AI Shipyard: Finite State Machine AI Engine for Ai Cap'n

Final Thesis
Presented to
the Faculty of the College of Computer Studies
De La Salle University Manila

In Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Science in Computer Science

by

BENAVIDEZ, KARL
CARONONGAN, ARTURO
CHUA, KELVIN
GOTAUCO, PATRICK

Prof. Paul INVENTADO
Adviser

Aug 26, 2010

The thesis entitled

AI Shipyard: Finite State Machine AI Engine for Ai Cap'n

developed by:

Benavidez, Karl

Caronongan, Art

Chua, Kelvin

Gotauco, Patrick

and submitted in partial fulfillment of the requirements of the Bachelor of Science in Computer Science degree, has been examined and recommended for acceptance and approval.

Mr. Paul Inventado, Adviser

_____, Date

The thesis entitled

AI Shipyard: Finite State Machine AI Engine for Ai Cap'n

after having been reviewed, is hereby approved by the following members of the thesis committee:

Mr. Paul Inventado
Lead Panelist

Date

Mr. Solomon See
Panelist

Date

Dr. Raymund Sison
Panelist

Date

The thesis entitled

AI Shipyard: Finite State Machine AI Engine for Ai Cap'n

after having been recommended and reviewed, is hereby approved by the Software Technology Department, College of Computer Studies, De La Salle University:

Ms. Ethel Ong	Dr. Rachel Edita Roxas
Chairperson	Dean
Software Technology Department	College of Computer Studies
Date	Date

Acknowledgement

First and forehand, we would like to acknowledge our adviser, Paul Inventado, for the guidance, advice, and help he gave use throughout our thesis period. We would like to acknowledge our panelists. We thank Sir Solomon See for providing us a venue for testing through the Learning Innovation Center (LIC) and giving us advice. We thank Dr. Raymund Sison for helping us find a number of suitable samples through his INTROAI classes and helping us acquire resources in some of our activities.

We'd also like to acknowledge the students who reside in the Learning Innovation Center for their moral and technical support on the project, as well as being test subjects. We'd like to acknowledge the equipment at the Learning Innovation Center, the same equipment which was used to create these documents for submission and aid in the development process.

We'd like to thank the university and it's faculty, for not only educating us, but providing the venue for our development as Lasallians and responsible citizens. We'd also like to thank our families for giving us the ability to study in such a university. Most of all we'd like to thank God for His continued presence in us as we worked on our thesis, as a guiding light as we strived to do what we needed to do.

Abstract

The research is about on the creation of a visual programming tool for the game AI Capn. The game teaches artificial intelligence concepts (such as path-finding and decision trees) and agent-based artificial intelligence by allowing users to create their own agents. The research addresses the difficulty faced by users who create agents through manual coding. The decision pattern of the agents in this visual tool will be using finite state machines, a graphical interpretation of system states. Using this representation, the program should be able to show the user a finite-state machine representation of their desired agent. After its formulation, the output should be a java bot class which can control an instance of a ship in the game.

Keywords: Object-oriented programming, Visual Programming Tools, Artificial Intelligence

Table of Contents

List of Figures

List of Tables

1 Research Description

This section provides an overview of what the research is about and aims to explain to the reader the significance of the study currently in its field.

1.1 Overview of the Current State of Technology

There are existing visual programming applications such as Carnegie Mellon University's Alice, an FSM-based programming tool. Alice is a training tool designed to teach inexperienced programmers with the help of 3D Objects to define their behavior as well as the chronological ordering of its actions (Conway et al., 2000) . Another example, Kara, is an application designed to introduce inexperienced programmers by using Finite State Machines. This application uses one agent (the ladybug) and it can be programmed using finite states to direct the agent into what path to choose to navigate a map (Hartmann, Nievergelt, & Reichert, 2001) . The graphical nature and immediate feedback presented in these programs enable visual programming and identification of errors without having to type the code (Moskal, Lurie, & Cooper, 2000).

Finite State Machines (FSM) can be defined as the representation of data at discrete points in time connected via conditions which lets an agent traverse through these states. A decision-making concept such as Finite State Machines is represented visually into different states connected by conditions in the context of agent development. FSM presents data into defined states and a deterministic finite automata which can be used to make a model for representing decisions made by an agent (Lu & Druzdzel, 2009). Artificial Intelligence from its many applications, such as learning algorithms, can also utilize simple decision trees in performing tasks. The simplistic structure of traversing between states triggered by an input, makes FSM a favorable candidate as basis for an AI model and for use in a visual programming environment.

Ai Cap'n is a game designed as a means to develop concepts in an existing environment focusing on agent development (Inventado & See, 2009). At its current state, developing the agents in the system requires the knowledge of the Java programming language. Due to this, users unfamiliar with the programming language find it difficult. There is a need to implement a universally, if not familiar, interface to enable development for a wider array of users.

Based on a survey in (Inventado & See, 2009), The absence of a visual tool or representation for creating agents in AICapn makes agent-generation hard for users to create their own agent since no other way to create AICapn agents aside from manually programming the agent classes.

1.2 General Objective

To design and implement a visual tool to generate agents for Ai Cap'n using finite-state machines.

1.3 Specific Objectives

- To study Ai Cap'n platform.
- To evaluate common AI game algorithms implementable in FSM & AI Cap'n.
- To study existing finite state machine authoring tools.
- To identify the features to be included in the visual tool.
- To consider the data representation for finite state machines in the visual tool.
- To evaluate the simplicity of the visual tool through peer evaluation.

1.4 Scope and Limitations of the Research

The research aims to identify how to implement AI Cap'n agents using finite state machines in order to model intelligence and decisions in a visual manner. The study will cover artificial intelligence in games to show the applications agent-based intelligence in a game environment. The research will include the analysis of features in visual tools to aid in designing the AI Cap'n visual engine.

The AI Cap'n game engine will not be revised and the proposed system will only serve as an extension to it. The research will mostly include decision-making algorithms for agent design, search algorithms for agent movement, and customizability options for agent development. The evaluation of visual tools will only include game, story or artificial intelligence simulation software such as Kara or Alice. The features to be considered in the project is not limited to the features found in the evaluated software and may add more features when deemed as necessary. Research on data representation will only consider how to represent the output, which is FSM, as a readable format for the engine to be created.

1.5 Significance of the Research

Students and programmers are going to benefit from this research as they will have a visual interface that will aid them in understanding the fundamentals of finite state machines and artificial intelligence. Figure 1 shows that less than 3% of students find agent creation as simple or help in visualizing agent-based intelligence and decision-making concepts. This research aims to provide a visual interface that students may use to create bots and, at the same time, have a more enjoyable time learning these concepts in AI Cap'n.

Table 2. Features of Ai Cap'n identified by students

Features of Ai Cap'n	Student Percentage
Fun Factor	35.96%
Application of concepts learned	22.80%
Challenging	8.77%
Understanding of AI Concepts	8.77%
Competitiveness	6.14%
No need for implementing GUI	4.38%
Usefulness for future work	4.38%
Appreciating the significance of AI	4.38%
Simplicity	2.63%
Visualizing AI concepts	2.63%

Figure 1 A survey evaluation from students about AI Cap'n

Teachers and learning institutions are going to benefit from this as they may use it as a tool for demonstrating some concepts of finite state machines and artificial intelligence to students who are and are not inclined in programming. This research focuses on creating a finite-state machine for the AI Cap'n game engine. This research aims to provide the same benefits of visual tools in teaching such concepts to students.

In the scope of artificial intelligence in general, the field aims to design systems to behave rationally and like one with intelligence. Our (human) intelligence can be separated into states. All our preceding actions affect our succeeding events and actions. Therefore, in modeling through finite state machines, we can more effectively create an agent with artificial intelligence. The research serves as a mean to modeling intelligence in an environment such as AI Cap'n.

1.6 Research Methodology

This section pertains to the methods by which the research shall be implemented given a software development model. The creation of a visual tool is subject to several builds that must be tested to an audience of users to ensure the quality of the software. The task of iteratively identifying problems and reconstructing the model is necessary in the methodology to be chosen. Thus the chosen model for the methodology of this research is the **Iterative and Incremental Development Model**. The research model supports the redesign of the system based on user evaluations and input, which is one objective of this research. This model consists of the following processes:

1. Initial Planning
2. Planning
3. Requirements
4. Analysis & Design
5. Implementation
6. Testing
7. Evaluation
8. Deployment

After step 1, Initial Planning, It proceeds to: Planning, Requirements, Analysis & Design, and Implementation. After the system is finished, it will proceed to Testing and Evaluation. When the system does not pass the Evaluation based on criteria, then there is a need to return to Step 2 and repeat the process once again. When all specifications and criteria have been satisfied, it proceeds towards the final Deployment stage.

1.6.1 Initial Planning

This step involves the initial conception of the program specifications. Involved in this process are the research objectives, scope and limitation, and the identification of the research problem. The compilation of literature for reference in the development and the designing of the activity calendar is also taken in this step.

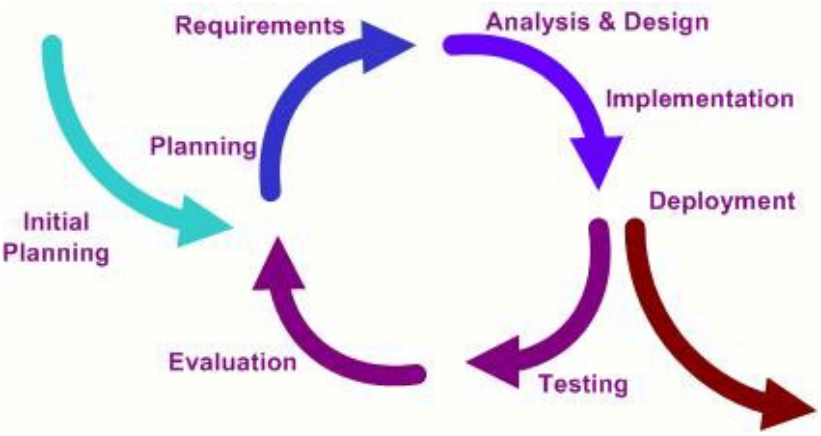

Figure 2 Iterative and Incremental Development Model

1.6.2 Planning

This step involves the allocation of tasks and generation of long-term goals for the project. This also involves closely working with the faculty adviser regarding the expected impediments of the project.

1.6.3 Requirements

This step involves the identification of final system specifications and research milestones. Once the goals have been defined, the requirements are appointed to different parts of the calendar of activities.

1.6.4 Analysis & Design

This step involves the designing of the structure for the artificial intelligence bot. Here, the formulation of a structure is created along with a prototype to determine its flexibility in terms of user inputted data. The system user interface's design will also be considered in this step.

1.6.5 Implementation

This step involves the actual creation of the system based on the design created in the previous step. Each member will be given a module to create which will be put together at the end of the step. The creation of the AI Engine system, the AI agent, and the debugging of the output is included in this step.

1.6.6 Testing

This step involves the testing of the AI agent and the verification of the algorithms generated by the state machine. This step also focuses on finding faults in the software.

1.6.7 Evaluation

This step involves the evaluation of the system by users of the system. Based on the feedback provided by the users, the architecture may be revised and changed based on the feedback

given. This step loops back to the Planning stage if the feedback is not favorable, otherwise, proceed to the Deployment step.

1.6.8 Deployment

In the Deployment step, the system will be ready to use for other users and will be stable enough for distribution.

1.6.9 Calendar of Activities

Table 1 illustrates the projected schedule of development based on the steps presented previously. Subject to change based on circumstances which may deter from the assumed schedule.

Table 1 Calendar of Activities

	2009					2010					
Step	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
Initial Planning	++++										
Planning	++	++++				++					
Requirements		++	++++			++	++				
A&D		++	++				++	++			
Implementation			++	++++	++++		++++	++++	++		
Testing					++	++			++	++	
Evaluation					++	++			++	++	
Deployment									++	++	

Legend: + = approx. number of weeks

2 Review of Related Literature

The review will revolve around three aspects of the proposed system. Visual Tools will be reviewed to research on aspects needed to represent various data graphically. Finite State Machines will be reviewed as a representation tool for intelligence. Artificial Intelligence will be reviewed as a basis for the agents to be created by the engine.

2.1 Visual Tools

There are instances in which students find a hard time understanding the concepts regarding the fundamentals of programming. Mainly because of the lack of computer experience before enrolling in any computer-related course or another factor is that the student is poorly prepared in mathematics prior entering the said course (Cooper, Dann, & Paush, 2003). Therefore, several approaches in teaching have been made to solve this problem. One popular approach is to abstract the student from programming syntax and focus on solving any give problem via visual representation that gives immediate feedback. Two learning tools are presented in this review namely Alice, a visual storyboarding tool and Kara, a finite state machine-based visual programming tool.

Alice is a storyboarding tool for beginners where users can set attributes on 3D Objects and use drag and drop to program these objects to make actions (Cooper et al., 2003). The user can put 3D Objects on the world and the movements for the objects are made via storyboarding and drag-and-drop style of input hence immediate feedback and visual representation is achieved. The storyboard may branch out based on user input, thus the process is likened to a tree. This software has been developed so that students can easily be introduced to object-oriented concepts of programming. An application is done by addressing an issue that was investigated and experimented by (Moskal et al., 2000): there was a problem regarding novice computer science students regarding programming due to lack of experience or the lack of introduction to mathematics. These students were at risk of failing in a programming subject; therefore, an experiment is done to investigate if the approach of Alice can be applied to this situation. The result was students who have used Alice had a better chance of passing than those who did not take Alice. Students can easily design the scenes on the world and experiment by trial and error hence, making them try out possibilities of solving a problem without worrying syntax of a programming language.

Kara is a tutorial tool that aims to intro duce beginners to programming by using Finite-State Machines (FSM) (Hartmann et al., 2001). The reason of using FSM as an approach to introduce to programming is because they are purely graphical in nature and revolves around the concept that for every input, there is an output. The decisions made by the ladybug o ccur in the generated finite-state machine. The software presents only one agent, Kara the ladybug, in which the users program her by a series of drag-and-drop of sequence actions with a combination of a pseudo if-else statement for a state encountered. One paper made by (Hartmann et al., 2001) indicates that programming should be a part of general education and not for only professionals so that users can find the meaning behind developing such program or to identify processes in making one. Kara helps realize the notion of programming by introducing it in a possible simplest manner. In the experiment results shown by the paper, upon comparing Kara to a text-driven programming language such as Pascal or Java, the users found it more motivating to learn programming using the tool. It was possible for users to concentrate on providing a solution to any given problem rather than concentrating solving and minding the

syntax of a particular programming language.

Table 2 Comparison of Selected Visual Tools

System	Purpose	Intended Users	Coding Paradigm	Decision Pattern
Alice	OOP	Beginners	Object-oriented	Tree
Kara	AI	Beginners	FSM	FSM

The two systems presented takes into account the difficulty of introducing students into programming concepts. Both of which used visual representation and immediate feedback of the mo del of the solution hence, allowing students to do trial and error in find the solution of any given problem. What differs on these systems is that the former is aimed to introduce object-oriented programming while the latter is aimed to intro duce programming via finite-state-machines. Comparing the two systems as seen in Table 2 shows the differences between the two systems as mentioned in their respective descriptions.

2.2 Finite State Machines

By definition, A finite state machine is a device, or a model of a device, which has a finite number of states it can be in at any given time and can operate on input to either make transitions from one state to another or to cause an output or action to take place. A finite state machine may only be in one state at a given time (Buckland, 2005). The idea behind finite state machines is to represent the behavior of an object into several transitions or states. The transitions between states are represented as inputs which, when the finite state machine receives, lead to another state. In our particular application, these inputs that the finite state machine receives are inputs from the world such as number of enemy units, power-ups, current health, and terrain.

Finite state machines are the preferred instrument of choice to illustrate AI agents for its simplicity to code, ease in debugging, and its flexibility to changes in the diagram. Since these are the aims of our application, we chose to utilize finite state machines. The simplicity of the graphical nature of Finite State Machines makes it an optimal candidate.

Finite state machines are not restricted to artificial intelligence (Gladyshev, 2005). They can also be used in approaches to analyze digital evidence, in the field of forensic analysis. With the arguments that the ad-hoc analysis of digital evidence is not appropriate in complex investigations due to the tendencies of being error prone, more scientific methods of analysis have been taken into consideration, one of which is the use of finite state machines.

Certain approaches have already been described in using a finite state machine in analyzing digital evidence finding weaknesses in informal forensics. Despite potential advantages such as analysis rigor and automation, the approaches require additional effort for modeling the incident and simplifying the said model. The potential benefits obtained from these approaches also need to be weighed against the possible cost of the additional efforts needed.

2.3 Artificial Intelligence in Games

Artificial Intelligence exists as one of the primary components of many interactive programs, especially games. The extent of which they are used may differ by genre and the object of each game. The use of artificial intelligence in games contributes to the overall gameplay by providing variety and autonomy to objects in the game setting. Artificially intelligent objects, known as

agents, are intelligent enough to react appropriately to the players' actions in that world (LaMothe, 1995).

There has been an attempt at modeling human-decision making by means of a Rapid Planning Process based on the psychological structure of decision making (Moffat & Medhurst, 2009). The study was conducted for the military and was aimed at providing decision-making in a constantly changing real-time environment. Presented in their study are approaches in handling single-decision and multi-decision algorithms for interpreting artificial intelligence.

Intelligent systems require the use of several parameters in order to make decisions (Ahn & Choi, 2009). Proposed solutions regarding the simplification of decision-making algorithms in a programming setting include considering the decision as a multiple decision tree. A study by (Stout, 1997) presents that the proponent could even re-evaluate the problem itself in order to simplify the needed information.

Common application of artificial intelligence implementation is the inclusion of search algorithms (Woodcock, 2000). Varying algorithms exist for different games depending on their own implementation. Algorithms such as breadth-first and A* come to mind with search algorithms. In (Stout, 1997), artificial intelligence should be capable handle user input and environmental changes during the algorithm.

The AI Capn game is an example of an agent-based AI creation engine (Inventado & See, 2009). The game was designed as an instruction tool for artificial intelligence search algorithms. The game enables creation of agents using the Java programming language as separate java classes which can be uploaded unto the game itself to be initialized as active players. The agents perform actions based on the algorithm programmed into them by a user. Through surveys and evaluation by the users, there is still an issue on the simplicity of creating agents. This is where a visual tool concept stems from in this study.

Artificial Intelligence is a broad field. As research continues on different algorithms and paradigms are performed, implementation of AI in applications such as games may improve in terms of speed, efficiency, and memory usage. With continuous improvements in technology human-like decision making may be possible for future AI agents (Moffat & Medhurst, 2009).

3 Theoretical Framework

This chapter discusses several concepts from some related literature which grabbed the interest of the researchers. Concepts may be related to or influence the final output of the designed system and will be discussed here in this chapter. The framework lists the features of certain systems and their utility in their respective systems.

3.1 Alice

Alice, developed by Carnegie Melon made use of a visual and an interactive environment that enabled users to get a visualization of the animation they are creating via the scripting engine. The interface enables a variety of users, even non-programmers or users with no defined knowledge in scripting to create an interactive environment through object-oriented programming.

The interactive environment is controlled through a script that is hidden from the user. The embedded script is used to control the actions or the setting of the user's desired output. Though the script is hidden from the user, they are able to modify the script's contents by dragging and dropping the objects (or characters) available to the user or by making use of the well-defined command buttons available.

```
obj.move(forward, 1)
obj.move(forward, 1, duration=3)
obj.move(forward, 1, speed=4)
obj.move(forward, speed=2)
# change of coordinate system
obj.move(forward, 1, AsSeenBy=camera)
# different interpolation function
obj.move(forward, 1, style=abruptly)
```

The embedded script is responsible for everything: making objects, placing them in the scene for their initial locations and orientations, and making the simulation start. The interface interacts with a built in scripting engine which generates Figure 3 displays the environment that is available to the user, while keeping the script generator hidden from the eyes of the user.

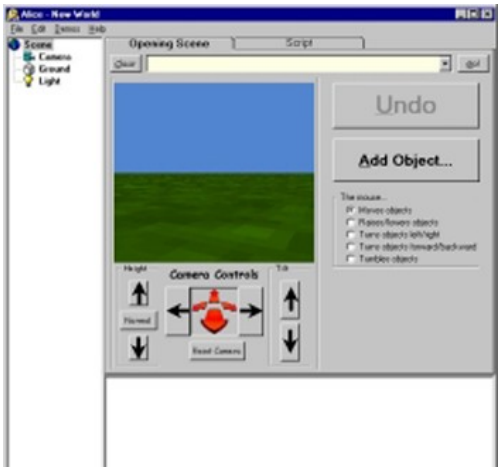

Figure 3 Alice Interface

3.1.1 Alice and Python

Alice uses a general purpose, interpreted language called Python . The developers of Alice decided to make use of Python as the language possessed certain qualities such as:

1. A modern language with a rich set of built in data types (Maps, strings, lists) and operators for those types.
2. Freely distributable and available without royalty.
3. Extensible in C/C++. For example, Alice uses the Coriolis collision detection library.

The developers made two key changes in Python implementation, as user testing data displayed the need to make such improvements effective. First, Python's integer division was modified, so users could type 1/2 and has it evaluate to 0.5, rather than zero. Second, Python was made to become case insensitive. Over 85% of the users made case errors and many continued to do so even after learning that case was significant. Most novice-oriented systems (e.g. Hypercard, Pascal, LOGO) are designed to be case insensitive, a lesson seen being ignored in the proposed standard for VRMLScript.

3.1.2 Alice's animation engine

The Alice animation engine interpolates data values from a starting-state to a target-state over time, with default duration of one second. When two or more animations run in parallel, the Alice scheduler interleaves the interpolations in round-robin fashion. This allows a user to evaluate a command while another command continues to animate, without any explicit thread management. The use of these implicit threads is a major contribution of the Alice system. Alice itself is a single-threaded application. Developers built an experimental Alice prototype using native Windows 95 system threads, one per animation, but it exhibited poor load balancing between threads, giving rise to poor-quality, lurching animations.

Animated Alice commands return an animation ob ject:

```
scoot = bob.move(forward, 1)
```

These objects respond to several metho ds (stop, start, loop, stoplooping) and can be composed with other animation objects:

```
DoInOrder(anim1, anim2,...animN)
```

, which causes the animations to run in sequence

```
DoTogether(anim1, anim2,...animN)
```

, which causes them to run in parallel

Of course, DoInOrder and DoTogether animations can also be composed, giving rise to more interesting animations. For example, given a world with an object called Bunny, the following script could make the bunny beat his drum by writing:

```
ArmsOut = DoTogether(
Bunny.Body.LeftArm.Turn(Left, 1/8),
Bunny.Body.RightArm.Turn(Right, 1/8) )
ArmsIn = DoTogether(Bunny.Body.LeftArm.Turn(Right, 1/8),
Bunny.Body.RightArm.Turn(Left, 1/8) )
BangTheDrumSlowly = DoInOrder(ArmsOut,ArmsIn,Bunny.PlaySound('bang') )
BangTheDrumSlowly.Loop()
```

Each commands may be broken down into sub commands (As the user may wish to do so) and eventually just combine these sub commands into one large command to prevent the repetition of calling a certain action several times in a continued fashion.

3.2 Finite State Machine

3.2.1 Definition

Finite State Machines (FSM), also known as Finite State Automation (FSA) models of the behaviors of a system or a complex object, with a limited number of defined conditions or modes, where mode transitions change with circumstance. (Brownlee, 2003)

Finite state machines consist of 4 main elements:

- States which define behavior and may produce actions
- State transitions which are movement from one state to another
- Rules or conditions which must be met to allow a state transition
- Input events which are either externally or internally generated, which may possibly trigger rules and lead to state transitions.

A finite state machine must have an initial state which provides a starting point, and a current state which remembers the product of the last state transition. Received input events act as triggers, which cause an evaluation of some kind of the rules that govern the transitions from the current state to other states. (Brownlee, 2003) A sample implementation is shown in Figure 4.

3.2.2 Use in Decision-Making

Finite state machines have been used in decision making. Because of the ease in implementing these, they have often been applied in an AI Bot / Agent's logic, most specifically in games.

A certain scenario is represented as a State in the finite state machine. Possible choices that can be made given a certain scenario are treated as the rules required to traverse from one state to another, and the effect of making such choice are the state transitions (the process from moving from one state to another) in the given FSM.

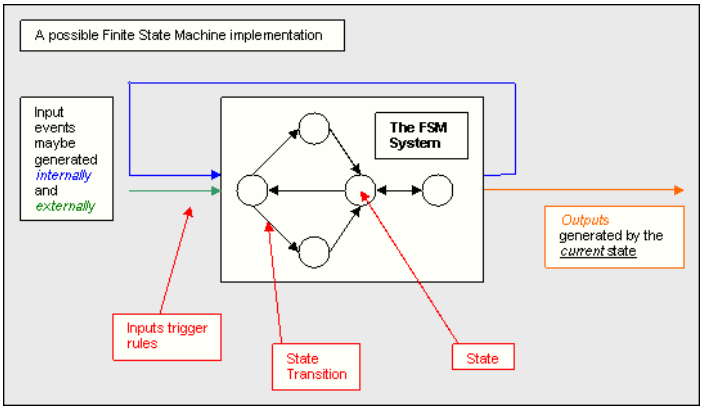

Figure 4 A finite state machine control system implementation

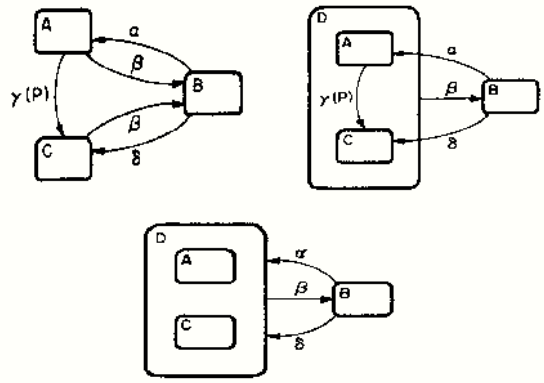

Figure 5 The transformation of A regular FSM to a grouped FSM

Though Finite State Machines are best to describe its action-reaction process, there is still difficulty in expressing the FSM in a formalized and simple way such that the operations of the FSM are abstracted from computer technical terminology (Harel, 1987). This problem also arises when a single FSM is consisted of multiple states and transitions such that it becomes too complex to describe actions for instance in a transformational system. Therefore, to solve this problem, the concept of state charts is introduced. State charts work by abstracting certain sub FSM states and transitions in such a way they are treated as one group. This ensures that these groups would work together to a single out come and thus making transition to other states generalized rather than the individual components of the composed states transitioning to other states. This not only makes the state machine easily understandable but it also optimizes

processing as all of the sub FSM states would ultimately produce a single outcome rather than taking into account all the duplicate transitions of different states towards a single state.

Take for example the diagram in Figure 3.4, this is a sample state where State A and State C only returns one transition to State B and State A transitions to State C only communicates when Condition P is satisfied. However, take note that State A and State B are virtually identical in terms of result and therefore, it is a waste to simply duplicate transitions that only leads to one state. This is where creating a super-state comes in. In the other figure, State A and State C is abstracted under State D. State D then only outputs whatever results State A and State C produces and passes it to B. Notice that State B does not need to know the workings of State D and State B only knows that it can travel to State D under two transition conditions.

3.3 AI Cap'n

AI Cap'n is a Java based game API power by the Golden Gate Game Engine (GTGE). It is a multiplayer agent-based game which pits multiple user-developed bots to battle in a death match type environment. It was developed by Solomon See, and Paul Inventado, both faculty members from the College of Computer Studies in De La Salle University Manila. Figure 6 shows how it looks like.

The aim of the game is to make use of multiple decision making algorithms, as well as path finding algorithms to score the most points. Points are earned through killing or damaging another bot in the game. Bots are defeated when they absorb enough damage to make their hit points hit zero. Items are available in the game which would aid the bots in scoring / defeating other bots such as items that would add more points to their hit points, enable them to move faster, and enable their shots to reach a wider range.

The GTGE library is required to run the Ai Cap'n engine. Aside from the GTGE library, the pre-made Ai Cap'n library is required to begin creating the bots to be attached to the game.

A bot is created by extending the predefined Bot class from the Ai Cap'n library. The predefined Bot class has three attributes: a ShipLogic object, a WorldState ob ject, and a String.

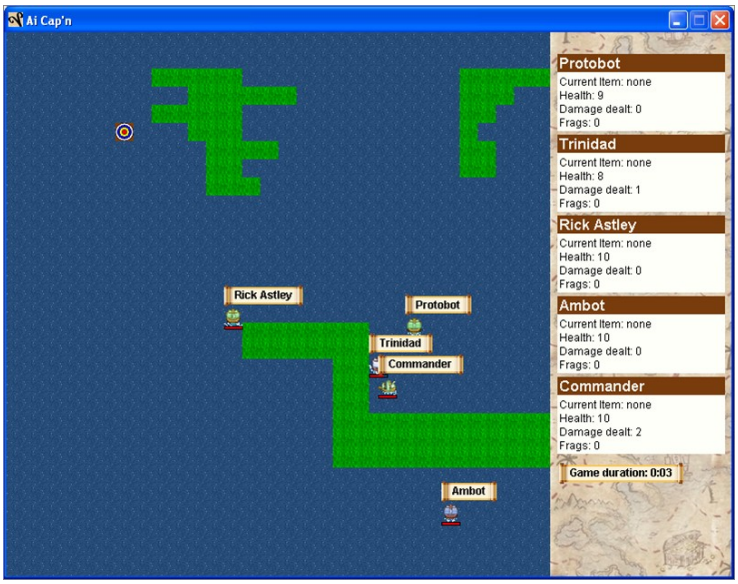

Figure 6 AI Cap'n in Action

The ShipLogic class contains all the methods that the player can do, such as fire, moveLeft, moveRight, getX, getY, useItem, setDirection, isAlive, and many more actions. The methods from ShipLogic for moving the bot only move the ship one tile at a time; therefore, the bot must call these methods repeatedly to accomplish a long trip. However, an action moveTo will be implemented which is a sequence of basic movements by the ship. Such method is done to relieve users who are new to using the system from the burden of creating their own path finding FSM. However, users who have become adept with making use of the basic features to develop bots may extend their knowledge and use this action to eventually develop their own desired path finding scheme.

The WorldState object is used as a reference. It represents what the whole world looks like since Ai Cap'n provides complete information to each user. The method getMap returns a 2-dimensional array of characters which represents the world: 0 representing passable block or tile, 1 for impassable blocks, and T for a target block. The method getTile(int x, int y) can be used to identify what character is currently on the specified coordinates. The isPassable(int x, int y) method is useful in determining the path of the bot. It tells whether the location of the specified coordinates is passable or not. The methods getMaxWidth and getMaxHeight is important so that the bot will not accidentally access the coordinates not existing in the world, which triggers an OutOfThisWorldException.

The getShips method returns an ArrayList of ShipState objects. The only accessible methods in the ShipState object (if it is not your ShipState) are getHealth, getDirection, getID, getX, and getY. Basically, only what is visible in the screen can be accessed. To obtain items, the method getCrates is needed. This method returns an ArrayList of CrateState objects. Each of these objects have methods getType, which identifies what type of crate it is, getX and getY.

The code below shows a sample bot named ”Abubot” and a part of the code used in implementing it. Included there is the action method used by the AI Cap'n engine to interpret actions. The only action that Abubot is programmed to do is to go to the first target on its list of targets.

```
package abubot;

import aicapn.exceptions.OutOfThisWorldException;
import java.awt.Point;
import java.util.ArrayList;

public class Abubot extends aicapn.bot.Bot{

@Override
public void action() {
        try{
        ArrayList<Point>
        shortestPath=findPath(world.getTargets().get(0).x,world.getTargets().get(0).y);
        for(Point p:shortestPath){
                sailTo(p.x,p.y);
        }catch(OutOfThisWorldException o){
                o.printStackTrace();
        }

        }
...
}
```

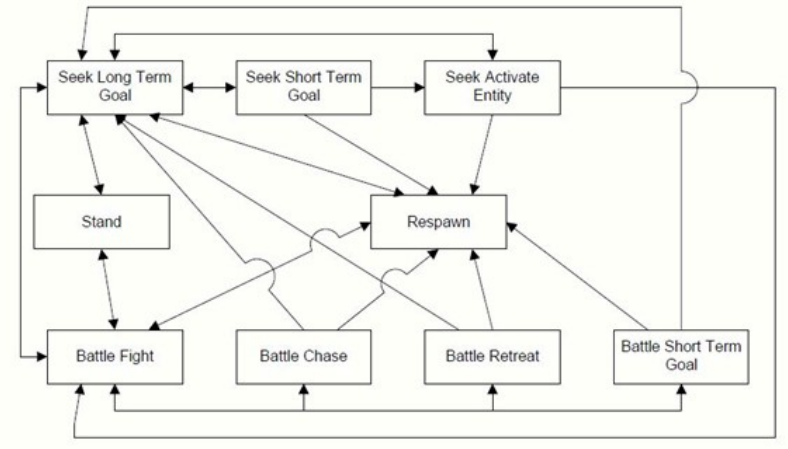

Figure 7 Quake III AI Network

3.4 The Quake III Arena Bot

3.4.1 Basic Actions

According to (Waveren, 2001), the bots in Quake III Arena possess a set of fundamental actions that are executed whenever the bot executes its AI. These actions are present both for the player and the bot. While human players use input devices such as a mouse or keyboard to execute these actions, the bot has practically no means to execute these actions using an external device, therefore, the bot has a set of primitives that in turn will execute these actions by the game engine as an outcome. Basic actions are significant to any gameplay because these primitives form the basis to generate strategy and as well as to make the decisions of the AI concrete to the user playing the game. For instance, one example of an advanced move in

Quake III Arena is called Rocket Jump in which the player must perform a move forward, a select weapon move, if the player has not armed his/her rocket launcher, a view move looking down to the o or and a fire and a jump move done simultaneously. This move will propel the player higher than the usual jump action making it possible to reach unreachable areas or executing it as a part of a tactic. There are other possible tactics that can be done using one or more basic actions. Refer to Table 4.1 in the Graphs and Tables Chapter for a list of actions.

3.4.2 AI Network

The Quake III Arena Bots features a design in such a way, acts as the processing unit whenever the bot needs to think on doing some decisions or actions. This design is modeled using finite state machines and uses a series of if-then-else statements at source code level. The bot at every frame will check itself if it is still under a state by executing those if-then-else statements. One state may lead to one or more states and if a certain criterion leading to the other state is met, the bot enters that state and again will repeat the process until the round ends.

The graph representation of the AI Network is represented in Figure 7. The boxes show the states that a bot can be in; the arrows represent the paths of transition a bot may go into. Upon the start of the game or after being killed, the bot goes into the Respawn state. Then, it may do some actions and go to the Seek Long Term Goal state. From there, the transition of the bot from one state to another may vary according to the situation the bot is in.

Table 3 The 11 States of the AI network of the Quake III Bot

Respawn	The bot enters this state if the bot resurrected from death or the game initially started
Seek Long Term Goal	This is the default state, the bot enters this state and will achieve try to achieve the goal, depending on the set game type
Seek Short Term Goal	The bot may encounter short term goals such as items and will enter this state if the bot can get there on a very short time.
Seek Activate Entity	The bot enters this state if the bot encounters buttons etc. that leads to puzzles.
Stand	The bot enters this state if it wants to initiate a chat.
Battle Fight	The bot enters this state if it encounters an enemy.
Battle Chase	The bot enters this state if it wants to pursue an enemy.
Battle Retreat	The bot enters this state if it wants to retreat from an enemy.
Battle Short Term Goal	The bot enters this state if a short term goal needs to be eliminated or battled with.
Observer	The bot enters this state if it goes in observer mode.
Intermission	The bot enters this state if the game goes in Intermission mode.

This finite state machine has shown instances of abstraction to reach its states. For example, the game would execute a function called AIEnter Respawn with optional messages and proceeds to the AINode Respawn function, which leaves the agent in a Respawn state. After

which, the bot will enter its Pre-Seek Long Term Goal state through AIEnter Seek LTG. In this state, the bot will search for goals to follow. If the bot finds no goal to follow, it will enter into the Seek Long Term Goal State by AIEnter Seek LTG.

3.4.3 Goals

According to (Waveren, 2001), the Quake III Bot classifies different elements in the game as Short Term Goals and Long Term Goals. Short Term Goals are any goals that will help aid in accomplishing a Long Term Goal. Long Term Goals in turn are any goals that will help a player win the game. An example scenario of a Short Term Goal is that if a bot spots a Quad Damage powerup, it will go to its lo cation and obtain the item. In turn, if it spots the enemy, which is a Long Term Goal, it will engage the enemy.

The Goaling system for Quake III Arena is implemented using a Stack-Based system and also includes distance and time calculation to reach a certain goal. The bots obtain goals if it sees one and calculates the time and the path the goal is found. The Bot continuously do this until either the Bot achieves the goal or the Bot sees the goal as already non-existent or has been dealt with. Also, the bot will discard a goal if it is taking to o long to achieve the goal.

```
void AIEnter_Respawn(bot_state_t *bs, char *s)
{
	//chat code ommited
	if (BotChat_Death(bs))
	{
		bs->respawn_time = FloatTime() + BotChatTime(bs);
		bs->respawnchat_time = FloatTime();
	}
	else
	 {
		bs->respawn_time = FloatTime() + 1 + random();
		bs->respawnchat_time = 0;
	}
	//set respawn state
	bs->respawn_wait = qfalse;
	bs->ainode = AINode_Respawn;
}
int AINode_Respawn(bot_state_t *bs)
{
	// if waiting for the actual respawn
	if (bs->respawn_wait) {
		if (!BotIsDead(bs)) {
			AIEnter_Seek_LTG(bs, "respawn: respawned");
		}
		else {
			trap_EA_Respawn(bs->client);
		}
	}
	else if (bs->respawn_time < FloatTime()) {
		// wait until respawned
		bs->respawn_wait = qtrue;
		// elementary action respawn
		trap_EA_Respawn(bs->client);
		//
		if (bs->respawnchat_time) {
			trap_BotEnterChat(bs->cs, 0, bs->chatto);
			bs->enemy = -1;
		}
	}
	if (bs->respawnchat_time && bs->respawnchat_time < FloatTime() - 0.5) {
		trap_EA_Talk(bs->client);
	}
	//
	return qtrue;
}
void AIEnter_Seek_LTG(bot_state_t *bs, char *s) {
	bot_goal_t goal;
	char buf[144];

	if (trap_BotGetTopGoal(bs->gs, &goal)) {
		trap_BotGoalName(goal.number, buf, 144);
		BotRecordNodeSwitch(bs, "seek LTG", buf, s);
	}
	else {
		BotRecordNodeSwitch(bs, "seek LTG", "no goal", s);
	}
	bs->ainode = AINode_Seek_LTG;
}
```

Figure 8 Quake III Arena Sample Code

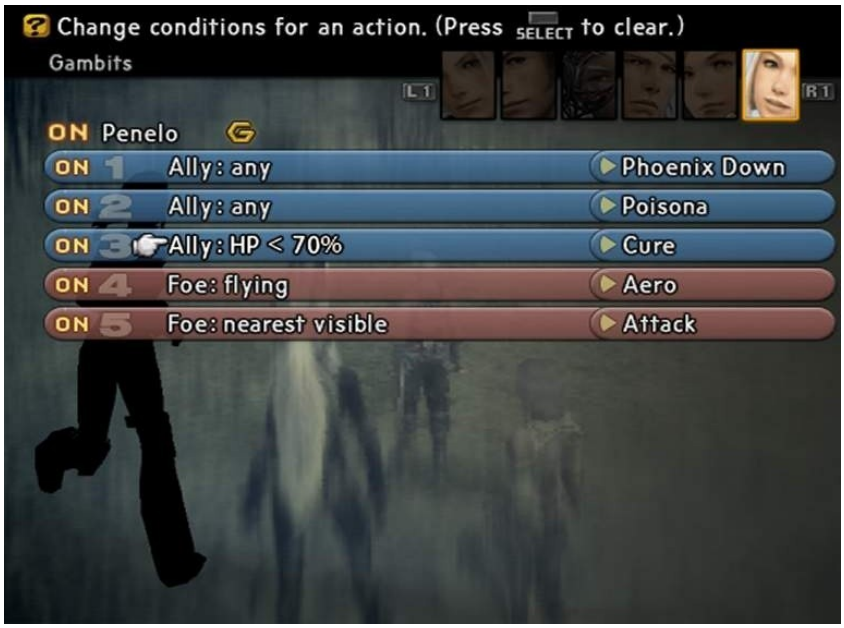

Figure 9 Final Fantasy XII Gambit System

3.5 Final Fantasy XII Gambit System

Final Fantasy XII is a PlayStation 2 game developed by the game company Square Enix. Within the game's system, lies a unique battle intelligence system which utilizes programming concepts as a means to control other playable characters called the Gambit System. Gambits are a set of rules which a character follows as a form of user-defined intelligence. It consists of the priority, target & condition, and action. In the list of rules/gambits, the rule with the highest (1 is highest) priority and is true will be repeated until it is false. If all rules are not true, agent is idle. Depending on the combination of gambits being used, a character can be configured to any type of behaviour in terms of using items, fighting defensively, or using magic o ensively among other types.

The configuration of gambits in Figure 9 can be translated as:

1. If any ally needs Phoenix Down (revives from death), use it; *
2. If any ally needs Poisona (cures poisoned state), use it; *
3. If any ally has less than 70% HP, use Cure (heal HP). *
4. If there is a ying foe, use Aero. *
5. If there is a near and visible enemy, attack it. *

*Loops While True

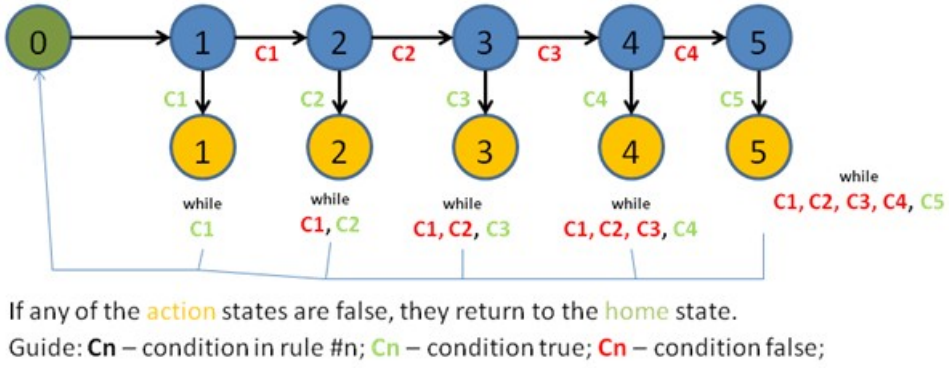

Figure 10 FSM Representation of the Gambit System

Each gambit is just composed of one target & condition and only one action. Target & Conditions target three objects/entities: Self, Ally, and Foe. Self targets the gambit owner, Ally targets friendly units in your party, and Foe targets enemies. Following the target parameter is the condition to apply to the target. (Pringle, 2009) lists these conditions and classifies them by target. The list has been added in the appendix (Section 5.3)

Upon tracing the way gambits work, Figure 3.9 shows how the previous example can be depicted as a finite-state machine. There exists an idle state within the states presented (in green) which is invoked when no other condition is satisfied. Based on the priority arrangement of these gambits, when the first priority gambit is true, then the agent performs the action continuously until any higher priority gambit is true. In the case of the first gambit, it will continue to do the assigned action as long as it is true and prevents lower priority actions to execute.

The states are travelled continuously in a loop (as shown in an image) and continuously checks the conditions which are true. When it finds a true condition, it performs the action and returns to its starting point.

4 AI Shipyard

The main system being created in this research. The main purpose of the tool is to create bots for the AI Cap'n game to address issues of manually programming them.

4.1 System Overview

The AI Shipyard is designed as an artificial intelligence agent development tool for the game Ai Cap'n that uses a series of finite state machines to model the agent's behavior. The tool will be aimed at users who have no background on any programming language. AI Shipyard will attempt to simplify the process of developing agents by mainly abstracting the user from text-based development environment and providing the user with a set of graphic-based, menu-driven development environment to model their custom-made agent. The software, in turn, will output such a file containing such a representation of the agent. This representation will be translated by a code generator into the Java programming language. The said programming language is used by the actual AI Cap'n game to represent agents.

4.2 System Objectives

4.2.1 General Objective

To create an environment utilizing a finite state machine framework to create AI agents for AI Cap'n.

4.2.2 Specific Objectives

- To provide a scope for displaying a finite state machine representation of an AI agent.
- To provide a default single-state FSM which the users can use and extend to create their own finite state machine.
- To provide a module which allow users to customize transitional conditions of their states and define actions their agents can use.
- To provide a means to simulate an agent's personality in a controlled environment.

4.3 System Limitations

The proponents will not change anything regarding how the AI Cap'n engine works. The proponents will only be concerned in the agent-creation phase of AI Cap'n. The system supports a bot framework patterned after a finite state machine's architecture. Therefore, the agent will not utilize threads and rely on a set of states defined by the user. The system will not support stochasticity in actions and remain deterministic in nature for simulation analysis purposes. The conditions to be used remain propositional to mirror the logic used by AI Cap'n.

4.4 System Terminology

This supplemental section lists down several terms which may be used in describing the system functions. In order to remove redundancy in definition, refer to the following terms.

- Battle Mode - These are synonymous to the states of an FSM. These are composed of several action sets arranged by priority. This defines the actions of an agent at a given point in time. Upon initializing the system, there is a default battle mode with an empty

action list known as the **Idle** battle mode.
• Battle Plan – These are composed to conditions and a target battle plan. This facilitates the conditions which the bot will move to another battle mode during a fight.
• Strategy - These are composed of conditions, a target, and a corresponding action. These are shown in the system as lists of strategy in each strategy.
• Condition - defined as the proposition/s that must be true in order for the action in the same action set to be performed
• Target - defines the receiver of the action. Choosing Enemy refers to the enemy to which the ”Enemy” conditions (e.g. Enemy: HP ¡ 50%) are true. Choosing Item coincides with ”Item” conditions (e.g. Item: Health exists). Choosing Self is generally used for actions without targets in mind.
• Action - defines the commands to be executed when the condition of the action set is considered true. It can be a simple action which uses only one command, or a combination of commands done serially.

4.5 Architectural Design

AI Shipyard outputs a java file to be used by AI Cap’n. The process of AI Shipyard utilizes an FSM Representation which is used by an internal Code Generator upon request to be translated into the corresponding output Java File. Inside AI Shipyard lies three visible modules, the FSM Viewer, the State Editor, and the Simulator, each with their own functions and correspondence with each other.

The main user interface of AI Shipyardis composed of the following parts (as shown in 4.1):

FSM Viewer. Shows a visual representation of the states in a given bot project.

State Editor. The main module for editing the contents of the strategies. Allows users to arrange the AI actions by applying the conditions, targets, and actions based on a priority queue. It shows an Action Set table, which lists all the actions being performed in that state based on a priority queue represented in a table. Another process which can be used in this state is the Customizer. This process allows users to define some pre-existing conditions or targets, and users may also create their own conditions/actions by combining several of the native conditions/actions, respectively. The editing of these codes is done in a Java code level and can be defined by a user with the aid of some instructions in the description. Custom conditions can be saved individually for future use.

Simulator. This module allows users to test how your agent reacts to conditions which the user can define. This module simulates a game where parameters can be defined. The output will be shown through the Debug Viewer to be able to trace how the bot moves. The following parameters can be defined for the simulator:

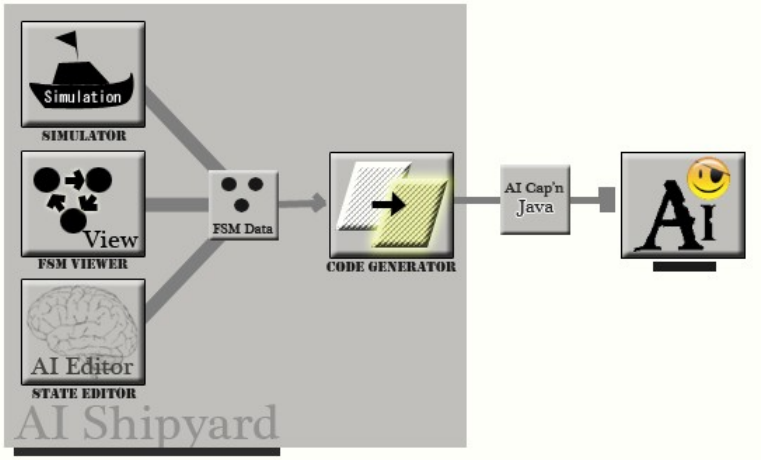

Figure 11 AI Shipyard System Framework

- the agent's health;
- the items present on the field, their positions
- the enemies health;
- the speed of the game, pausing the game;

The FSM Data being used between the modules is a textual representation of the state machine. The data is formed by strategies. A strategy has several action sets. Each action set consists of conditions, target, and action. Each action set has a priority measure, where the TOP MOST condition is checked first. This data is saved in the workspace.

Code Generator. This backend script translates the FSM Data into a Java file (AI Cap'n Java in Figure 4.1) which is used by AI Cap'n to generate bots in the game map environment. The process of Code Generation utilizes the following format to create a bot:

```
if(inBattleMode)
{
        if( conditions ) //first battle plan
        {

        }
        else if( conditions ) //second battle plan
        {
        }
        //repeat for all battle plans
        else
        {
                if( conditions ) //first strategy
                {
                        //action
                }
                else if( conditions ) //second strategy
```

```
                    {
                            //action
                    }
                    //repeat for all strategies
            }
    }
    //repeat for all battle modes
```

AI Cap'n. AI Cap'n will not be part of the system, but is necessary to run the bot created by the system. The input will be the java file generated by the system.

4.6 System Functions

Refer to Appendix B for reference screenshots. Detailed processes are available in the user manual.

4.6.1 *Master Controls* Menu Functions

- **Open Blueprint**. Loads a project file (.aip) you may have previously saved.
- **New Blueprint**. Creates a new project. It has a single state called *Idle*.
- **Save Blueprint**. Saves a project into a project (.aip) file.
- **Import Battle Mode/s**. Allows you to load a battle mode file (.ais) into the current project.
- **Rename Ship**. Changes the name of the current ship.
- **Export Ship**. Allows you to save the bot as a java file. Along with its class file as well.
- **Abandon Ship**. Closes the project.

4.6.2 *Forge* Menu Functions

- **Strategy Database**. Loads a project file (.aip) you may have previously saved.
- **Test Cruise Ship**. Loads a project file (.aip) you may have previously saved.
- **Set Raw Materials**. Loads a project file (.aip) you may have previously saved.

4.6.3 *Help* Menu Functions

- **Open Help**. Opens the help file in your default browser.
- **About**. Displays a small info box about the authors of the program.

4.6.4 *FSM Editor* Functions

- **Add Battle Mode**. Adds a new battle mode.
- **Delete Battle Mode**. Deletes the currently selected battle mode.
- **Rename Battle Mode**. Renames the currently selected battle mode.
- **Save Battle Mode**. Exports this battle mode as an AI Shipyard State.

- **Add New Battle Plan**. Adds a new battle plan to the table view.
- **Edit Battle Plan**. Edits the selected battle plans' condition and target state.
- **Edit Battle Plan (Target Only)**. Edits the selected battle plans' target state.
- **Delete Battle Plan**. Deletes the selected battle plan.

• **Add New Strategy**. Adds a new strategy to the table view.
• **Edit Strategy**. Edits the selected strategy condition and target state
• **Edit Strategy (Action Only)**. Edits the selected strategy' target state.
• **Delete Strategy**. Deletes the selected strategy.

4.6.5 *Strategy Database* Functions

• **Add Action/Condition**. Lets one add a new action or condition.
• **Delete Action/Condition**. Deletes the selected action or condition.
• **Rename Action/Condition**. Renames the selected action or condition.
• **Edit Description.** Edits the current description of the selected action or condition.
• **Advanced Add.** Opens a text editor which lets the user code in JAVA.
• **Easy Add.** Opens a code helper program to easily make action or conditions.
• **View**. Lets you view the code of the currently selected action or condition.

4.6.3 *Debugger* Functions

• **Adjust Speed**. Allows user to slow down game time by two levels.
• **Pause Game**. Completely halts game time.
• **Change Bot Health**. Adjusts selected bot's health
• **Change Equipped Item**. Opens the help file in your default browser.
• **Add Item**. Add a specific item in the field at a specific x,y location.

The steps to making a bot are summarized as follows.

• **Create Bot/Ship**. Creates an empty ship project with a default state named "Idle", which serves as the starting state.
• **Add State**. Creates a new state if one is needed. Selecting the state shows an empty set of actions.
• **Add Action Set**. Adds a new action set to the state.
• **Select Condition**. Defines the conditions of the new action set by selecting one or more from a set of defined conditions. Combining of conditions can be defined disjunctively through grouping and defining relationship (AND or OR) before adding a condition or group of conditions.
• **Select Target**. Defines the target of the new action set.
• **Select Action**. Defines the action to be used by the action set by selecting one or more from a set of defined actions.
• **Generate Code**. Create the java output file to be used by AI Cap'n.

4.7 Physical Environment and Resources

Minimum Hardware Requirements:

• Any x86-based Dual Core Processor or greater
• 1GB RAM or greater
• 15MB of Free Hard Disk Space
• 64MB Graphics Memory or greater
• Mouse and Keyboard Input

Minimum Software Requirements:

• Microsoft Windows 2000/XP/Vista/7 Operating System
• Latest Java Development Kit Runtime Environment
• A Working copy of AI Cap'n for agent testing
• At least 1024x768 resolution for all the windows to appear without error

User Specifications:

• User must be able to understand college level English
• User must be computer literate, that is be able to use the computer
• User must know the basics of the AI Cap'n game

4.8 Agent Implementation

The agent assesses its actions through the workflow as shown in Figure 4.7. It follows a simple looping of collecting information (Sense), evaluating conditions before an action (Think) and implementing actions (Act). The state machine in Figure 4.7 should notably have no end state, as the AI Cap'n game stops its agents when there is exactly one bot left, ending the game.

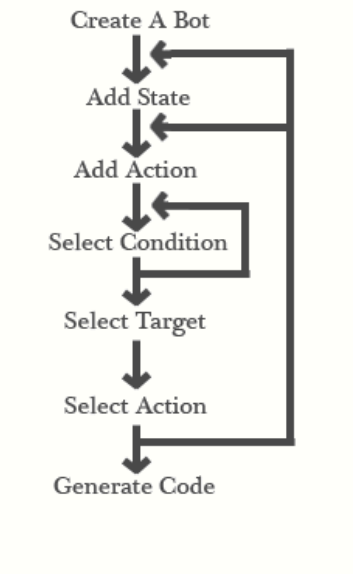

Figure 12 Flow of Agent Creation

Spawn. The start of the game wherein the bots are initialized. After which, the bot changes state to Sense.

Sense. The bot will collect data into pre-defined variables. The data utilizes the WorldState's (AI Cap'n) variables which includes, and may not be limited to, the list of all enemies and the list of items of the map. The user is able to customize this by defining conditions in the Customizer module. It should be noted that the variables used are dependent on the conditions used by the bot.

Think. This state is dependent on the variables initialized in the Sense state. Composed of the states horizontally parallel to it: Get Actions and Find 1st satisfied condition.

- Get Possible Actions. Collects the list of actions (composed of the condition and corresponding action) which was defined to it in the FSM Editor.
- Find first satisfied condition. Traverses the list of actions to find the first true condition, which may be composed of compounded conditions, in the list. If there are no existing satisfied conditions, they bot returns to its Sense state.

Act. Executes the action specified by the TRUE condition determined in the Think states. If there are still enemies existing after the action, the bot returns to the Sense state.

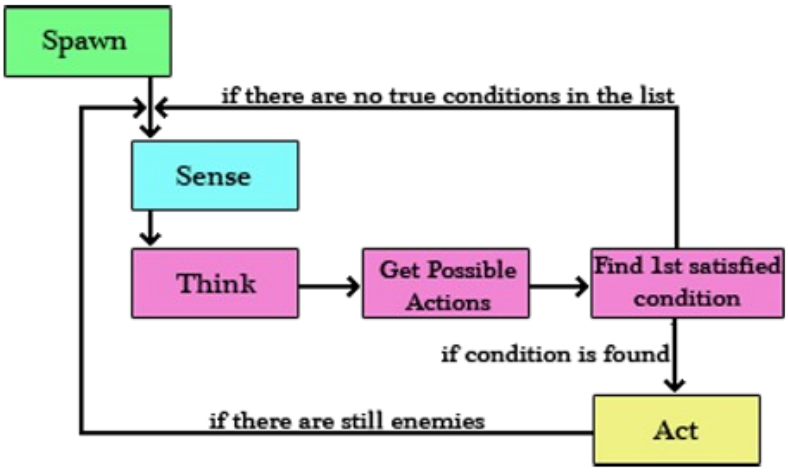

Figure 13 How The Bot Thinks

4.9 System Rationale

In order to illustrate the advantages of AI Shipyard as a system, the proponents justify that:

• AI Shipyard help students visualize the agent-based intelligence by separating the decision making into states and presented in a visual interface.
• AI Shipyard simplifies creating low-level actions of agents and therefore allowing the user to create higher-level actions.
• If, for example, a student has an idea on what his agent should do, but can't program it in Java, he can use AI Shipyard to model the agent and then check the output Java code to see how it is coded in Java language.

4.10 Debugger

Another option to set up the debugging environment would be to debug with an implanted JFrame code within the bots. The purpose of this debugging feature is to enable means to debug the bot and tracing its use. For this, two parts are created. A textual viewer similar system.out.println function call in Java for users who are not inclined to programming, automatically printing out the current state of their respective bot, enabling users to monitor their bots actions; and also a visual representation of its states, which highlight with the same input as the textual viewer. This is done by embedding a code for a JFrame with a text area inside the generated code.

The code for the Debug class contains the following methods:

- Debug. instantiates a new instance of the debug frame
- initComponents. initializes the components of the JFrame
- debug message. prints out a message in the text area.

The visual representation is called the same way as the textual viewer and can be interchanged with the textual viewer. The input used by the textual input takes note of the current state and highlights the particular circle or node of the FSM specified by it. Utilizing Java shapes and associating each one to a strategy enables the tracing process with changing their colors.

5 Design and Implementation Issues

In this section, issues encountered in the development stages of AI Cap'n are discussed, along with the solutions applied.

5.1 User Interface

5.1.1 UI Design and Implementation

The GUI Design is the oldest component in the software and has undergone several changes up to the point of an overhaul. There are notable components in the GUI namely, the Graph that is responsible for visualizing the States and the Stack system used in generating subconditions for adding conditions for transition/actions.

All GUIs with the exception of the Debugger were placed in the Main class. This is to give the Main class control over all GUIs. This is helpful especially when hiding other windows or disabling other windows when a window is opened. The first design of the GUI composed of mainly ListBoxes and Tables. The objective was to provide the user with all information at once. This was a direct application of the concept of the Gambit system from the game "Final Fantasy XIII". However, as development went by, it was noticed that the GUI was getting cumbersome to use and that it required altering core objects. Therefore, a complete modification of the interface was required to make the software easier to use.

The current design of the GUI was based on Blizzard's StarEditor for the game Starcraft. To give its credit, the GUI in that application was designed in such a way that the user is presented with a list of "Triggers", and the use can specify custom conditions and actions by selecting from a list. Therefore, this became a factor in the graphical overhaul of the system. Since the GUI was modified, it also made optimizations to the core objects such as removing some unnecessary methods and removing two objects namely: "CompoundCondition" and "CompoundAction", which were results of the old GUI. The data that is being held by the objects were also simplified up to the point where only Strings and Vectors we're being used.

Blizzard's design on the StarEditor also contributed in the designing of the Stacking system used in making subconditions. A class called EZOpsManager, is in charge of maintaining a stack of Condition-related windows which were the High Level Conditions editor and the Low Level Conditions Editor. In order to make subconditions possible, each instance of the window is called "Layers". Each Layer comprises of certain precedence in terms of mathematics. Therefore, the higher the Layer is, the equation is more prioritized when evaluated. When one editor is closed, depending on the operation, the data from that editor is passed into the previous editor; hence, it becomes a subcondition for the previous frame. The data being manipulated here is just purely strings as it goes in a series of concatenation.

Aside from the Stack system, one of the critical components of the GUI and the hardest to implement is the Graph. This module was not made until the discovery of "JGraphT", an open source library used to display graph theory concepts. The library is built on top of "JGraph" and made the system easier to use. In order for the Graph to work, all that was needed was to specify the graph to used, add edges and vertices using Strings and pass this graph to a JGraphAdapter. The JGraphAdpater will in turn, be passed to a JGraph object, hence, and will be rendered. The advantage of this library is that when something happens to the backend graph, it automatically updates, given the viewer is still running. To add the ability to highlight nodes, the JFrame that the Graph resides in can be added by a JPanel on its JGlassPane layer.

This will enable the developer to draw circles on the coordinate of the nodes. More of this will be discussed in the debugger which is also a product of the Graph.

5.1.2 Issues Encountered

The GUI Design has undergone several changes starting from its prototype form until the final version. It is the oldest component in the thesis. The old GUI follows a more list-filled design. This entailed the users to have all the data presented to them at once. In the long run of its development, it became cumbersome to use as it required the user to constantly click the states. It was deemed large and spacious given the data presented. In the middle stage of development, the GUI was overhauled and it was smaller and had less use of tables and lists and more on comboboxes. The GUI became easier to use, however, like any other program, the learning curve is still there. This also forced a change in the design of the core objects in a positive way. The problems pertaining to this aspect will be discussed later.

Another point that led to a series of problems is the design of the Condition/Action. Previously, the module followed a similar design: lists. The idea is that users should be able to "move" objects by the buttons. However, it became far more complex when adding compound conditions/actions. The user was compelled to again use the buttons to add or remove objects in the lists. What's more is that the user should toggle and/or buttons for every two lines. There were also some operations present in that module that will make the operation even more complex like "Merge and Split Conditions". This design has influenced the core objects by adding new objects just for compound conditions and actions. Later in the middle stages, along with the GUI overhaul, the core objects were revised and it ended up positively.

The second component of the thesis with the most revisions was the *core object design*. Although the design was made simple at start, it had undergone changes that were influenced by the GUI. To start, the basic design of the software as stated is simple in such a way that no Inheritance or any other forms of complex Object oriented concept was used. It was just a simple set of objects in which there are sets of relationships. However, the integrity of the design was put into disarray due to the changes in the Condition/Action module. This gave birth to two new objects that are inherited from their parents namely the "CompoundCondition" and "CompoundAction". These objects were added to suit the needs of compound conditions and actions. However, there came to a point that it became cumbersome to differentiate just Conditions and CompoundConditions and the same goes for actions. Due to this multitude of problems, the core design was changed in the middle stage and has removed several unnecessary methods and attributes, as well s the two introduced objects. The end result is a more simplified design that made use of string manipulation. Therefore, it became easier to maintain the software.

Arguably, the graph is the hardest to develop in the GUI component. This module did not appear until the discovery of the library "JGraphT". The current implementation of the graph was based on the demo applet of the library. There were several issues that arose while developing this module.

Firstly, the module wasn't running on a JFrame, but rather in the applet itself. It was modified in such a way that it used the data of the states. This is the stage where the module was experimented on to transfer the entire module into just a JFrame. At this point, the library is being studied and few improvements have been made such as being able to be moved or control what can be done with the library. Sooner this was achieved along with the ability to use a JFrame rather than a JApplet.

The second issue after the port was the ability to highlight the nodes in the graph. This feature is not present natively on the graph so research needed to be done in order to achieve this. Fortunately, Java has a built in feature called the “GlassPane”. It is basically a JPanel over a JPanel and this top layer panel can be used to draw Java shapes. In order to achieve this, the transfer from JApplet to JFrame needed be done, which after doing so, made the feature easy to install. However, in lieu with the debugger, the trace suffered problems, particularly on the communication of the debug bot to the program since both of the elements are not related to each other. The was later solved by making use of sockets.

The final issue that the graph faced is the cluttering of the text in the edges and the ability of the graph to scroll. These issues appeared during the testing phase. The text in the graphs may not be removed by the functions provided for the graph, but research found a sample to remove the text. It required rebuilding the graph and it was a complex code, thus solving the cluttered text. The scroll feature however was made on a trial and error basis. Since the graph is a subset of a JComponent, it should be easy to be docked inside a JScrollPane. It took several tries until realizing that any JComponent in Netbeans must have the setBounds to be visible in the form. This successfully added the scrollbars and therefore, removing the scroll limitation (Refer to Appendix B for comparisons of past and present versions).

5.2 Code Generation

5.2.1 Algorithm

AI Shipyard, being a bot generation program, requires several code generation algorithms. The code generation is underlying the main window of the AI Shipyard platform through several methods: Saving, Loading, and Bot Generation.

The AI Shipyard ship is represented in terms of structures defined in AI Shipyard. These include the list of Battle Modes (list of underlying Battle Plans and Strategies), list of all Actions, and list of all conditions. Using this information, the underlying parser (ASCodeParser.java) will translate them into code through its *createContent* methods and can read the output files through the *parseContent* methods. These createContent methods convert the data into readable XML files which can be saved to a file either as a project (.aip) or a state (.ais) using the file processor (ASFileProcess.java aided by the TextFilter.java). All reading (ASFileReader.java) and writing (ASFileWriter.java) implementations are handled by their respective classes.

The project differs from the state, in which the state is only a subset of the project file. The state file only refers to one state and all used actions and conditions without the transitions, while the project file encompasses all aspects of the program itself, including the transitions and graph coordinate placements.

In creating the bot’s java file, which is needed for compiling the class files or for debugging, it uses a different approach. It is stranslated into blocks of if-else statements, which traverses all the battle modes, their battle plans, and finally their actions. Wherein it searches for the first true condition and does it’s corresponding action, else they move on to the next condition and loop the process. It uses the Sense-Think-Feel paradigm of AI (see Figure 13, p30).

Later on, in order to handle the implicitness of actions to their conditions, the concept of get creating methods for checking their existence along with getting the perfect match is defined. These methods are split into the *getEnemyExist_n() / getItemExist() / getEnemyItemExist*() methods where n is the condition number it is referring to. A method to actually get the target is in the form *getEnemyMatch_n() / getItemMatch_n()* where n is the corresponding action to the

condition number of the Exist functions. These *getXMatch()* methods return the respective type of targets that are needed by the actions. If there is an action using enemy information in the strategies area, it adds the getEnemyMatch() for the target being searched. The getItemMatch() returns a CrateState which is the information of the item being searched for.

5.2.2 Issues

The first issue arose in first developing the parser itself. The structure was yet to be defined. When the structure was finally defined, an XML representation of states and their transitions and actions were developed, since XML was a simple way of representing data. Using a special parser to read through the contents of the XML file, the AI Shipyard structure is populated and finally displays all the information to the screen.

The second issue is with regards to the generation of the bot construct. First of all, the bot construct should have been defined into how the structure looks like. AI Cap'n only really utilizes the action() method for the bot's logic. As such, the structure was made to look like that of an if-else structure. External templates were created for the purpose of not making the templates hardcoded and easily switchable.

The third issue is with regards to how conditions and their targets can be defined. The conditions, even if one can just place the code directly to the conditions part of the code, will not be able to tell the person what the target will be. Thus, the solution was to at first parse the conditions given to create its own methods for targeting and verifying if the conditions are true. That is where the getXMatch and getXExist() methods were developed for, as explained earlier.

The last major issue is with regards to parsing conditions separately. In order to make sure there is no conflict between looking for either the Enemy or Item is found, the conditions were parsed separately. Earlier on, the error was that the parser would get confused when both enemy and item conditions existed in the same condition area as they were parsed at the same time. Now, the two are parsed separately for things to work.

5.3 Debugger

5.3.1 Algorithm

The debugging environment is made up of three parts, namely the state viewer / tracer, the controller, and the AiCap'n game environment.

Once the debugging procedure is called, a defaultbotxx.class (where xx is an integer) which contains the class file of the bot the user would wish to debug. This is done by invoking a "javac" command and generating the code of the bot in a separate folder. This is so that the bot will become accessible for the debugging environment to use.

Before initializing the debugging environment, a Starter object is called, which imitates the actions that the opening window displayed in the game of AiCap'n does where the user is free to add bots from the pre defined list into the environment. By default, a bot is already in the list of bots to be added to the environment, which is a bot named "DebugBot", which is the default bot name for their debugging bot, which follows the actions of the bot they intend to prepare. A maximum of five bots may be added, inclusive of the DebugBot.

Upon pressing the start button, the Starter object passes the string inside the bot listing text bot and parses the received string. By splitting the string received into an array of smaller strings by separating each token through the ‘,’ character, the client is able to add the desired bots into the debugging environment depending on the token in the array.

The AiCap’n environment makes use of the same environment version 2.17 used as well as the same powerups, same time limit, and the same limitations and scoring mechanics. However, lines of code were edited internally to enable the modification of the environment during runtime. Functions ‘addDebugItem’ and ‘initItemDebugger’ were some of the few functions added that enabled the adding of items when called upon. Bots also needed a special ‘setHealth’ function that would enable the modification of their health during run time. Special lines of code were also added into the main ‘update’ function to enable the pausing of the game or by altering the game’s speed.

The flags and new functions were called through the Controller object. The Controller object is a private attribute of the AiCapn object which was added to enable the user to modify the game’s environment as it progressed. The Controller may add items by making use of either ‘getDebugRage’, getDebugSpeed’, ‘getDebugHealth’, ‘getDebugSlow’, or ‘getDebugMine’ and calling it’s respective init function. The Controller is also able to control the speed by passing a flag value to the AiCap’n environment, namely indicated through which radio button in the speed modifier area is selected. The same is applied for the bots as the Controller makes use of the newly added function ‘setHealth’ and simply calls it given the bot’s index (selected from a list in the Controller) in the game along with the integer given by the user (placed in the text box).

The users are able to trace the bot’s behavior through the Debug Viewer. The visual debugger is an extension of the Visualizer found in the Main Window. It has a similar implementation to that found in the Main albeit with few minor changes. Aside from the frame itself, there is a JGlassPane which is in the form of a JFrame. This allows the frame to be painted through the outer layer of the JFrame. This is essential to make the circle that is put on top of the nodes in the graph. To guide the frame on which node is to be highlighted, a function has been made to do this and is repeatedly called when the debugBot is running; it is called “highlight (int, int, int)” where the three integer parameters are as follows: the first one pertains to what state is being highlighted, the second one pertains if the transitions or the actions are to be highlighted in the tables and the last parameter, given the previous parameter, will highlight the nth item. The frame also refreshes itself every time the method is called. This is possible through the sockets made and the messages given to the debugger are used for the “highlight (int,int,int)” function.

The bots developed communicate with the state viewer through sockets. In the generated bot for debugging purposes (named defaultBotxx where xx = an integer), a block of code is inserted to enable communication with the state viewer. Figure 14 Code Enabling Sockets displays the block of code. Through this socket that the bot will connect with are they able to send messages through a pw.println() statement to the state viewer regarding which state they are in and which action they are performing.

```
PrintWriter pw = null;
try {
    Socket s = new Socket(InetAddress.getLocalHost().getHostAddress(), 2000); /,
    pw = new PrintWriter(s.getOutputStream(), true);
} catch (UnknownHostException ex) {
    Logger.getLogger(defaultBot10.class.getName()).log(Level.SEVERE, null, ex);
} catch (IOException ex) {
    Logger.getLogger(defaultBot10.class.getName()).log(Level.SEVERE, null, ex);
}
```

Figure 14 Code Enabling Sockets

5.3.2 Issues during development

During the development phase of the debugger environment, questions arose regarding how the environment should be, the functions, and how it should be designed in such a way that it would be beneficial for the user. Upon coming up with the specifications through comments and suggestions from their thesis adviser, the development phase was undertaken. During the process, several issues arose.

The first issue was that the code for AiCap'n was outdated as due to an unfortunate event of the most recent and updated source code being lost due to a previous hard drive crash, the system were forced to make do with a severely outdated version of the game, a version which still contained numerous bugs and lacked many features present in the updated version which the proponents were in need of (ie, Dynamic class loader). This was solved through a careful modification of the source code such that the errors present were fixed but the modified code contained less features than what the most up to date version would have contained.

The second issue encountered was the requirement of heavy modification of the AiCap'n source code caused the game to break at times with bugs that cannot be traced from the code altered (as Ai Cap'n is heavily dependent on the GTGE game engine). There was a necessity to edit the code in such a way that it would enable the game to interact with several other frames, more particularly the controller frame. A workaround was found for such conditions (IE, slowing down the game, spawning items, modifying health) without the use of socket programming and by simply making the controller frame an attribute of the game itself. However, there were several changes to the game's original source code made, and at many times, the errors that would occur due to code experimentation would often break the game down, resulting in the necessity of archiving several backup files at all times. Through careful code modification and through careful testing after adding certain lines, it reduced the necessity to continuously add.

The third issue was getting the debug window tracer to interact with the bot developed and to inform the user the bot's current sate. The most difficult area that was faced was the actual interaction of the graph's highlighter and the actual bot itself. Through countless experimentation and researching for a solution that could be expressed in threads alone, socket programming proved to be the solution to get the module functioning correctly. However, there were times when a "lag" was encountered where the bot would switch states before the debug tracer would receive the message that the bot has changed states and caused a short delay in highlighting the state. The issue caused a "speed modifier" feature in the debug controller to enable the game to slow down so that users will have an easier time monitoring their bot's behavior.

The fourth issue was the layout of the frames: In its original design, there were supposedly four frames: The game window, the controller, the action list, and the debug window tracer. This however would clog the desktop, impeding the user's view of the game and the bot's current state as well as make it hard by requiring the user to switch from frame to frame. The game window's dimensions made it challenging to layout the frames correctly as the it was impossible to remove one frame as all four of them were very important in adding to the visual environment that AI shipyard had to offer. This was solved by combining the debug window tracer and the action list into one frame and redesigning the controller window in such a way

that the number of frames would be reduced to three and all frames except the game window would be reduced into smaller dimensions, easing the clogged appearance of the screen.

The fifth issue regarded the dynamic class loader. The program had the necessity of compiling and loading class files (of bot files) in places outside the set CLASSPATH. As the original code in the most up to date version of AiCap'n was lost, ways to modify the class path during runtime had to be researched upon. Researches regarding the JAVA classpath yielded results that it was indeed possible to modify the classpath during runtime through the use of the URLClassLoader object, enabling the loading the class defaultBot which served as a dummy class file for bots developed by the user that are being debugged.

The sixth issue encountered was the loading of updated .class files. While using the Class.forname() solution proved to be successful, it would not load an updated version of the defaultBot class file, where the necessity to close and rerun the program was present in order for the debugger to catch the updated version. This was because the class was loaded immediately into the memory and was accessed from memory instead of from the filepath given if it is loaded the second time around. This was solved by assigning the defaultBot class files with numbers written at the end of their filename, indicating a distinct and updated class file, signaling the Class.forname() command to load the updated class.

At one part of the implementation, there was a problem with closing multiple windows. In the debugger, several windows opened under AI Shipyard. The problem was to close the windows simultaneously. It wouldn't be a problem if only one of the windows is allowed to close all the others. The problem was to try and let any window close all the other windows. So, for example, we have windows A, B, and C. We want to let A close such that B and C will close as well, B such that A and C will close, and C such that B and A will close. Therefore, a solution tried was by adding the method *close()* for each window that closes the other windows if the current window is closed. However, the problem with this solution was that closing window A, triggering *A.close()*, will close windows B and C. Then, *B.close()* will try to close window A, which is already closed, and trigger an exception. So an alternative solution was made. What was done was that only one window has the *close()* function that actually closes the windows. The other *close()* functions simply call the *close()* function of the window with the unique *close()* function.

Since there are multiple windows, some windows need to communicate with each other in real-time. One of the solutions used is through using *notify()* and *wait()*. However, the problem with this is the compilation phase. It cannot compile since the class that it waits is in a different package. This was solved by using sockets where the windows communicated through a TCP communication within the system. This, however, led to a slight drawback. Since they communicated real-time, they sent messages hundreds of times per second. This was solved by saving the previous message it sent, where if the message to be sent is the same as the previous one, then it will not resend the message.

6 Results and Observations

6.1 Method of Testing

6.1.1 Preliminary Testing

The means of testing the program's effectiveness was through asking respondents to test its usability. The test was in a form of a survey, which respondents answered upon thoroughly testing the program. It should be noted that this survey is based on the parameters specified in a study using AI Cap'n as a learning tool (Inventado, 2009).

The respondents belong to CCS students who have either tried AI Cap'n before or have not. The testers who have not tried so before were INTROAI students who apparently have no idea what AI Cap'n was or how it worked before, along with unfamiliarity to what a finite-state machine even was. The main motivation from the younger students were mostly because of the incentives in their INTROAI class, but soon became more than just that when they realize how it can assist them in their AI Cap'n related assignment.

The testing results were split into phases, each representing an iteration in improvement based on what a previous group suggested. The following lists down the parameters at which the tests were done:

1) **SET 1** (07/17): It was originally fresh from alpha testing. Suggestions acquired in this set include: debugging targeting, fixing UI problems.7 students were able to try it out. 3 veterans, 4 novices.
2) **SET 2** (07/19-07/20): Conditions now differentiate between enemy and items. Suggestions acquired in this set include: adding the FSM viewer next to the FSM Editor, adding delay to the compiler Start button. 5 students were able to try it out, all novices. There is an uncounted one here, for she did not return or do the survey.
3) **SET 3** (07/23): Visualizer appended to main window, delay also added for the compilation. Suggestions acquired in this set include: negated conditions, conditions in actions. 7 students were able to try it out. 1 veteran, 6 novices.

A total of 19 students have tested it for all three phases. They were briefed beforehand about what AI Cap'n and Finite-state machines were before letting them to the program, introducing it's basic premise of using FSM's to create AI. Approximate time for all testers is around 30 minutes. Due to only having two testing machines (the other two members had faulty machines), the respondents were allowed to share the machine with others to save time. Afterwards, all were made to answer the survey. (See Appendix C for the questions)

A set of instructional videos were created later in the testing phase in order to help brief users about Ai Cap'n, Finite-State Machines, and an AI Shipyard program run-through. This was to prevent the redundancy of explaining every testing phase. These videos can then replace the preliminary briefing before testing.

6.1.2 Beta Testing

After a week from the initial defense, another set of tests were to be given. This time, the aim was to have a larger sample for evaluation. Thus, the program was applied to an AI Cap'n competition. The competition has been done in INTROAI classes since the introduction of AI Cap'n and pitted bots against each other to see which team of programmers can create the most powerful bot. With the help of the INTROAI classes of that term, the evaluation was put forth under the pretense to help them in one of their term projects, creating a bot for AI Cap'n. The following steps were observed throughout the process:

1) **Orientation**. The four (4) classes of INTROAI (s17, s18, s19, s20) were briefed on the use, functions and relevance of AI Shipyard to their projects. The general run-through of its features were done, and several questions about the programs were addressed. All attendees were given a copy of the program.
2) **Usability Testing**. The students were to use the program to help them build their bots for a week or so before the competition on the 20th. In this period of testing the program, the proponents acted as a help desk in case any student had any problems or concerns.
3) **Bot Evaluation**. 3 days before the tournament, the students were required to pass their bots for evaluation. The bots they gave were to be screened against bots the proponents made in order to qualify for the competition. Those who succeed the test will be given a place in the competition proper. This was to prevent any unintelligent or mediocre bots from entering the competition.
4) **Program Evaluation**. All participating members were given an evaluation sheet (see Appendix C) about the program. They were tasked to complete it, for use in the findings (to be discussed below).
5) **Competition Proper**. The competition was held to have the qualified bots fight each other. The following system was used. Note that the battles were repeated thrice, for reference to the students (who were tasked to analyze them) and to determine the winners.
    a. 5 sets of 3-bot FFA (free-for-all) battles. The 1st, 2nd, and 3rd contestants will be placed in their own matches.
    b. 3rd placers would fight in one FFA match. The ranking is based on the results.
    c. 2nd placers would fight in one FFA match. The ranking is based on the results.
    d. 1st placers would have a round robin match where all the 1st placers will fight each other one-on-one. By virtue of popular request, the bot "BukoBattlecruiser" was included in the bots each would fight, but it was not a counted match.
    e. The overall winner would fight the unbeaten champion of AI Cap'n in the past two years, Rick Astley. By popular request again, "BukoBattlecruiser" also fought him.
6) **Analysis**. All data based on the evaluation sheet were then compiled and analyzed, as to be discussed in a later chapter in this document.

6.2 Alpha Testing

6.2.1 Alpha Testing Summary
Rating: 1 (Not at all) 2 (Partly) 3 (Definitely)

Table 4 AI Shipyard Survey Results: Concept

Concepts that were apparent in AI Shipyard	Set 1 rating	Set 2 rating	Set 3 rating
Search Algorithm	3	3	3
Planning	3	3	3
Heuristics	2	2	2
Algorithm Analysis	2	2	3
Knowledge Representation	3	3	2

Table 5 AI Shipyard Survey Results: Features

Features that AI Shipyard contributed	Set 1 rating	Set 2 rating	Set 3 rating
Was it fun?	2	3	3
Were you able to apply concepts in class?	3	2	2
Is it challenging?	2	3	3
Able to understand AI concepts	3	3	3
Does it make you feel competitive?	3	2	2
Do you find it useful for future work?	2	3	3
Are you appreciating AI's significance?	3	3	3
Is it simple to use?	2	2	2
Does it help visualize AI concepts?	3	3	3

Table 6 AI Shipyard Survey Results: Impressions

Question	Set 1 answers	Set 2 answers	Set 3 answers
Did it help you grasp concepts in Artificial Intelligence better? Which ones?	Yes. Search algorithm, finite state machine	Yes. Logic, finite state machine, search algorithm	Yes. Logic, state transition, search algorithm, analysis
Would you recommend it to future INTROAI students? Why?	Yes. Easier to use, learn AI concepts while enjoying	Yes. Makes creating bots easier, simple, fun, and easy to use	Yes. It is an introductory program like "Alice" for future students to get a grasp of AI concepts easier.
What do you think was missing in the program?	Reduce panel, bug fix	Interface issues, diversity of possible conditions and movements	More obstacles in the map for more complex logic, visualization of search algorithm, better

			interface

6.2.2 Alpha Testing Analysis

The first preliminary survey conducted is separated into three sets based on when the respondents visited. Then, using the results from the survey (forms are attached at the appendix), the answers were classified by sets. The first two tables above simply rate AI Shipyard based on various aspects. As indicated, the trend of the ratings is slightly increasing. This is because of the improvements added based on the results of the preceding surveys. The last table consists of more comprehensive questions to know how the respondents find our software. For the first question, it is asked if respondents discovered new or rediscovered old concepts about artificial intelligence through the software. Next, it is asked if the software affected their learning or understanding of artificial intelligence. Next if it is asked if they would recommend future students to use AI Shipyard. The last question is to help further improve AI Shipyard.

Next, it is imperative to compare against the findings in the paper of AI Cap'n (Inventado, 2009). To note, those findings apparently come from unstructured essay questions compared to the structured nature of this software's survey. Instead, these comparisons will just be a basis on what concepts or properties are apparent/evident in AI Shipyard more so than just AI Cap'n. These are not competitive comparisons, but refer to improvements/strengths which AI Shipyard has in lieu of AI Cap'n. The table below shows the comparison . In this case, the "**definitely"** results of the final testing phase are used as the parameter. The others are not included as they represented old versions of the AI Shipyard, moderately different from its current iteration.

It is apparent in the Concepts Learned Table that in comparison to AI Cap'n, the strength of AI Shipyard lies mostly on Planning and Algorithm analysis. This, in the case of system objectives, is what AI Shipyard is supposed to be teaching after all. The FSM's basic and simple structure is ample for Planning and Algorithm Analysis, to which these results can be attributed to it being the preferred format of the system. The Features table shows that AI Shipyard does achieve its objective of helping the user visualize AI concepts in the form of finite state machines, being that it's the most agreed upon by all users.

Table 7 AI Cap'n Compared to AI Shipyard

Concepts learned	**AI Capn**	**AI Shipyard**
Search Algorithms	69.29%	57.14%
Planning	42.10%	85.71%
Heuristics	22.80%	42.86%
Algorithm Analysis	13.15%	85.71%
Knowledge Representation	3.50%	42.86%
Features	**AI Capn**	**AI Shipyard**
Fun Factor	35.96%	71.43%
Application of Concepts Learned	22.80%	42.86%
Challenging	8.77%	71.43%
Understanding AI Concepts	8.77%	71.43%
Competitiveness	6.14%	42.86%
Usefulness for future work	4.38%	71.43%
Appreciating significance of AI	4.38%	42.86%
Simplicity	2.63%	42.86%
Visualizing AI Concepts	2.63%	85.71%

6.3 Beta Testing

6.3.1 Beta Testing Survey Results

S17

Answers: 1 (Not at all) 2(Partly) 3(Definitely)

User Experience	
> FSMs help me visualize the bot's behavior	2.4
> FSMs are a simple way to show agent behavior.	2.4
> Is the software simple to use?	2.1
> Does it make you feel competitive?	2.3
> Is it fun creating agents for the game?	2.3
> Does it make you more interested in learning agent-based decision making concepts?	2.4
> It was easy to create bots	1.9
> The debugger helped me formulate the bot's design effectively	2.1

Please do rate the following functions of the program from 1-5 (5 being the highest) based on helpfulness, attractiveness, simplicity, along with other parameters you feel are relevant.

Functionality Testing	
FSM Editor (Battle Mode, Battle Plan, Strategies)	3.6
FSM Viewer	3.6
Actions/Conditions Editor	3.5
Debugger	2.8
Help Files	3.3

S18

Answers: 1 (Not at all) 2(Partly) 3(Definitely)

User Experience	
> FSMs help me visualize the bot's behavior	2.4
> FSMs are a simple way to show agent behavior.	2.3
> Is the software simple to use?	2
> Does it make you feel competitive?	2.2
> Is it fun creating agents for the game?	2.4
> Does it make you more interested in learning agent-based decision making concepts?	2.5
> It was easy to create bots	2.1
> The debugger helped me formulate the bot's design effectively	1.7

Please do rate the following functions of the program from 1-5 (5 being the highest) based on helpfulness, attractiveness, simplicity, along with

S19

Answers: 1 (Not at all) 2(Partly) 3(Definitely)

User Experience	
> FSMs help me visualize the bot's behavior	2.5
> FSMs are a simple way to show agent behavior.	2.2
> Is the software simple to use?	2.1
> Does it make you feel competitive?	2.3
> Is it fun creating agents for the game?	1.9
> Does it make you more interested in learning agent-based decision making concepts?	2.2
> It was easy to create bots	1.8
> The debugger helped me formulate the bot's design effectively	1.7

Please do rate the following functions of the program from 1-5 (5 being the highest) based on helpfulness, attractiveness, simplicity, along with other parameters you feel are relevant.

Functionality Testing	
FSM Editor (Battle Mode, Battle Plan, Strategies)	3.1
FSM Viewer	3.5
Actions/Conditions Editor	3.2
Debugger	2.7
Help Files	3.2

S20

Answers: 1 (Not at all) 2(Partly) 3(Definitely)

User Experience	
> FSMs help me visualize the bot's behavior	2.5
> FSMs are a simple way to show agent behavior.	2.2
> Is the software simple to use?	2.1
> Does it make you feel competitive?	2.3
> Is it fun creating agents for the game?	1.9
> Does it make you more interested in learning agent-based decision making concepts?	2.2
> It was easy to create bots	1.8
> The debugger helped me formulate the bot's design effectively	1.7

Please do rate the following functions of the program from 1-5 (5 being the highest) based on helpfulness, attractiveness, simplicity, along with

other parameters you feel are relevant.

Functionality Testing	
FSM Editor (Battle Mode, Battle Plan, Strategies)	3.5
FSM Viewer	3.7
Actions/Conditions Editor	3.2
Debugger	2.6
Help Files	3.7

other parameters you feel are relevant.

Functionality Testing	
FSM Editor (Battle Mode, Battle Plan, Strategies)	3.8
FSM Viewer	3.7
Actions/Conditions Editor	3.4
Debugger	3.2
Help Files	4.1

Any Comments/ Recommendations for the program?

- To fix some of the problems with the items and the resources the program takes up.
- Optimization for better performance during debug. Consumes too much memory.
- Warning system for bots with erroneous code
- Still best to be used just to create a framework for the bot, then add personalized code manually.
- Renaming actions and conditions to give a more specific description of what they do

Would you recommend it to future INTROAI students? Why?

- Yes, because it makes the development of bots much easier.

Which AI concept or concepts did AI Shipyard help you understand better? Please explain.

- Game search
- FSM
- Strategizing

6.3.2 Beta Survey Analysis

After conducting the survey for INTROAI students, the researchers have come to several conclusions based on the answers provided by the respondents.

1. AI Shipyard is not yet efficient after prolonged use . This is probably due to a problem in clearing memory that is no longer needed after exiting the debug phase.
2. The students expect error reports for tracing. This presents a problem since most of the errors are logical errors which even well-used programming languages are unable to show to a programmer in error reports. It is recommended that there be more specific errors for each error case.
3. Some of the names assigned to actions and conditions are vague.
4. AI Shipyard is generally accepted by the respondents as evident with the ratings they gave. The ratings are above average. For the first set of questions, the average of the ratings is around “2” with the highest being “3”. The same goes for the second set, where the average is above “3” with the highest being “5”.

7 Conclusions and Recommendations

Although AI Shipyard focuses on aiding students implement bots for AI Cap'n, which is under the course INTROAI, the researchers found out that it can also be used for other courses in CCS. It is apparent in the results that the students learned and understood the basic functions of finite state machines through AI Shipyard.

Based on the survey, respondents from each set answered finite state machine or state transition in the *concepts learned* question in the last table. Hence, AI Shipyard can not only aid the course INTROAI but also THEOCOM (Automata Theory, Formal Languages, and Computation complexity), which introduces finite state machines. In relation to this, respondents from all sets also said, based on the "would you recommend AI Shipyard" question, that the students found it easy to implement and visualize AI concepts through AI Shipyard. Students call it "user friendly", "makes it easy to grasp concepts", and "that it helps visualize AI". Therefore, it can be established that finite state machine is a simple and easy-to-use model in representing artificial intelligence.

The Beta Test period also revealed a more realistic light on the student opinion on more improvements to the software itself. The testers were indeed expecting the perfect program for use in their needs. "It is good enough for now, but still has a lot of room to improve" Seeing that they all responded "yes" when asked if they would recommend AI Shipyard means that it is usable enough. However, some things that they noted can still be fixed to further improve AI Shipyard. One in particular is the vague error messages, which is recommended to be more specific in terms of the error's origin. Tracing of the FSM will be needed before compilation.

For future implementations of this program, there are many ways to extend this study. Introducing the concept of chance to the strategies can provide stochastic results, making the bot intelligence less linear. Some students also recommend on hoping for being able to add conditions inside actions along with the implementation of loops through the Easy Add function, so that non-programmers can do it as well. Furthermore, in order to extend this program above just decision making and logic, there could also be more customizable search algorithms available to the user which they can edit themselves. Further improvements could possibly include a better user interface to fit all needed windows for debugging and perhaps other ways to make the designing process more intuitive.

Appendix A: Tables

AI Shipyard

Table 8 AI Shipyard Default Conditions

Condition	Description
HP_LESS_THAN _50	If your ship's HP is less than 50%...
HP_MORE_THAN_50	If your ship's HP is more than 50%...
ENEMY_EXISTS	If there are still enemies on the map… (all of them)
HEALTH_ITEM_EXISTS	If there is a health item on the field…

Table 9 AI Shipyard Default Actions

Action	Description
Attack	Runs after the enemy and shoots at them **
Run	Runs away from any nearby enemy**
Get_Nearest_Item	Gets the specified item *

Table 10 AI Shipyard Default Base Actions

Base Action	Description
Ship Moves Up	Move ship up by one step
Ship Moves Down	Move ship down by one step
Ship Moves Left	Move ship left by one step
Ship Moves Right	Move ship right by one step
Ship Faces Up	Makes ship face upward
Ship Faces Down	Makes ship face downward
Ship Faces Left	Makes ship face leftward
Ship Faces Right	Makes ship face rightward
Ship Fires	Ship fires its cannons (fires from its side)
Ship Engages Nearest Enemy	Ship chases a target and fires at an enemy**
Ship Grabs Nearest Item	Ship moves to the item*
Ship Uses Held Item	Ship uses the item it is holding
Ship Follows Nearest Enemy	Ship chases a target**
Ship Faces Parallel To Enemy	Ship faces its enemy with its guns ready to fire**
Ship Flees	Ship stays away from enemy**

* item condition dependent
** enemy condition dependent

Quake 3 Arena

Table 11 Quake III Bot Actions

Bot Action	Description
Attack	This action equals to a player holding down the fire button. The bot will fire the weapon the bot currently holds.
Use	The bot will use the currently held holdable item.
Respawn	This action causes the bot to respawn when the bot is dead. Human players use the fire button for this purpose.

Jump	This action makes the bot jump up
Crouch	This will make the bot crouch. The bot will crouch while this basic action is used. Just like a human player would hold down the crouch button.
Move Up	This action makes the bot move up when swimming. This equals to holding down the jump button for human players.
Move Down	This action makes the bot move down when swimming. This equals to holding down the crouch button for human players.
Move Forward	When the bot walks this action will make the bot move in the horizontal view direction. However when under water this action makes the bot swim in the 3D direction the bot is viewing.
Move Back	When the bot walks this action will make the bot move in the opposite of the horizontal view direction. However when under water this action makes the bot swim backwards relative to the 3D direction the bot is viewing.
Move Left	This action makes the bot move sideward to the left.
Move Right	This action makes the bot move sideward to the right.
Walk	While the bot uses this action the bot will walk instead of run.
Talk	While the bot uses this action a chat icon will appear above the bot's head and the bot also will not be able to move.
Gesture	When this action is used, the visible in game model the bot uses will show a special animation of a gesture.
View	This action will set the bot's view angles. This action uses the desired view angles as a parameter.
Move	This action is always used with two parameters: the direction of movement and the speed. The action makes the bot move in a certain direction with the specified speed. Note that unlike the common human interface the movement direction for this basic action is independent from the view angles.
Select Weapon	This action sets the bot's weapon to be used. The weapon is selected with a number, which is the parameter to this action.
Command	Several commands can be typed on the console in the game. The bot can use these commands with this action, which has the command string as a parameter.
Say	With this action the bot can say someone to all the other players. The message to be displayed is the parameter to this action.
Say Team	This action is similar to the Say action however now the message will only be visible to the bot's team mates.

Final Fantasy XIII Gambits
As mentioned in (Pringle, 2009)

Table 12 Final Fantasy XII Ally Gambits

Ally Targets	**Description**
Ally: Any	Targets any ally, if it requires the action
Ally: HP/MP less than X%	Targets an ally of a certain HP/MP value.
Ally: Status=Status	Targets an ally in a certain status.
Ally: <Name>	Targets an ally of the same name.
Ally: lowest <parameter>	Targets an ally of a certain parameter. Strongest Weapon / Lowest Defense / Lowest Magick Resist / etc
Ally: Item > X	Uses an item on an ally until the equal to X items is no longer greater than X value

Table 13 Final Fantasy XII Self Gambits

Self Targets	**Description**
Self	Targets self with action if required
Self: HP/MP < X%	Targets self at a certain HP/MP value.
Self: Status=Status	Targets self while in a certain status

Table 14 Final Fantasy XII Foe Gambits

Foe Targets	**Description**
Foe: Party Leader's Target	Allies will target party leader's (your) target
Foe: Nearest Visible	Allies will target the nearest enemy
Foe: Any	Allies will target any enemy
Foe: Targeting leader	Allies will target any enemy targeting you self / ally
Foe: nearest	Allies will target enemies based on distance
Foe: highest/lowest <parameter>	Allies will target enemies based on HP/MP, HP, MP, Max HP, Max MP, level, strength, magick power, speed
Foe: HP <modifier> X	Allies will target enemies based on HP value
Foe: Status=status	Allies will target an enemy in this status
Foe: X-weak	Allies will target an enemy in weak to an element or is in a state (X-vulnerable, undead)
Foe: Flying	Allies will target flying enemies
Foe: character HP=100%,	Allies will target enemies based on a specific state (Item greater than 10, HP/MP less/greater than X%, Status=status, HP Critical)

Appendix B: Screenshots

Old Versions of AI Shipyard

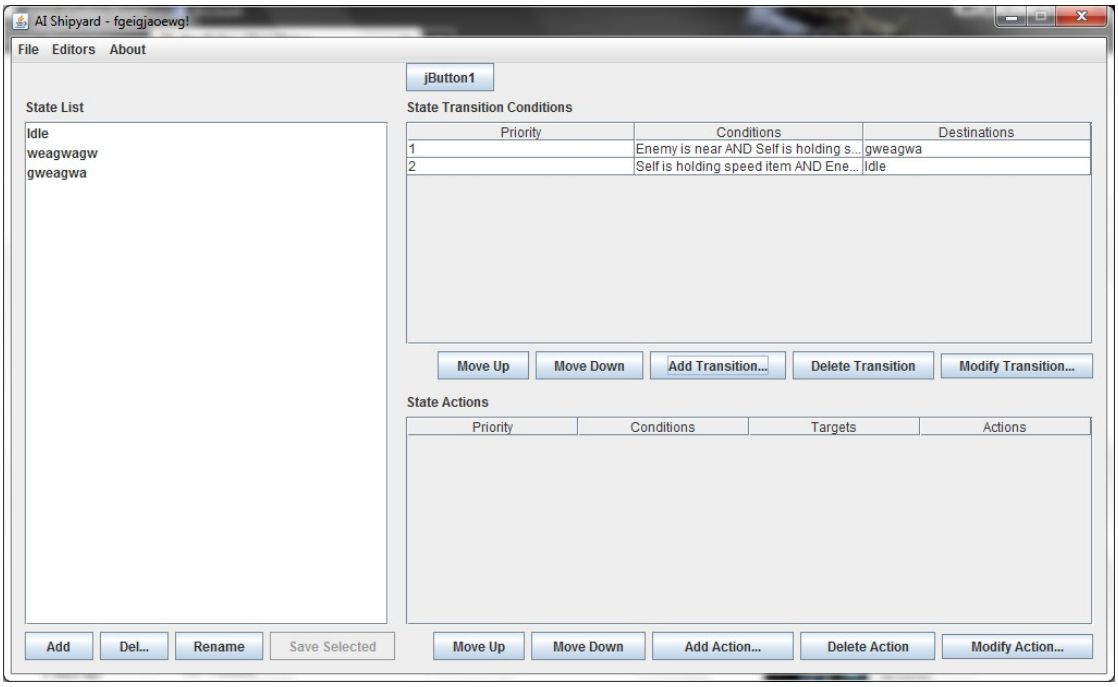

Figure 15 Old AI Shipyard Main Window

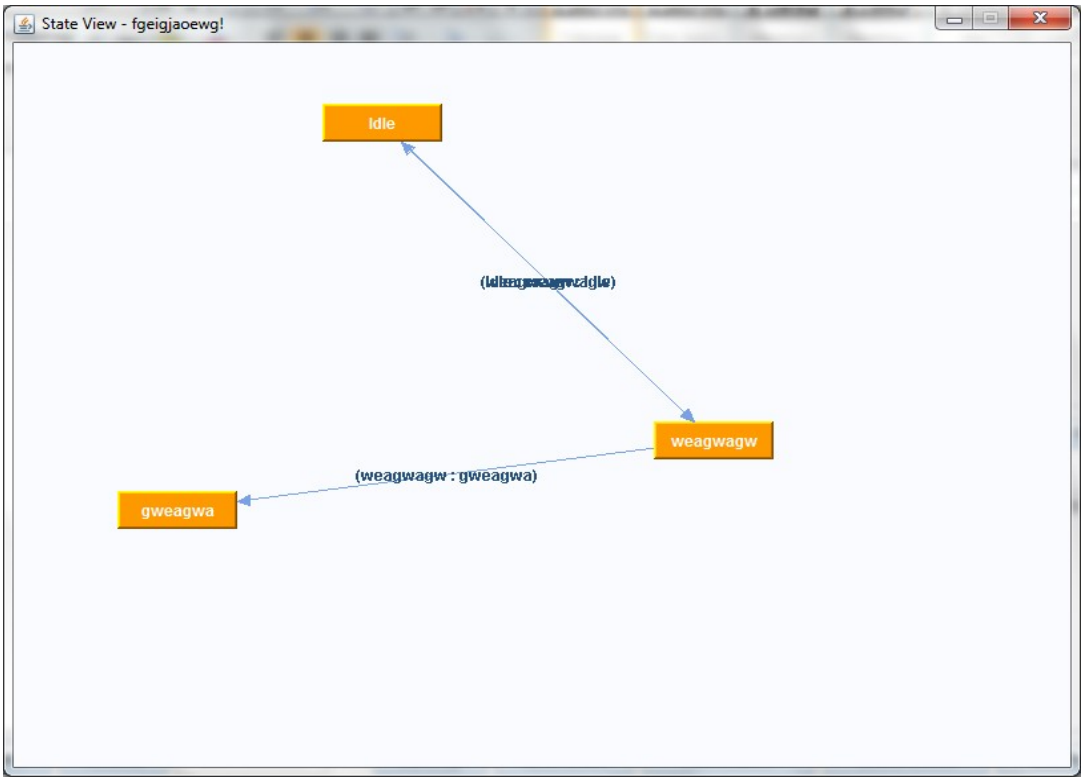

Figure 16 Old AI Shipyard Graph Viewer

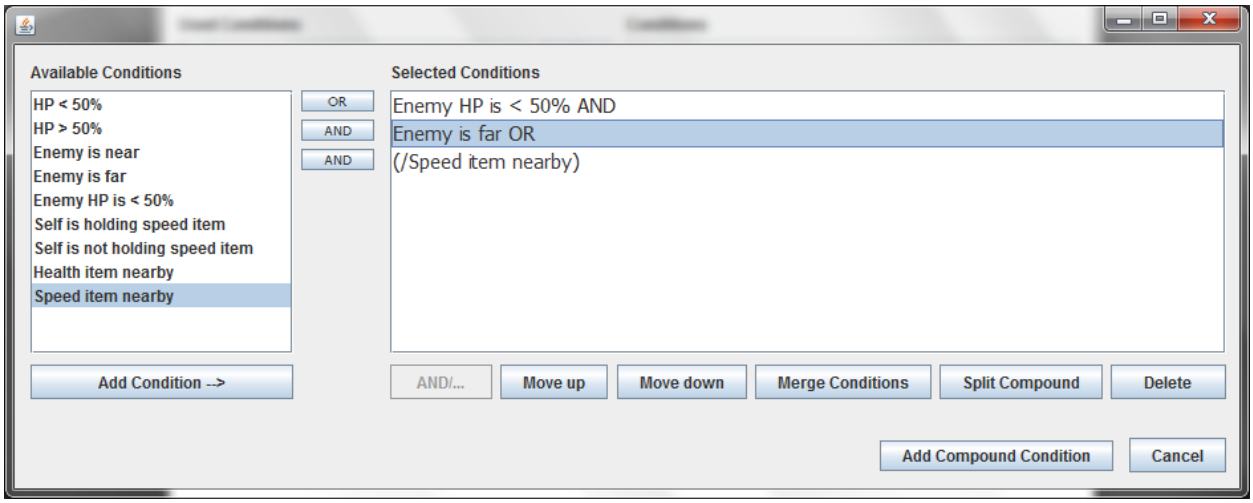

Figure 17 Old AI Shipyard Compound Condition

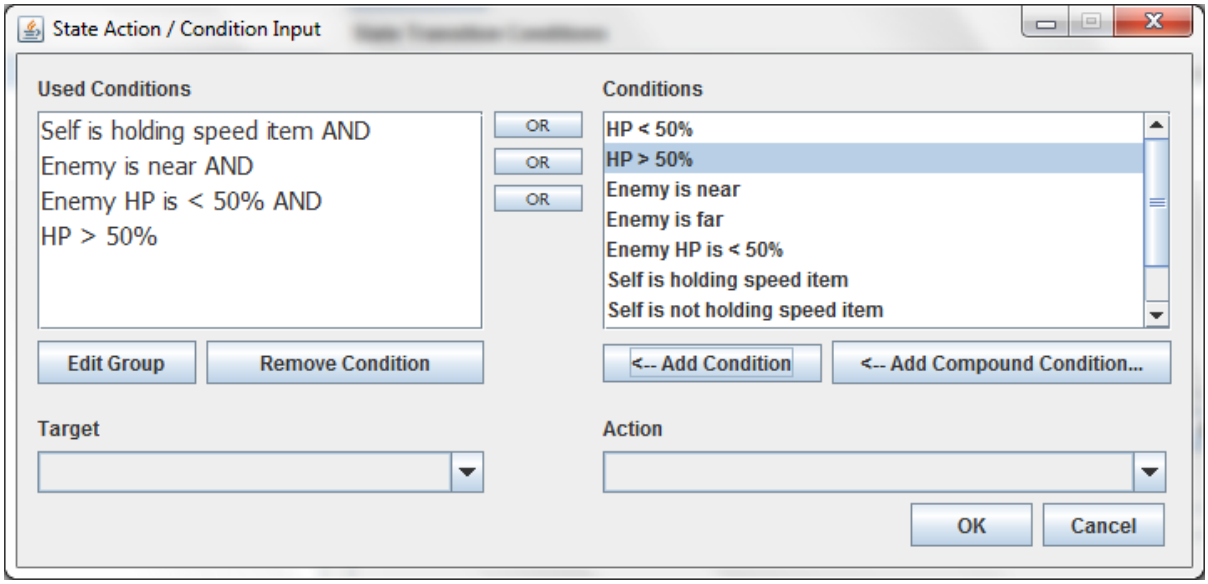

Figure 18 Old AI Shipyard Action Adding

Current Version of AI Shipyard

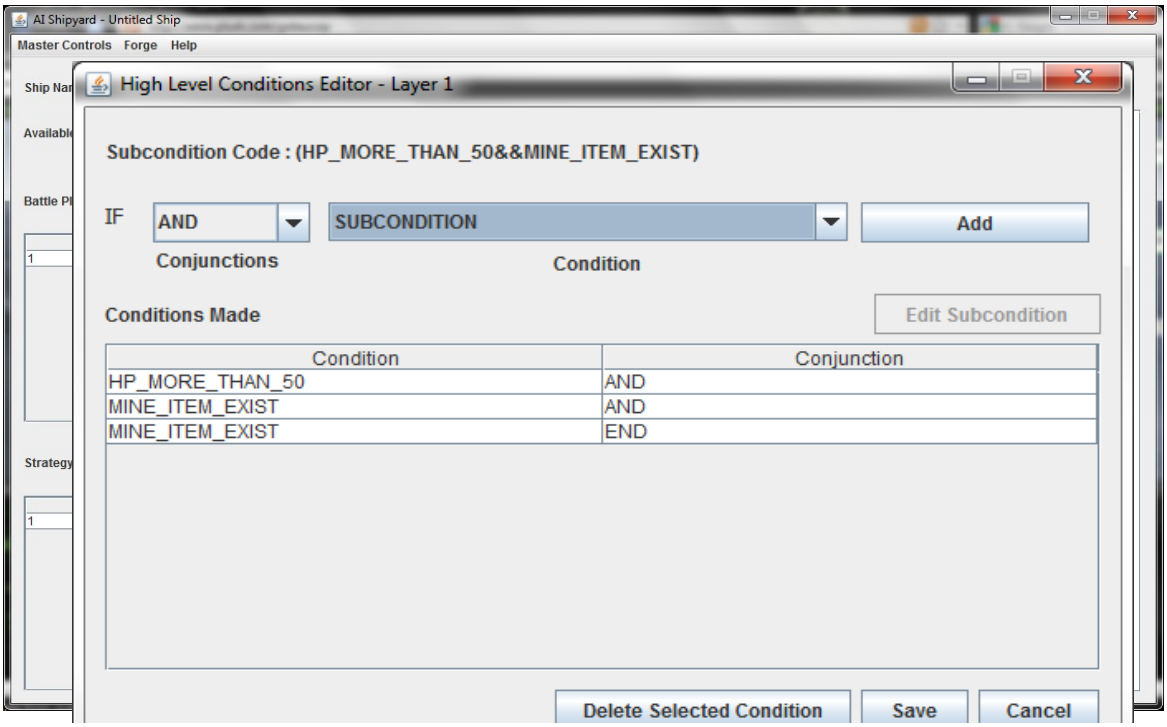

Figure 19 Main Window With Built-in Graph Viewer

Figure 20 Conditions Editor with Subconditions

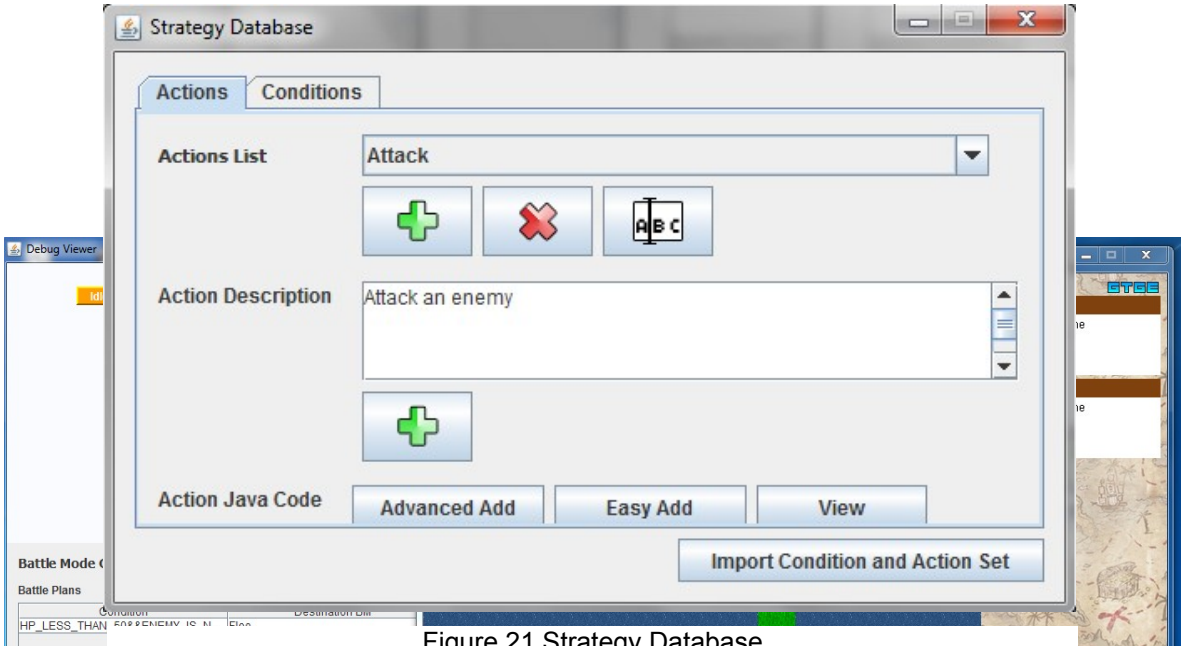

Figure 21 Strategy Database

Figure 22 Overview of Debugger

References

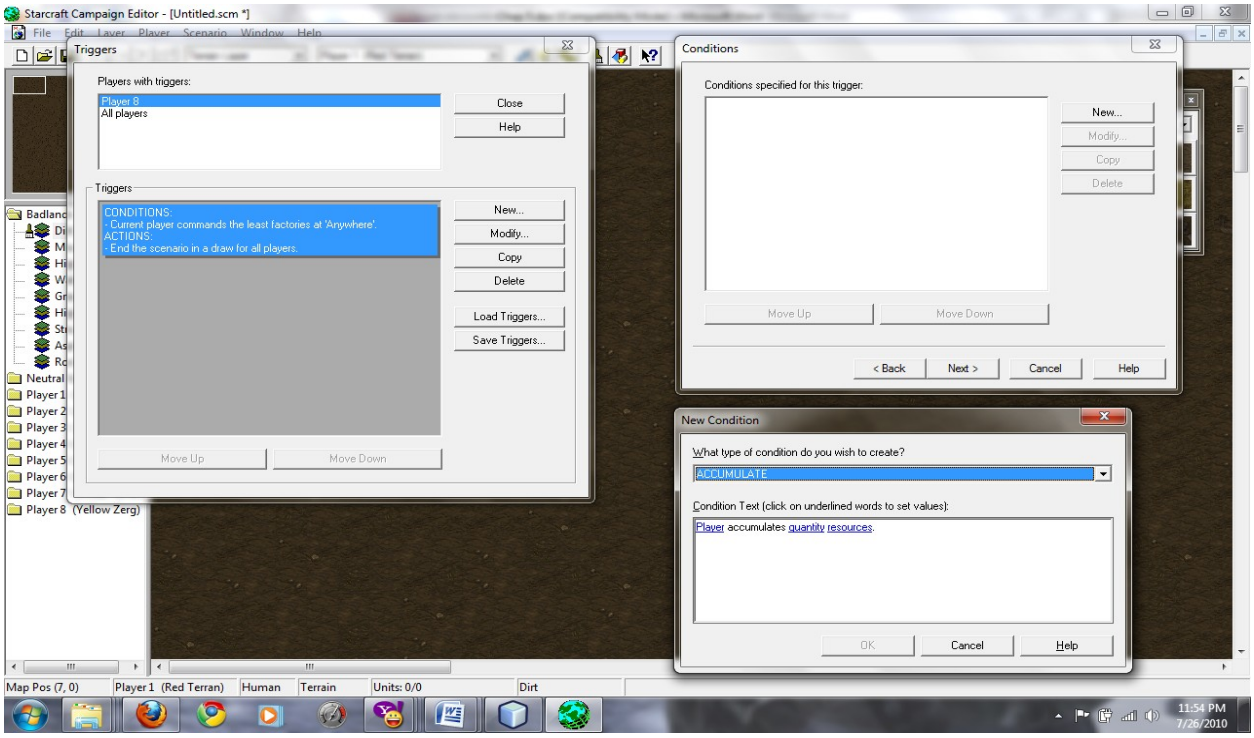

Figure 23 StarEdit UI Design

Appendix C: Survey Questions and Results

AI Shipyard Survey Form

Disclaimer: This form is for survey purposes only and no information here will be disseminated to non-concerned parties and be will be kept confidential.

Name:_____

Email:_____

Age: ____ Gender: _____

Have you used AI Cap'n before? _____

Do you belong to one of Dr. Sison's classes? _____

If yes, what subject and section? _____

Answers: 1 (Not at all) 2(Partly) 3(Definitely)

The concepts were apparent by AI Shipyard

Concept	**1**	**2**	**3**
Search Algorithm			
Planning			
Heuristics			
Algorithm Analysis			
Knowledge Representation			

General Comments: _____

_____

Which features does AI Shipyard contribute?

Features	**1**	**2**	**3**
Was it fun?			
Were you able to apply concepts in class?			
Is it challenging?			
Able to understand AI Concepts?			
Does It make you feel competitive?			
Find it useful for future work?			
Are you appreciating AI's significance?			
Is it simple to use?			
Does it help visualize AI Concepts?			

General Comments: _____

_____

Did it help you grasp concepts in Artificial Intelligence better? Which ones?

_____

_____

_____

Would you recommend it to future INTROAI students? Why?

Attached after this page are photocopies of the filled up survey forms.

What do you think was missing in the program?

_____

_____

_____

Please return this form to LIC (G411) or to whoever gave you this.

Thank you for taking the survey. ☺

Updated Survey Form

AI Shipyard Survey Form

Disclaimer: This form is for survey purposes only and no information here will be disseminated to non-concerned parties and be will be kept confidential.

Name:_____

Email:_____

Age: ____ Gender: _____

Have you used AI Cap'n before? _____

Do you belong to one of Dr. Sison's classes? _____

If yes, what subject and section? _____

* send a copy of the email to Dr. Sison of your filled-out questionnaire at raymund.sison@delasalle.ph. The subject of the e-mail should be: INTROAI - Shipyard - <section> - <last name> <first name>. Note the space before and after each dash. For example: INTROAI - Shipyard - S17 - Sison Raymund.

Answers: 1 (Not at all) 2(Partly) 3(Definitely)

User Experience	**1**	**2**	**3**
> FSMs help me visualize the bot's behavior			
> FSMs are a simple way to show agent behavior.			
> Is the software simple to use?			
> Does it make you feel competitive?			
> Is it fun creating agents for the game?			
> Does it make you more interested in learning agent-based decision making concepts?			
> It was easy to create bots			
> The debugger helped me formulate the bot's design effectively			

Please do rate the following functions of the program from 1-5 (5 being the highest) based on helpfulness, attractiveness, simplicity, along with other parameters you feel are relevant.

Functionality Testing	**1**	**2**	**3**	**4**	**5**
FSM Editor (Battle Mode, Battle Plan, Strategies)					
FSM Viewer					
Actions/Conditions Editor					
Debugger					
Help Files					
Others? _____					

Any Comments/ Recommendations for the program?

_____

_____

_____

_____

_____

Would you recommend it to future INTROAI students? Why?

_____

_____

_____

Attached after this page are the results answers of the filled up updated survey forms.

Which AI concept or concepts did AI Shipyard help you understand better? Please explain.

_____

_____

_____

For more information on AI Cap'n, go to its website at: http://sites.google.com/a/delasalle.ph/aicapn/home

Please return this form to LIC (G411) or to whoever gave you this.

Thank you for taking the survey. ☺

Appendix D: Resource Persons

Mr. Paul Inventado
Adviser
College of Computer Studies
De La Salle University-Manila
paul.inventado@delasalle.ph

Mr. Solomon See
Faculty
College of Computer Studies
De La Salle University-Manila
solomon.see@delasalle.ph

Dr. Raymund Sison
Faculty
College of Computer Studies
De La Salle University-Manila
raymund.sison@delasalle.ph

Appendix E: Personal Vitae

Karl Stephan G. Benavidez
43 Yakal St. Greenwoods Exec. Vill., Cainta, Rizal
09064207353
vikifanatic@yahoo.com / karlbenavidez@gmail.com

Arturo P. Caronongan
122 Don Rufino Ave., Tahanan Village, Parañaque
09164085565
r2roboy@yahoo.com

Kelvin C. Chua
2313 Taft Ave. Corner Castro Street, Malate, Manila
09228558658
klvnchua@yahoo.com

Patrick C. Gotauco
Blk42 Lt22 Munic St. Town and Country Southville Subd, Biñan, Laguna
09053156545
gotauco_p@yahoo.com

References

Ahn, B. S., & Choi, S. H. (2009, November). Con ict resolution in a knowledge based system using multiple attribute decision-making. Expert Systems with Applications .

Brownlee, J. (2003). Finite state machines. Retrieved October 30, 2009, from http://ai-depot.com/FiniteStateMachines/

Buckland, M. (2005). Programming game ai by example. Jones and Bartlett.

Conway, M., Audia, S., Burnette, T., Cosgrove, D., Christiansen, K., Deline, R., et al. (2000). Alice: Lessons learned from building a 3d system for novices. CHI .

Cooper, S., Dann, W., & Paush, R. (2003). Teaching objects-first in introductory computer science. In 34th sigcse technical symposium on computer science education (p. 191-195). Reno, Navada: ACM.

Gladyshev, P. (2005). Finite state machine analysis of a blackmail investigation. International Journal of Digital Evidence.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. Science of Computer Programming , 8 (3).

Hartmann, W., Nievergelt, J., & Reichert, R. (2001). Kara finite state machines and the case for programming as part of general education. In Ieee symposia on human centric computing languages and environments (p. 135-141). Stresa, Italy.

Inventado, P., & See, S. (2009). Ai cap'n: A game platform for learning artificial intelligence.

LaMothe, A. (1995). Building brains into your games. Retrieved July 26, 2009, from http://www.gamedev.net/reference/articles/article574.asp

Lu, T.-C., & Druzdzel, M. J. (2009, December). Interactive construction of graphical decision models based on causal mechanisms. European Journal of Operational Research , 199 , 873-882.

Mo at, J., & Medhurst, J. (2009, August). Modeling of human decision-making in simulation models of con ict using experimental gaming. European Journal of Operational Research , 1147-1157.

Moskal, B., Lurie, D., & Cooper, S. (2000). Evaluating the effectiveness of a new instructional approach. Retrieved July 14, 2009, from http://www.alice.org/index.php?page=publications/publications

Pringle, B. (2009). Final fantasy xii gambit system. Retrieved October 27, 2009, from http://billpringle.com/games/ffxii gambits.html

Stout, B. (1997). Smart moves: Intelligent pathfinding. Retrieved July 26, 2009, from http://www.gamasutra.com/features/19970801/pathfinding.htm

Waveren, J. P. van. (2001). The quake iii arena bot. Unpublished master's thesis, Delft University of Technology.

Woodcock, S. (2000). Game ai: The state of the industry. Retrieved July 26, 2009, from http://www.gamasutra.com/features/20001101/woodcock01.htm