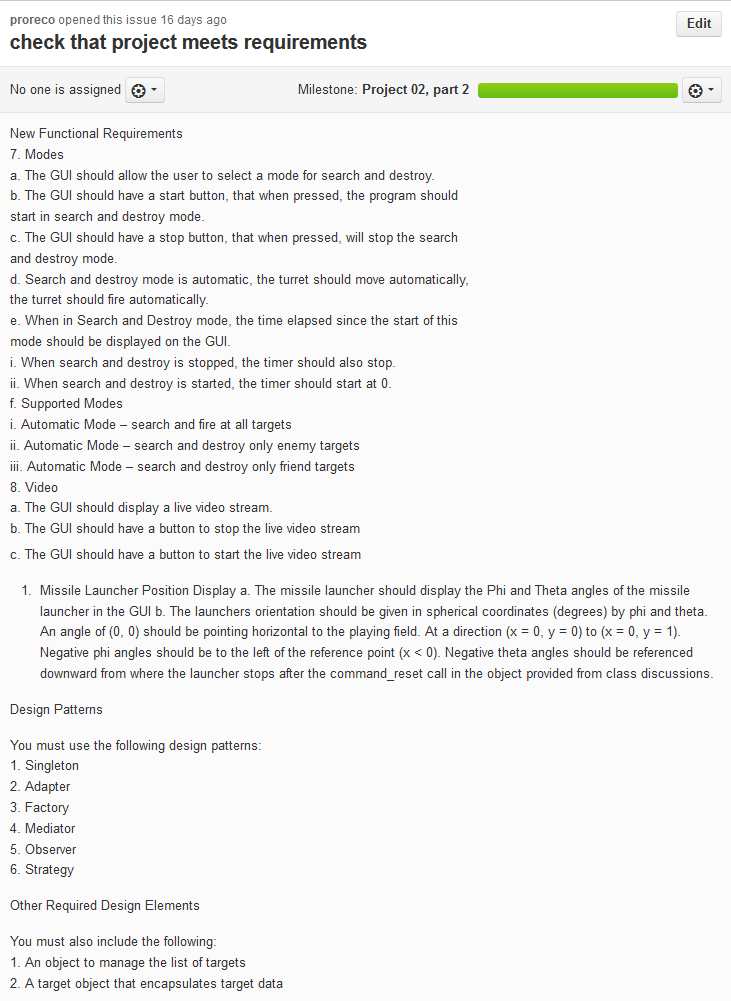
https://github.com/proreco/ThunderCat/issues/14



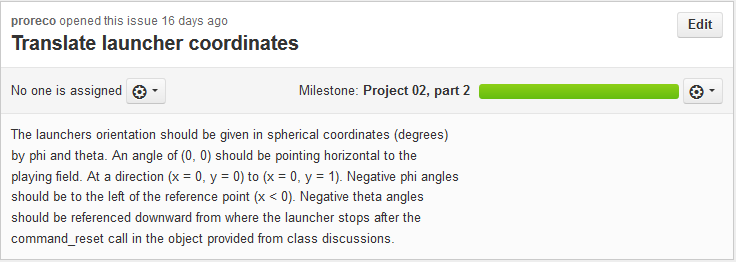
https://github.com/proreco/ThunderCat/issues/31



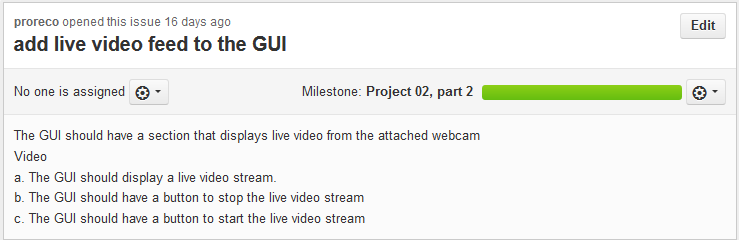
https://github.com/proreco/ThunderCat/issues/29



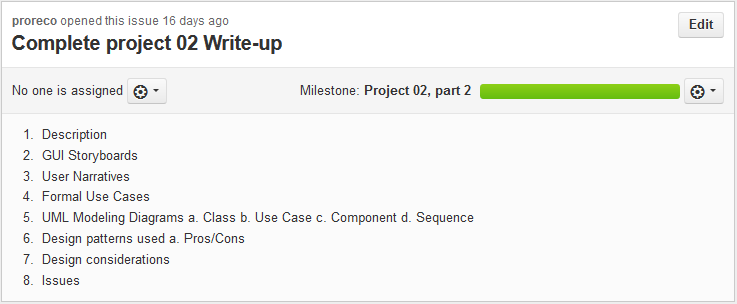
https://github.com/proreco/ThunderCat/issues/30



https://github.com/proreco/ThunderCat/issues/28



https://github.com/proreco/ThunderCat/issues/27



https://github.com/proreco/ThunderCat/issues/21