# ROLE PLAYING GAMES AS COOPERATIVE CAPSTONE DESIGN PROJECTS

By

STEVEN WELCH

Bachelor of Science in Electrical Engineering

Oklahoma State University

Stillwater, Oklahoma

2008

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2012

ROLE PLAYING GAMES AS COOPERATIVE CAPSTONE

DESIGN PROJECTS

Thesis Approved:

Dr. Alan Cheville

Thesis Adviser

Dr. Damon Chandler

Dr. Sohum Sohoni

Dr. Sheryl A. Tucker

Dean of the Graduate College

# PREFACE

This thesis diverges from the normal structure of a research paper. Typical thesis structures are designed for hypothesis or question-driven research projects. The thesis written in this style usually begins with an introduction providing an overview of the hypothesis being tested, an explanation of the problem, and background research. The experimental results are then presented and discussed, the results compared to relevant theory, and finally the work is summarized and the conclusions presented. This form of thesis presentation reflects the structure and conduct of question-driven research.

The work presented in this thesis is organized around an engineering system perspective to better align with the goals and results of this project. In this perspective, the function of the system is described and a comparison is made with other solutions to highlight potential advantages and disadvantages. Such advantages are context and constraint dependent. Next the system is described in terms of its functional components in increasing levels of detail starting from a functional overview and ending on the schematic level. The thesis concludes with a discussion of further research to be conducted.

Using a system design lens to examine the work done in this thesis has led to the following organizational structure: Chapter one describes the perceived problem the system addresses, why the problem is important, and design parameters of the system that are required to address these issues. Chapter two is a discussion of the evolution of the system. Chapters three through five discuss the technical implementation details of each sub-system. Finally chapter six summarizes the work completed and provides ideas for further development.

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

# CHAPTER I

## INTRODUCTION

In the process of updating and maintaining an electrical engineering capstone course, our group encountered several difficult challenges. One of the biggest challenges was developing new projects every semester that were well supported by our teaching staff and that students' found consistently interesting. Another challenge was that students were focusing too much on their individual part of the project at the expense of understanding the system they were designing. Since completely new projects were created every semester it was impossible to implement projects with large amounts system integration. The cost, both in terms of supplies and time, for developing completely new projects and supporting technologies while maintaining student interest levels was unsustainable. Also, only a limited number of sub-disciplines with electrical engineering could be incorporated into any given project. To solve these problems, we began developing our own capstone design project, which evolved into an ecosystem [1]. This paper reports on a unique capstone project that addresses these issues and is designed to help students learn design more effectively while remaining interesting and fun.

Creating a capstone design project that meets these criteria as well as the necessary support structure for the course is at heart a technically challenging research project and, many technical challenges had to be overcome. For example, one challenge we faced was how do we get students to be engaged and motivated to work on engineering problems. The solution we came up with was to design the project around games. The reasoning behind using games will be discussed later. After much discussion and debate, it was decided that the project would implement a game that is mobile and in which students could physically engage other students in real world

environments. This decision led to another set of design challenges. Given that within a game there are lots of different interaction methods between players, we needed to design a platform that could accept multiple input types. We also wanted to incorporate as many cutting edge technologies as possible so that students would get a fresh view of what was actually going on in engineering today. The need for modularity quickly became prevalent as we brainstormed on how to keep the project fresh and continue to support new technologies as they became available. We quickly realized that we would need a common data structure since we would be designing games in a real world environment with computers and many different communication methods.

The data structure would need to be short enough to accommodate all our communication methods and be relatively robust, but complex enough to support the functions of a game. We spent weeks coming up with an initial data packet structure. Considering what communication methods we would be using, what kind of interfaces the hardware would have, and what data we would need while continuing to work on the course. In addition, we wanted to follow the modular design concepts that we had with our hardware so the packet structure would need to accommodate many different types of software scenarios, not just games. In the end we developed a simple yet structured data packet that could be many different communication methods in an abundance of ways.

These challenges give a flavor of the work that is to be presented. Considerably more detail and explanation will be provided throughout the rest of the document. While developing this project we literally pushed the limits of technology. Some of the technologies used in the final product did not even exist when the journey to develop a new capstone design project began.

The rest of this chapter will discuss the guiding principles behind the MAGE System, give a big picture sense of what was accomplished, and show how the MAGE system takes a more systematic and systems approach to design.

## A. GAMES IN TEACHING CAPSTONE DESIGN

There has been some recent interest in using games for learning due to the fact that games are ubiquitous in the lives of young people [2]. Research has shown games cross ethnic, gender, and cultural boundaries the way few other cultural icons do[2]. This study showed how ubiquitous games are to our target demographic—98% of teenage boys and 94% of girls play electronic games—and shattered stereotypes about games as being restricted to a "solitary, nerd" subculture [2]. Since students from all backgrounds understand games, they have a large amount of experience to draw from when creating games. Therefore, it seemed logical to conclude games would form a good basis for capstone projects.

Role-Playing Games (RPGs) are a very common class of game in which a player takes control of a fictional character in a constructed world. The original RPGs were pen and paper games such as *Dungeons & Dragons*®[3]. Live Action Role-Playing Games (LARPs) are an extension of standard RPGs where players physically perform their characters actions. These two forms of RPGs usually have a game master (GM) that implements the rules and creates the fictional world. Other RPGs like *World of Warcraft*® are computer based and graphics intensive and allow players to team up with people around the globe. In massively multiplayer online role-playing games (MMORPG) players use computers to control fictional characters in an artificially created computer environment and interact through the computer with many hundreds or thousands of other players by means of the Internet [4, 5].

The popularity of role-playing games is incredible. Over twenty million people have played *Dungeons & Dragons*®, the most famous pencil-and-paper RPG [6]. *World of Warcraft*®, arguably the most popular MMORPG, had a population of 6.3 million active residents in July, 2008 [7]. The population grew to 12 million active players by October of 2010 [8]. That population ranks it as the 6$^{th}$ largest city in the world, with a larger population that in any city in the United States. Put another way, in any given month *twenty five times* as many people are

playing *one* game, World of Warcraft®, as are majoring in engineering at *all* US colleges and universities combined [9].

But this is only one game. Between January 1, 2000 and March 28, 2009 PlayStation sold 1.5 billion units of its console the PS2 with Nintendo following close behind with 550 million Nintendo DS consoles and 370 million Wii consoles sold [10]. Games are truly a popular form of entertainment.

In order to integrate games into a capstone design course, the Modular Artificial Gaming Environment (MAGE) was created. MAGE is constructed of several sub systems, or modules. This type of system organization helps teach the principles of functional decomposition[11]. MAGE is artificial in the sense that, like MMORPGs, the game system creates an artificial environment (i.e. computer hardware, software, and custom designed peripherals) with the potential to add many players. Since the environment MAGE creates can be programmed the game can be easily changed from semester to semester. The system is designed around the principles and ideas that through the use of MAGE students can create new simulated environments.

Unlike pencil and paper or computer games, the MAGE system implements a real life RPG where players move in the physical world. In essence, all RPGs are simply a specific set of rules that govern the way the game is played. To create a real-life game, each player wears a set of hardware modules that allows them to interact with the other players in the game. The computer hardware and software implement the rules of the game, and these rules can be changed and customized, while students work on building additional pieces to the system. As the students' needs advanced and the course expanded, so too did the MAGE system. MAGE was created over a period of several years and its systems were developed one at a time. Our original idea of what MAGE should be changed drastically throughout the course of its development. As we studied

literature on design, gained experience in how students learn, and what it takes to sustain a design project our ideas of what makes a good design project changed as well. Eventually we developed a stable set of ideas about what the requirements and characteristics for design projects in a capstone design environment should be.

## B. CHARACTERISTICS OF GOOD DESIGN PROJECTS

Research has shown that good design projects have specific characteristics [12, 13]. When we design projects for students a large factor of what makes the project good is whether or not students can successfully complete the project. The characteristics discussed in this section have been proven through research and experience to improve the success of student design projects.

In general it is very difficult for students to work with very open ended projects [14]. If a team of students is given a problem statement like "Build an MRI" it is highly likely they will not do well. Students need context in which to place the problem in order for them to understand the problem and be able to work with it [15, 16], which games easily provide. They also need a limited set of directions in which to proceed [14]. In the case of a problem statement such as "Build an MRI" there are far too many directions for students to proceed. In order to prevent impossible projects, clearly defined constraints must be provided.

There are many reasons to incorporate cutting edge technologies into a project. It has been reported that incorporating "modern and emerging technologies with which most of the students would have some familiarity" is an important factor in the success of a project [17]. Incorporating these technologies could bring a sense of relevance to the project thus making students more engaged in the project [15, 16].

Projects should be related to the engineering discipline and, if possible, should lie within students area of specialization [12]. The complexity of the project should be strongly correlated with the need for knowledge and experience that is not directly covered within the students course work

[13]. Thus, the farther outside the student's area of experience you go, the higher the likelihood the student will perform badly on the project. This presents a huge problem. How do you make complex projects that remain within the capabilities of the students?

The ability of a team to perform multiple iterations of a design project is an extremely important factor in the success of design projects. Projects that do not allow multiple iterations greatly reduced students' chances of success [13]. Given the time frame of a typical design class (16 weeks), the complexity of independent projects is limited.

One of the most challenging problems faced by our team was the classification of a project and the completeness of the project. It was often very difficult to judge a project that was classified as "research" or a "prototype" fairly since everyone's opinion of what a prototype should be is different. We had no set of standards with which to classify the state of a projects readiness. We also had no way to track and compare the successfulness of projects classified as "research" to projects classified as "prototypes" or "finished product". To fulfill this need we used the TRL (Technology Readiness Level) scale as a framework [18]. The scale is tiered to determine the complexity level of a project as illustrated here:

- **TRL 1** Basic principles observed and reported [18]
- **TRL 2** Technology concept and/or application formulated [18]
- **TRL 3** Analytical and experimental critical function and/or characteristic proof-of-concept [18]
- **TRL 4** Component and/or breadboard validation in laboratory environment [18]
- **TRL 5** Component and/or breadboard validation in relevant environment [18]
- **TRL 6** System/subsystem model or prototype demonstration in a relevant environment [18]
- **TRL 7** System prototype demonstration in a relevant environment [18]
- **TRL 8** Actual system completed and "flight qualified" through test and demonstration [18]

- **TRL 9** Actual system "flight proven" through successful mission operations [18]

NASA developed the TRL scheme for the assessment of the maturity of technologies relevant to space flight. For this reason, TRL levels 8 and 9 do not typically apply to educational environments, but represent a polished final product. TRL levels 5 through 7 focus on the testing of a design in a relevant environment, specifically not a research lab, and represent a well-polished prototype. TRL levels 1-4 represent the qualities of a research project, where the final outcome may only be knowledge not an actual product. Research has shown that projects on the outer edges of the TRL scale tend to be less successful than those that fall within the 5-7 range [13]. By designing projects with outcomes that will fall within a TRL of 5-7 we can help make it easier for students to succeed.

After four years of developing the MAGE system and using it in a capstone course we have discovered several features that support deep and meaningful learning of design. Some of these features support teaching design and some are specifically for supporting design classes.

When teaching design, many factors can contribute to the success or failure of any design project. Constraints are a critical aspect of design, yet it can be difficult to create projects with real rather than artificial constraints. Here we discuss two classifications of constraints, System Constraints and Project Constraints.

System constraints are those that are imposed on the program, instructor, or course. The primary constraints on a design course/project are as follows:

- **Affordability** - Any Capstone Design project must have a low startup cost so that it can be affordable to universities.
- **Sustainability** - Meaning the recurring costs of supporting a team must be low.
- **Support** - In order to accommodate the limited availability of faculty, staff, and teaching assistants, a curriculum, software, and hardware support structures must be available to

the students. It takes an enormous amount of man-hours to create completely new design projects each year. Even after the projects are decided, the students would still require instructions and possibly labs for the technologies they would have to use within the project, if the project is to have any complexity to it at all and it is to be completed in a single semester. Creating documentation and activities on that scale requires months, even years, not days. It is expensive and simply not sustainable to operate in that manner. Having the technical resources available allows students to be brought up to speed more quickly and eliminates some strain on faculty, staff, and teaching assistants. Trying to assign students complex projects without the resources they need only results in the failure of the project.

- **Safety** - Safety must be a primary concern when designing any system that students will work on. Any person creating a project that would be presented to students will have to consider the potential safety concerns, such as high voltage, optical radiation, and hazardous chemicals and take appropriate steps to ensure proper safety precautions are taken.

Project constraints are given to students to help teams more effectively learn design. Projects must have hardware and software integration. These projects are becoming a necessity as almost all electrical engineering projects have some mix of hardware and software. Also given that our teams are usually small (4-5 students per team) this gives people who are more software-oriented a chance to contribute to the project. It also allows us to incorporate the computer engineering specialization:

- **Accountability and Complexity-** If projects are going to be complex and the individual students held accountable for their part of the project the project must be separated into parts. Using Functional Decomposition (see Appendix A) gives us that ability. By breaking the project down into functional blocks and creating a detailed block diagram,

students are able to work relatively independently for a short time. This independent working time puts accountability on each student as to whether or not the project will be completed and it gives him or her a sense of ownership in the project as well. Also, breaking the project down into smaller parts allows more complex projects to be given. (See Figure II-V for an example. The HIU is part of the MAGE System and the HIU is also broken down into its constituent blocks.)

- **Human factors** - Something that is generally not considered when creating student design project. However, most things engineers' design will need to be used by non-engineers. Also, there are specific times when the Human Factors of a device are part of its' core functionality. The things students' design must be usable in a real environment, even a lab environment[19].
- **Rules -** All engineering systems have rules that govern the system. These rules can be operational rules or software and hardware protocols. Teams must implement rules in their system so that they match the requirement of their project to the larger system. They may have to use a specific protocol such as CANBus, or students might be implementing the three laws of robotics.
- **Student Motivation** - If the project motivates the students to be engaged, the students will likely learn more [15, 16]. Research has shown that elements of MMORPGS can help motivate students [4]. See Chapter IA.
- **Cooperation Between Students -** Students will often be asked to work on a team with people from very different backgrounds. Many times it can be difficult to find a common interest among the team. Also, getting teams to cooperate with each other without unnecessary amounts of competition can be extremely challenging[20].
- **Original Projects -** Some design competitions feature a recurring design task, such as NATCAR [21]. Design courses that participate in those competitions often suffer from a

lack of original design. Students simply copy what was done previously. These competitions remain in use because it is very difficult to design an original competition each year.

- **Support Multiple Specializations -** Most complex engineering systems today require the combination of at least two sub-disciplines of electrical engineering. One distinct advantage to this "inter-disciplinary within a discipline" approach is that it is simple to demonstrate meaningful interactions within interdisciplinary teams as required by ABET [22].

We have found that when all of these desired characteristics are present in a system level design project it makes creating sustainable, educational, and fun projects much easier.

## C. GENERAL DISCUSSION OF GAMES IN THE MAGE SYSTEM

This section will provide background information on games, specifically Role-Playing Games (RPGs), and help bring into context the importance of games in the MAGE System. MAGE was designed around RPGs and therefore draws many elements of its design and organization from the concepts and structure of games. Without a good understanding of RPGs many of the design decisions and organization choices made in this project will not make sense. If you have a good understanding of MMORPGs and *Dungeons & Dragons*® you may choose to skip this section and continue at subsection section 2.

### 1. ROLE PLAYING GAMES

MAGE was designed first and foremost as a tool for education, that incorporates the ideas and concepts Role-Playing Games (RPGs). RPGs are a class of games in which each player creates a character whose identity they will take on during the game. Every player is responsible for creating a history for his or her character and to some extent developing the characters statistics. *Dungeons and Dragons*® (D&D) was the first commercially available RPG and most other RPGs

follow, to some extent, the same design as D&D. In RPGs, players generally come together in a small group to play the game.

One of the players is called the Game Master (GM), or, in the case of D&D, the Dungeon Master (DM). The GM is responsible for implementing the rules and creating the setting in which the players characters will exist. The setting includes a rough outline of the storyline of the game, any challenges the characters will face, and any traps or enemies that must be defeated.

The GM starts the game with an introduction of all the characters and a broad description of the setting. Players describe their characters and the character's actions during gameplay. The game begins with some sort of problem presented to an individual character, or in most cases a team of characters, and their decision on how they will approach the problem. The GM then describes the setting, any addition non-player characters, and consequences of player's actions. For example, a team of players could be exploring a dungeon. The GM would have to give descriptions to the characters of what they are seeing so that the characters can decide what to do next. Some characters may decide to stick together while others may want to explore closed doors or hallways. All the while, the GM is responsible for keeping the story going and following the rules once the characters come in contact with other non-player characters, such as monsters.

The rules of the game will determine the outcome of some of the characters actions. If, for instance, a character that went wandering off on its own encountered a monster the ensuing battle would be governed by the rules of the game. The outcome of the fight would be dependent on many factors including the abilities of the player's character, the abilities of the monster, the combined attack of a team of characters. These abilities are factored into the battle through the use of different types of dice according to the rules of the game. If the player uses magic, they may have certain abilities to attack or defend against a monster. By rolling the dice, the GM would determine whether or not the player was successful. Some rolls are equal in that the player

just needs to roll a dice higher than what the monster rolled while others are more complex requiring a certain number to be thrown or a certain percentage. And in other cases, a team effort may be required. Using a turn based system players take turns trying to dwindle down the monster's strength before the monster gets a chance to attack one of them. Sometimes the combined effort results in an early victory, while other times it could prove fatal to a player. The GM's knowledge of the rules and the player's knowledge of their character's abilities are essential during a sequence of this nature.

Upon completion of a quest or goal, characters increase all of their statistics at once by gaining a level. Level gaining can happen during the course of the game or only when the game is finished. Gaining levels helps characters to defeat stronger monsters and receive greater rewards.

The gameplay will continue until the characters meet the final challenge set up by the GM. The challenge could be any number of things such as the solving of a puzzle or the discovery of a treasure. The characters might be required to defeat a specific monster as a team or simply to survive whatever punishment the GM puts the characters through until the end of the storyline. Games generally last only a few hours, but they could be part of a series that lasts through several sessions. When the end does come, the GM will describe to the characters the effect their actions had on the game world.

The Live Action Role Playing game or LARP has the same basic rules as a standard RPG. The main difference is that the players actually act out their characters actions, similar to improvisational theatre. In a LARP costumes are common and the game takes place in a real environment. The environment may resemble the fictional setting and the players will generally use props to make their environment more closely match the fictional one. Combat in LARPs is sometimes carried out with simulated weapons such as foam swords or air soft guns.

The number of players in a Live Action game can range from a few, like standard RPGs, or large groups. Large groups would normally require a larger space and people generally separate into teams.

Massively Multiplayer Online Role Playing Games (MMORPGs) allow large groups of people to interact over the Internet. In *World of Warcraft*® people from all over the world can cooperate in an RPG. Using computers it is relatively easy to create a fictional environment and characters can be fictional creatures. In the MMORPGs the computer takes the role of the GM and implements the rules and keeps score. Just like standard RPGs, MMORPGs have a storyline and objectives. Player's use their characters to "act out" a story line like in LARPs. Also like other RPGs, there are many objectives that cannot be completed without the help of a team. In MMORPGs in game props, or items, take the place of real props. MMORPGs have taken elements of LARPs and RPGs to create a simulated environment that player's can enjoy.

## 2. INTEGRATION OF R.P.G.S AND MAGE

MAGE regains the use of the physical world in role-playing. In LARPs, a GM implements the rules and the players rely mostly on imagination for combat. You can't really shoot someone with an actual bow and arrow and expect to be welcome next time. In an MMORPG, the characters you control are entirely computer driven the only interaction you have with them is through the computer. MAGE combines these elements into what we call a Live Action MMORPG, or LAMMORPG.

At its core level, a game is simply a set of rules that allow actions by players to be judged in order to determine favorable or unfavorable outcomes. The MAGE system allows for instantaneous calculations of in game occurrences, such as damage taken, while still allowing the player to maintain physical engagement with the environment. By designing a modular hierarchical block based system, we have created a system that is easily understandable, expandable, and

modifiable. This creates a project that is user friendly, portable, and provides instant gratification for the player. The MAGE system consists of electronic hardware worn by players, a central server that manages overall game information, and other hardware in the gaming environment which players interact with. Each player thus communicates with the environment and the other player's wearable computers. Communication between players and the environment is done via data packets that contain all the information needed to implement actions such as attacks, spells, using items in the environment, and sending any other needed data. The idea is similar to computer and console-based games except with far more motion.

## D. USING MAGE TO TEACH CAPSTONE DESIGN

The primary goal of any design project is to teach students engineering design. We designed the MAGE system in a way that facilitates the design process. Our design cycle shown in Figure I-I demonstrates a simplified process of design that can be easily taught in a fifteen-week capstone course. Experience and research [13] indicates that a project that allows teams to undergo multiple iterations have much higher probabilities of success.

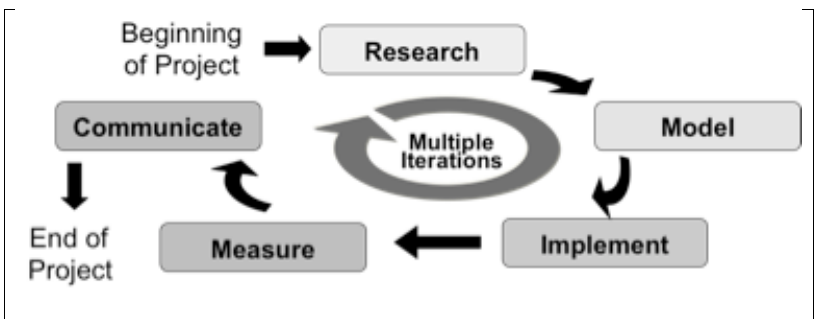


**Figure I-I: Engineering Design Cycle**

Students who participate in MAGE projects integrate their work into the system at multiple levels from local, on-player subsystems to distributed data networks. Beyond teaching design in the constraints of an existing system, MAGE additionally integrates content across electrical engineering and computer engineering degree programs. The range of hardware, software, and

communication channels in MAGE integrate computer engineering with the five areas of specialization offered in the electrical engineering program: communications, controls, and signal processing; computer engineering; lasers and photonics; electronics and mixed signal VLSI; and energy and power.

MAGE was created as a game because games encourage friendly competition and cooperation [4, 15, 16]. As mentioned previously games cut through ethnic and gender barriers [2]. Through game development students gain advantages of competition without the rankings or potentially negative consequences. Since students will use what they create in playing a game, the human factors of the design have a strong impact on performance.

MAGE allows for several different ways to offer engineering projects to students. One example of this is a project that adds new functionality to an existing system, such as constructing a new peripheral. The project could be used to build a new device that would act as an IR grenade or a large shield with a display, thus creating a device that performs a fundamentally new function. In this type of project, students build from the ground up a new device that must adhere to the specification of the existing system. Forcing students to use standards and existing specifications creates context and clearly defined constraints. See Chapter IVC.

Students could also create projects that modify the functionality of MAGE. An example would be a project that modifies the game code to report the position to the server from a USB attached GPS module. Another example would be modifying the code on a players GUI to display the position of all the members of their team on the battlefield. Students could also change the game software entirely and create a new type of game. This type of project requires students take one part of the system and modify it to extend the existing functionality of the system. As with the last type of project students must adhere to the specifications of the existing system. In short

students are able to work with an existing and well-supported system while at the same time not having to repeat past projects.

MAGE was designed as a mixed hardware-software system to make the system more portable and easily customizable. To create a project with only hardware or only software would most likely only be possible if projects are not given as a complete and separate device. To do this we would have to violate our constraint that Project is a subsystem of a larger system that can be broken down using functional decomposition.

Since the MAGE system integrates a variety of communication, display, RF, software systems, power management, and control technologies, it is easy to assign students roles on teams that require knowledge from different sub-disciplines of electrical and computer engineering. For example, during one semester the course can have one group work on a new RFID antenna design, another group work on a new type of weapon such as an IR grenade, and another group work on a GUI (software graphical user interface). The system level approach also allows the MAGE System to be integrated into the curriculum starting at the first semester. For instance, students in a Circuits class could design a simple LED display that could connect to a pre-existing communication and control module. Also, given the nature of system level design there is also the potential for MAGE to be introduced to secondary education institutions.

One of our requirements for a good design project is that projects must be part of a larger system. We have discovered that having experience with systems engineering is critical to the success of design projects. However, giving projects that are a part of a large system has some constraints of its own. The system must be able to be broken down into small functional subsystems and the system must have clearly defined constraints and be restricted to a limited set of technologies.

The philosophy behind many of our decisions regarding the structure of MAGE come from our desire to match the systems and processes used in the project to the high level goals of the

capstone design course. Functional decomposition, or breaking complex systems down into separate but interacting parts, is a critical aspect of design. The MAGE system itself teaches functional decomposition of a complex system and it is designed so that students can break their projects down into blocks as well. Creating a working final product is as much an integration problem as a lesson in teamwork. Each student is expected to be responsible for one or more blocks of the team's project. This approach gives each person a personal investment and a responsibility to the project. It allows individual assessment with the benefits of a team project. Using functional decomposition we can easily break the system down and adapt it to support the high level goals of the capstone design course.

Compared with paper proposals or computer simulations the MAGE system allows student teams to not only go from initial conception of an ideal to fabrication of a complete hardware or software solution but also allow multiple iterations of the design in one semester. Typically this is very hard to achieve due to the limited time in a one-semester capstone course. The MAGE system provides a significant knowledge base, technical support, and structure to students through written documentation, teaching assistant expertise, and a base of former students.

MAGE is designed so that it will meet the requirements of a good design project. We have discussed how MAGE has been designed to be modular and contain subsystems that are part of a larger system. This type of modular design meets our requirement that projects must be part of a larger system and be able to be broken down using functional decomposition. MAGE is a capstone design project for primarily electrical engineering curriculums and it is made up of electrical systems. Given this, our requirement for projects being related to the engineering discipline is met. By including Zigbee, RFID, and CANBus as technologies we meet the requirement that cutting edge technologies be included in projects.

When teaching design the idea of a live action game system emphasizes several aspects of design that can be difficult to otherwise achieve. Reliability, Quality, Robustness, and Human Factors are essential to successfully play the game, wearable systems must be robust. The design imposes constraints on size and weight; it is not fun to lug around a car battery. This forces students to consider portability. The less power the device uses the longer a team can play and the less weight they have to carry in batteries; forcing students to consider power management. Human Factors force students to look at the playability of the game in a real scenario.

By using playable games as a base MAGE ensures projects will fall within a TRL of 5-7. These levels represent prototypes. These levels of TRL are achieved by setting standards for project types and providing clearly defined constraints (another requirement of successful design projects), like being able to wear your creation in the field. Setting the standards and constraints for the projects creates a scenario where many types of projects can be created and provides students the necessary background and context to bring them out of the TRL 1-4 range. Projects usually fall within the TLR 1-4 range when developing new technologies for use in MAGE. These types of research-based projects require more experience with design and would be well suited of graduate students.

These next two conditions for successful design projects are very much dependent on the timeline of the project and the complexity of the project. When creating projects it is very important that the team be given enough time to complete multiple iterations of the design cycle [13]. Therefore the project creators must not give projects that are so large and complex the students will never have enough time to complete it. It is also important that projects be complicated enough that the project itself can be broken down using functional decomposition. If a team is going to work on a subsystem then the project given to the team must be able to be broken down as well. Functional Decomposition gives us a method to do this. By breaking the project down in to functional blocks and creating a detailed block diagram students are able to work relatively independently for a

short time. This independent working time puts accountability on each student as to whether or not the project will be completed and it gives him or her a sense of ownership in the project as well. Projects will go much more smoothly if the team creates a very good detailed block diagram. These two conditions are critical to the fluid operation of a successful design course.

CHAPTER II

EVOLUTION AND DEVELOPMENT OF MAGE

At the highest-level MAGE is a game. Composed of hardware and software systems that players use to play the game. Players are people who take on the role of a character in the game and attempt to complete a task or a quest. The player carries wearable hardware with them that allows the rules of the game to be implemented. Using that hardware players interact with other players and Non-player characters. NPCs are devices like remote weapons (E.g. IR Grenades) or vendors of goods. NPCs supplement the gaming environment and allow players to purchase goods or may provide a static enemy for players to battle.

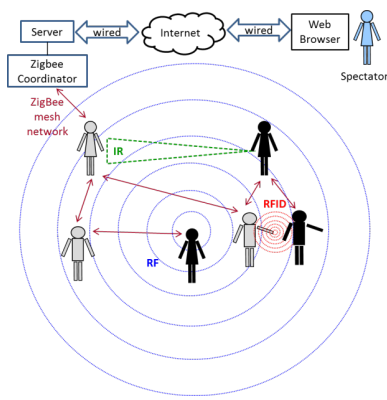


**Figure II-I: Players in the MAGE environment playing the game through interchange of data packets**

Most role-playing games, such as *Dungeons & Dragons*® and *World of Warcraft*®, have three basic types of attacks: Melee, Ranged, and Area of Effect (AoE). Melee combat is the name given to all close-quarters combat. Think of melee combat as a medieval sword fight. Ranged combat

will be referred to as any combat that affects only a single distant opponent and thus needs to be highly directional to preserve the game mechanics. In ranged combat weapons like the bow and arrow or guns are used. Magic can also be ranged depending on the spell. AoE combat is a weapon or spell that affects players within a specific area. AoE attacks usually take the form of a spell or grenade type weapon. Translating these combat modes into a usable hardware and software platform is one of the key reasons why games can create challenging electrical engineering projects.

Players equip themselves with peripherals that can simulate the Melee, Ranged, and AOE combat effects. Each of these combat methods uses a different communication technology as described in Chapter III. Communication is the area that is most highly dictated by game play. The player's peripherals transmit and receive signals to other players and each player's Wearable Hardware. All of the different communications technologies are combined with one packet structure (see Chapter III) that allows a transparent exchange of data. MAGE Peripherals are plugin devices that allow users to easily expand the functionality of the MAGE system. Peripherals don't just enable combat; they can be as simple as a device that blinks an LED to complicated IR Transmitters with adjustable range. No matter what the peripheral does, the principal idea is to allow the player to do or show something that they could not do without it.

Peripherals are limited as they can only expand on existing functionality and are designed primarily as communication devices. In order to allow players to interact with each other and implement game rules more hardware and software is needed. This core hardware is where most of the fundamental changes in MAGE have taken place. The following sections explain the evolution of the core hardware and software that make MAGE function.

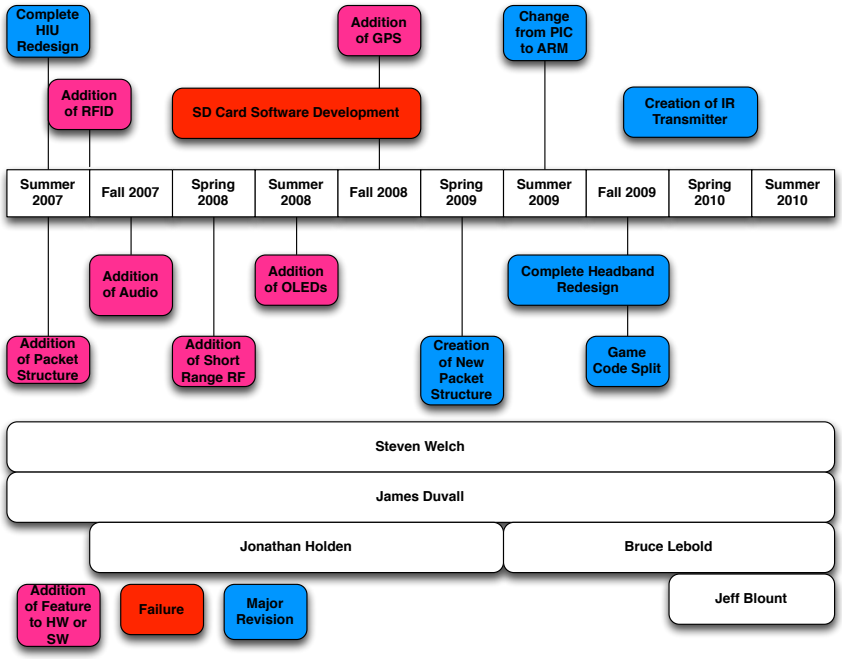


**Figure II-II: MAGE Development Timeline**

## A. MAGE V1.X

I joined the MAGE development team in the summer of 2007. All of the original team had moved on and left little to work with. A fellow student, James Duvall, and Dr. Alan Cheville, formed the new development team. We spent some time to take stock of the current platform, and we created a system level block diagram (see Figure II-III). The first prototype of the MAGE System had only two major systems, the HIU (Hardware Interface Unit) and peripherals. In this version of the HIU is the center of the system. The peripherals communicate with the HIU and served to expand its functionality. These two subsystems were used to create all the functionality of the MAGE system.

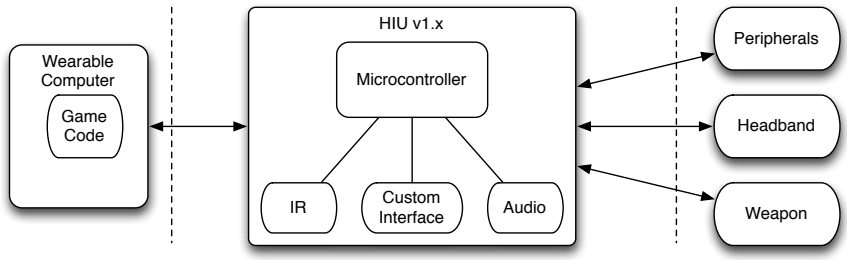


**Figure II-III: MAGE v1.x**

During a game, a player would use peripherals to send packets. Another player's peripherals would receive that packet and pass it to the HIU. The HIU then processed the packet using the game code and an appropriate response was generated. At that time the game code was very under-developed and a response could only could be an increase or decrease in health.

There are two peripherals that a player needs to make his or her hardware complete. The first is what we call a Headband. The Headband is a piece of headgear that receives IR data and relays it to the wearable computer. The Headband also has some LEDs that allow other players to know a player's current health status. The second is the Weapon. This Weapon is required to cast spells from the Touch Screen Interface. Each player must construct his or her own weapon (similar to the Jedi in the *Star Wars* universe [23]), but one IR Weapon design is provided as a reference. The Weapon must have the ability to adjust the brightness of the IR LED allowing the range to be limited depending on how strong the character is. The Weapon also has a reprogrammable RFID chip. This RFID chip allows players to emulate sword type weapons.

Having finished our design review we began to examine the problems with the current system. First and foremost was communication, the protocol used between the peripherals was completely custom made. It was unreliable and badly documented. The structure of the packets used by the system were also badly documented and often changed to suit the situation. The HIU was not very functional and had many design flaws. The core hardware design and communications

system was discarded. We kept the design of the peripheral we call the headband and the ideas and principals of using games in capstone design. We created a new communications system and a new system block diagram. These choices led to the second version of the MAGE system.

### B. MAGE V2.X

Our choices in communications technology and our desire to expand the functionality of MAGE lead to a redesign of the HIU. The HIU would become a platform for testing and developing the new technologies we had chosen. The rest of the 2007 summer was used to create Version 2.x of the MAGE system. We kept the relationship between the HIU and the peripherals that we had in version 1.x, but the communications between them transitioned to CANBus. The new packet structure we developed allowed us to transmit data uniformly over all our communications technologies. The hardware design of the HIU was my responsibility and I created a new block diagram (see Figure II-IV). After the block diagram was created I went on to research and test the components of the individual blocks. For communication our new design included an RFID reader/writer, a Zigbee Module, an IR transmitter and receiver, CANBus, GPS. An OLED display and an audio system were added to provide more interaction with the players. An SD Card was added for storage and everything was controlled by one of the most powerful 8bit microcontrollers available. When the summer was over I had finished the design and fabricated a prototype.

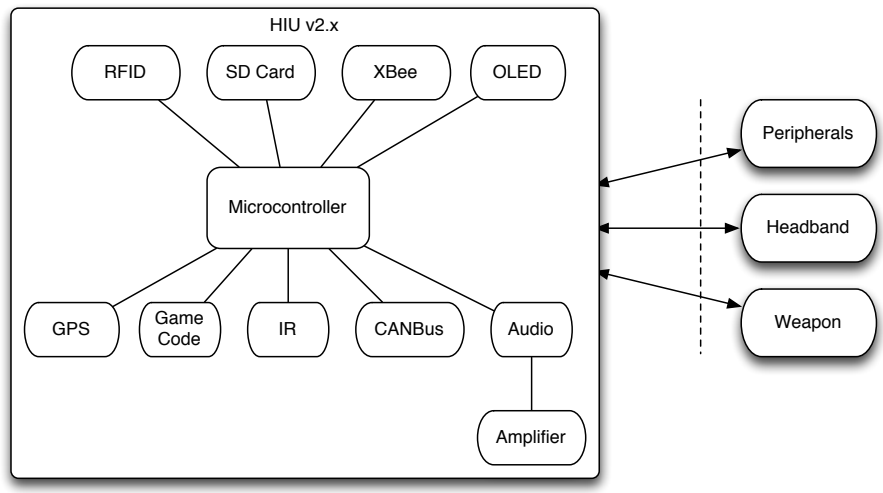


**Figure II-IV: MAGE v2.x**

The design was a success and all the subsystems worked electrically. However, up to this point there was only enough software developed to see that the hardware worked. We spent the next few semesters trying to get everything working.

The RFID, CANBus, and IR blocks had the benefit of being used in past student projects. We were able to use the software from those projects to quickly get our hardware functioning. The software for the RFID block was almost completely functional. It only lacked some error checking which was not added until the final version of the MAGE system. A library for CANBus communications was provided with the compiler. There was no trouble writing software to communicate over CANBus. The IR block functioned, but badly. It was not until the fall of 2009 that Bruce Lebold joined the team and created a system that worked (see Weapon). All of these blocks were available for students to use in the Fall 2007 Semester.

Also in the fall of 2007, Jonathan Holden joined the development team. He helped our team develop the software to record and playback audio. The audio feature relied on a special IC that

could store sound clips and then play them back later. The IC functioned well, but had limitations. Recording the audio to the chip had to be done manually and the audio chip could only store about a minute of sound. Also, recording good quality audio to the chip was not easy as the gain had to be carefully matched. While the Audio system worked these limitations made it impractical for our needs.

Once the Audio Block was working I began to experiment with point to point and mesh networks. I had almost no problems writing software for our XBee module since only serial communication was needed. After the XBee module was configured it functioned as a serial cable would. The simplicity and the ability to quickly change from a mesh network to a point-to-point link made the XBee module very consistent and adaptable.

Our next step was to get our OLED Displays up and running. Writing software to configure the OLED and display colors was relatively straight forward, even though it took the entire summer of 2008 to create. At the end of the summer I could draw simple shapes and text and make everything move around the screen. However, anything we wanted to display had to be hard coded. Text was easy, but fonts had to be custom made and there was not enough space to store images. We knew we would eventually need more space that is why we added the SD Card.

Libraries to support an SD Card and a FAT file system were provided with the compiler, but they were only beta versions. Jonathan started the development of that software and was, in some respects, successful. The problem was with the FAT file system library. There were many errors and it was not compatible with PCs. James and I joined Jonathan to help with the effort, but eventually we gave up.

We never got around to integrating a GPS into the HIU and only minimal progress was made with the Game Code. After our failure with the SD Card, we had a system that did not meet our expectations. In the spring of 2009 we stopped development of the MAGE 2.x platform

### C. MAGE V3.X

Until now my role in MAGE system development was that of an engineer. During the spring and summer of 2009 while we designed a new MAGE system my role began to change. I became more of a project manager and systems engineer rather than a hardware and software developer. I was able to better guide the development of MAGE and we began to plan out our system. First we spent some time doing research into the poorly understood systems of MAGE, like the IR transmitter discussed previously. At this point devices like the iPhone were just appearing and we began to see potential for hand held computers. So we revisited the idea of using wearable or hand held computers. The HIU was scaled back and many design requirements like storage and interfacing with the user were moved to the HHC (hand held computer). Once we had an idea of what we wanted we created a plan for the development and deployment of the various MAGE system components. Before any design work began, the whole system was planned out and the hardware and software designs were carefully documented. Finally, we updated the designs of our peripherals, the Headband and the weapon. This shift in thinking is the biggest change in the MAGE v3.x system. The reset of this chapter, and the chapters to come, explain in technical detail the operation of the MAGE v3.x platform.

Figure II-V shows a block diagram of the MAGE v3.x system. The blocks in the v2.x HIU have been distributed over two major subsystems, the HIU and a new HHC. The HHC now handles all of the Display, Audio, and storage we tried to integrate into the HIU. The HIU is more like a peripheral that expands the communication options of the HHC. Several versions of the HHC have been tried, but it has always been something we purchased instead of built. The first HHC was a low cost ARM7 development board running an embedded Linux. The Linux support of this system was not very good and we moved to the Gumstix platform after two semesters. The Gumstix platform allowed us to use newer technology and the Linux support and support from the manufacturer was much better. The HHC can be any device that supports serial

communications. So, we would often use PCs to develop Game Code. The recent appearance of Mobile devices like Android and iPhones brings an opportunity for MAGE.

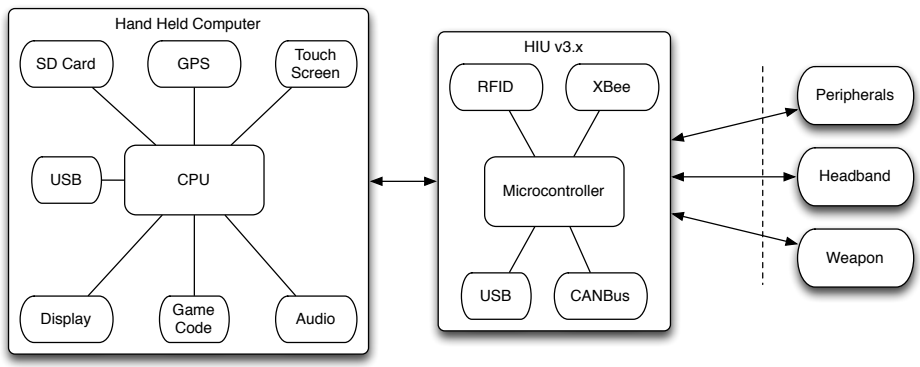


**Figure II-V: MAGE x3.x**

Using iOS and Android devices as our HHCs make MAGE even more accessible and usable as there are already design ecosystems built up around these two platforms.

The MAGE system is represented as a highly simplified level-one block diagram in Figure II-V. The system worn by players consists of three major sub-systems and power. The wearable computer (**WC**) contains two of those subsystems: a hand-held computer (**HHC**) and a hardware interface unit (**HIU**). The third major subsystem is peripherals. The rules for the MAGE system are encoded into a database and software running on the HHC. If an action is directed at the player, an incoming data packet is decoded by the game code on the HHC. The HHC exchanges data with the HIU, which supports communication with MAGE's peripherals, other players, and the environment. In playing the game, player actions are encoded into 64 bit data packets, which are transmitted between players or the environment using RF, IR, or RFID communication (see Chapter III). The result of the action changes a player's statistics, represented by a set of

numbers, which symbolize factors such as health, strength, etc. In a standard RPG, the GM would carry out these calculations. The player is informed of actions taken through status indicators and through a graphical user interface (GUI). The GUI also allows the player to take actions through a touch-screen interface. The Game Code is written in C and is encapsulated in such a way that it is easily portable to many platforms. The Game Code could be easily ported to iOS or Android to support a wide variety of devices. Peripherals in the game are designed by players, and can serve a multitude of functions such as status indicators, sensors, or weapons. Peripherals are connected to the HIU through a four-wire interface that supports CANBus communications and supplies power to the peripherals. Peripherals have been designed that allow players to send and receive IR data, display their status, and take actions through a series of motions detected by accelerometers, similar to the Nintendo Wii.

Just as the player is the center of the MAGE system, the wearable computer is the center node for each player. It enforces the rules of the game and regulates the other subsystems. The wearable computer consists of two major parts:  an HHC running the Game Code (see Chapter V) and the HIU (see Chapter IVB). The wearable computer and its peripherals are the basic components needed to transmit, receive, interpret, and act on data. Without these systems the game could not function.

The HHC part of the system is a small hand held computer. It allows us to coordinate all the functions of the game through the Game Code (see Chapter V) and to provide a GUI for the player to implement an action. The Game Code is what drives all of the action in the game. When the player receives an incoming packet the Game Code processes it. If the player casts a spell at another player, the Game Code generates a packet and sends it to the HIU to be transmitted. A GUI allows the user to get information about the character they are playing. Statistics such as health and armor are displayed on the interface. The interface is also used to or cast spells. Each

team is able to customize their own GUI because a large part of game play is dependent upon human factors.

### D. MAGE IN CAPSTONE DESIGN

Now that we understand what went in to the development, let's take some time to explain how this relates back to the course work. In the current structure of our Capstone Design Course, each team constructs two peripherals. The course is separated into two major sections: six weeks of training and nine weeks of design. In the first section, students are given the schematics of a headband and receive the necessary training to construct the peripheral. First, all students learn to create schematics and PCB. Then the training becomes more specialized. Some will learn skills like soldering, milling and etching PCBs, and test and measurement. Others will learn how to program microcontrollers and design 3D objects to be printed. We call these training courses certifications and they are essential to the success of a team in our capstone design course. Certifications are continually under development and new certifications are added as old ones are dropped. Since we began the certification system, the success rate of the projects in the first six weeks has risen to 100%. After the first six weeks of the course, students have most of the skills they need to design and build their own peripheral.

The last nine weeks is spent with the students designing and building a single peripheral. Students are given a generally vague description of a project, usually a description of the basic function, and any design requirements such as specific interface protocols or size and weight. They are then largely left to complete the hardware and software design and construction on their own. The course is highly structured with deadlines for specific phases. This allows students creativity in their project, the ability to get hands-on experience, and maintain a level of difficulty and accountability with specific criteria and constraints.

The point is that a single MAGE peripheral can be sufficiently difficult for a Senior Design Project. Difficulty can be adjusted up or down by limiting or expanding the functions of the peripheral. Making sure the peripheral won't be too hard is a very challenging task. In recent semesters, the failure of a project in the last nine weeks has become uncommon. When a project does fail, it is usually due to the difficulty level of the project being set too high, or a lack of sufficient documentation and development platforms for technologies in use in that particular project.

The MAGE system may seem complicated, but actually playing the game is much simpler. Let's imagine for a moment a duel between two players. Each player is equipped with a wand (IR Weapon) and a sword (RFID Weapon). When the players are at a distance they use the touch screen to cast spells. As the players move closer they engage in a sword fight. For safety reasons the swords must only come within a few inches of the players RFID reader to register a hit. Students should not actually be hitting each other with swords. So it is not only a matter of getting close to your opponent with a sword, but some skill with a sword will be required as well. Each time a hit is registered the indicator lights on the players costume change. If this were the only iteration of the game, players would soon get bored. The ability to expand on the system has been built into the peripherals.

A list of all Design of Engineering Systems (DoES) projects to-date are listed in the appendix.

# CHAPTER III

# COMMUNICATION

As mentioned previously, the technologies that were chosen for communication were driven by game play. For direct, medium range (across the room), point-to-point communication that mimics single shot weapons (wands, bow and arrow, gun) data is sent via infrared (IR) light. For Omni-Directional and long-range communication, data packets are sent over 2.4 GHz RF using the 802.15.4 protocol. This type of communication simulates AoE spells or weapons (a grenade or buff for people near the player). For very short-range data that simulates physical contact (such as swords), programmable RFID tags are used. Figure IV-I illustrates players interacting using these different types of attack methods. Communications between the wearable computer and the player's peripherals are carried out through CANBus. Figure II-V shows how the peripherals connect to the Wearable Computer. The rest of this chapter will discuss in detail what communications protocols were chosen, why, and how they are integrated into one system.

## A. HARDWARE IMPLEMENTATION

When choosing a technology to implement melee combat, several types of technology were investigated. Very short range RF, touch type devices, and direct contact data transfer were all considered as options. Given that most melee type weapons are sword or axe type, it was decided that a technology requiring direct contact with the player was not favorable. It was very easy to see how a player could get carried away and end up injuring each other accidentally. That decision left short range RF as the only logical choice. There are two basic types of devices in the short range RF category: active and passive. Most small low power RF transmitters come in either the passive or active type, not both. RFID, however, comes in both active and passive

types. This made RFID an ideal choice as it was very short range and could be used in many situations throughout the game. Using RFID also helped meet a critical characteristic of good design projects: the inclusion of cutting edge technologies.

Low frequency RFID, using the ISO/IEC 11784/11785 standard, was chosen for two reasons. First, the 135 kHz operating frequency made developing the hardware necessary for the game much simpler as RF design concerns were minimal. The 134kHz frequency allowed complexity to be added in the design of the antenna. Secondly, the RFID tags were available in two types: a standalone tag and reprogrammable IC tags. The reprogrammable IC tags were perfect for use on a player weapon as the tag could be reprogrammed easily for each player's characteristics. Otherwise, when the player changes equipment or gains a level, the player would have to return to a station with an RFID programmer to update his or her weapon. Standalone tags work well for items, as they are independent and do not require an onboard power source. This along with the low cost of RFID tags allows them to be easily embedded into clothing and small containers the players can use, such as potions and other game items.

Finding a type of technology to implement ranged combat was fairly straightforward. Very few systems can match the restriction of target area at a distance as an optical system can. Optical communications systems have been used for many years in remote control and short-range data transfer such as IrDA. Optical systems can be used to very effectively limit the range and the area that data is transmitted to. This makes it a very good candidate for implementing bow and gun type weapons. While many different IR modulation protocols are used, there is only one wavelength for which many IR receivers are available. A 950 nm IR receiver module was chosen because of the integrated carrier frequency detection and filtering available. Using a module with the analog portions integrated, instead of designing one from the ground up, made development much faster and allowed time to focus on other complexities of the optical communications system. A software protocol was developed to get the system functioning. Refining or creating a

new protocol is an example of a complex project within the MAGE system that could be offered to students. While IR LEDs are available that can limit the angle of light that is emitted, adding an optical system to increase the focus of the light is another example of allowing the project to modify the existing system.

AoE attacks have several characteristics that must be implemented through the technology of MAGE. These requirements drive the type of technology we chose to implement. AoE attacks must have a variable range and they must be omnidirectional. Any technology used to implement AoE attacks must transmit data in a circular pattern around the attacking player. The radius of the circle must be adjustable and the data that is transmitted must be easily changed. By adding an RF system not only are the requirements of an AoE attack met but it also allowed us to bring in the Electromagnetics sub discipline of electrical engineering.

Designing an RF system from a component level was considered too difficult and did not match up with the system level design approach used throughout the MAGE system. Many different single chip and modular solutions were explored. Most of the single chip solutions that were researched required at least some development of strip line components, so to further simplify the design a decision was made to use a module. Not only did this decision reduce development time, but it also allowed for a more modular systems level approach to the integration of an RF system.

When searching for an RF module several key specifications were desired. Minimization of size, cost, and power consumption were paramount. A module with a serial interface was also required, as it would communicate with a microcontroller. The XBee line of RF modules from Digi International was the best choice available at the time. Using the XBee modules also added a lot of flexibility as they can be configured to use an 802.15.4 Point to Point protocol or a Zigbee Mesh Network. Using a module with such bleeding edge protocols again helped meet a critical characteristic of good design projects, the inclusion of cutting edge technologies.

When MAGE was first being developed communications between the HIU and its peripherals were handled by a protocol written in house. This protocol turned out to be a very bad method of communication. Requirements for a replacement were a simple, noise resistant, standards based protocol that was in wide spread use. CANBus, specifically CAN v2.0, was the best fit for our needs. It is used heavily in the auto industry and can withstand a great deal of noise. Also, CANBus is simple, having only a two wire differential pair that can be many meters long.

## B. SOFTWARE IMPLEMENTATION

Having so many different communications technologies means that there are many different communications protocols and packet structures. For simplicity's sake, a single packet structure was developed that could be used over all the various communications technologies in the MAGE System.

The IR communications protocol was developed in house so it was easy to make sure it would support whatever general packet structure we came up with. The 802.11.15 protocol can handle up to 100 bytes per packet [24]. So there is a lot of room there. The RFID tags come with different memory sizes with most of them being 80 bits or 10 bytes [25]. On RFID tags the last two bytes are used for a checksum. That leaves 8 bytes for user data. Considering the size limitations of RFID an 8-byte packet was developed.

All communications between players occur through the 8-byte (64 bit) packets. These packets represent all allowed actions within the game. Each part of the packet is described in detail in the sections to follow. The structure of the packet is highly dependent on the way the game is played and the structure of the Game Code (see Chapter V).

### 1. PLAYER TO PLAYER DATA PACKET STRUCTURE

Each packet is divided into six distinct values Team Number, Player ID, Subroutine ID, Process ID, Query Return Value, and a Checksum. Each of these values has a different size measured in

nibbles. A byte is an 8-bit number that can represent 256 unique values. A nibble is half of a byte (4 bits) and can represent 16 unique values. The purpose of the value and the reason for its size is discussed in the following text.

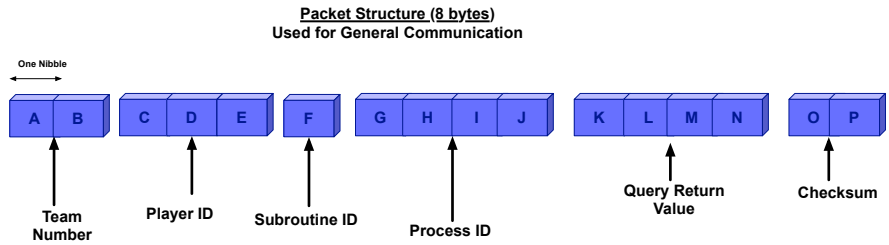


**Figure III-I: MAGE Player to Player Data Packet Structure**

### *a) TEAM NUMBER*

Every player or object in the game will have a team number. The team number indicates the team of the player sending the packet. When determining a size for the Team Number value it seemed likely that there could be more than 16 teams playing at once. However, it seemed very unlikely that there would ever be more than 256 teams playing at once. Therefore the size of the Team Number value was set to two nibbles. The team numbers in Table III-I are reserved; all others are open for assignment to a team.

**Table III-I: Team Number Values**

| Hex Value | Description |
|---|---|
| FF | Reserved for internal communication of data |

### *b) PLAYER ID*

The player's ID number distinguishes between individual members of a team. Since the Player ID has three nibbles allocated to it, up to 4096 players can exist on a team (0-4095). The size of the Player ID was assigned with the idea that there could be very large teams combatting each other. The idea that each team could be so large came from thinking universities could compete against each other in competitions. There are no reserved Player ID values.

### *c) SUBROUTINE ID*

The Subroutine ID tells the game code what subroutine is supposed to be run for this packet. There are a limited number of actions that can be taken in the game. As such, one nibble is sufficient to store the Subroutine ID. Some values of the Subroutine ID have been assigned to one of eight possible subroutines. The subroutines are listed in Table III-II along with a brief description of what they do.

**Table III-II: Subroutine ID Values**

| Hex Value | Name | Function |
|---|---|---|
| 0 | Melee | Contains processes that follow the standard rules for melee combat. (I.e. attacking with a mace, polearm, or fists.) |
| 1 | Request Statistic | Contains processes that allow the player's HIU to request a statistic from another player. |
| 2 | Receive Statistic | Contains processes that allow the player's HIU to receive a statistic from another player. |
| 3 | Special Execution | Indicates that the process is not a normal subroutine and will need to be executed as a unique case statement. |
| 4 | Buff/Debuff | Contains processes that increase or decrease the stats of the player. |
| 5 | Magic or Trap | Contains processes that follow the standard rules for attacks and traps that use saves rather than AC to determine a hit. |
| 6 | Item Request | Contains processes that allows an item to request statistics from a players HIU |
| 7 | Item Receive | Contains processes that allow a player's HIU to send statistics to an item. |
| 8 | Consume Item | Used when a player consumes an item such a potion. |
| 9 | Environment | Used when a player interacts with the environment. |
| F | Server | Reserved to send data to server |

### *d) PROCESS ID*

The Process ID indicates which process will be executed under the selected subroutine. The combination of the subroutine ID and process ID uniquely specify an action in the game. Since the processes depend on the subroutine ID, a process ID does not uniquely identify a specific action. There are no reserved Process IDs. Given that there could be many different items or

spells it was appropriate to assign 4 nibbles to the Process ID. Having 16 bits, 65536 possible processes may be assigned to any given subroutine, numbered from 0 to 65535.

### *e) QUERY RETURN VALUE*

The Query Return Value is a payload. It is used to store information that is needed by a subroutine from another player in the game. The Query Return Value is also used to transmit additional, common information with packets in order to minimize the number of queries and speed up operation of the game. The size of the QRV was set to four nibbles to ensure that there would be enough room to transmit most values used by the game. The use of the QRV by each Subroutine is shown in Table III-III.

**Table III-III: Query Return Values**

| Subroutine ID | Subroutine Name | Additional Data |
|---|---|---|
| 1 | Melee | To Hit (byte 1), Plus Damage (byte 2) |
| 2 | Request Statistic | |
| 3 | Receive Statistic | Requested Statistic Value |
| 4 | Special Execution | |
| 5 | Buff/Debuff | |
| 6 | Magic or Trap | |
| 7 | Item Request | |
| 8 | Item Receive | |
| 9 | Consume Item | Used when a player consumes an item such a potion. |
| A | Environment | Used when a player interacts with the environment. |
| F | | |

### *f) CHECKSUM*

When originally designing this packet structure, it was not known that the checksum value on the RFID tags could be calculated and written to the tags by the user. So, a checksum was implemented using the last available byte (two nibbles). Now that it is known, RFID tags have a possible payload of 10 bytes. It is recommended that the Checksum size be increased to four nibbles and a new checksum algorithm be used.

## 2. INTERNAL PLAYER DATA PACKET STRUCTURE

Packets sent over the player's internal network have a slightly different packet structure, with modified uses for each packet value. This was done so that changes to the player's hardware setting could be made while the game is running.

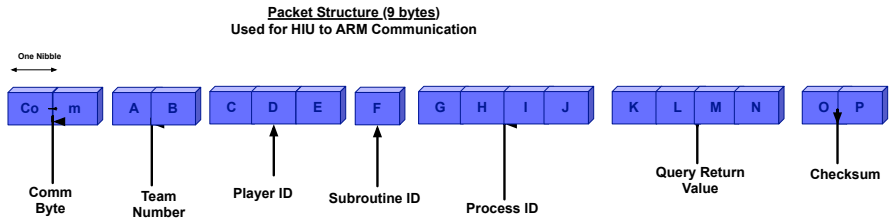


**Figure III-II: MAGE Internal Data Packet Structure**

### *a) COM BYTE*

The COM Byte is used by the HIU to route outgoing packets to specific peripheral devices. The HHC attaches the COM Byte and sends the packet to the HIU. The COM Byte also lets the ARM board know which device a packet was routed from. The first eight values (0-7) of the COM byte specify particular devices on the HIU while the remaining values are used for addressing particular CANBus devices. The COM byte is removed by the HIU before passing the packet to the peripheral to be transmitted. If the packet is to go over the CANBus network, the COM Byte remains intact and is used as the CANBus address for the destination CANBus device. Most of the COM Byte values are left open for teams or players to use in designating custom-built equipment; however, some values are reserved for game-wide devices or development tools as outlined in the table below. The COM byte is stripped off when packets are sent between players.

**Table III-IV: Com Byte Values**

| Hex Value | Device |
|---|---|
| 00 | Received IR Data |
| 01 | RF Module |
| 02 | RFID Module |
| 03-06 | Reserved for future expansion |
| 07 | CANBus – Broadcast Statistic (Shout!) |

| | |
|---|---|
| 08 | CANBus- IR transmitter narrow beam |
| 09 | CANBus- IR transmitter wide beam |
| 0A-0F | CANBus- Reserved for future expansion |
| 10-FE | CANBus- Open for user-developed devices |
| FF | Reserved for errors, flags, etc. |

Packets that begin with 07 as the COM Byte are “Shout! Packets” and are used to broadcast the players statistics to peripheral devices that may need them. One example would be when a player’s health changes a “Shout! Packet” is transmitted to all the peripherals. This packet would be received by the player’s headband for instance and the headband would update the display according to the player’s new current health.

***b) TEAM NUMBER***

As discussed in Table III-I, Team Numbers equal to 0xFF (or 255) are reserved. If a team number of 255 is encountered by the HIU, or one of its peripherals, the packet will not be transmitted. Instead, the firmware on that device will use the information contained in the packet to modify its settings in some way. Changing the power level of the RF transmitter would be a good example. The content of the data in the rest of the packet, with the exception of the checksum, is up to the developer of the device receiving the data.

CHAPTER IV

# DETAILED HARDWARE DESCRIPTION

This chapter describes in great detail the hardware of MAGE. Many standards are used in the hardware and software of MAGE As standards are encountered explanations will be given regarding why they were chosen.

Figure IV-I shows a level one block diagram of two players in the game. The figure shows several fundamental blocks required for each player to have so that they can send and receive all of the necessary signals. The Headband, HIU, Weapon, and HHC make up the fundamental components, which each player must carry with them at all times.

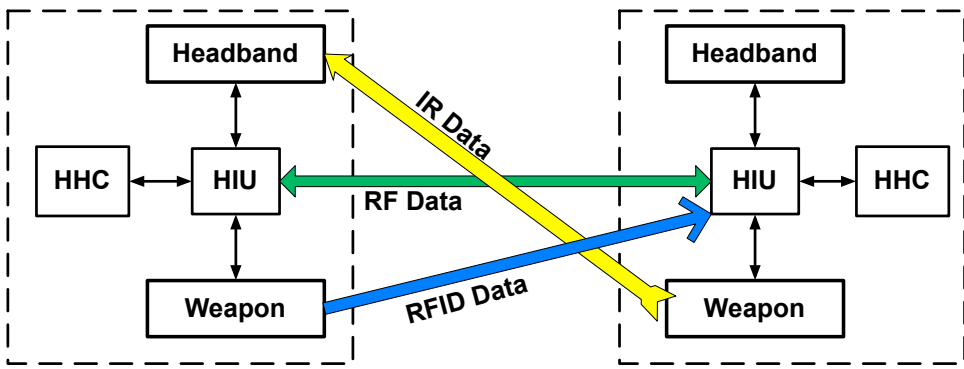


**Figure IV-I: Two players interacting during a game using the MAGE system.**

The Headband is the main IR Receiver and Player Status Indicator. It allows you to receive an incoming IR attack and it also shows other players your health status. It communicates with the HIU over CANBus and with the IR Weapon over 940nm IR.

The Weapon is exactly what it sounds like. It allows you to deliver attacks in the form of short range and long range IR, as well as very short range RFID. It also communicates to the HIU over CANBus. The IR is delivered at 940nm and the RIFD is on a 134 kHz carrier frequency.

The HIU is the center of all the communications traffic. It allows incoming and outgoing packets to arrive over RFID, 2.4 GHz RF, CANBus, and Serial. The RFID receives attacks from the weapon and also allows a player to read and write to RFID Tags that might be in their gear or items. The RF is used to communicate with other players and to deliver area of effect spells. CANBus is used to talk to all the peripherals.

Because the MAGE System was designed using a modular block approach, the specific hardware implementation is not necessarily important. This section will explain the function of each device that makes up a player's hardware in such a way that the implementation of the hardware is not a critical factor. By using standard protocols and portable code, the MAGE system can be implemented on almost any hardware. This approach teaches system design and higher level design skills. The communications power requirements and device function will be detailed in each of the following sections.

### A. HHC

The HHC is the brain of the MAGE system. It stores all of the player's information and implements the rules of the game. The HHC also provides a critical graphical user interface that allows the player to view current statistics and issue commands. The HHC only communicates with the HIU. That communication is carried out through a serial interface using the USB to serial port present in the HIU. Other methods, such as Bluetooth, for implementing the serial connection between the HHC and HIU have been considered. This is mainly because having wires running all over a player decrease the reliability and usability of the gaming system.


**Figure IV-II: HHC Block Diagram - Level 0**

The HHC is where all the processing takes place. The packets transmitted and received by the HHC follow the: MAGE Internal Data Packet Structure (see Chapter IIIB.2). When a packet is received, the HHC will process it and update the player's statistics. The new statistics will be displayed on the HHC's display and any changed statistics will be broadcast to the player's peripherals. When the player casts a spell, a packet will be transmitted to the HIU where it will be disseminated.

Considering the necessary skills and typically short time frame of educational programs, building an HHC that is both small enough and rugged enough to be used in a game would not be practical. Therefore we decided that a prebuilt system should be used. Since the basis of the MAGE system is block based modular design using a prebuilt HHC should work perfectly if it has the required serial interface. Most of the hardware implementation details of the HHC are not explicitly defined. There are only a few carefully defined interface requirements of the HHC. This was done purposefully so that any platform with a serial port may be used.

During the development of MAGE many things have been used as an HHC. First we used an ARM 7 [26] based development board that was rather large when compared to mobile devices like the iPhone. We abandoned the ARM 7 in favor of the ARM Cortex A- Gumstix platform [27]. The Gumstix was much smaller and more powerful, but not as robust. Our last efforts in HHC development were to begin using the iOS and Android platforms. These platforms let us use relatively inexpensive devices and they already have a strong development community. Utilizing

these two platforms will allow us to focus on software development. There are only a few requirements for the HHC's interfaces which are shown in Figure IV-III.

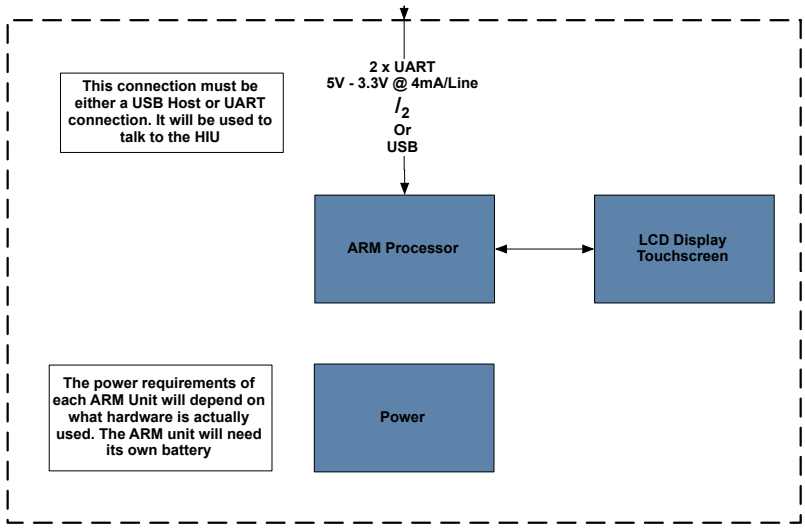


**Figure IV-III: HHC Hardware Block Diagram**

(1) DETAILED DESCRIPTION OF DISPLAY

The display block of the HHC must do two things. First, and most important, is to display the information from the game. Second is to accept input from the user. Since the graphical user interface that is shown on the display is designed and created by the player, the method of input is also up to the player. Any number of input methods are available, though buttons and touchscreens are the most common, the choice will depend on the chosen hardware.

(2) DETAILED DESCRIPTION OF ARM PROCESSOR

The ARM platform was chosen because of several reasons.

- ARM CPUs are extremely energy efficient which is needed in a mobile environment
- ARM CPUs, unlike microcontrollers, can support higher level programs and are much faster

- Most small mobile platforms with LCDs use ARM processors.
- Many different free Open Source compilers.

As with the display, the specific ARM processor used is not critical. As long as the system has a compatible compiler and an accessible serial port, any CPU can be used. In our test setups we frequently use a PC with a serial port running the HHC Software.

(3) DETAILED DESCRIPTION OF POWER

The power block will have requirements added to it by the other components of the HHC. The power block will need to supply enough power to the HHC to keep it running for several hours. Also, whatever battery or power source used will need to be light enough as to not weigh the player down.

**B. HIU**

The HIU is the communications hub of the MAGE system. It connects all of the peripherals of the system to the HHC. Incoming communications are received via CANBus and RF and relayed through a serial connection to the HHC. The HIU is also responsible for distributing power and communications from the HHC to the peripherals of the system.

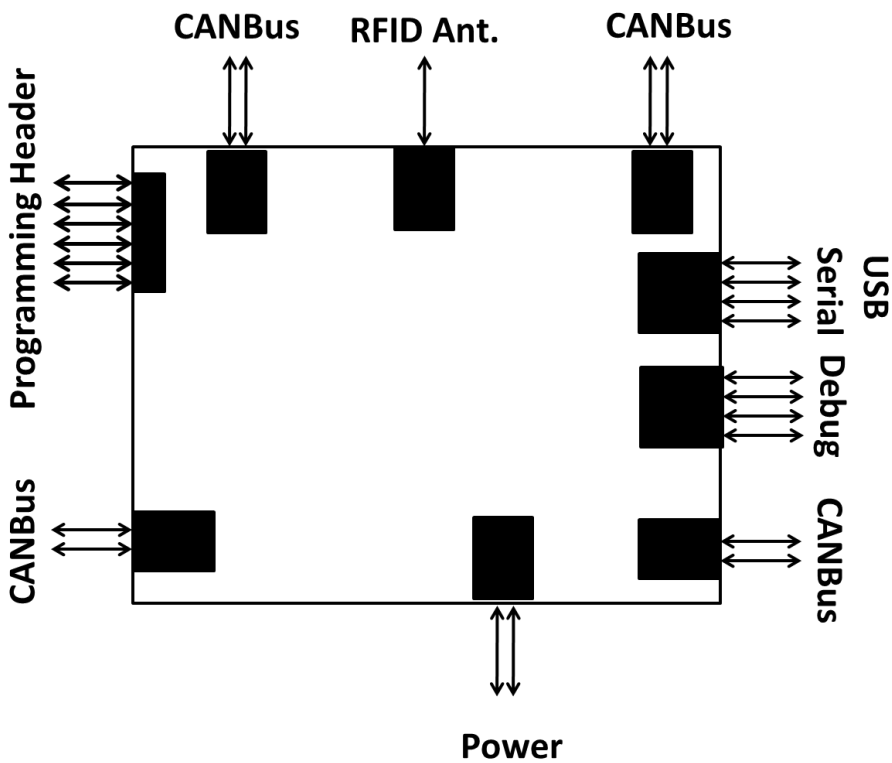


**Figure IV-IV: HIU Block Diagram - Level 0**

The HIU can receive packets from CANBus, RF, or RFID, or from the HHC over serial. All incoming packets from the environment are processed by the microcontroller on the HIU. The standard 8-byte packet is assessed and a COM byte is assigned depending on where the packet is coming from. The new 9-byte packet is then transmitted over serial to the HHC. The HHC will then process the new incoming packet and may send a response back to the HIU. The HIU will use the COM Byte to determine where the packet should be sent. These packets can go many places. An outgoing packet has possible destinations of a device on the CANBus, the RF transmitter, or the RFID writer. “Shout! Packets” that broadcast player statistics to MAGE peripherals get sent to all devices on the CANBus. Packets can also include settings for the

hardware devices in the HIU and peripherals. An example would be setting the power level of the RF transmitter.

The user mounts the HIU boards and connecting cables onto belt or other clothing. Beyond the CANBus interface and RFID antenna connection the HIU board also has a Debug Connector that contains a connection for a Brainstem TTL to RS232 converter, a connector for an In Circuit Serial Programming Connector for debugging and programming the microcontroller, and a power connection for a battery or other power source.

### *a) HIU DETAILED HARDWARE DESCRIPTION*

The HIU is broken down into several blocks (Figure IV-V), each of the blocks show in the figure will be described in detail later. The block approach helps simplify the design and create defined specifications for the inputs and outputs of the system. The most important function of the HIU is the handling of packet traffic. This is the only way data from the environment can be communicated to the HHC. The HIU has a connection for power along with five other IO connections, Serial Data, CANBus Data, RFID Data, and RF Data. Any peripheral connected to the HIU receives power at 5V with a current limit of approximately 100mA. This limitation drives the power usage requirements of the MAGE peripherals. CANBus is used to communicate with the MAGE peripherals and the interface protocols are defined by Industry Standards. Standard 134 kHz Low Frequency RFID is used to write to RFID tags (Items such as potions) and to read incoming RFID attacks. A low power RF transmitter is built into the HIU in the form of an add-on module. This allows the player to communicate with other players and with the server. A standard 802.15.4 point-to-point scheme is implemented through the use of an XBee module. Using this platform allow for more modularity as well as the option for choosing a higher power module and mesh network support.

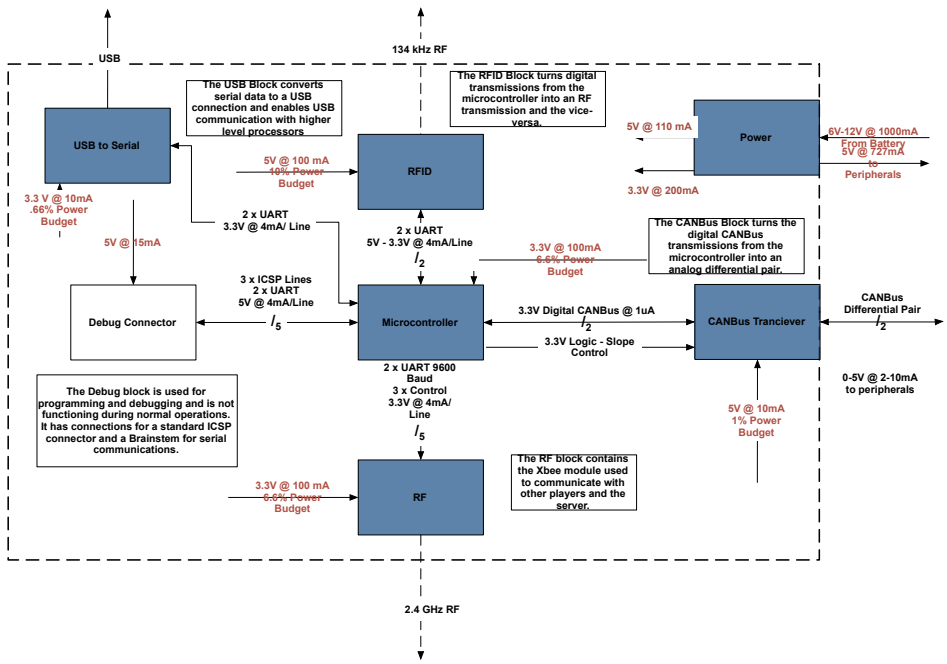


**Figure IV-V: HIU Hardware Block Diagram**

(1) DETAILED DESCRIPTION OF CANBUS TRANSCEIVER BLOCK

The CANBus Transceiver Block of the HIU must accept the standard differential CANBus signals from the HIUs peripherals and translate those signals into digital signals that can be interpreted by the microcontroller. The same process works in reverse. The Microcontroller sends digital CANBus data to the CANBus Block. That digital signal is then converted to the differential analog voltages required by the CANBus protocol.

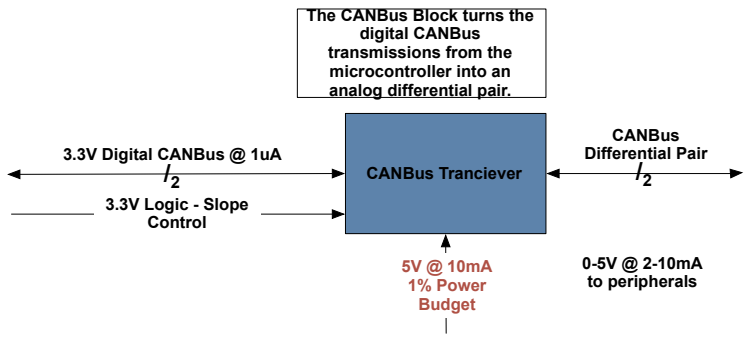


**Figure IV-VI: HIU Hardware Block Diagram – CANBus**

The CANBus Transceiver block can be implemented with any 5V CANBus transceiver such as Microchip's MCP2551 High-Speed CAN Transceiver [28]. Multiple connection possibilities exist. There are connections for the CANBus Differential Pair that connects to the HIUs peripherals and there are connections for the digital CANBus that connect to the CAN Controller in the Microcontroller. There is also a connection for CANSlope.

The connections for the Digital CANBus and Differential CANBus are direct connections to other devices. The CANSlope connection is however slightly more complicated. The slope control allows user control of the rise and fall times of the Differential CANBus signals. Changing the value of an external resistor changes the slew rate. Typically the CANSlope pin can

also put the CAN Transceiver in High Speed mode (90° angles on waveform) or sleep mode. However, this functionality will vary with the transceiver model.

A 120Ω resistor should terminate the CANBus if it is the last node on the bus. The CANBus must be terminated on both ends of the bus. It is very important that the CANBus only be terminated at each end.

(2) DETAILED DESCRIPTION OF RFID BLOCK

The RFID Block (see Figure IV-VII) of the HIU must be able to read 134 kHz passive RFID tags and transmit that data over a serial connection to the microcontroller. The RFID block must also be able to receive serial data and then write that data to an RFID tag.

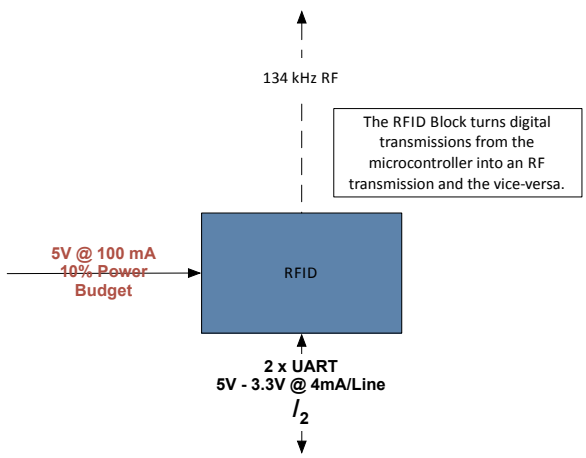


**Figure IV-VII: HIU Hardware Block Diagram – RFID**

Like all other blocks, the discrete components that make up the RFID block don't particularly matter as long as the components that make up the block handle the IO correctly. An example LF RFID base station IC is the TI TMS3705. This IC works well but the datasheets can be hard to find and the communication scheme used is somewhat complicated. We found that it could be

very difficult to get information on Active RFID devices due to security issues. Since Active RFID components are used in many security systems, companies were often unwilling to give undergraduates access to that technology.

The block has connections for serial communication with the microcontroller, a connection for an antenna and a power connection. The serial connection to the microcontroller must be in the 5V – 3.3V range and should be a baud rate of 115200 bps or less. Anything faster and it is likely that the microcontroller will not be fast enough to receive the data stream.

The characteristics of the connection for the antenna are not critical as the frequency is so low. Typically the shorter the connection from the IC is to the antenna connection, the better. As for the antenna, coils are the standard choice for short range LF RFID. These coil antennas usually have an impedance of 400-700 uH.

The most complicated component of the RFID block is conserving power. When the block is actively scanning for an RFID tag current consumption spikes well over the 500 mW. Using a small ultra cap could minimize this large current spike. The goal is to conserve power while not missing any incoming RFID information. The read frequency is controlled through the firmware and can therefore be adjusted to minimize the power consumption.

(3) DETAILED DESCRIPTION OF RF BLOCK

The RF Block of the HIU must communicate over a serial connection to the microcontroller. It must also transmit and receive RF signals at 2.4 GHz using the 802.15.4 point-to-point protocol. To ensure simplicity, modularity, and upgradability, a module such as an XBee is recommended for use.

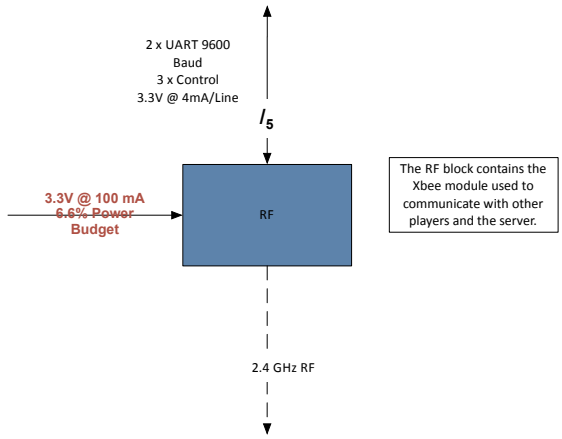


**Figure IV-VIII: HIU Hardware Block Diagram – RF (XBee)**

There are many good reasons to use a module as part of a PCB design.

- The simplicity of purchasing a low cost module verses designing a complex RF system is advantageous when building systems and using the block approach to problems.
- The advantages of having all the RF communications protocols handled inside the module instead of on our microcontroller simplifies the design and creates a more segregated block.
- By using modules, one can easily change it out for one with a greater output power or for a different communications protocol. For instance, changing from a mesh network to a point-to-point transmission.

The RF Block must have connections for a 3.3V serial interface. Using a 3.3v interface creates a measure of flexibility since most 5V serial interfaces will work with only 3.3V. A baud rate of 9600 bps will allow communications with a microcontroller. This was done purposefully because default baud rate of the XBee Modules is 9600 [24].

(4) DETAILED DESCRIPTION OF MICROCONTROLLER BLOCK

The Block Diagram for the microcontroller drives the selection of its implementing hardware. There will need to be support for CANBus, either built into the microcontroller or external to it. Four serial ports will be required with at least two of them being hardware based, as interrupt driven communication is not required by the RFID block. The microcontroller will also have to be low power. Another requirement will be speed. The microcontroller will be receiving all sorts of communications data and will have to be fast enough to process and transmit it without losing any packets.

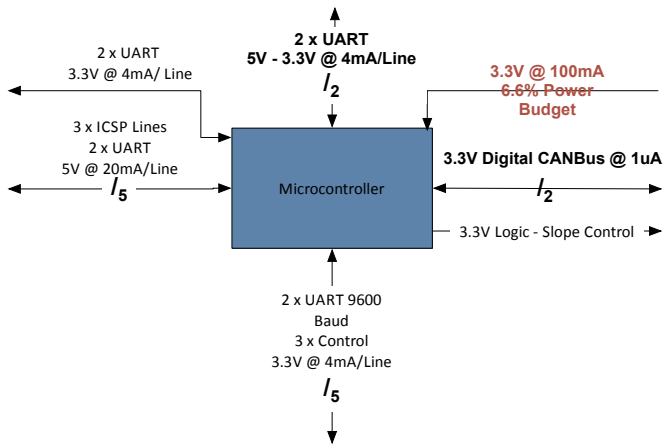


**Figure IV-IX: HIU Hardware Block Diagram – Microcontroller**

The IO requirements for our microcontroller block are fairly steep. It will need two hardware UARTs that can support both 3.3V and 5V operation. This is because the RF block accepts a maximum of 3.3V as an input and because the RFID block will be returning 5V on its transmit line. The RFID block should only transmit data when reading an RFID tag. Since a read request is made by the microcontroller, and a response is expected directly following that request, interrupt driven communication is not required. That allows the use of a software serial port. The RF and USB connections require interrupt driven communications since data could arrive at any time.

The hardware UARTs will be reserved for these two blocks. The last connection is the Debug connection. Since the reliability of debug data is not critical to the overall function of the system a software serial port can be used. Given the low speed of debug output, data integrity should not be compromised by using a software serial port.

The CANBus support must be provided in the microcontroller block as well. There are several CANBus chips that can provide this function at the required 3.3V interface levels. Microchip's MCP2515 CAN Controller is an example. Any 3.3V IO pin can be used for the Slope Control Line that connects to the CAN Block.

Most microcontrollers are very power efficient and can meet the power requirements. However, there are not very many that have built in CANBus, 2 UARTs, operate at 3.3V and have 5V tolerant IO. One example is the Microchip PIC25HJ128GP502.

(5) DETAILED DESCRIPTION OF POWER BLOCK

The power block must take the incoming 6V -12V from the battery and regulate it down to 3.3V and 5V for the HIU and its peripherals. The efficiency of the power block must be very high to maximize run time and minimize overall system weight. Using typical Low Drop Out regulators (LDOs) the power loss to heat would be large since LDOs have a very low efficiency. A power converter with a very high efficiency and current limit of at least 1A is needed.

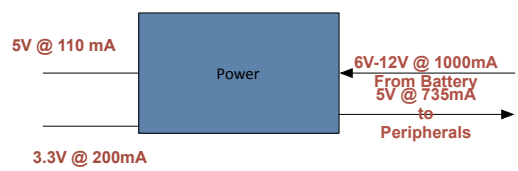


**Figure IV-X: HIU Hardware Block Diagram - Power Block**

The Power block has an input of 6V – 12V with a current of 1A. This power will come from a battery or set of batteries in series to make up the necessary voltage requirement. The power block will also need to be able to handle the high current requirements of the system.

An output of 5V as 845mA or about 1A will be required to power some HIU components and the MAGE Peripherals. Several HIU blocks also require 3.3V so it too must be converted from the incoming power. Only about 200 mA of 3.3 volts is required.

To provide the necessary power at the required current levels, and with the required efficiency, a switching converter is probably necessary. An example of a power converter that would work is the TI TPS54291. It can convert the required power with greater than 90% efficiency. These types of switching regulators require some careful PCB layout to meet the RF requirements of the circuit. There were many failed power circuit designs because of bad PCB layout practices. This turned out to be a valuable learning experience for students.

(6) DETAILED DESCRIPTION OF USB BLOCK

The USB block provides a connection from the microcontroller to the HHC through a more standard interface. It must accept a 3.3V serial connection from the microcontroller and provide a USB connection. The USB block has a very small power budget so the chosen hardware implementation must be efficient. A USB to Serial converter IC such as FTDIs FT232R would easily meet the requirements of this application.

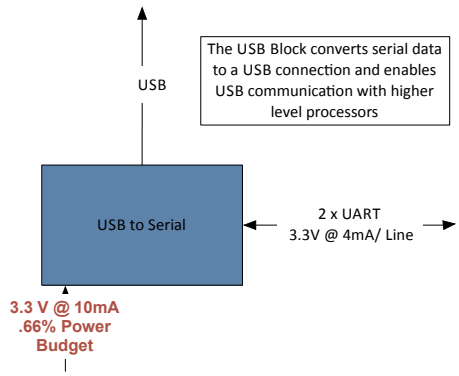


**Figure IV-XI: HIU Hardware Block Diagram - USB Block**

### *b) HIU DETAILED FIRMWARE DESCRIPTION*

The HIU is controlled by firmware that is stored in the microcontroller's flash memory. The processes of the HIU are driven by incoming data (see Figure IV-XII: HIU Data Flow). The HIU receives data from many sources: CANBus, RF, RFID and Serial from the HHC. All sources, except the RFID, are connected to interrupts. MAGE Internal Packets (see Chapter IIIB.2) are collected from the CANBus. These packets will be processed by the HIU and sent on to the HHC. Data from the RF and RFID blocks contain a standard MAGE packet (see Chapter IIIB.1) and will also be passed on to the HHC over serial. The packets received over serial from the HHC will either take some action on the HIUs hardware or be transmitted as CANBus, RF, or RFID packets.

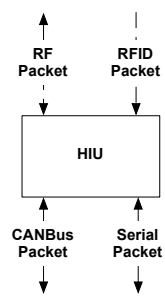


**Figure IV-XII: HIU Data Flow**

The HIU runs interrupt driven firmware. This allows for extremely fast response time to incoming data ensuring that no data is lost. It does however complicate the flow of data. The HIU firmware is split into two distinct “processes”. The interrupts all retrieve data and place it in a buffer. The main program loop checks to see if any data is in the buffer and acts on it.

### (1) DETAILED DESCRIPTION OF INTERRUPTS

The interrupt routines shown in Figure IV-XIII all activate based on external stimulus. The serial, CANBus, and RF interrupts are triggered when new data is detected. The RFID interrupt triggers on a timer. These interrupts have a unified purpose, to gather data, check its validity, and store it into a buffer for the main loop to process. Incoming serial and CANBus packets should all have a COMID in the packet. RFID and RF packets will have a COMID assigned to them before they are placed in a buffer.

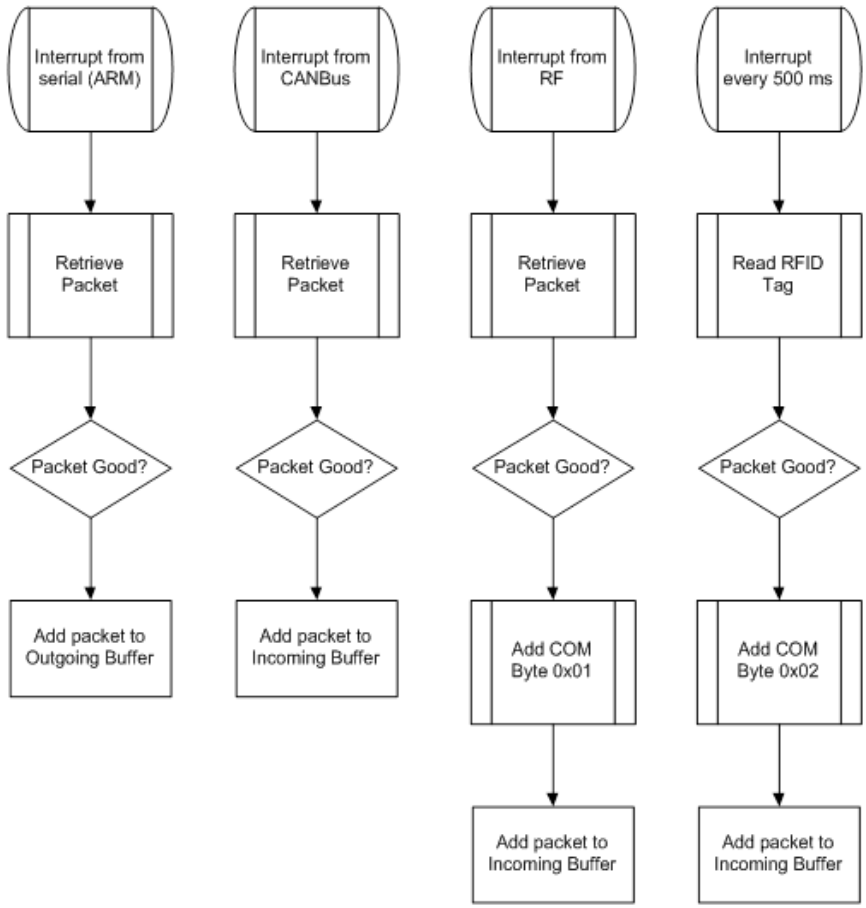


**Figure IV-XIII: HIU Firmware Flowchart - Interrupts**

(2) DETAILED DESCRIPTION OF MAIN LOOP

The main loop is what controls the HIU. Upon power up, the HIU goes through an initialization routine. The next step checks to see if there is data in the outgoing or incoming buffers. If neither of these conditions is met, then the firmware will simply loop indefinitely checking for data and acting on that data.

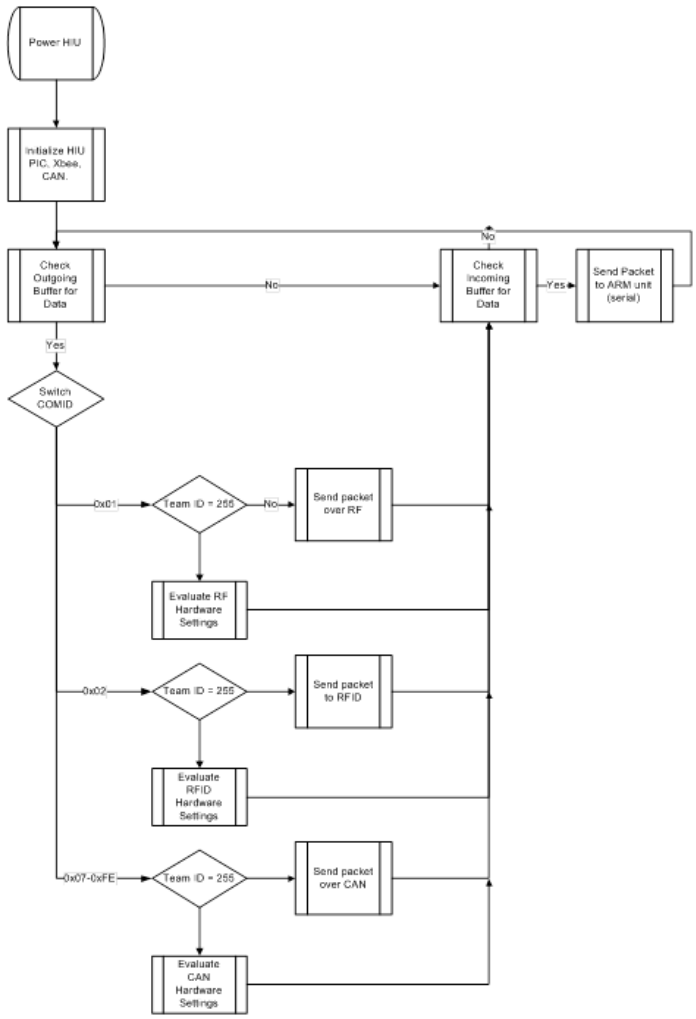


**Figure IV-XIV: HIU Firmware Flowchart - Main Loop**

When data is available in the outgoing buffer, it is evaluated to see where that data will be transmitted. Depending on the value of the COMID the data will be sent to the RF, RFID, or CANBus routines for further processing. If the Team ID has the value 255, then the packet is intended to modify the settings of the subsystem specified by the COMID. These settings could

be anything hardware related such as RF power level. If there is data waiting in the incoming buffer it is simply passed on over serial.

## C. PERIPHERALS

### 1. HEADBAND

The headband is a peripheral to the MAGE gaming system that is used both to detect incoming IR data packets and visibly display player health. The communications from the attacking player will be received using infrared and relayed to the receiving player's wearable computer using CANBus. The headband will receive a health packet that indicates the player's health status and display it using LEDs visible to the other players in the game.

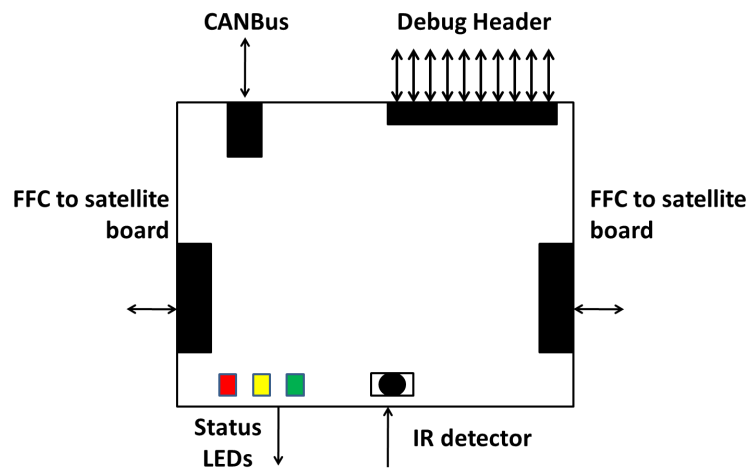


**Figure IV-XV: Headband Block Diagram - Level 0**

The headband system consists of one main communication and processing board and two satellite display and detector boards. Infrared light data packets are detected by the IR sensors and then read by an onboard microcontroller. The microcontroller then relays the incoming data packet over CANBus to the MAGE system. The MAGE system responds by sending the headband a 72-bit packet that contains player health. For details on data packet format see the MAGE system

documentation. When a properly formatted data packet is received over CANBus, the headband microcontroller red, yellow and/or green LED's light up on the main board and the satellite boards to visually display player health status.

The user mounts the headband boards and connecting cables onto a hat or other headgear. The three boards are equally spaced at approximately 120°. The use of three IR sensors allows detection of incoming radiation over 360°. Beyond the CANBus interface, LED's and IR detectors, the headband processor board also has a Debug Connector that contains a connection for a Brainstem TTL to RS232 converter and an In Circuit Serial Programming Connector for debugging and programming the microcontroller.

### *a) DETAILED HARDWARE DESCRIPTION*

The headband is broken down into several blocks. The block approach helps simplify the design and create defined specifications for the inputs and outputs of the system. The headband has three inputs Power, CANBus, and IR Data. The specifications for the CANBus and Power are inherited from the HIU. Any peripheral connected to the HIU receives power at 5V with a current limit of 100mA. This limitation drives the power usage requirements of the Headband. CANBus is used to communicate with the HIU and the interface protocols are defined by Industry Standards. The most important function of the headband is the IR Data reception. This is the only way a player has to receive IR Data. The protocol used is custom and resembles that used by remote controls. The only hardware defined part of the IR system is the 56 kHz carrier frequency. The Headband also functions as a display device and allows other players in the game to ascertain the health status of any player easily.

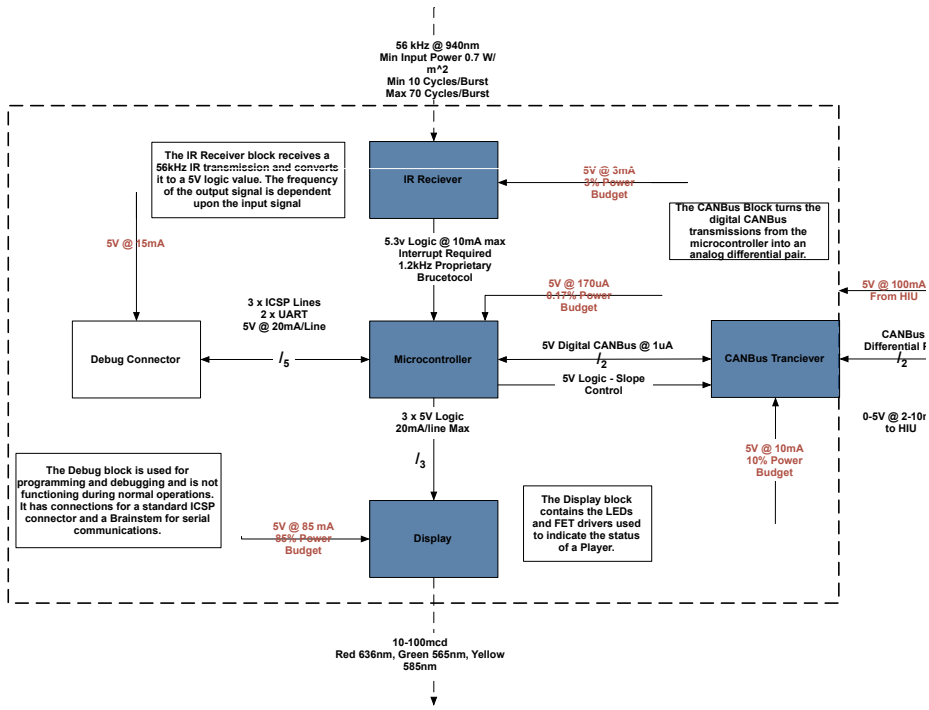


**Figure IV-XVI: Headband Hardware Block Diagram**

(1) DETAILED DESCRIPTION OF IR RECEIVER BLOCK

The reception of IR signals is the primary purpose of the headband. The IR Receiver block (see Figure IV-XVII) provides the facilities to receive these signals and convert them to digital signals. The digital output of the IR Receiver block is fed into the microcontroller and then decoded to retrieve a standard MAGE Packet.

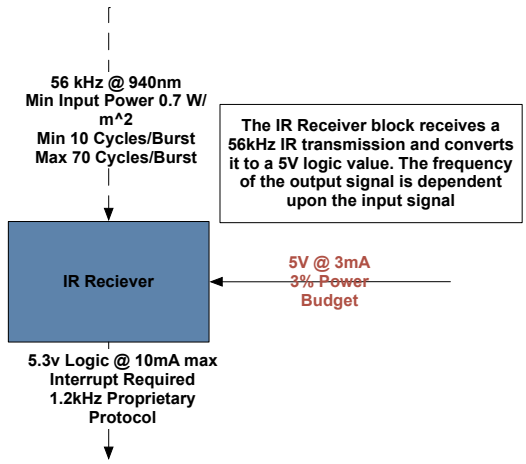


**Figure IV-XVII: Headband Hardware Block Diagram – IR Receiver**

The IR Receiver block can be implemented using a standard 940nm 56 kHz IR receiver such as the TSOP36256 in the AGC2 series from Vishay. This receiver was chosen after considerable lab testing to reduce noise. The receiver contains circuitry for automatic gain control and a bandpass filter. We found that this extra circuitry is necessary to receive a clean signal at a distance greater than a few feet. This receiver removes the complexity of implementing a discrete amplification and filtering circuitry. However, any IR receiver can be used if the proper filtering and demodulation techniques are implemented.

56 kHz modulated IR transmissions at a wavelength of 940nm with a minimum irradiance of $0.7\,{}^{W}/_{m^2}$ is the minimum required detectable signal. The 56 kHz carrier signal is demodulated

and the original input signal is retrieved. The decoded transmission is then output as an active low 5V digital signal that is fed into the PIC. The viewing angle of each IR Receiver is different. If reception from any angle around the player is desired at least three sensors will be needed.

Bruce Lebold researched many topics on how interference with fluorescent lights, different types of communication modalities, etc. affect the transmission quality. He spent many months perfecting the transmission protocol.

Elizabeth Welch 4/28/
**Comment [1]:**

(2) DETAILED DESCRIPTION OF DISPLAY BLOCK

Displaying a player's status is also an important function that the headband provides. Green, Yellow, and Red LEDs in different patterns are used to express the player's current status. The display (see Figure IV-XVIII) block consumes more power than any other part of the headband. The input to the block is three logic lines form the microcontroller block. Each logic line turns on one color led in the display sub system.

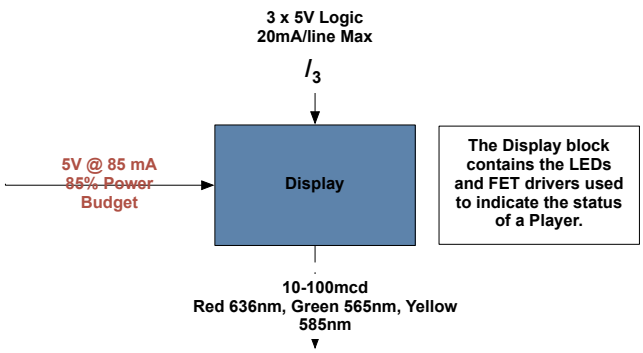


**Figure IV-XVIII: Headband Hardware Block Diagram - Display**

The display block only has 66 mW of power is available for each color LED. In order to create a display that can be seen from any orientation more power will be required. Some simple FET switches such as the FPF 2004 by Fairchild Semiconductor could provide the necessary extra power. The input to the FET Circuit must be 5V logic. The FPF 2004 FET Switch has both an over current shutdown and a thermal shutdown. This makes it ideal for powering our LEDs, as

these extra features will help prevent a catastrophic failure of the FET switch. This will greatly increase safety since the Headband will likely be mounted on a person's head.

(3) DETAILED DESCRIPTION OF MICROCONTROLLER BLOCK

The microcontroller block is at the center of the Headband. It is the brains of the device, handling the interfaces between other Headband blocks. The microcontroller block contains connections for all other blocks, including connections for a debug interface.

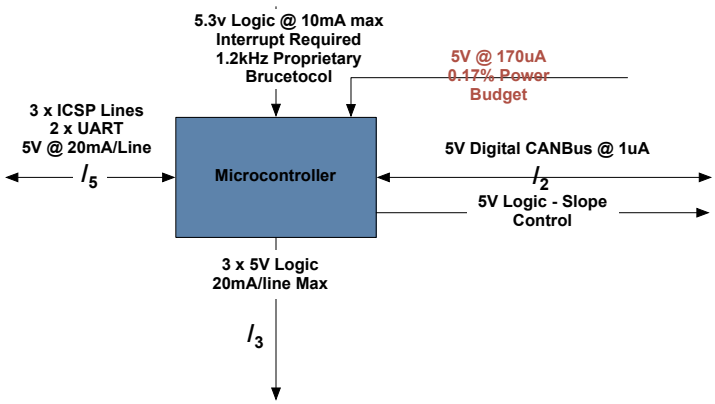


**Figure IV-XIX: Headband Hardware Block Diagram – Microcontroller**

The debug interface contains pins for the microcontroller's in-circuit serial programming (ICSP) and pins for a serial port. The MCLR pin is a master reset of the microcontroller and is connected to the extra switch pin in the CANBus Block. The connections for the Display block are difficult to see in the schematic, as they are part of a bus instead of individual pins like the CANBus block connections.

(4) DETAILED DESCRIPTION OF CANBUS TRANSCEIVER BLOCK

The CANBus block is reused in many of the hardware designs please see B.a)(1) DETAILED DESCRIPTION OF CANBUS TRANSCEIVER BLOCK for details.

### *b) DETAILED FIRMWARE DESCRIPTION*

The Headband is controlled by firmware that is stored in the microcontroller's flash memory. The processes of the Headband are driven by incoming data (see Figure IV-XX: Headband Data Flow). The Headband receives data from two sources: the CANBus and the IR Receiver. Both are connected to interrupts. MAGE Shout! Packets (see Figure IV-XXI) are collected from the CANBus. These packets tell the Headband the player's maximum and current health. Data from the IR Receiver contains a standard MAGE packet (see Figure IV-XXII).

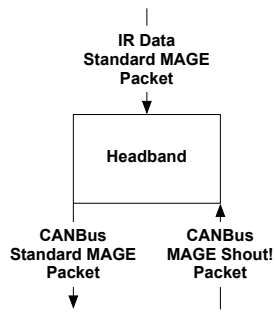


**Figure IV-XX: Headband Data Flow**

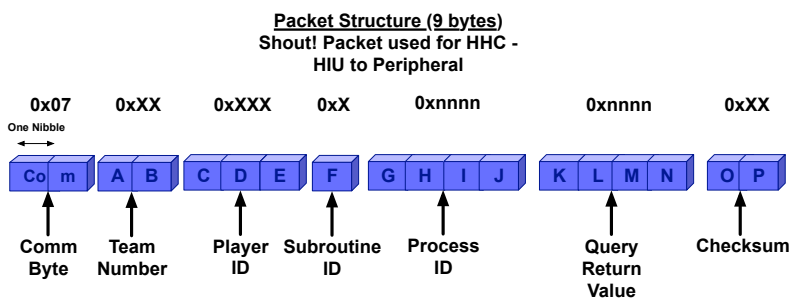


**Figure IV-XXI: MAGE Shout! Packet Structure**

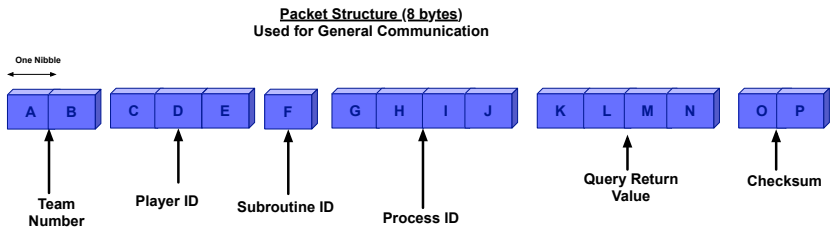


**Figure IV-XXII: MAGE Standard Packet**

The headband runs interrupt driven firmware. This allows for extremely fast response time to incoming data ensuring that no data is lost. It does however complicate the flow of data. The headband firmware is split into three distinct "processes". The main program loop checks to see if either IR data or CANBus data has been captured. A CANBus interrupt is triggered on incoming CANBus data and the data is quickly read in. The external interrupt is triggered at the beginning of an IR Data packet.

(1) DETAILED DESCRIPTION OF CANBUS INTERRUPT

When CANBus data is received an interrupt is triggered. The interrupt then collects the CANBus data from the buffer. If the CANBus ID is set to Shout! then the global flag CAN Data Waiting is set and the interrupt ends.

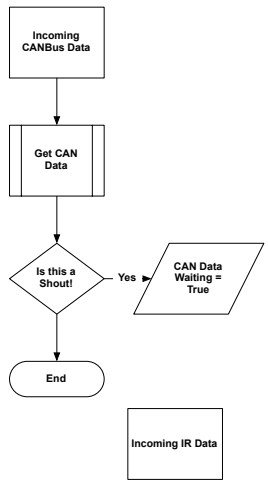


**Figure IV-XXIII: Headband Firmware Flowchart – CANBus Interrupt**

(2) DETAILED DESCRIPTION OF EXTERNAL INTERRUPT

This interrupt is triggered when incoming IR data is detected. The scheme used for IR communications in MAGE was created by Bruce Lebold and will not be discussed in detail here.

(3) DETAILED DESCRIPTION OF MAIN LOOP

The main loop is what controls the headband. Upon power up the headband goes through an initialization routine. The next step checks to see if IR data or CANBus data has been collected. If neither of these conditions is met then the firmware will simply loop indefinitely checking for data and acting on that data.

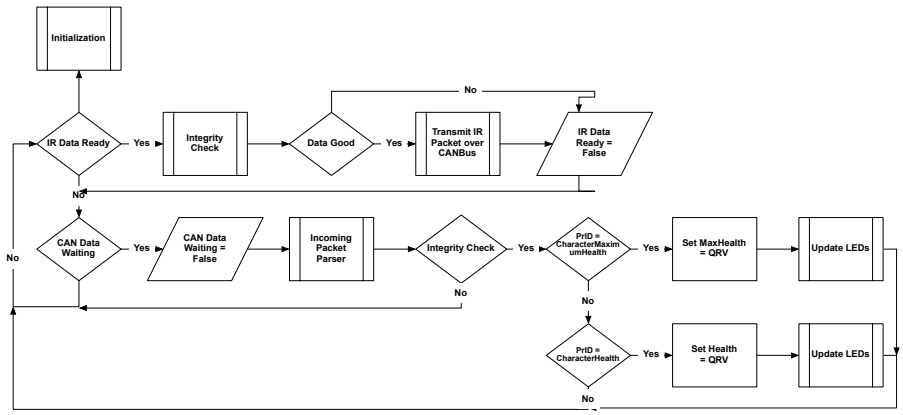


**Figure IV-XXIV: Headband Firmware Flowchart – Main Loop**

When IR data is ready a branch that handles the data is run. The first step in the branch is to check the received IR packet for data integrity. If the packet is not corrupted then a COM byte of 0x00 (IRData) is assigned. The COM Bytes tells the HHC and the HIU where the packet came from. Then the packet is transmitted over CANBus. If the packet is corrupted then the branch progresses to the last step, which is to the reset the IR Data Ready flag.

If CANBus data is waiting, then another branch that handles data runs. The first step is to reset the CAN Data Waiting flag. Then the packet is parsed using a modular portable C library. Next, an integrity check is run on the packet. If the packet contains the Players Maximum Health or the Character Health then the respective variables are set and the LEDs are updated and the branch ends. If the packet does not contain Health data then the branch also ends.

## 2. WEAPON

The weapon is another MAGE peripheral. It is the only way that IR and RFID packets can be transmitted to the other players in the game. The other player's Headbands receive the IR transmissions and the RFID transmissions are received by the other player's HIU. The weapon receives packets from the HIU over CANBus and relays those over either IR or RFID.


**Figure IV-XXV: Weapon Block Diagram - Level 0**

The weapon is controlled from the GUI on the HHC. When a player selects a spell to be cast, that spell (IR data packets) is easily transmitted using the IR transmitter block. When a player chooses a different melee weapon, such as a sword, to be equipped the RFID must be reprogrammed. This is somewhat inconvenient, as the player would have to reprogram the RFID tag manually. An RFID tag that can be reprogrammed without a traditional RFID reader/write system could be extremely useful here.

### *a) DETAILED HARDWARE DESCRIPTION*

Like the headband, the weapon is constructed of blocks. The block approach makes the design simpler and allows us to reuse blocks from other designs. The weapon has only one standard MAGE CANBus connection that includes power. The specifications for the CANBus and Power are inherited from the HIU. Any peripheral connected to the HIU receives power at 5V with a current limit of 100mA. This limitation drives the power usage requirements of the Weapon. CANBus is used to communicate with the HIU and the interface protocols are defined by Industry Standards. The most important function of the weapon is the transmission of IR and RFID data. The IR protocol used is custom and resembles that used by remote controls. It

operates on a wavelength of 940nm with a carrier frequency of 56 kHz. The RFID part of the system uses a standard 134 kHz RFID protocol.

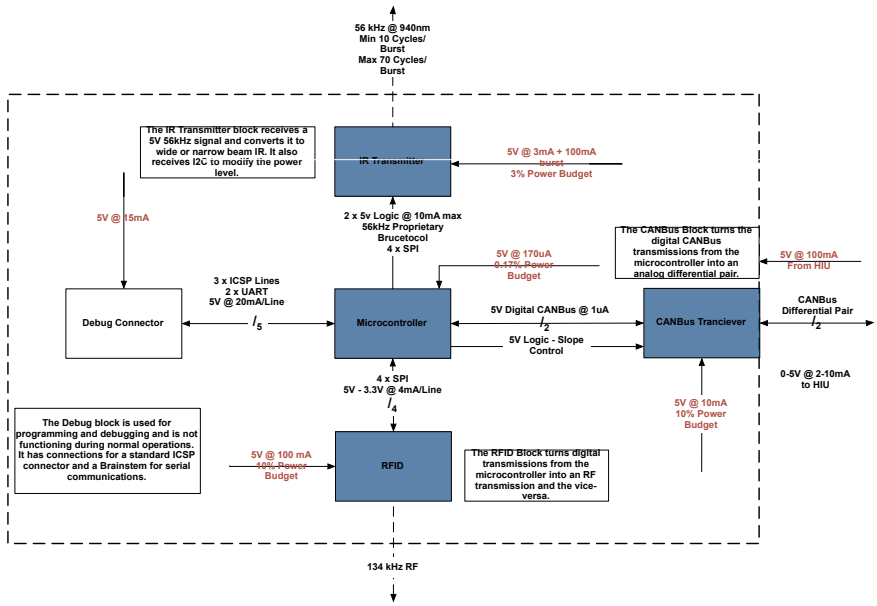


**Figure IV-XXVI: Weapon Hardware Block Diagram**

(1) DETAILED DESCRIPTION OF MICROCONTROLLER BLOCK

The microcontroller block is at the center of the Weapon. It is the brains of the device, handling the interfaces between other weapon blocks. The microcontroller block contains connections for all other blocks, including connections for a debug interface.

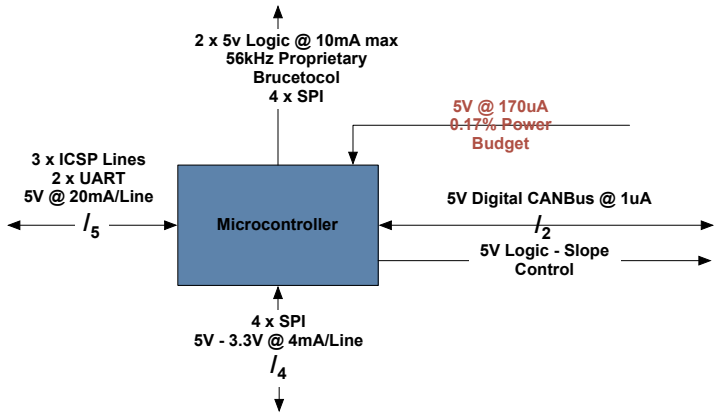


**Figure IV-XXVII: Weapon Hardware Block Diagram – Microcontroller**

The debug interface contains pins for the microcontroller's in-circuit serial programming (ICSP) and pins for a serial port. The connections for the IR transmitter include 4 SPI and two logic pins. The CANBus connections are standard Digital CANBus and one logic line for slope control. Connections to the RFID block are 4 SPI lines. The RFID block used in the weapon is different than that of the headband.

(2) DETAILED DESCRIPTION OF IR BLOCK

The transmission of IR signals is one of the primary purposes of the weapon. The IR transmitter block (see Figure IV-XXVIII) provides the means to transmit wide or narrow angle IR data. The block accepts SPI, which is used to change the power level and data pins, which are used to select the narrow or wide IR LEDs.

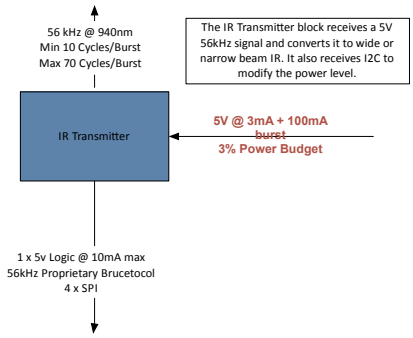


**Figure IV-XXVIII: Weapon Hardware Block Diagram - IR Block**

The IR transmitter block can be implemented many ways. Several designs have been tried. I will describe the simplest and most robust design created to date. The IR transmitter block is required to generate a 56 kHz modulated signal with adjustable current. The simplest way to digitally adjust current is with a BJT connected to a digital potentiometer. If that adjusted current is then fed into a FET switch that is used to generate the 56 kHz signal. The required data stream can be created.

(3) DETAILED DESCRIPTION OF THE RFID BLOCK

The main goal of the RFID block is to implement a melee type weapon. This is done through the use of RFID. The RFID block acts not as an RFID transmitter per say, but as a reprogrammable RFID tag. The block has one SPI interface to receive data from the microcontroller and it sends out standard 124 kHz RFID data.

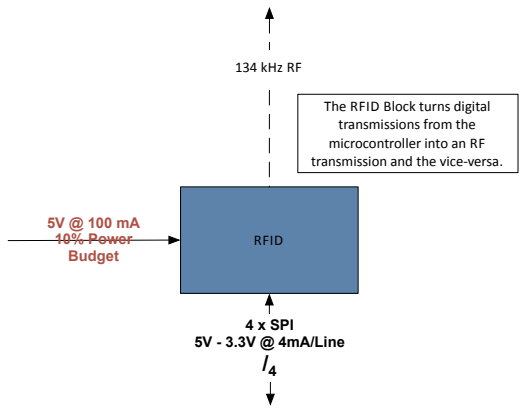


**Figure IV-XXIX: Weapon Hardware Block Diagram - RFID Block**

The RFID block has been implemented in several ways. The standard but complicated way is to use an RFID tag and a programmer for the tag. This takes up lots of space and is not very efficient. RFID emulators have also been used but these require much analog design. The TMS37157 from TI is reprogrammable RFID transponder with an SPI interface. Essentially it is an RFID tag that is capable of being reprogrammed over an SPI interface. This is the suggested hardware implementation as it is much simpler and less costly that the other options.

(4) DETAILED DESCRIPTION OF OTHER BLOCKS

One of the benefits of using modular block based design is that many blocks can be reused. This allows designers to design hardware at the system level instead of the component level. This type of system level design can make the process of creating a prototype much faster. A description of the CANBus Transceiver block can be found in section B.a)(1). A short description of the debug block can be found at the end of section 1.a)(3).

### *b) DETAILED FIRMWARE DESCRIPTION*

Just like the Headband, the Weapon is controlled by firmware that is stored in the microcontroller's flash memory. The processes of the Weapon are driven by incoming CANBus

packets. MAGE Packets (see Figure IV-XXII) are collected from the CANBus. These packets contain the information to be transmitted and tell the Weapon what method to use to transmit the packet.

(1) DETAILED DESCRIPTION OF CANBUS INTERRUPT

When CANBus data is received, an interrupt is triggered. The interrupt then collects the CANBus data from the buffer. If the CANBus packet is not corrupted then the global flag CAN Data Waiting is set and the interrupt ends.

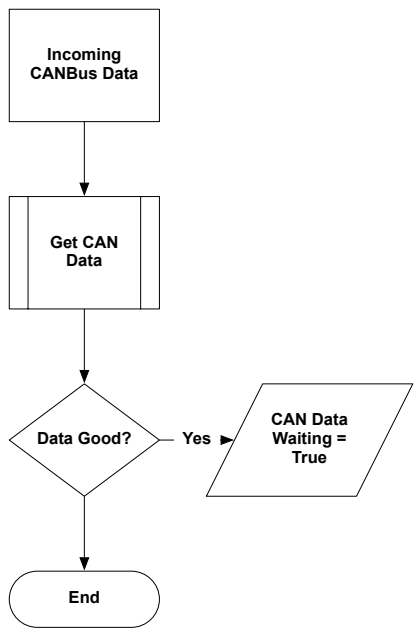


**Figure IV-XXX: Weapon Firmware Flowchart – CANBus Interrupt**

(2) DETAILED DESCRIPTION OF MAIN LOOP

The main loop of the Weapon firmware is very simple. The first step is initialization of the Weapon Hardware. From there the firmware waits for data. Once data is received the new packet is parsed. The COM ID is then used to determine where the packet should be transmitted. If the

COM ID is 0x08 the packet is transmitted over Narrow Beam IR. If the COM ID is 0x09 the packet is transmitted over wide beam IR. If the COM ID is 0x0A, the RFID tag is reprogrammed with the new packet. If the packet does not match any of those COM IDs, the packet is discarded. After the packet is transmitted the firmware continues waiting for new CANBus data.

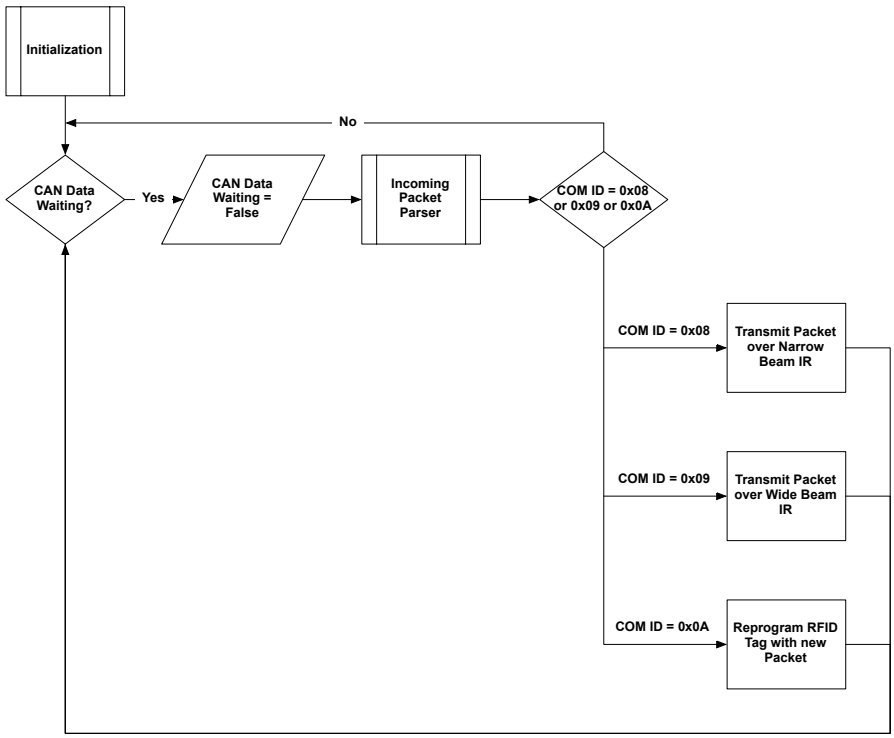


**Figure IV-XXXI: Weapon Firmware Flowchart - Main Loop**

Hopefully these paragraphs have helped you to understand the design and operation of MAGE. The hardware and firmware that make up these systems are the core components of MAGE. Without them MAGE could not function. It has been suggested that we expand these systems into a platform that would allow us to create more than just games. The prospect of a new peripheral platform for mobile devices is discussed more in the conclusion. However, if MAGE is going to

work we need some logic that implements the rules of the game. This need brings us to the last component of MAGE, a software system that runs on the HHC. We call this software the Game Code.

# CHAPTER V

## GAME CODE

In the beginning the game code for MAGE was like many of the other systems very simple, lacking features, and monolithic. The game code inherits its' essence from D&D. The original code ran on the first HIU and it was a single piece of firmware code. Everything was written in C for the PIC microcontroller platform. As we discussed earlier the original communication structure was custom and not based on any architecture. It would often be changed from semester to semester to fit the needs of the current projects. This was obviously not sustainable. In Mage v2.x the game code evolved slightly and began to become more stable. Communications protocols were defined (Mage Packet) and Standards adopted (CANBus), but we were still working with a piece of monolithic code that ran on a microcontroller. That made feature expansion, adaptation (to support specific needs), and addition of game items difficult. We eventually moved to an SD card based memory system for the microcontroller so we could support images and sound files, we never got the SD card to work correctly. Our microcontroller was trying to drive OLEDs, process packets, and execute game code. Keep in mind everything is one large code base with very little independent structure. One change to fix a problem with the SD card or OLED and everything would stop working. It was clear that we were asking too much. We decided we needed a new more powerful system. We chose embedded Linux running on an ARM processor. Suddenly we had tons of memory, built in display routines, databases and more room to expand than we knew what to do with. All of our MAGE game code was written for the PIC microcontroller. We needed to adapt our game code for our new Linux platform.

Instead of starting over from scratch we tried to port our code to our new platform. This kind of worked, but by now we could see a clear need for more structure emerging. The rate at which we wanted to expand and use new features was much greater that what we could implement with a single piece of code. It was time for a new system. We started over, a ground up rebuild with a completely different architecture. Subroutines and structures were salvaged, but almost no original code would remain. We kept our packet structure and some of our other low level interfaces to the microcontrollers that had become well developed. We would now adopt a modern game structure consisting of two parts. A game engine that would handle the work of interfacing to various subsystems, Audio, Display, User Interface, Peripheral Communication, and access the database was the first part. The other piece of code would handle the actual game. It would implement the rules, handle items and not need to be bothered with anything else. This is how most modern computer games work[29]. Using this architecture gave us all the flexibility that we needed and proved to be stable and easy add or remove features. Our flow charts started to make more sense and were easier to follow. Everything was working great. Our new system is the one that will be explained here and functions very well.

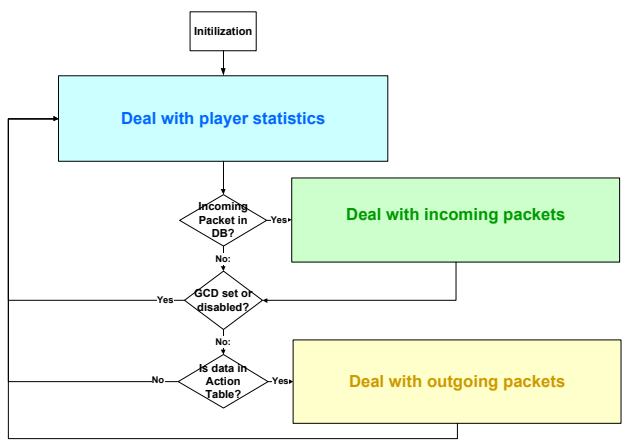


**Figure V-I: Game Code Software Flow Chart**

The Game code is written in C and consists of three major sections or subroutines shown in Figure V-I. Each subroutine has a specific set of tasks. The “Deal with Player Statistics” subroutine handles all the updating and broadcasting of player statistics. The Deal with incoming packets subroutine runs a given subroutine depending on the contents of the packet. Outgoing packets are all handled by the Deal with outgoing packets subroutine. Other minor subroutines and logic checks will be discussed as they come up in the discussion of each subroutine. The Game Code relies on a database to store all of the many statistics and parameters that it need to operate. The database type was considered carefully and SQLite was chosen for its small size, portability, speed and not requiring running a SQL server. SQLite can be used on any system with a C compiler, and many platforms have built-in support for SQLite databases.

The following paragraphs describe in detail the Game Code. It is important to understand that I was not the only person developing the Game Code. James Duvall, Alan Cheville, Bruce Lebold and I developed the overall structure of the Game Code and wrote the flow charts. James Duvall wrote all of the “Deal with Incoming Packets Subroutines” and developed the actual database. I wrote the remaining parts of the Game Code. The Game Code was a collaborative effort.

(1) DETAILED DESCRIPTION OF INITIALIZATION

The initialization subroutine needs to accomplish several single run tasks. First, any system dependent tasks will be taken care of in the Entry subroutine. Second, the SQLite Database will need to be opened. Then, the statistics stack will need to be loaded from the database into a statistics stack structure. The last task that the initialization subroutine must attend is transmitting the statistics or “Shout!” the statistics to all of the MAGE peripherals. This is done so that any initial data needed by the peripherals is given.

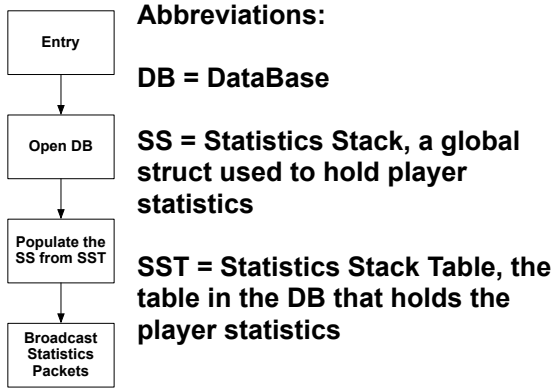


**Figure V-II: HHC Software Flowchart - Initialization**

The entry subroutine takes care of any platform dependent initialization. Things like parsing command line parameters and opening serial ports are taken care of here. The Open Database subroutine is where the database is opened. This could also be a platform dependent subroutine as many platforms have a built in interface for SQLite and do not use the standard SQLite calls. QT and iOS are examples of such platforms. The MAGE Game Code includes a set of structures for each table in the database. Since the statistics stack is just a database table it is a simple matter to load all the entries from the database into a structure. The precise coding of the database call will differ from platform to platform but that is to be expected. The last task of the Initialization subroutine is to "Shout!" all of the statistics in the statistics stack to the player's peripherals. This is done to ensure that each device has all necessary data to begin the game. An example would be the Headband. It needs two statistics to correctly display the player's current health status. The headband will recognize and record the statistics and then update its LEDs.

(2) DETAILED DESCRIPTION OF DEAL WITH PLAYER STATISTICS

The Deal with Player Statistics subroutine is responsible for making sure that the database is updated with the players current statistics as well as keeping the MAGE peripherals up to data. This subroutine is executed every time the program loops.

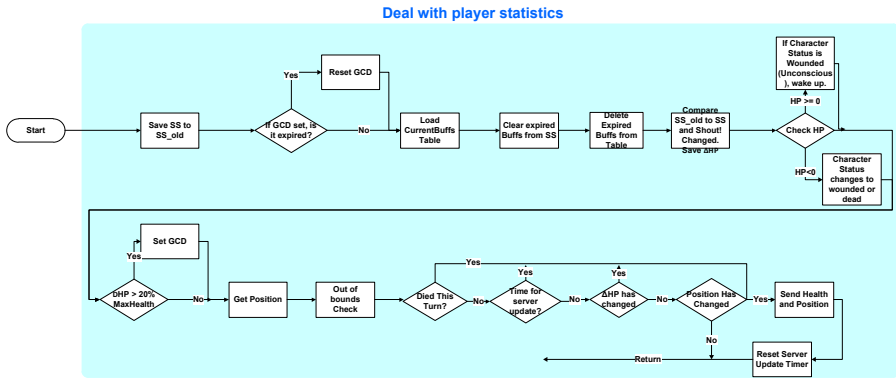


**Figure V-III: HHC Software Flow Chart - Deal with Player Statistics**

When the subroutine runs the first thing that is done is to save a copy of the statistics stack, as it exists when the subroutine is executed. The Game Code implements a global cool down so that players cannot simply cast spells over and over. This time can be adjusted but would generally be in the range of a few seconds. The GCD is checked for expiration and reset to inactive if it is expired during the Deal with the Player Statistics subroutine. The next step is to check for expired buffs or special enhancements to a player's statistics. If any expired buff is found it is removed from the statistics stack. The modified statistics stack can now be compared to the original statistics stack that we saved at the beginning. If any changes are found, they are broadcast to the MAGE peripherals. The next step is to begin a series of checks on the change in the player's health, the change in the player's position, and the time since the last server update. Each of these conditions could trigger an outgoing packet that will be transmitted to the server for display.

(3) DETAILED DESCRIPTION OF DEAL WITH INCOMING PACKETS

The incoming packets subroutine is where all the packet handling happens. Figure V-IV shows all the planned subroutines, while Figure V-V shows only a few. When a packet is received it is processed by this subroutine, depending on the contents of the packet, different actions will be taken.

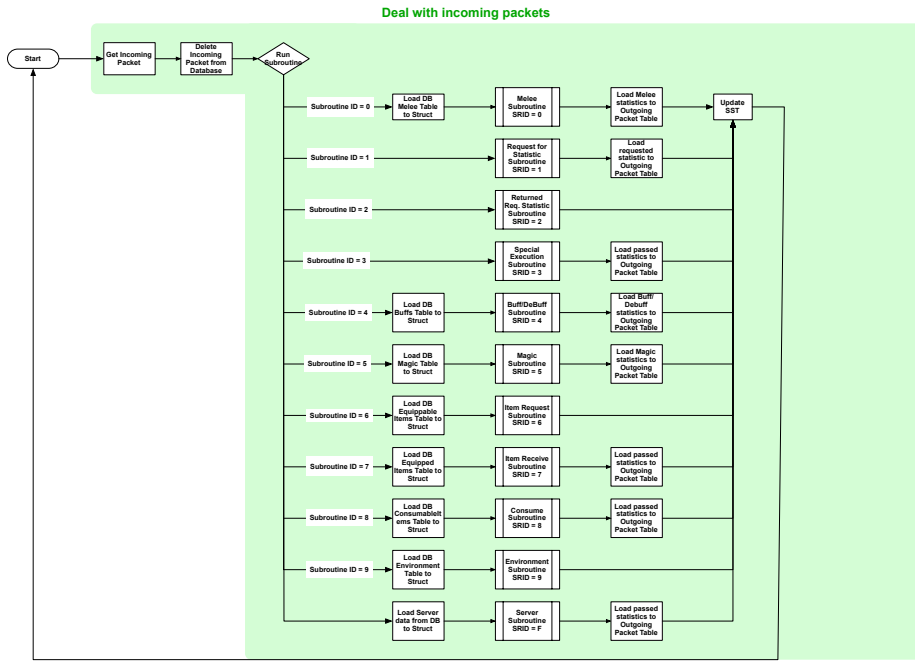


**Figure V-IV: HHC Software Flow Chart - Deal with Incoming Packets**

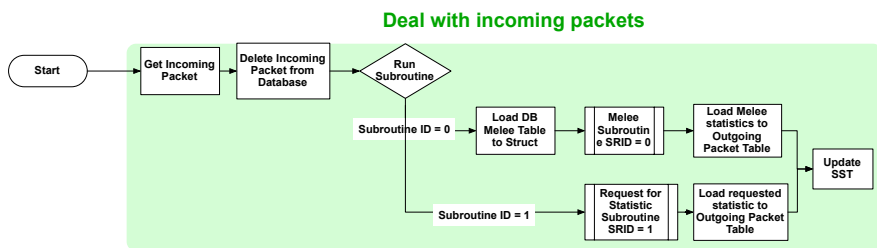


**Figure V-V: HHC Software Flow Chart - Deal with Incoming Packets - Short**

Packets can come from either the serial port or the Actions Table in the database and will be treated the same after they have been retrieved. When a packet is received, the Subroutine ID is extracted and used as a descriptor telling the Game Code what subroutine should handle that particular packet. We will use the Melee subroutine as an example. The incoming packet will

have a Subroutine ID of zero and the Process ID we will set to 5. When the packet is received the Subroutine ID will be recognized and Melee Packet processing will begin. First, the Melee Table entry number 5 (based on the value of the Process ID) will be loaded from the database into a structure. Then, the Melee Subroutine will be executed and the Melee Table Entry and the Incoming packet will be passed to the Melee Subroutine. When the subroutine has finished, the statistics stack will have been updated and a new "Shout!" packet will need to be sent. The packet is then transmitted over the serial port. The Deal with Incoming Packets subroutine is now complete and it will return to continue the rest of the program.

(4) DETAILED DESCRIPTION OF DEAL WITH OUTGOING PACKETS

In order to keep the user from having to have an in-depth knowledge of the Game Code and modifying it too much, the Actions Table was created as a sort of abstracted code interface. This table gives the player a way to construct a GUI without having to modify any Game Code at all. They can simply place an entry in the Actions Table and it will magically work its way through the system.

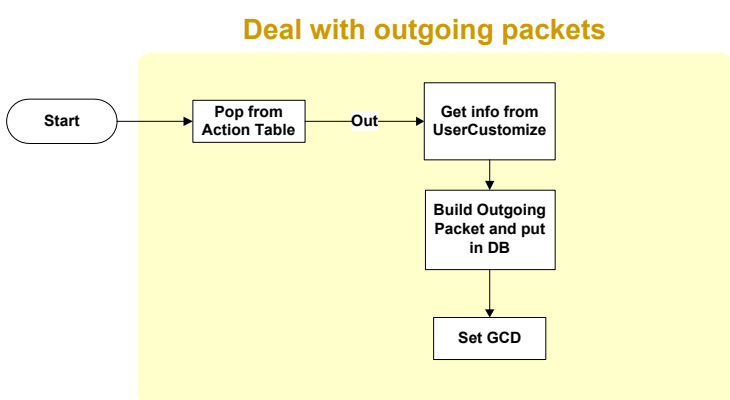


**Figure V-VI: HHC Software Flow Chart - Deal with Outgoing Packets**

If there is no Global Cool Down set, then Outgoing packets can be processed. Outgoing packets are selected from the Actions Table. Information about what COM Byte will be used is retrieved from the User Customize table. Once all the necessary information is retrieved an outgoing packet is created and transmitted. The GCD is set to ensure that no other action can be immediately taken and the subroutine exits to continue program execution.

(5) GAME PALY PACKET FLOW EXAMPLE

The Game Code runs in a loop constantly looking for incoming and outgoing packets. When a player receives a packet it is processed by the Deal with Incoming Packets subroutine, the packet will be processed through the appropriate subroutine, the Melee subroutine for instance. Once the packet has been processed by the Melee subroutine the player's statistics are updated. The Game code then checks for outgoing packets. If outgoing packets are found they will be processed through the Outgoing Packets Subroutine. Outgoing packets are transmitted from the HHC to the HIU and out to enemy players. The process then begins again with the enemy players reading in packets and passing them to the Deal with Incoming Packets subroutine. This basic process runs on all players all the time. There are other subroutines and user interfaces that facilitate the flow of these packets. However, the most fundamental repeating process of packet reception, change in statistics, then packet transmission never stops.

# CHAPTER VI

# THE END OR THE BEGINNING?

It should be clear that this paper was intended to be the story of a journey through the development of the MAGE System, from good idea to bad idea and back again. It was my intent to illustrate the evolution of a simple idea into a complex system. I have described the technological journey, but there was also personal one. During this project I grew and gained experience with design. I can now see new opportunities and realize our mistakes. I have gained a better vision of what a project could be. MAGE has gone through many cycles of change and once again, MAGE needs to be expanded to reach its potential. If implemented, with careful planning and purpose I believe we can turn MAGE into something revolutionary and having a great impact on the way we think about and interact with mobile systems. In these last paragraphs I will describe what we have learned and the new directions in research and development I believe are needed.

## A. WHAT WE LEARNED

While I was working on MAGE, I spent most of my time just trying to get things working in time to support the projects of students. Only at the end of each semester would we have time to review. We would invite all students to come into the lab after grades were posted to comment on the course. The comments from students helped shape the course and the MAGE project. In 2010, I stopped development work on MAGE. Since then I have had a lot of time to reflect on the growth of MAGE. I find myself asking what knowledge was gained from the development of MAGE? There are two categories these ideas fall into: education and technology.

What did MAGE teach about designing electrical engineering systems? This is perhaps the most important question. MAGE was designed to facilitate education, but more research is needed to determine if that goal was achieved. A comparison study between courses implementing MAGE, standard lecture courses, and open-ended project courses may tell us more about MAGE's impact on education. There are, however, many ways in which MAGE needs improvement. First of all, MAGE was not completed. This made actual competition impossible, as we always lacked the necessary hardware and software systems. Also, since MAGE was under continuous development, keeping documentation up to date was nearly impossible. Since most software projects are also continuously under development we may be able to adapt some techniques to help stabilize our documentation. Students often required data on how MAGE operated that didn't exist. In many cases the incompleteness of MAGE and its associated documentation made the students task of developing peripherals difficult. During the development of MAGE, the technology behind mobile computing platforms was progressing rapidly. When we began development there were no acceptable mobile computers available. A lot of time was spent trying to develop an acceptable way to integrate the features of computers into a wearable platform. Now, many advanced mobile platforms exist, development efforts can be shifted away from wearable computers and into other parts of MAGE. Once some of the technical and logistical problems are solved we can again approach the question of how MAGE supports the education of students.

As I studied the development phases of MAGE several needs became clear. First of all, a more structured hardware platform is needed to combat continuous iteration. Initially we underestimated the complexity of what we were creating. This led us to a design cycle that repeated continuously. Unfortunately, it was somewhat necessary as the design of the system was evolving constantly.

Another need we discovered is set of rigidly defined interfaces. As components of the MAGE system matured, a design freeze was made. This allowed proper documentation be created. Perhaps the most important interface was the packet structure used for all communication. Having this one piece of the system defined and frozen still works well with our iterative model, as these interfaces change at a much slower rate than other parts of the system. Having these interfaces allowed the students to design against something that would not change. It also gave us an interface with which to judge conformance. It quickly became clear that defined interfaces were a necessary part of system design.

One of our goals was to teach students to integrate their designs with existing designs. Having defined interfaces allowed everyone to easily integrate their design with existing MAGE systems. When new developers joined the team they usually wanted to redesign MAGE from the ground up as well. There were always differences of opinions about how MAGE's systems should work and how the interfaces should be designed. As it turned out, we spent just as much time trying to get TA and developers to work with existing systems as we did with students.

The need for a more structured software design became even more apparent as the MAGE system developed. Distinct structure in the software began to develop even before we thought it should be there. Having two platforms to write software for was a large part of the structure that emerged. The firmware on the microcontroller based devices contained two distinct parts. Low level interfaces from the microcontroller to hardware devices, such as an RFID reader and code that would implement the mage system. The same was true for the HHC, hardware drivers and an OS at the core of system functionality and the actual code that would implement the rules of the game. As I later learned, computer games have similar structures. Most games have a game engine that handles the operation of the virtual environment. Separately, they have a piece of software that handles the game rules. These two software components are integrated heavily, but are distinctly different. Having this type of structured system becomes somewhat necessary as the

complexity of the system increases. MAGE would have benefited greatly from having software design experts involved in development from the beginning. It could be very beneficial if software system design courses were incorporated in to the ECE curriculum.

Emerging from these needs are several distinct platforms. One is a set of hardware peripherals for mobile devices (PDA, Smart Phones, etc.). These peripherals will need an accompanying software platform to support their function and enable rapid development. A Game Engine to support the unique requirements of MAGE type games and finally the actual MAGE Game. These platforms are the future for MAGE.

## B. WHAT STEPS NEED TO BE TAKEN NEXT?

It is important to separate the development process of MAGE and development of the support ecosystem system in the classroom. We need a large amount of resources to support the development and operation of a Capstone Design course. However, the requirements are not insignificant. We needed training materials, facilities, and support personnel. All of these things were needed regardless of what project was used. Over the last few years we shaped the ecosystem of the design lab and MAGE was developed somewhat independently. Every design facility will have its own ecosystem that is shaped by the community of people that use the facility.

When we began, we were designing and building a system to accomplish an educational goal. I eventually realized that instead we needed to build a platform with educational goals in mind. I also realized we were creating an ecosystem of design resources. It was not enough to create the technology. We needed an entire ecosystem to make it successful. This means that it is likely that not every design project given will have the complete support ecosystem needed for the project to succeed. By adjusting the amount of support you give to students on a given technology, the difficulty level of the project can be adjusted. Given these issues, a strategic plan of the MAGE

ecosystem and our proposed design competition must be created before development can continue.

The first step in the creation of this ecosystem must be to consider the final goal. MAGE was originally developed with the hopes that it will one day become a national design competition. Students from different universities could first design and construct devices and then compete against each other in a friendly game. These devices will have to conform to specifications and integrate with existing systems. How well student teams do in the game would depend both on their game play abilities, but also how well they designed the game systems they will use.

As we discussed earlier, MAGE would benefit from being broken down into separate platforms: a mobile device peripheral platform and a game engine to support the unique requirements of a live action role-playing game. Each of these building blocks are complex and will need careful planning if they are to become a reality.

First, let us discuss the mobile device peripheral platform. The ecosystem for the peripherals should include the physical hardware, modular software, a community of developers, a support community, and of course a user group. We need to create modular blocks of technology that can be used for many applications. These peripherals must be easy to use, and a good way for developers and users to integrate them quickly into a new device must be provided. The development activities and operation of the peripherals must be intuitive. Users should be able to operate the device without special knowledge and almost like there is an inherent, natural knowledge. One example is the use of CANBus in our current system. It works well as a method of interconnection and is fairly user friendly. However, as a practical solution it does not work well at all. For a player to equip themselves wires connecting all the peripherals are needed. Therefore cables run all over a person. They end up looking and feeling more like a cybernetic life form rather than a player. The connections are then easily broken as a player moves around. If

we change CANBus to a wireless connection we eliminate many of those issues. Our solutions to problems should reduce complexity, not add to it. Imagine if creating a new blood glucose monitor was as simple as adding a simple sensor to a mobile phone and creating a simple piece of software to read out the data. That kind of quick, easy development is the purpose of the peripheral platform.

There are many open source and commercial games and game engines on the market today. Unfortunately, they do not meet the needs of a game like MAGE with the elements of MMORPGs and live action RPGs. Most game engines today are designed for 3D video games and do not include the elements needed for live action RPGs. Therefore, a new game platform is needed. As with the mobile device peripheral platform, an ecosystem needs to be created. A strong community of developers, testers, and users must be present.

As we discussed in Chapter V, most games today are separated into game engines and game code. This structure has a special advantage for a platform like MAGE. It allows new games to be quickly developed by only changing the game code and not having to rewire all the hardware interfaces. Also, unlike most games, the software that interfaces with MAGE peripherals could be very useful to other projects. The blood glucose meter is a good example. If the each MAGE peripheral has a module of code that is independent of the game, that module can easily be reused. Having these working modules of code can greatly decrease the time it takes to develop a new device. Games and game engines are complicated software systems. However, if time is taken to plan and structure the software into reusable, well-documented modules, we can benefit from simple and reusable code.

One of the principle ideas in MAGE is open source design. This allows anyone to be a user or a developer. Groups of users with knowledge of MAGE will be able to share their expertise freely. Hopefully this will enable a broad spectrum of people and programs to participate. A tiered

adoption scheme is planned that would decrease the start up cost and allow universities to buy some systems and assemble others depending on what tools and facilities they have available. Slowly, over time, a university could advance to the point where they would not need to purchase any systems or components and all of the fabrication could be done on site. If universities begin to adopt MAGE there will be various entrepreneurship opportunities available. Supplying the materials and components for universities with limited facilities could become quite a large business opportunity, particularly if the MAGE system can be developed into a national competition. Given the presence of games in the life of young people and the success of project based learning, I believe, the development of MAGE should be continued.

## REFERENCES


[1] J. McCormack, "Creative ecosystems," *Proceedings of the fourth international joint workshop on computational creativity, London, UK,* pp. 129-36, 2007.

[2] A. Lenhart, J. Kahne, E. Middaugh, A. R. Macgill, C. Evans, and J. Vitak, "Teens, Video Games, and Civics " Pew Internet & American Life Project, Washington, D.C.2008.

[3] M. Copier, "Connecting Worlds. Fantasy Role-Playing Games, Rituals Acts and the Magic Circle," presented at the Proc. 2005 DiGRA: Changing Views--Worlds in Play, Vancouver, 2005.

[4] M. D. Dickey, "Game design and learning: a conjectural analysis of how massively multiple online role-playing games (MMORPGs) foster intrinsic motivation," *Education Tech. Research Dev.,* vol. 55, pp. 253-273, 2007.

[5] c. Wikipedia. (10 July 2011 01:44 UTC). *Massively multiplayer online role-playing game*. Available: http://en.wikipedia.org/w/index.php?title=Massively_multiplayer_online_role-playing_game&oldid=436576619

[6] R. S. Dancey, "Adventure Game Industry Market Research Summary (RPGs) v1.0," February 07 2000.

[7] W. Realms. (2008, July). *Warcraft Realms Census* [Internet]. Available: http://www.warcraftrealms.com/census.php

[8] B. Entertainment. (2010, 10/07). *WORLD OF WARCRAFT® SUBSCRIBER BASE REACHES 12 MILLION WORLDWIDE*. Available: http://us.blizzard.com/en-us/company/press/pressreleases.html?101007

[9] N. S. Board, "Science and Engineering Indicators 2008," National Science Board, Washington, D.C.2008.

[10] J. Mazel. (2009, July 03). *The 50 Best Selling Console Games of this Decade (So Far)*. Available: http://www.vgchartz.com/article/3364/the-50-best-selling-console-games-of-this-decade-so-far/

[11] R. M. Ford and C. S. Coulston, *Design for Electrical and Computer Engineers*. New York: McGraw-Hill, 2007.

[12] J. A. Marin, J. E. Armstrong, Jr., and J. L. Kays, "Elements of an optimal capstone design experience," *Journal of Engineering Education,* vol. 88, Jan 1999.

[13] A. Cheville, "Designing Successful Design Projects," presented at the 117th ASEE Annual Conference & Exposition, Louisville, KY, 2010.

[14] D. R. Woods, *Problem-based learning : how to gain the most from PBL*, 2nd ed. Waterdown, Ont.: Donald R. Woods, 1997.

[15] B. L. McCombs, "Alternative perspectives for motivation," in *Developing Engaged Readers in School and Home Communities*, L. Baker, P. Afflerback, and D. Reinking, Eds., ed Mahwah, NJ: Erlbaum, 1996.

[16] P. R. Pintrich and D. Schunk, *Motivation in Education: Theory, Research, and Application*. Columbus, OH: Merrill Prentice-Hall, 1996.

[17] D. L. Beaudoin and D. F. Ollis, "A Product and Process Engineering Laboratory for Freshmen," *Journal of Engineering Education,* vol. 84, July 1995.

[18] J. C. Mankins. (1995, Technology Readiness Levels, A White Paper.

[19] G. Hustwit, "Objectified," ed, 2009.
[20] K. A. Smith, S. D. Sheppard, D. W. Johnson, and R. T. Johnson, "Pedagogies of Engagement: Classroom-Based Practices," *Journal of Engineering Education,* vol. 94, pp. 87-101, Jan 2005.
[21] R. Spencer. (2010, May). *Natcar Home*. Available: http://www.ece.ucdavis.edu/natcar/index.html
[22] ABET, "Criteria for Accrediting Engineering Programs," Engineering Accreditation Commision, Baltimore2008.
[23] c. Wikipedia. (10 July 2011 17:30 UTC). *Jedi*. Available: http://en.wikipedia.org/w/index.php?title=Jedi&oldid=438438212
[24] D. International. (2009, Jan 14). *XBee Documentation*. Available: http://ftp1.digi.com/support/documentation/90000982_B.pdf
[25] T. Instruments. (2011, Jan). *Transponders*. Available: http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?family=rfid&sectionId=475&tabId=2102&familyId=1352
[26] (2010). *FriendlyARM*. Available: http://www.friendlyarm.net/
[27] (2010). *Gumstix Products*. Available: http://www.gumstix.com/store/catalog/index.php
[28] Microchip, "High-Speed CAN Tranceiver MCP2551," 2003.
[29] M. McShaffry, *Game coding complete*, 3rd ed. Australia ; United States: Charles River Media/Course Technology Cengage Learning, 2009.

# APPENDIX A

## DESIGN OF ENGINEERING SYSTEMS AND PROJECTS

### A. FALL 2007

**Optical Transmitter Module** – Students built a modular IR transmitter that was able to provide 64 different brightness levels to two different IR LEDs: one wide angle and one narrow angle. This project was given in the hopes that it could be used as a component in future projects.

**Non-Player Character** - The NPC interacts with the outside world and the players in the game via RFID tags, IR modulated signals, and Passive Inferred motion detection. The NPC also contained an audio subsystem allowing it to play sounds, music, or recorded messages. The NPC could be programmed to be a monster that would attack people when in proximity or other functions like healing.

### B. SPRING 2008

**RFID Sword** - The sword implemented RFID technology to provide wireless melee combat. The project had to implement an RFID tag programmer and an RFID tag jammer to prevent attackers from overriding the global cool down.

**Motion Activated Wand** – Similar to the current Wii motes, the device would take inputs in the form of movement of the wand. The wand would detect specific patterns and then transmit a preprogrammed spell over IR.

### C. FALL 2008

**GPS Tracking** – This project required students to create a GPS module that could interface with the HIU over CANBus. The data would then be relayed back to a server over a Zigbee Mesh Network and the player's position and statistics would be displayed in Google Maps.

**Mote Swarm** – This project was a second attempt at an NPC. The team chose to use a distributed, modular concept. Each "Mote" was a small device that communicated with a central controlling unit. There were several different motes, each with a different function. One was an Audio/Visual mote that could make noise and emit light. The IR Mote was capable of directional IR attacks. The team also built area attack Mote that used Zigbee and would affect everyone nearby.

### D. SPRING 2009

**Touch Input Wrist Band** – This project was our first attempt at a touch screen interface. Packets are sent and received over CANBus that contain player information, screen layouts, and icons. Spells can be cast through the touch screen and the player's current health, as well as other statistics, can be seen on the screen. The screen used was a 1.5" x 1.5" OLED display.

**Wand** – This was the second attempt at a motion-activated wand. This version included the accelerometer for input of spells and a modified version of the IR transmitter module. The module was capable of powering a narrow angle and wide angle LED. The output of the LED could be adjusted to multiple brightness levels.

### E. FALL 2009

**Holy Hand Grenade of Antioch (HHGA) -** The HHGA is a virtual stand-alone weapon. It emits an IR data packet omni-directionally, which is received by a player's headband. When the virtual "explosion" occurs the HHGA emits bright flashes of light and sounds a buzzer to notify

neighboring players of the discharge. The Grenade can be programmed with any standard packet making it either capable of delivering damage or healing players within its radius.

**Touch Screen Interface** – This attempt at a touch screen interface was built on an ARM 9 development platform. The OS was Unix based and the GUI was built in Qt. This was also the first attempt at interfacing external electronics with a higher-level OS and more advanced processors. Like the previous Touch Input Wrist Band, this interface was also capable of displaying and selecting spells as well as displaying the player's current health and other statistics. The display was a 3.5" LCD, a great improvement in size and usability over the OLED used in previous designs.

### F. SPRING 2010

**Sanctuary** - The Sanctuary is designed to operate as a safe zone. When a player comes within range of the Sanctuary, he or she becomes immune to attack, and his or her wearable computer is enabled to communicate with the server in order to buy and store weapons and potions. When in Sanctuary, players can change equipment and their health continuously regenerates. The unit also serves as a data relay for the entire gaming area by collecting player GPS coordinates and other information through a Zigbee network and transmitting them to the server.

**RFID Target** – The RFID target was a large area RFID antenna. The antenna had to read tags that were at least 3 inches away and over an area of 6'x6'. In addition to the target, there was also a display component. The display would let the attacker know if they were successful in damaging their opponent. Different patterns could be programmed for a hit, miss, or critical hit.

VITA

Steven David Welch

Candidate for the Degree of

Master of Science

Thesis: ROLE-PLAYING GAMES AS COOPERATIVE CAPSTONE DESIGN PROJECTS

Major Field: Electrical Engineering

Education: Completed the requirements for the Master of Science in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma in May 2010.

Completed the requirements for the Bachelor of Science in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma in July 2008.

Name: Steven Welch Date of Degree: May, 2012

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: ROLE PLAYING GAMES AS COOPERATIVE CAPSTONE DESIGN PROJECTS

Pages in Study: 097 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: The purpose of this study was the design and construction of a new capstone design project for electrical engineering students based on games. Games provide context and a common interest for our students. Many characteristics of good design projects can easily be found in the structure of games. In a fundamental way games represent design. A design for a wearable gaming platform, called MAGE, was created. Several prototypes of the system were designed, built, and used by students in a capstone design course.

Findings and Conclusions: During the course of development many lessons were learned. Students struggled with projects based on MAGE because the system was incomplete. Continuous development meant that creating and maintaining up to date documentation was nearly impossible. There is positive anecdotal feedback in the form of high project success rates, which suggest MAGE, supports the education of students. However, more research is needed to make any definitive determination.

Through the course of development, the need for several new distinct platforms emerged. One is a set of hardware peripherals for mobile devices (PDA, Smart Phones, etc.). These peripherals will need an accompanying software platform to support their function and enable rapid development. A Game Engine to support the unique requirements of MAGE type will also be needed. More development is needed to fully realize the potential of using games as projects design courses.

ADVISER'S APPROVAL: Alan Cheville