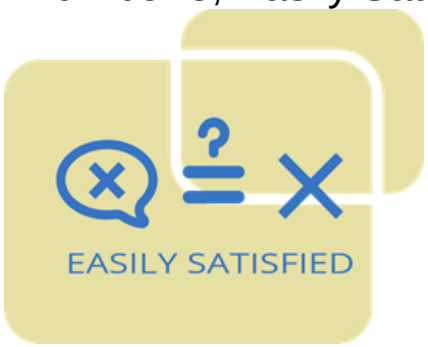
Easily Satisfied

Dr. Christopher Brown
wcbrown@usna.edu
CS Department Faculty (Professor)

Team Number 8, Easily Satisfied



Oriole, A.J. (CS),
Cimmiron, Caleb (CS),
Bourne, Jason (CS/IT (dual major)),
McCloud, Ian (IT) (Team Leader)

Customer's initials: _____ Date initialed: _____ "I have reviewed the Teams Capstone Proposal and am satisfied with its contents including, in particular, the team's Acceptance-Testing-Focused Functional Requirements Trace Table. I understand that the team will be following the Agile Software Development approach and that I may (and am encouraged to) ask for the addition, removal, or modification of any functional or non-functional requirements and/or acceptance test cases at any time as the Capstone Project progresses."

Table of Contents

Abstract

Satisfiability Modulo Theory (SMT) solvers are tools which solve problems expressed in first-order logic. We propose creating a theory solver, a component of SMT solvers, which can perform fast simplifications and satisfiability determinations in the theory of real non-linear constraints. A challenge will be in adapting algorithms that were not designed for the architecture of modern SMT solvers, implementing them, and then incorporating them into existing SMT solvers.

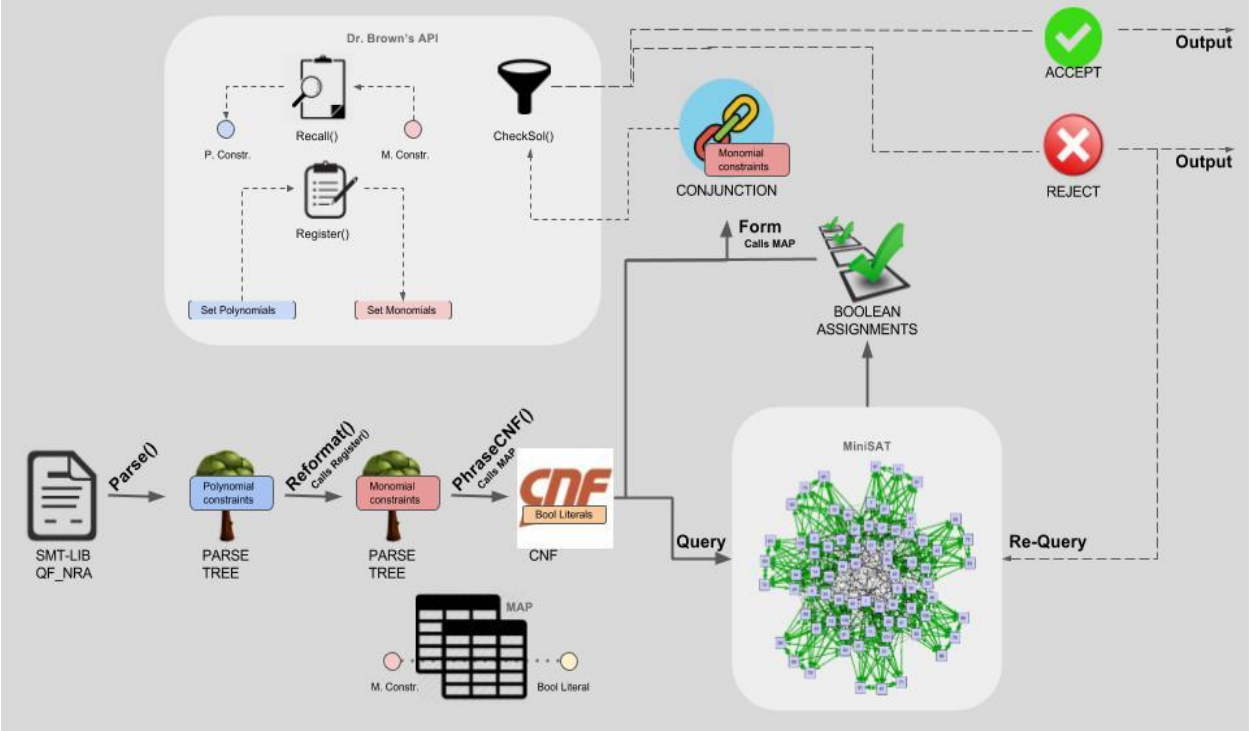
Motivation

Our project is intended to provide Dr. Brown the functionality of a Satisfiability Modulo Theory (SMT) solver, and to aid in his research pertaining to nonlinear polynomial constraint (NPQ) problems. We take an NPQ problem, which will be given in the format of the SMT-LIB language commonly used in this field of research, and generate an equivalent conjunctive normal form (CNF) boolean logic expression that can be solved by a conventional SAT solver program. The boolean literals of the CNF expression will be mapped to the polynomial inequalities which they represent. These polynomials and the boolean values assigned to them by the SAT solver will be used as input to Dr. Brown's theory solver to receive a satisfying assignment to all variables of the problem. We find this project interesting since it will allow us to deeply explore the SMT_LIB language, create novel algorithms for large data structures, and generate SAT assignments, all of which go far beyond our algorithms and theory of computing coursework.

Overview

A high-level view of our project is given in Figure 1. As shown in Figure 1, all communications between the software we create and the software Dr. Brown uses will adhere to the application programming interface (API) that we will develop in coordination with Dr. Brown. Specifically, we will scan and parse SMT-LIB input into a usable parse tree; we will register the polynomial constraint inequalities with Dr. Brown's software to reduce, simplify, and condense those inequalities; we will convert these simplified inequalities into an expression of boolean literals. We will generate CNF expressions from those boolean literals that can be passed to the SAT solver program MiniSat, which we hope to later modify to reduce time-costs by implementing a programmable heuristic algorithm which can streamline the SAT-solving process in the context of the NPQ Problem. Finally, we will pass an assignment set of literals along with the inequalities that they represent to be solved by the theory solver. We will have a driver program, or "mediator," that uses the APIs of the MiniSat program and Dr. Brown's Theory Solver software as well as the parser and transformers that we develop to eventually provide either an assignment of variables to satisfy the NPQ problem posed by the user, or confirm that the problem is inherently unsatisfiable.

Figure 1: High Level Diagram of the SAT Solver Project



Glossary

- **Satisfiability (SAT)**: A formula is satisfiable if it is possible to find an interpretation (model) that makes the formula true.
- **Self-reducible:** Each algorithm which correctly answers whether an instance of SAT is solvable can be used to find a satisfying assignment is considered self-reducible.
- **Mini-SAT**: A minimalistic, open-source SAT solver that we will modify to perform the SAT solving.

Justification

This project is a good candidate for a capstone project because it will prompt us to use our cursory knowledge from theory and algorithms to solve a much more complex and systematic software need. Each member will be delving into unfamiliar academic material and apply their coding experience to a vastly different kind of program. This project actively assists Dr. Brown and his research on real non-linear polynomial constraint problems and also has the ability to be scoped to a broader set of applications. Table A gives a discussion of the existing software that we will use, and also the software that we will develop from scratch.

Customer's Current Process

The Customer currently has no SMT solver for SMT-LIB problems in the QF_NRA logic.

Topical Areas

We have identified 4 major tasks:

1) Parsing in SMT-LIB and propositional logic - language processing
2) "Hacking" MiniSat to use modular heuristics - reverse engineering
3) Co-creating an API for Dr. Brown's SW - API development
4) Create a driver to mediate our solvers - inter-application comm.

We will be building heavily on course knowledge from Theory, Algorithms, and Artificial Intelligence

Existing / To Be Built Software

Table A discusses the existing software we will use, and the software that the team intends to develop as part of the project.

Table A. Existing Software to be Used and Significant Software to be Developed

Pre-existing software the capstone team intends to use with little or no modification	**Software the capstone team intends to develop** on its own, or significantly modify
MiniSat - MiniSat is a minimalistic, open-source SAT solver that we will use to perform the SAT solving. We will utilize the API to find assignments for our CNF-formatted expressions of boolean literals. We will make small modifications to the solver to be open it up to heuristic modifications that benefit the theory solver.	SMT-LIB parser - A parser to take input in SMT-LIB format and parse it into its component parts and store it in a data structure. Developed from scratch
QF_NRA Theory Solver - The customer's existing algorithm to solve problems in the set of real, non-linear, polynomial constraint problems.	CNF-Transformer - A program that can produce a CNF-formatted expression of boolean literals from the data structure. This transformer communicates with the theory solver to produce the simplest possible CNF-expression. Developed from scratch.

Required Resources

- The C++ Programming Language 4th edition
- Academic sources on SMT, SAT, Driver Programming
- SMT-LIB website for documentation (http://smtlib.cs.uiowa.edu/)
- Dr. Brown's Theory Solver software
- MiniSat software (open-source)
- Human resources provided by Dr. Brown's colleagues.

Functional Requirements Trace Table

Table B gives the Functional Requirements for the project, as well as the Acceptance Test Cases that will be used to demonstrate that the indicated Functional Requirements have been met by our system. The primary developer as well as a secondary developer has been indicated for each functional requirement. The build number corresponds to the order in which we intend to develop the various subsets of the project. We also include all necessary preliminary steps, such as 6 - Learn MiniSat API, as Functional Requirements.

Table B. Acceptance-Testing Focused Functional Requirements Trace Table

Functional Requirement	Acceptance Test Plan	Build #
1. Parse SMT-LIB: Read in an SMT-LIB problem and be able to parse it into a readable, infix notation expression. Give proper errors for bad problems. *Primary*: MIDN Cimmiron *Secondary*: MIDN McCloud	1.1. A problem is given, but is not in proper SMT-LIB format. **Expected result** -> The parser returns an appropriate error message. (*Abnormal*) 1.2. A problem pertaining to a logic other than QF_NRA is given in SMT-LIB format. **Expected result** -> The parser returns an appropriate error message. (*Abnormal*) 1.3. A properly formatted problem in the QF_NRA logic is presented. **Expected result** -> The problem is parsed and printed as a readable, infix notation expression. (*Normal*)	1
2. Theory Solver API: Formally define an API that enumerates the functions that can be called on the theory solver. Create documentation that is approved by both parties. *Primary*: MIDN McCloud *Secondary*: MIDN Oriole	2.1. The customer and the developer formally review the documentation. **Expected result** -> The API is deemed satisfactory. (*Normal*)	1

3. Expression Simplification: Register inequalities with the theory solver to replace complex expressions with inequalities of a single variable and 0. *Primary*: MIDN Oriole *Secondary*: MIDN Bourne	3.1. A set of complex inequalities is given. **Expected result** -> The set is printed in simplified form. (*Normal*) 3.2 A set in simplified form is compared to the set that it was derived from. **Expected result** -> The sets are found to be equivalent. (*Normal*)	3
4. Boolean Transformation: Represent a set of single variable inequalities as an expression of boolean literals. Create a map from these boolean literals to the inequalities that they represent. Recognize the relationship between different inequalities on the same variable to refrain from adding unnecessary literals. *Primary*: MIDN Bourne *Secondary*: MIDN Cimmiron	4.1. A set of unique single variable inequalities is given. **Expected result** -> Represent them as boolean literals and create an expression of these literals. (*Normal*) 4.2. An expression of boolean literals derived from a set of inequalities is converted back to a set of inequalities. This set is compared to the original set of inequalities. **Expected result** -> The two sets of inequalities are found to be equivalent. (*Normal*) 4.3. Two inequalities are given with the same variable. **Expected result** -> One boolean literal is created to represent both inequalities. (*Abnormal*)	2
5. CNF Transformation: Transform an expression of boolean literals to CNF-format utilizing Tseitin- Transformation. *Primary*: MIDN McCloud *Secondary*: MIDN Bourne	5.1. An expression of boolean literals not in CNF-format is given. **Expected result** -> The expression is transformed to CNF-format. (*Normal*) 5.2. A transformed expression is compared to the expression that it was derived from. **Expected result** -> The expression is shown to be equivalent to the original expression. (*Normal*)	2
6. Learn MiniSat API *Primary*: MIDN McCloud *Secondary*: MIDN Oriole *Note: This is a necessary preliminary step*	6.1. A CNF-format satisfiable problem is given to be solved through the MiniSat API. **Expected result** -> A solution is received from the MiniSat solver through the MiniSat API. (*Normal*) 6.2. A user requests to receive all solutions to a CNF-format satisfiable problem. **Expected result** -> All solutions from the MiniSat solver are iterated through via the API and output to the user. (*Normal*) 6.3. A CNF-format unsatisfiable problem is given to	1

	be solved through the MiniSat API. **Expected result** -> The user is alerted that there is no solution to the problem. (*Normal*)	
7. Theory Solver: Be able to pass the theory solver an expression and receive a solution or a reason for failure. *Primary*: MIDN Bourne *Secondary*: MIDN Cimmiron	7.1. An expression is passed to the theory solver that is satisfiable in the theory. **Expected result** -> Receive a satisfying assignment to the problem. (*Normal*) 7.2. An expression is passed to the theory solver that is not satisfiable in the theory. **Expected result** -> Receive an explanation of failure that formulates a learned clause. (*Normal*)	4
8. SMT-LIB Conversion: Be able to convert a CNF-formatted expression to SMT-LIB *Primary*: MIDN Cimmiron *Secondary*: MIDN Bourne	8.1. A CNF-format expression is given. **Expected result** -> The expression is converted to an SMT-LIB statement. (*Normal*) 8.2. An SMT-LIB statement is converted back to a CNF-format expression and compared to the expression that it was derived from. **Expected result** -> The two expressions are equivalent. (*Normal*)	3
9. Heuristic Modification: Modify MiniSat to allow for heuristic manipulation. *Primary*: MIDN Oriole *Secondary*: MIDN McCloud	9.1. A modified set of heuristics for MiniSat is given. **Expected result** -> The heuristics are plugged into the MiniSat solver. (*Normal*) 9.2. The solver is run on a problem to receive a solution with the new heuristics. **Expected result** -> The result is in keeping with the heuristics expected outcome. (*Normal*)	4

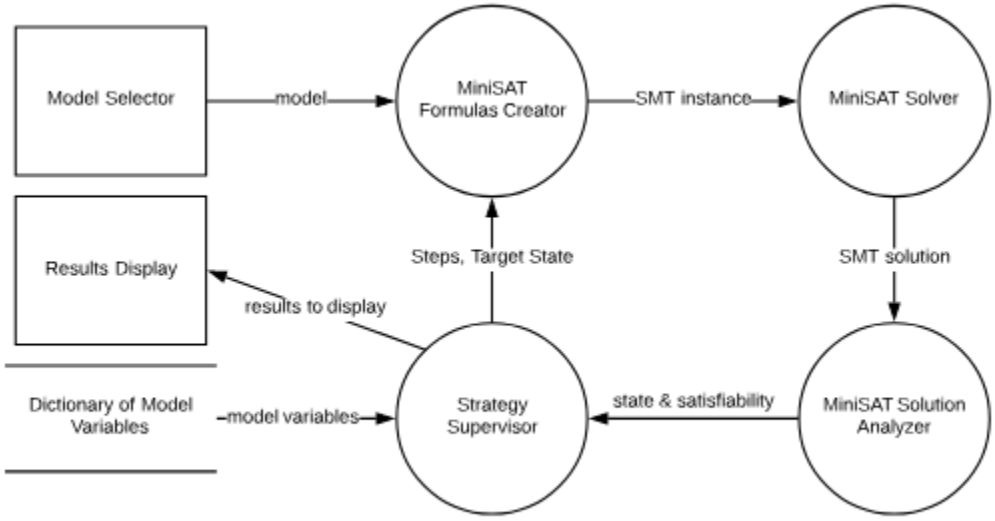
Design (of the most complex part of the project)

The most complex part of our project is Functional Requirement 7: Theory Solver, as it requires a determination of whether a SAT solution exists for any properly formed expression passed to it. Our MiniSAT Solver Solution Algorithm fulfills this functional requirement. Figure 2a gives the overall design for the Selection and Analysis subsystem, while Figure 2b gives the specific design of our MiniSAT Solver Solution Algorithm. A detailed description of our design is as follows:

1. As shown in Figure 2a, the input to our system comes from our Model Selector data source which allows the user to select from either predefined or user-defined models,

2. Once a model had been selected and provided as input, MiniSAT Formulas Creator process generates a formula for an SMT instance which is then sent to our MiniSAT Solver process.
3. The resulting SMT solution (including the empty case if no solution is found) is sent to our MiniSAT Solution Analyzer process to determine the state and satisfiability of the solution.
4. These results are sent to our Strategy Supervisor process which determines whether the target state has been reached, and if not, the number of steps remaining. Our data store a dictionary of standard model variable values that may be looked up as needed by the Strategy Supervisor.
    a. If the Strategy Supervisor determines that additional refinements to the solution are possible, the steps for refinements and the current target state are sent to the SMT Formulas process which continues the analysis as described above.
    b. Otherwise, the Strategy Supervisor displays the current results.

Figure 2a. Design of SMT Selection and Analysis subsystem using a dataflow diagram

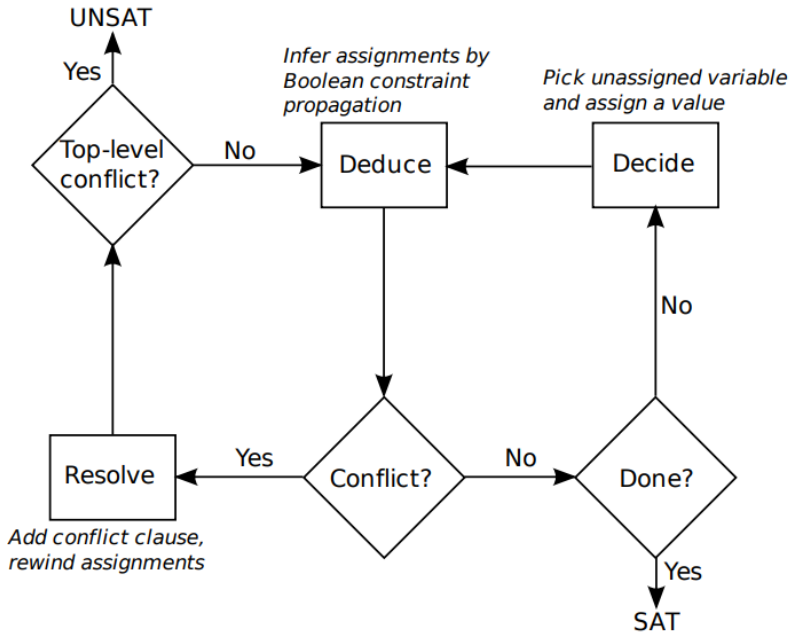


As shown in Figure 2.b, the design of our MiniSAT Solution Analyzer algorithm includes the following steps:

1. Perform a depth-first search through the space of possible variable assignments. Stop when a satisfying assignment is found or all possibilities have been tried. Optimization choices include:
    a. Skip branches where no satisfying assignments can occur.
    b. Order the search to maximize the amount of the search space that can be skipped.
2. Repeat the following steps until SAT or UNSAT is returned:
    a. Decide: Select some unassigned variable and assign it a value.
        i. If all variables are assigned, return SAT.

b. Deduce: Infer values of other variables that follow from that assignment and detect conflicts.
c. Resolve: In case of conflict, record a new clause prohibiting that conflict; undo the assignments leading to the conflict.
    i. If it's a top-level conflict (the conflict clause is empty), return UNSAT.

Figure 2b. Design of our MiniSAT Solution Analyzer algorithm using a flowchart



Risk Management

In Table C, we consider all of the risks that we anticipate in the development of the project, identify a risk management technique for resolving each risk, and give the current status of the team in mitigating each risk.

Table C: Risk Management

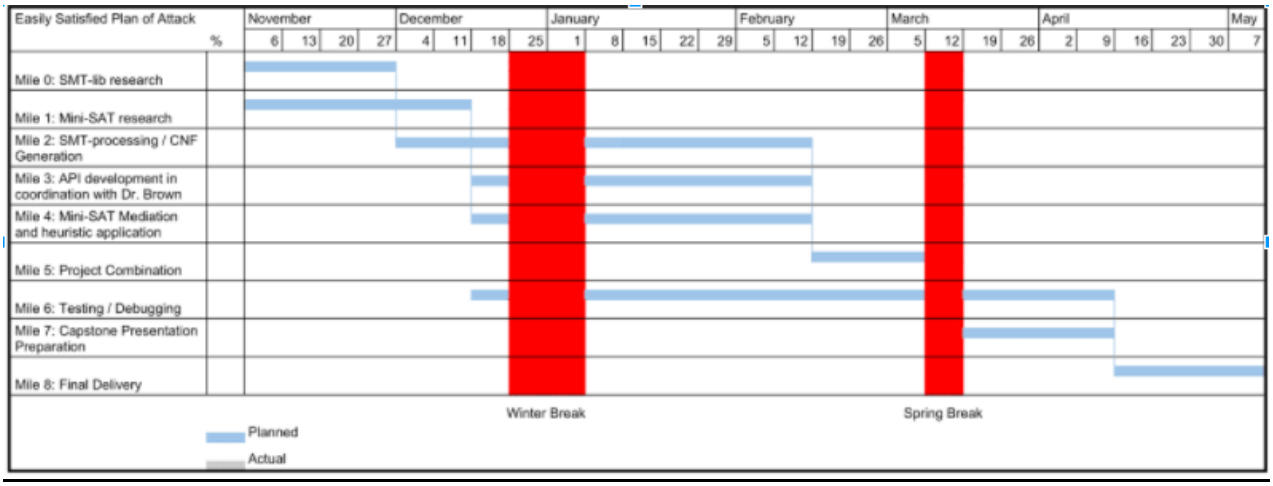
Priority	Risk	Risk Management Technique	Status
1 **Probability:** Medium	Scanner and parser do not handle certain inputs correctly due to wide	Continual testing on inputs throughout development	All team members in Programming Languages are learning Flex and Bison in their

Severity: High	range of SMT-LIB data types		Programming Languages class
3 **Probability:** Low **Severity:** High	Scanner and parser pass up bad input causing errors further along program execution that are hard to trace back to an apparently working scanner/parser	Careful analysis of errors and never assuming scanner and parser are 100% correct	Scanner and Parser going through re-development since Milestone 2
8 **Probability:** Low **Severity:** Medium	Compressor of map has a bottleneck in time because of inefficiency	Algorithm analysis to help determine time cost	Early Stages: Searching for best c++ map library for our use
4 **Probability:** Low **Severity:** High	Map that Dr. Brown's Program has and our program are not in sync	Make one copy of the map at the beginning that persists through our process, and is stored in Dr. Brown's Software.	Consulting Dr. Brown of our plan for the map, and when we process those expressions
7 **Probability:** Low **Severity:** Medium	CNF translation takes exponential time	Algorithm analysis to determine time cost	Functional requirement lead for this step responsible
5 **Probability:** Low **Severity:** High	CNF translation is translated with exponential space	Algorithm analysis to determine space cost	Functional requirement lead for this step responsible
2 **Probability:** Medium **Severity:** High	Mediator and API between programs cause errors on certain inputs not accounted for and go unnoticed	Create an API that always catches errors and informs user as opposed to catching errors and moving on	Functional requirement lead for this step responsible
6 **Probability:** Medium **Severity:** Medium	MiniSat spends too much time waiting on mediator and not doing work.	Test MiniSat or find out through MiniSat research how much CNF data it can handle at a time	Currently everyone is researching MiniSat

Project Plan & Gantt Chart

Figure 3 presents the project plan timeline as a Gantt Chart. As shown in Figure 3, before the group begins development on this project, everyone will need to research the particulars of some of the components involved. Specifically, the group will look into the nuances of the SMT-LIB language to effectively construct SMT processing software that will parses the user input, as well as the MiniSat program which will be modified to allow for users to define a set of heuristic specifications. By early to mid-December, this research should be sufficiently complete and the group will move into the development phase of the major components of this project. The SMT-processing and CNF generation can begin after researching SMT-LIB. Similarly, both the API to communicate with Dr. Brown's code as well as the software handling mediation between the MiniSat program and the theory solver can begin once the group has figured their way around MiniSat. These developments are expected to take the most significant amount of time and will be tested and debugged along the way. It is expected that the development phase will end around mid-February, and then the group will begin the combination of these separate functions over the next few weeks before Spring Break in March. From March to mid-April the group will continue to debug any outlying issues, and prepare the capstone presentation in earnest to be prepared for final delivery during the last weeks of the second semester. This plan is demonstrated with a more time oriented design as shown in Figure 3.

Figure 3: Project Plan Timeline as a Gantt Chart



Quality Assurance

MIDN Oriole served as the quality assurance team member and reviewed all parts of the document. MIDN McCloud reviewed the portions of the document developed by MIDN Oriole.

Customer Acknowledgement

By signing below, the customer acknowledges that the project developed as part of this capstone coursework becomes the property of the DoD, and that the CS Department does not assume any responsibility for maintaining the software produced for the client. The client may use the software within the context of their USNA affiliation, and may not distribute it without approval from the USNA legal office.

Capstone Team Leader Name_____

Capstone Project Title _____

Customer Name (printed) _____

Customer Contact Info (email/phone) _____

Customer Affiliation _____

Customer Signature _____

Date_____