

Bringing J2ME Industry Practices into the Undergraduate
Classroom
(Auburn OK Program)

Except where reference is made to the work of others, the work described in this
project report is my own or was done in collaboration with my advisory
committee.

Nischita R. Meda

David A. Umphress, Chair

Associate Professor

CSSE Dept

James H. Cross II

Professor & Chair

CSSE Dept

W. Homer Carlisle

Associate Professor

CSSE Dept

This work was funded by NSF Grant DUE 0311339

Bringing J2ME Industry Practices into the Undergraduate
Classroom
(Auburn OK Program)

Nischita R. Meda

A project submitted to
the Graduate Faculty of
Auburn University in
Partial Fulfillment of the
Requirements for the
Degree of Master of
Software Engineering

Auburn, Alabama

April 23, 2004

Bringing J2ME Industry Practices into the Undergraduate
Classroom
(Auburn OK Program)

Nischita R. Meda

Permission is granted to Auburn University to make copies of this project report at its discretion, upon the request of individuals or institutions and at their expense. The author reserves all publication rights.

Signature of Author

Date

Copy sent to:

<u>Name</u>	<u>Date</u>
-------------	-------------

Abstract

This project aims to improve the state of education for wireless engineering courses at Auburn University by adapting wireless industry practices in the classroom. After reviewing the industry best practices for mobile phone application certifications, Nokia's OK Java 2 Platform Micro Edition (J2ME) application certification process was chosen for the quality assurance program for a core set of wireless software engineering courses.

There are two main parts of the project. The first part is the Auburn University J2ME Development Guide which enumerates the standards to be followed for Auburn University wireless application certification. It also includes some guidelines for efficient MIDP programming. The second part is the mobile phone game Castle Quest which acts as a model to illustrate the use of the guidelines and the standards outlined in the guide.

The wireless industry standards were successfully adapted and incorporated into the Auburn University J2ME guide and their use explained with the help of the example code of Castle Quest.

TABLE OF CONTENTS

1	BRINGING J2ME INDUSTRY PRACTICES INTO UNDERGRADUATE CLASSROOM	7
	1.1 Introduction.	7
	1.2 Why Cell Phones.	8
	1.3 Why J2ME.	9
	1.4 Why Standardize.	9
	1.5 Organization of this report.	11
2	LITERATURE SURVEY	13
	2.1 J2ME.	13
	2.2 Industry Practices.	16
	2.3 Nokia OK.	18
	2.4 Academic Practices.	21
3	AUBURN OK	22
	3.1 Auburn J2ME Development Guide.	22
	3.2 Castle Quest.	22
4	CASTLE QUEST DESIGN SPECIFICATIONS	24
	4.1 Introduction.	24
	4.2 Statement of Scope.	27
	4.3 Class Diagram.	31
	4.4 Sequence Diagrams.	32
	4.5 Deployment Diagram.	35
	4.6 Design Affected by Standards?	35
5	CONCLUSION	40
	5.1 Observations.	40
	5.2 Future Work.	43
	REFERENCES	45
	APPENDICES	48

A	APPENDIX A	49
	A.1 Auburn University J2ME Development Guide.	50
B	APPENDIX B	69
	B.1 Source Code.	69
C	APPENDIX C	70
	C.1 NSF Proposal.	70

Bringing J2ME Industry Practices into the Undergraduate Classroom

1.1 Introduction

The work described in this report is part of an NSF-funded project (NSF 2003) to help advance the state of education for wireless system software development. It aims to adapt and implement wireless industry best practices in the classroom. Auburn University initiated the nation's first undergraduate wireless engineering degree program in August 2002. This project targets the bulk of the courses comprising the software engineering track of the wireless engineering degree.

The following goals were set for the NSF-project so that the students will experience and understand the situations themselves:

- To adapt industry application standards for classroom use as examples of what is expected of industry-quality software.
- To incorporate industry software development tools into a student-friendly software development environment.
- To implement in-house certification procedures as a basis for assessing student projects and providing feedback.[Umphress, Cross, Jain, Meda, Barowski, 2004]

After carefully considering the available wireless industry practices, Motorola [Motorola 2003], Nokia [Nokia 2003a], and Ericsson [Sony Ericsson 2003] were short listed for their certification programs. Of the three, Nokia OK's Java 2 Platform Micro Edition (J2ME) application certification process was selected for adaptation into the software engineering courses [NSF 2003].

A part of the project solution, 'Auburn University J2ME Development Guide' addresses the first issue of adapting Nokia OK application standards for classroom use by helping the students understand real life scenarios, developmental practices, and the standard of the industry software.

In addition to this, an application was also developed which implements the guidelines and conforms to all the standards listed in the Development Guide. This application is a game called Castle Quest written in J2ME for mobile phone devices. It will act as an example application for the students to understand the concepts outlined in the Development Guide as well as different issues of developing applications for small devices.

1.2 Why Cell Phones

Close to a billion cell phones are in use today around the world and the number is expected to grow. This proves the pervasiveness of the medium and so is a

natural choice to be included into the wireless curriculum. Also, the constrained computing environment decreases the size and scope of the applications targeting these devices and will let each student to be a part of the complete development lifecycle in contrast to software for conventional hardware.

1.3 Why J2ME

Sun's Java 2 Platform Micro Edition is a form of the Java language that is optimized for small devices such as mobile phones and PDAs. It vastly improves the ability of mobile phones to support applications. It allows far better control over the interface than SMS or WAP with its cross-platform capabilities. It is an appealing choice for applications running on a wide variety of cellular phones and operating systems [Nokia 2002b].

Most phone manufacturers have made a strong commitment to Java phone deployment. Tens of millions of Java-enabled phones are already in consumers' hands with the ability to add capabilities to their phones without having to purchase new ones [Umphress et al., 2004]. Although J2ME is not the only language deployed on phones, it is an industry standard backed by many manufacturers and therefore offers a large and growing installed base.

1.4 Why Standardize

Wireless device manufacturers are turning to defined software development, user interface, and quality assurance standards to control the quality of the software being deployed onto their handsets. This aspect of software development practices is to be included in the curriculum to give the students a rounded exposure to real-world practices [Umphress et al., 2004].

The industry attempts to have standardized applications - application certification programs, show that there is a failure rate of 80% to 90% on the first certification test attempt of a J2ME application [Grott 2004]. This shows the difference in the standard of certified and non-certified software. In industry, certification guarantees the vendors and the users that the applications meet preset requirements. In an academic setting, we can use such certification to:

- make the students aware of the kind of quality of the software that they will be required to write once they join a company
- let the students get used to quality control
- improve the academic software standard in emerging technology like J2ME and create a baseline for wireless software engineering education
- let the students get used to the technology that they are likely to encounter in their first years of work

- encourage them to induce quality assurance practices into their software development process, rather than viewing quality assurance as an outside policing factor

1.5 Organization of this Report

The organization of this report into five chapters is as follows:

Chapter 1 gives an outline of the issue being addressed i.e. injecting J2ME standards into the undergraduate classroom. It also describes the rationale for choosing cellular phones, J2ME, and standardization as the building blocks to prepare the students for the foray into the industry practices.

Chapter 2 provides supporting background information and attempts to justify the work done here and also how the students are going to benefit from it. It includes literature surveys in the areas of current wireless practices in academics, available wireless industry practices, and the expected outcomes.

Chapter 3 explains the approach taken towards arriving at a solution to the issue addressed in the first chapter. This includes the principles behind the synthesis of ‘Auburn University J2ME Development Guide’ and also the development of ‘Castle Quest’ game for mobile phones. It also includes a brief description of the technology used.

Chapter 4 consists of the design document for the game in question and also describes how it conforms to the standards in the Guide.

Finally, **Chapter 5** concludes by citing some observations of problems encountered, scope for improvements and outcomes of the project.

The **Appendix** includes the Auburn University J2ME Development Guide document, source code for Castle Quest game and the Project Proposal document submitted to NSF.

Literature Survey

MIDP 1.0 on CLDC devices is used in this project. This technology and the terms are explained in the following sections. Further down, the industry practices are described and the choice of Nokia's OK program is justified followed by the current academic practices in this area.

2.1 J2ME

J2ME is a reduced version of the Java API and Java Virtual Machine that is designed to operate within the sparse resources available in the new breed of embedded computers and microcomputers [Keogh 2003]. It is a natural choice for mobile application development. J2ME is mainly targeted towards small computing devices such as screen phones, set-top boxes, cell phones, PDAs, etc.

To support such a wide variety of devices, J2ME is organized into configurations and profiles. Both profiles and configurations define a set of Java API classes which can be used by the applications.

2.1.1 Configuration

A configuration is the Java run-time environment and the core classes that operate on each device. There are two configurations: Connected Limited Device

Configuration (CLDC) for handheld devices and Connected Device Configuration (CDC) for plug-in devices. CLDC 16-bit or 32-bit devices, such as cell phones, have between 128KB and 512KB of available memory and are battery powered. These devices use KJava Virtual Machine (KVM) which is a stripped down version of JVM.

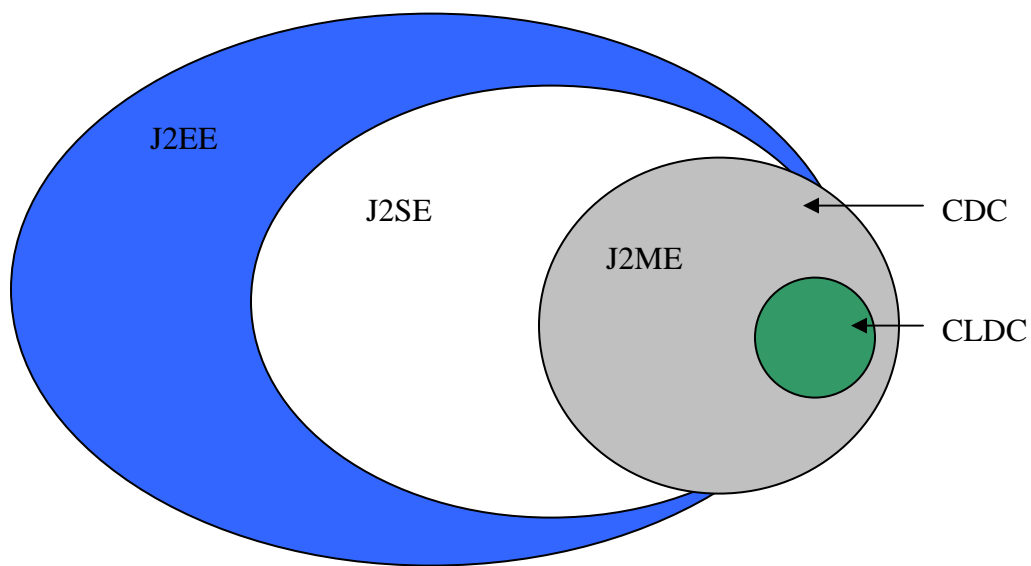


Fig.2.1 Java Editions and Configurations [Barr 2002]

Fig.2.1 illustrates the relationship between CLDC, CDC and the Java editions of Java 2 Platform Enterprise Edition and Java 2 Platform Standard Edition. As shown in the figure, the major functionality in CLDC and CDC has been inherited from J2SE. So we can say that J2ME is basically a slimmed down version of J2SE. An obvious example of the functionality avoided from J2SE is that of the Abstract Window Toolkit - many mobile devices do not have the screen capabilities to provide user interface components such as overlapping

windows and drop-down menus [Muchow 2002]. In addition, CLDC and CDC introduce a number of features, not drawn from the J2SE or J2EE, designed specifically to fit the needs of small-footprint devices. An example of this would be the Generic Connection Framework included in the `javax.microedition.io` package that provides support to connection types [Ortiz 2003].

2.1.2 Profile

A profile is a set of Java APIs which, together with a configuration, provides a complete application runtime environment targeted at small computing devices, such as mobile phones and entry-level PDAs etc. Applications are written for a particular profile and are thus portable to any device that supports that profile. A device can support multiple profiles and profiles can be nested, i.e. profiles may include other profiles. There are seven profiles (Foundation profile, Game profile, Mobile Information Device profile, PDA profile, Personal profile, Personal Basis profile and RMI profile) that are available and they continue to evolve as the proliferation of small computing devices continues. Of these seven, Mobile Information Device Profile (MIDP) is the one supported by cell phones. It provides local storage, user interface, and networking capabilities.

Among the two versions of MIDP available at the time of this project, MIDP 1.0 is widely used. Very few MIDP 2.0 compatible cell phones are available now. MIDP 2.0 has more classes included in it and so provides more features such as

multimedia, scheduling and secure networking. But working on lesser number of classes as in MIDP 1.0 is more constraining and will help the students understand the features required by mobile phone applications and map them to the features offered by this profile. The students can appreciate the new features offered by MIDP 2.0 and also will have a better foundation because they are already familiar with MIDP 1.0.

MIDP assumes the minimum display size on the device as 96x54 pixels. It can take input via touch-screen, keypad, or keyboard. The minimum memory requirements are: 128 KB ROM, 8 KB EEPROM (for persistent data), and 32 KB RAM. The device must also support wireless networking [Barr 2002].

2.2 Industry Practices

There are about 136 J2ME enabled phone models in the market, manufactured by the 18 cellular handset manufacturers (Casio, Fujitsu, Hitachi, Kyocera, LG Electronics, Mitsubishi, Motorola, NEC, Nokia, Panasonic, RIM, Samsung, Sanyo, Sharp, Siemens, Sony Ericsson, Toshiba, TTPcom). Of these 136 phones, only 6 phones support MIDP 2.0 [Sun 2004a]. The number of phones supporting J2ME is expected to grow and by 2006, 80% of the phones are expected to support Java [Lawton 2002] and by the end of 2003, 250 million people are expected to have Java-enabled phones [Ericsson 2003].

At the time of this project there was no unified testing process available in the industry to test and certify J2ME applications, although JavaVerified program was announced in June 2003 by the Unified Testing Initiative group members Motorola, Nokia, Siemens, Sony Ericsson and Sun Microsystems [JavaVerified 2004]. Some of the major handset manufacturers such as Nokia, Motorola, and Sony Ericsson implemented in-house Certification programs. Nokia's OK program [Nokia 2003b] and Motorola's Application Certification program [Motorola 2003] are fairly equivalent, but Nokia provides automated testing tools which give it an edge. Also, Nokia's website (<http://forum.nokia.com/main>) provides a lot more documentation on development and certification.

Some non-handset organizations like cellmania.com and synclast.com also provide J2ME application testing. Synclast offers free certification for J2ME applications, which includes testing for download, installation and minimum user satisfaction [Synclast 2004]. Cellmania too offers certification and expert review of applications, but does not provide much information on the extensiveness or criteria of testing [Cellmania 2004].

After looking at the different testing processes available, Nokia's OK program was chosen for adaptation into the academic process as it is the most mature program and has a number of features that can be cultivated for classroom. We

discuss the contents provided by the website below, and also describe how they are organized.

2.3 Nokia OK

Nokia divides its handset models based on the screen size, memory size and the key layout into Series 30 [Nokia 2003c], Series 40 [Nokia 2003d], Series 60 [Nokia 2003e] and Series 90 [Nokia 2003f] phones. The Nokia OK process addresses these differences in the user interface style guides provided for the different series. These style guides give an overview of the UI components of each series and also how to use these components in an application running on different handsets. The key layout varies from a two-soft key UI in a Series 30 phone, two soft key, four-way navigation key UI in a Series 40 phone and a two soft key, five-way navigation key UI in a Series 60 phone to a QWERTY keyboard in a Series 80 phone.

The figure below shows the revised series in 2004 [Nokia 2004a]. We can note the wide variation between the user interface dimensions of the different series. A Series 40 phone can have a 128x128 pixel screen where as Series 60 phones have 176x208 pixel wide screens. From these different series and the issues addressed in these guides, students can understand how to build core functionality on different technologies and then optimize the applications for different target devices.

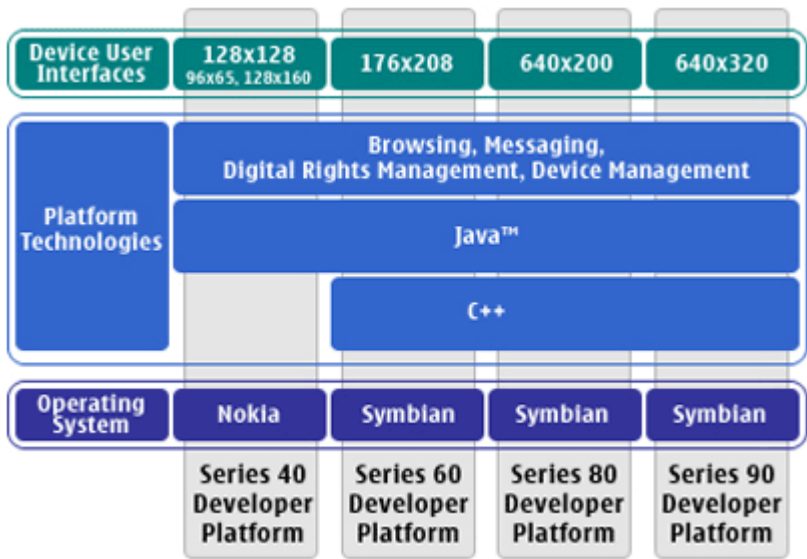


Fig 2. Nokia Handset Series and Technologies [Nokia 2004a]

The Nokia OK process also outlines the J2ME application requirements and standards [Nokia 2003g] which define the restrictions on application content, performance latency, usability, security, operation sequencing and installation. This can help the developers to test their applications before submitting for certification. It aids in constant attention to the quality of the application and therefore assures better products and consistent quality.

The Nokia website also provides APIs, emulators and test gremlins for handsets supporting J2ME [Nokia 2003h], [Nokia 2003i]. With the Nokia testing suite, developers can emulate user activities in the mobile phone with test scripts consisting of commands and key presses.

The certification process of Nokia OK lets the developers sign up and upload the application for testing [Nokia 2003j]. The Nokia OK process flowchart describes the process of reading the requirements and license agreement, submitting the application for certification, paying the testing house, and receiving the test report to see if the application was rejected or accepted [Nokia 2003k]. The page also has links to detailed descriptions of the license agreement, pre-testing, testing house information and the documents involved. Only a few months after the announcement of the Nokia OK program, some 130 applications had been submitted and 50 of them have been certified [Nokia 2003l]. The number of applications that failed the tests and also the number of retests for the certified applications indicates the success of the program. But the Nokia OK certification is not without cost. The average price for testing an application varies from \$2000 to \$4000. Thus, it is highly inaccessible to students. This is one more reason to come up with an academic certification program to let the students experience the rigor of industry practices.

Nokia also offers developer support in the form of discussion boards, professional support and training. These can help the students interact with professional developers, when included into academic wireless engineering teaching practices.

2.4 Academic Practices

Examination of universities with course catalogs provided on the web, conference proceedings, and research digests shows a flurry of research on software development for mobile devices and cellular phones at the graduate level, but very little evidence of wireless software certification integrated into the undergraduate curriculum [NSF 2003]. Many major universities teach Java, and also J2ME in upper level undergraduate courses, but no university offers any kind of industry standards or certification processes in the undergraduate classrooms. Auburn University is the first university in the nation to offer an undergraduate degree in wireless engineering. With the Auburn OK program in place, it will also be the first university to provide certification to student applications developed in these wireless engineering courses.

Auburn OK

As discussed in Chapter 1, the Auburn University J2ME Development Guide was synthesized to address the goal of adapting Nokia OK application standards for the undergraduate classroom. From here on we will call this process of adapting the standards and certifying the student applications as Auburn OK keeping in view its derivation from the Nokia OK process.

3.1 AU J2ME Development Guide

The Auburn University J2ME Development Guide is written for the students of AU wireless engineering courses. It introduces them to mobile phone application development environment, general guidelines that will help in streamlined programs, standards that need to be followed and the need for them.

The guide describes the limitations of the medium (the cell phones), such as form factor, memory size etc, and suggests the use of standards to overcome these. The guidelines are the programming practices used in the industry for following the standards and efficient implementation. The guide is attached to this report as Appendix A.

3.2 Castle Quest

A MIDlet called Castle Quest is also included in this project along with the guide. A MIDlet is a Java application designed to use the MIDP libraries. This game is developed for the benefit of the students so they can understand the quirks in the development of a mobile phone application. Implementation of most of the guidelines is explained in the guide by using the code of this MIDlet as example.

Castle Quest is a simple game yet it implements most if not all of the features included in a typical mobile phone game or application. It includes menu structure and text displays which are the high-level user interface components, graphical canvas display which is a low-level user interface component, persistent storage with RMS (Record Management Store), implementation of localization, use of sounds, vibration and back light, and timer management.

Nokia phones 3650 and 6610 were used as bench mark phones to test this game. The code of this MIDlet is attached to this report as Appendix B for further details.

opponent's pawns are marked by alphabetic characters a, b, etc. The game provides a default number of pawns for the first time the game is played. This number can be changed by the user before the start of the game by using the Settings option of the main menu as shown in Fig.4.4. For subsequent games, he can choose a different number of pawns or continue with the number of pawns used in the last game.



Fig.4.3 Main menu screen

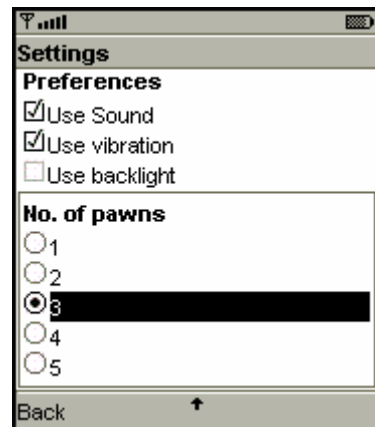


Fig.4.4 Settings screen

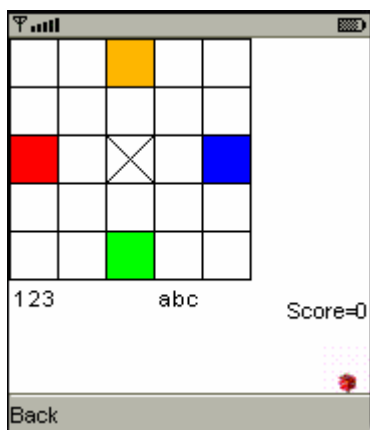


Fig.4.5 Game screen 1

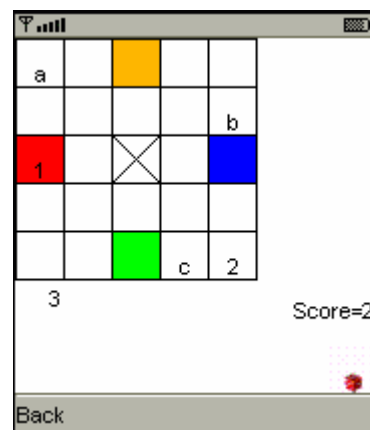


Fig.4.6 Game screen 2

The objective of the game is to roll the dice and select a pawn for each score obtained, and eventually take all of one's pawns into the castle at the centre of the board. The pawns travel in the path shown in Fig.4.2. The user and the opponent alternate in rolling the dice. Left and right game keys work as toggle keys to roll the dice.

The pawns can also kill each other if they are placed in a cell occupied by an opponent's pawn, except in safe boxes which are the crossed cells. The players' pawns can coexist in these safe boxes which are like rest houses. After each move, the status is checked to see if the game is over and the outcome, if the human player has won or lost, is displayed to the screen. Accordingly the high scores screen is displayed asking the user to enter his name or to dismiss and go back to the menu screen to play a new game or quit.

The main menu also includes Instructions (Fig.4.7) which describes the objective of the game and help on how it should be played. Selecting About from the main menu shows details about the MIDlet, its version and the author (Fig.4.8).

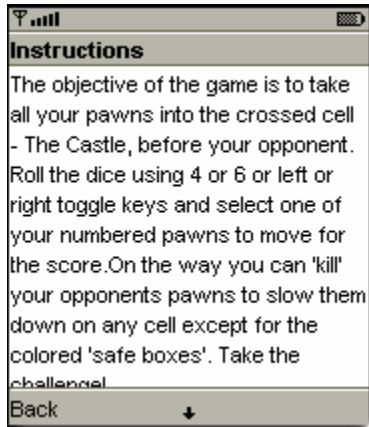


Fig.4.7 Instructions screen



Fig.4.8 About screen

4.2 Statement of Scope

Castle Quest game provides the following functionalities:

1. Splash screen display
2. Change sound, backlight, vibration settings
3. Select number of pawns to play with
4. Game saved and continued
5. View instructions
6. View vendor information
7. View high scores
8. Rolling dice
9. Selecting the pawn
10. Count number of moves

A brief description of each of the above functionalities follows:

1. Splash Screen: This is the first screen displayed when the midlet is started. The screen shot is shown in Fig.4.1. This screen will be automatically dismissed after 3 seconds or the player can dismiss it by pressing any key on the phone pad. Splash is implemented as a Canvas screen and the Main Menu screen which follows splash is implemented using a List.

2. Sound, backlight, vibration settings: The 'Settings' option in the main menu allows the user to either switch on or switch off the sound, backlight and vibration functions for this game. These settings are saved in the phone, so when the user opens the midlet in future, it will have the same settings. The Settings screen is a Form with two ChoiceGroup items in it.

3. Select number of pawns: This functionality is provided by the other ChoiceGroup of the Settings Form, where the user selects the number of pawns that he wants to play with. This value is also saved for the next game. The first time the user plays the game, a default value is provided by the midlet.

4. Game save and continue: While the user is playing the game, he can go back to the Main Menu and change settings such as sound or vibration, and come back to play the saved game. If the user selects a different number of pawns, then a new game is started and the old game is deleted. If a game is

saved, Main Menu screen will show an additional item 'Continue' to let the user play the paused game. The user can always choose to start another session with 'New Game'. A game state is also saved when there is an incoming call.

5. View instructions: 'Instructions' option in the Main Menu opens a text box screen to show help text on how to play the game. This screen is vertically scrollable.

6. View vendor information: This option shows the name of the midlet, version number, and the author name in a text box.

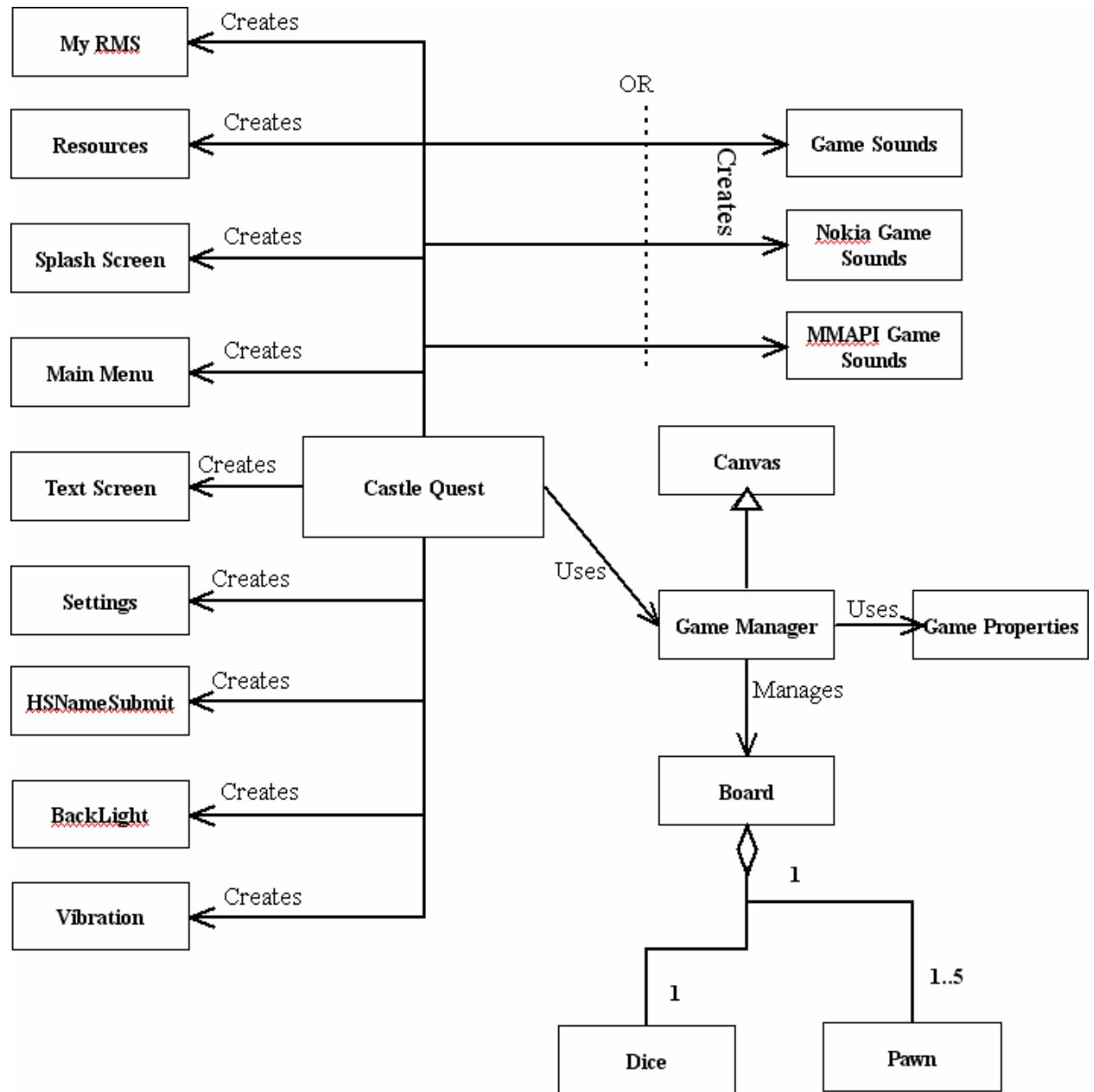
7. View high scores: This screen shows the top three high scores and the names of the players. Scores are counted on number of moves on the board. The least number of moves will be the highest score. Entering the name for a high score is optional.

8. Rolling dice: The player can roll the dice in his turn with the Left or Right game actions as toggle keys. In Nokia 3650, the left arrow, right arrow, 2, and * keys act as Left or Right game actions. In Nokia 6610, left arrow, right arrow, 4, and 6 keys act as Left or Right arrow keys.

9. Selecting the pawn: The player can select the pawn to be moved for the present score by pressing the appropriate key for the number of the pawn. If the pawn selected is not a valid one for the score then a 'Select a different pawn' alert is displayed.

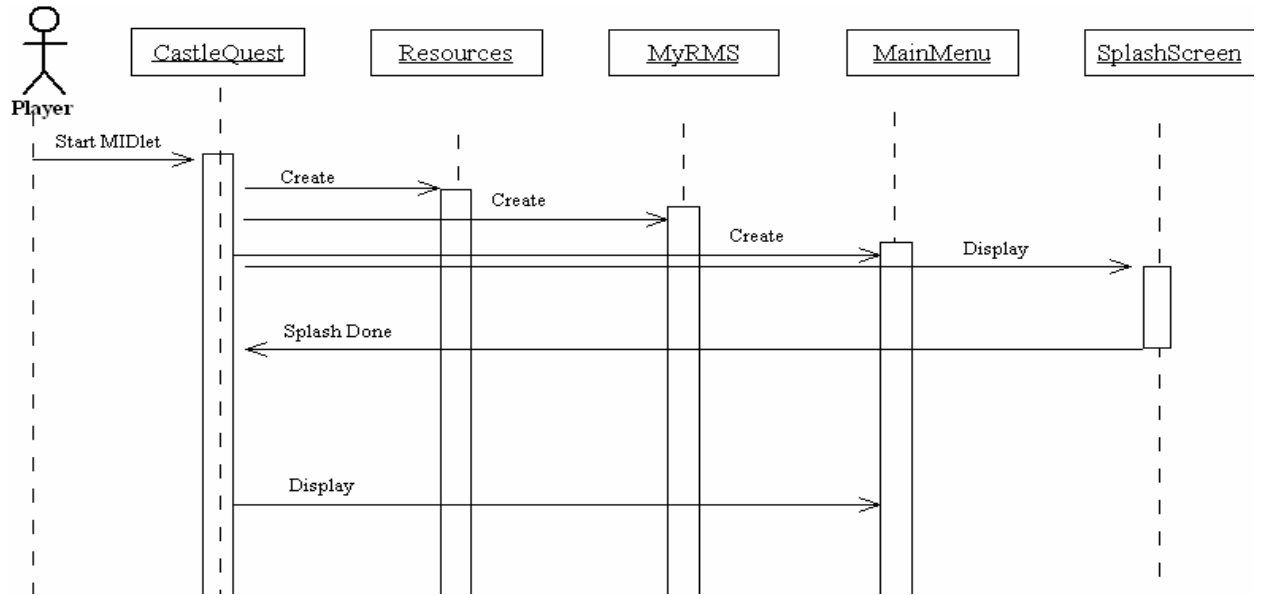
10. Count number of moves: The number of moves used by the player is counted and if at the end of the game the player qualifies for a high score then a screen for text entry is displayed.

4.3 Class Diagram

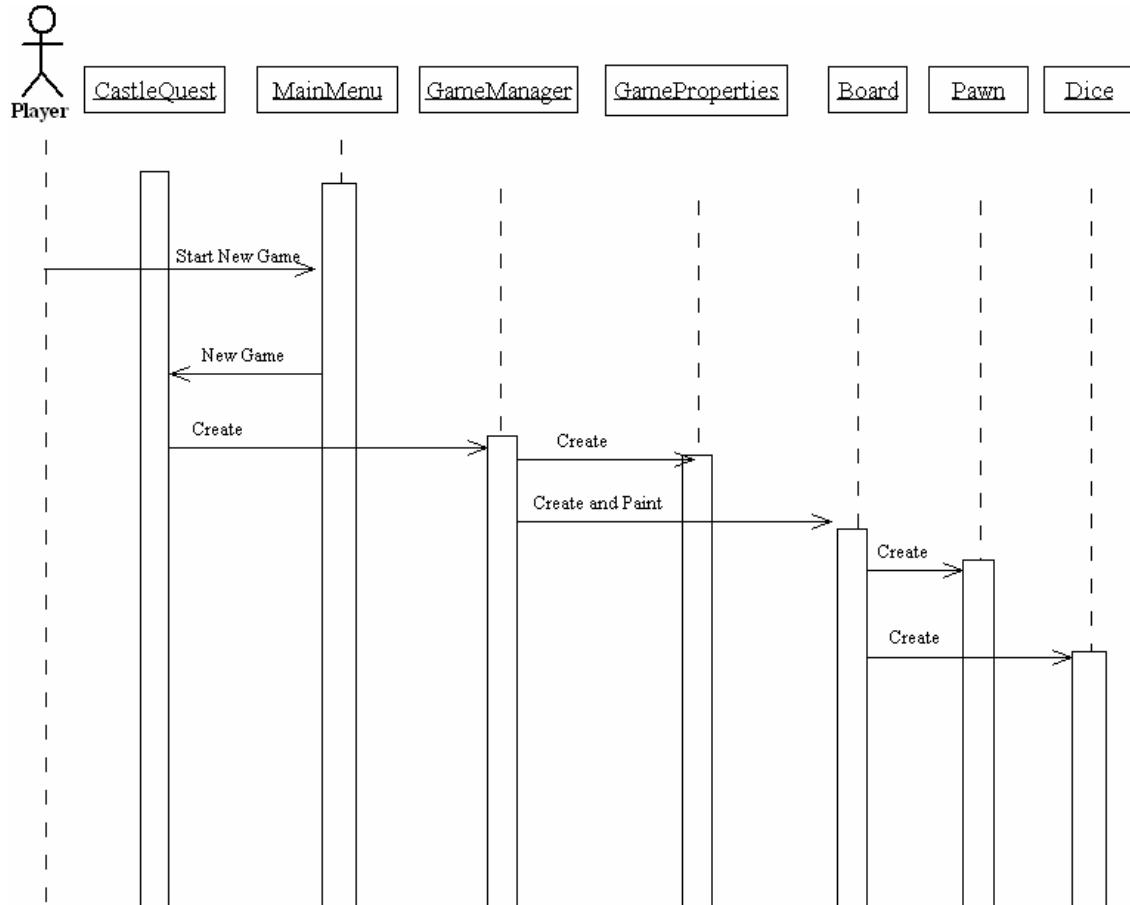


4.4 Sequence Diagrams

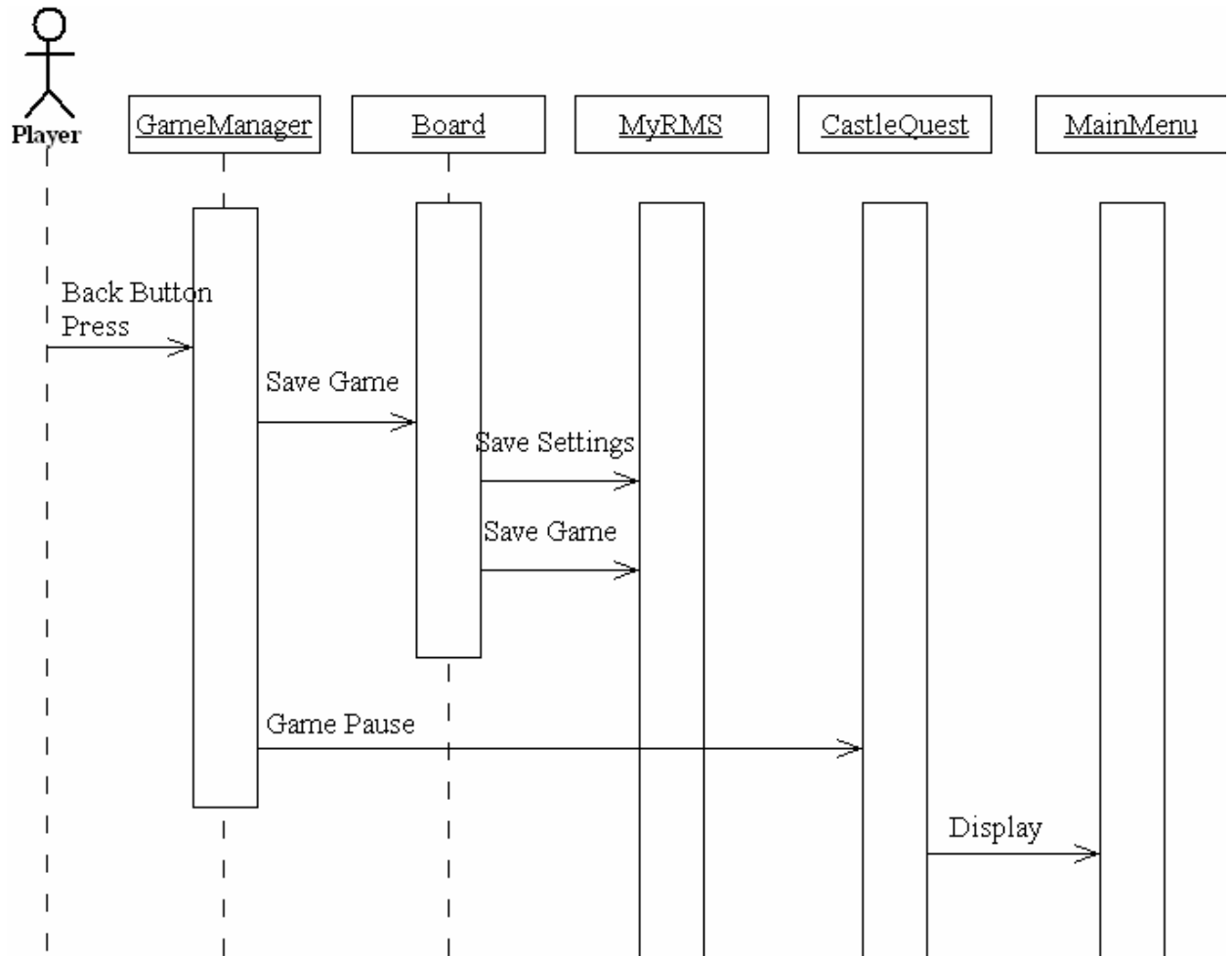
1. Starting MIDlet



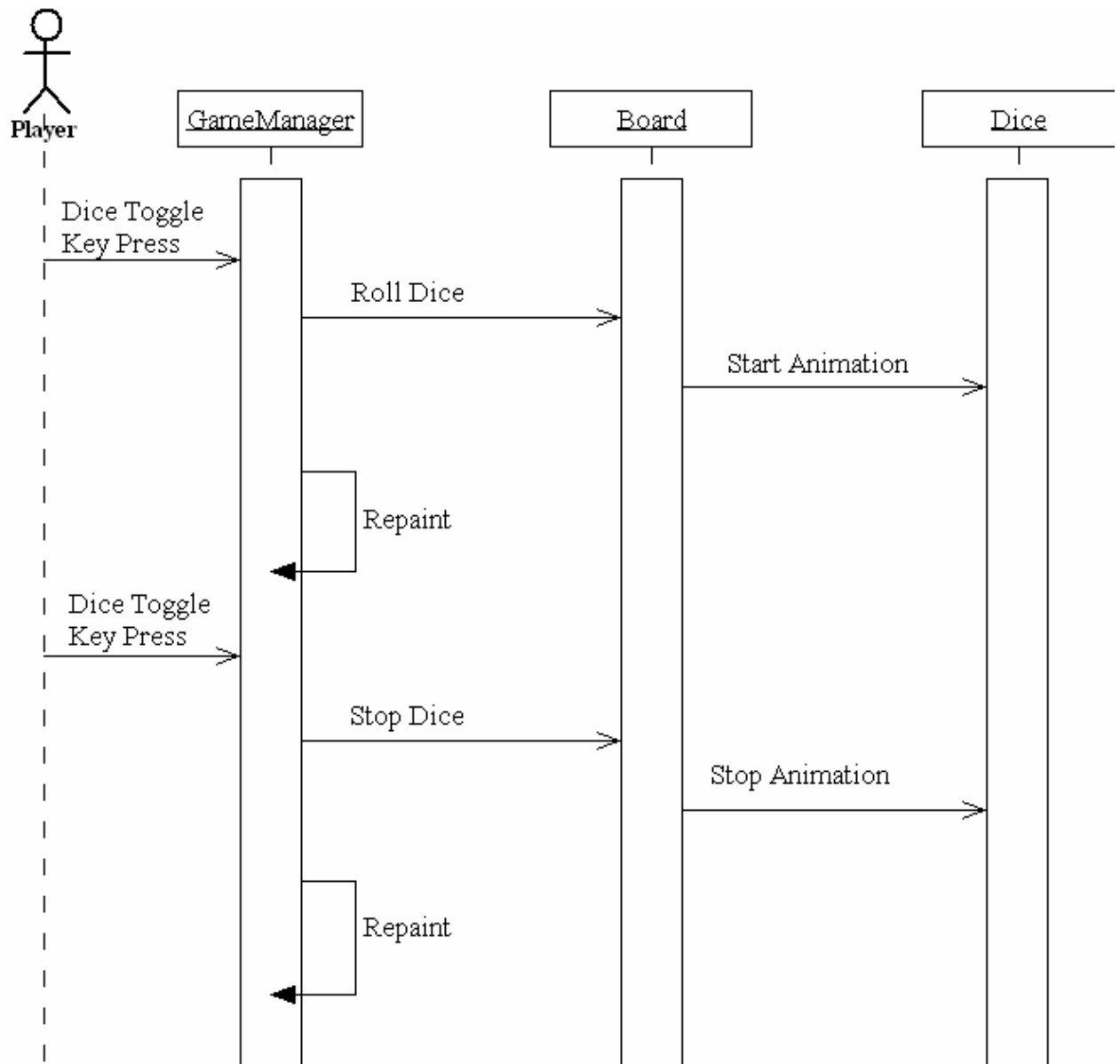
2. Start New Game



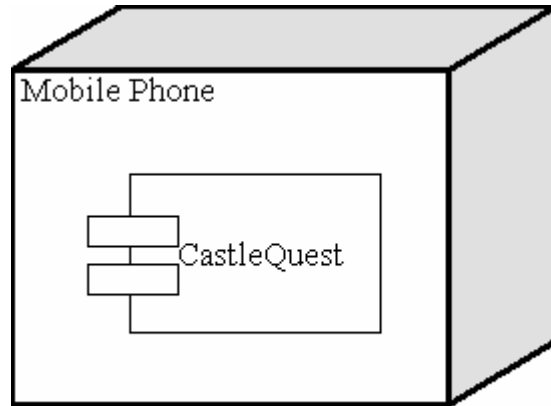
3. Save Game



4. Dice Animation



4.5 Deployment Diagram



4.6 Design Affected by Standards?

Being aware of the standards and the certification details before starting the development of Castle Quest had a major impact on its design. The characteristics of a mobile phone game described by the standards clearly define the core requirements that must be satisfied by any mobile phone game. In addition to that, the usability standards helped dilate the user interface design down to the particulars like the position of the soft key commands. The user interface standards section of the guide lists and describes the features of the different screens that must be included in a game. Following is a detailed description of the effect of the standards and the guidelines on the design of this game.

4.6.1 Fundamental Standards

The fundamental standards identify what the content of a mobile phone game should and should not be. In other words they describe the behavior of the game. This helped in designing the game itself, what it should be about and how it should be played.

Castle Quest has been derived from an ancient game called Ashta Chemma played in India. It has a mythological significance in the country of its origin. Played during the era of the kings, this game was used to improve the hand-eye coordination and to teach teenagers war tactics and strategy. Originally, the game was played on a cloth-knit board or a wooden board with tamarind seeds for dice and wood-carved bright colored pieces for pawns. It was designed to fit the cell phone's display and key pad layout and renamed as Castle Quest to suit the audience.

4.6.2 System Specific Standards

This section outlines the behavior of a mobile phone application during occasional or exceptional tasks. The design features like the behavior of the game on the event of incoming calls, or red soft key selection are defined here.

4.6.3 Usability Standards

The design details such as the speed of performance, the amount of time each screen must appear, the consistency of the soft key labels in different screens, and usability requirements of sound or vibration have been decided based on this section. The negative soft key commands such as Back and Cancel were implemented for the right soft keys and positive commands such as Select were implemented for the left soft keys as the guide says that this functionality must be consistent with that provided by the other Nokia phone applications. The sound and vibration use in the game was kept to a minimum and used only where appropriate.

4.6.4 User Interface Standards

This is the most useful section in that it describes the specific features that must be provided by any mobile phone game. It lists the screens that must be implemented and traces the order of appearance of these screens. The design of the class diagram was based on the information from this section. The classes are divided into display classes such as Settings, SplashScreen, TextScreen etc., computation classes such as GameManager, Board, MIDlet class CastleQuest, and helper classes such as Resources, MyRMS, GameProperties, etc.

The user interface of each screen was clearly depicted to the last of the details such as the soft key commands, title and content, and the behavior on the event of any key press at any point of time.

4.6.5 Guidelines

The guidelines included in the guide affected the design as well as the implementation details of the MIDlet. The effects are described below.

1. Resources class was added to the design to handle localization and internationalization issues described in Writing World-Aware Applications section.
2. To achieve device independence, hard coding was avoided and based on the display size of the mobile phone the game screen is computed.
3. To take advantage of vendor API, the availability of Nokia API is checked before using MMAPI for sound implementation. At runtime NokiaPlayer class is implemented if Nokia API is available and if not MMAPIPlayer is implemented. If neither is present, then a dummy class CQPlayer is used.
4. Nokia 3650 does not support vibration and backlight functionality. But these are supported by Nokia 6610. Before implementation, "microedition.platform" is checked to confirm the phone model and accordingly the features are used.
5. MyRMS class was introduced to conform to the standard of saving and retrieving a game from persistent storage.

6. For better performance, repaint of clipping rectangles was implemented for painting the animation of the dice.
7. OO vs. Size: Each empty class declaration takes up 200 bytes of space after compilation. Keeping in view the extreme restriction on the size of the JAR file, the number of classes used was minimized. This affected the object oriented-ness and hence the comprehensibility of the MIDlet's design. The small footprint requirement forces that each MIDlet should fit the individual phone specification exactly. So there is no scope for reuse of any part of these MIDlets.

Conclusions

At the time of this writing, Auburn OK process was not yet introduced in the classroom and so there is no feedback available from the students. All the conclusions are derived from the understanding and experiences of the author.

5.1 Observations

- There are a variety of certification processes available in the industry for J2ME applications on mobile phones that are supported by handset manufacturers as well as non-handset vendor organizations. Almost all of these processes involve a) Understanding the certification program and process b) Testing the product informally c) Applying for certification and d) Formal testing and submission of results. The effectiveness of each process depends on:
 - The maturity level of the process i.e. a well-depicted process with clear terms and conditions and adequate amount of information to help the developers understand the requirements for the certification.
 - Support provided by the organization in the form of discussion boards, professional help, FAQ documents, and access via email.

- Prompt and well documented results and feedback that describe if and how the application failed to meet any standards.
 - Comprehensive checklists for informal testing before submission to make sure that the application meets the relevant conformance requirements and is ready for entry into the certification program.
 - Additional tools or documents that will aid the developers in the process of development or testing the applications.
- Certification processes in the industry are affecting the development of mobile applications and are successful in setting standards for such software.
 - Universities can take advantage of the industry practices such as testing and certification processes by adapting and integrating them into the academic classrooms. It is easier to adapt J2ME application standards and guidelines into classroom, as they are small and manageable and so can be distilled for classroom with little loss of industry intent.
 - Having an academic certification process can be especially useful for undergraduate students who can benefit from getting a taste of the industry standards very early in their course work. These are the same benefits that we talked about in Chapter 1.
 - Adapting industry practices for classroom use involves evaluating all the information at hand and deciding on the relevance of the various elements. Adapting the Nokia's OK process required removing references

to Nokia or its specific procedures, and adding explanations to standards or guidelines that are unclear to a non-Nokia audience. For comprehensiveness, additional guidelines from other sources were also added.

- The Auburn University J2ME Development Guide describes the basic behavior, necessary user interface features and the essential usability features of any mobile phone application. To conform to the Auburn OK standards a lot of changes were made to the design and implementation of the Castle Quest game as discussed in section 4.6 of Chapter 4.
- The Castle Quest game illustrates the use of most of the standards and can be used as an example by the students developing such applications.
- Developing a MIDP application for a mobile phone is not a simple task.
 - The small footprint of the phone can place a constraint on the design method used for building the application. Object oriented design approach is not always possible as some of these practices increase the size of the final application.
 - The constraint of limited memory also affects the quality of graphics used. For example, the dice images in Castle Quest were downsized, leading to a loss in sharpness. In this case it did not cause any problem but in an application with a lot of images or animations this can be an issue.

- Often there is dramatic difference in performance values among various handheld devices even within the same family of products. The target devices used in this project are both manufactured by Nokia, but they differ in a lot of aspects. For example, MMAPi is supported in Nokia 3650 but not in Nokia 6610.
- Addressing the differences in the target devices can also increase the size of the application. Additional code has to be included to compute the graphics to support the different screen sizes or to check for the API availability.

5.2 Future Work

The most important task that should follow now is to introduce this work into the classroom. Students of introductory courses can be given a glimpse into the J2ME development to make them understand how it fits in with the rest of the Java development that they are learning. Auburn OK concept can be launched in the next level courses, where students can develop their applications based on some of the standards mentioned in the guide. Students of more advanced courses such as the Capstone design project can be asked to develop applications based on Auburn OK standards which will be verified and certified by a faculty member or a group of graduate students. Qualification of each senior design project can be based on whether it passes the certification. This will definitely

help the students both understand and experience the industry quality standards.

Most of the standards can only be tested manually, i.e., executing the application to see if it satisfies the requirements. Application Standards section of the Auburn University J2ME Development Guide can be used as a checklist to do this verification. However automated testing tools can be developed or existing tools like the Nokia Testing Suite [Nokia 2004b] can be used wherever possible to decrease the time and effort that goes into testing.

As more MIDP 2.0 phones are made available in the market and as the industry accepts the new technology, the guide can be improved and updated to include the new MIDP 2.0 issues. Further, the effect of object oriented-ness on the size of the application can be determined and heuristics can be developed to understand and arrive at optimal design practices for mobile applications.

References

- [Barr 2002] Michael Barr, "Java 2 Micro Edition, Course 330", Embedded Systems Conference, Chicago, June 3-6, 2002. <http://www.netrino.com/Papers/ESC2002/> . Accessed December 2003.
- [Cellmania 2004] Cellmania website. <http://www.cellmania.com/> Accessed April 2004.
- [Ericsson 2003] Increased focus on J2ME at the annual JavaOne conference, Ericsson Mobility World articles, July 22, 2003. http://www.ericsson.com/mobilityworld/sub/articles/other_articles/nl03jul01
- [Grott 2004] Fred Grott. High Costs of J2ME Certification. February 2004. http://jroller.com/page/shareme/20040221#high_costs_of_j2me_certification
- [JavaVerified 2004] Java Verified program for J2ME <http://www.javaverified.com/index.jsp> Accessed March 2004.
- [Keogh 2003] Keogh, James. J2ME: The Complete Reference. 2003.
- [Lawton 2002] Lawton, G. Moving Java into mobile phones. IEEE Computer, Vol. 35, Number 6, June 2002, pp. 17-20.
- [Motorola 2003] Motorola Application Certification for J2ME Applications Developer Guide, Version 1.0.2, Feb 11, 2003. Accessed September 2003.
- [Muchow 2002] John Muchow, Core J2ME Technology and MIDP, Prentice Hall PTR, 2002.
- [Nokia 2003a] Nokia Application Development Tools. <http://forum.nokia.com/main> Accessed August 2003.
- [Nokia 2003b] Nokia OK Concept. <http://forum.nokia.com/main> . Accessed August 2003.
- [Nokia 2003c] Series 30 MIDP Concept. <http://www.forum.nokia.com/main/0,6566,034-32,00.html> . Accessed October 2003.
- [Nokia 2003d] Series 40 Developer Platform. http://forum.nokia.com/main/1,6566,010_20,00.html#getting . Accessed October 2003.
- [Nokia 2003e] Series 60 Developer Platform. http://forum.nokia.com/main/0,6566,010_40,00.html Accessed October 2003.

- [Nokia 2003f] Series 90 Developer Platform. http://forum.nokia.com/main/0,6566,010_60,00.html
Accessed October 2003.
- [Nokia 2003g] Developer Checklist for J2ME Applications, 2002.
Nokia OK MIDP Application Requirements, 9/19/2002.
Nokia OK MIDP Application Requirements for Games, 9/11/2002.
Nokia OK Content Guidelines, September 2002.
<http://forum.nokia.com/main> . Accessed September 2003.
- [Nokia 2003h] J2ME Application Testing. http://forum.nokia.com/main/0,,22_10,00.html
Accessed November 2003.
- [Nokia 2003i] Nokia OK Pre-Test Sets.
http://forum.nokia.com/main/1,6566,00_25_10_35_40,00.html
Accessed November 2003.
- [Nokia 2003j] Nokia OK Certification Process.
http://forum.nokia.com/main/0,6566,00_25_10,00.html
Testing House Information.
http://forum.nokia.com/main/1,6566,00_25_10_35_40,00.html
- [Nokia 2003k] Nokia OK Process Flowchart.
http://forum.nokia.com/main/1,6566,00_25_10_35_10,00.html
Accessed October 2003.
- [Nokia 2003l] Nokia Developer Channel. 'How to get the Nokia OK on your Mobile Application.
<http://portals.devx.com/Nokia/Article/6729> . Accessed November 2003.
- [Nokia 2004a] Nokia Developer Platforms. <http://forum.nokia.com/main/0,6566,010,00.html#s40> .
Accessed March 2004.
- [Nokia 2004b] Nokia Testing Suite. <http://www.forum.nokia.com/main/0,6566,034-43,00.html>
Accessed January 2004.
- [NSF 2003] Umphress, David. "Bringing J2ME Industry Practice into the Undergraduate Classroom", NSF award 0311339, June 2003. Appendix C
- [Ortiz 2003] Enrique Ortiz, The Generic Connection Framework, Technical Articles and Tips,
<http://developers.sun.com/techttopics/mobility/midp/articles/genericframework/>
August 2003. Accessed February 2004.
- [Sun 2004a] Sun Microsystems. J2ME Devices. <http://jal.sun.com/webapps/device/device>
Accessed February 2004.

[Synclast
2004]

Synclast Java 2 Micro Edition. http://www.synclast.com/app_dist.jsp
Accessed April 2004.

[Umphress
et al 2004]

David A. Umphress, James H. Cross, Jhilmil Jain, Nischita Meda, Larry A. Barowski.
"Bringing J2ME Industry Practices into Undergraduate Classroom", Proceedings of
2004 SIGCSE Technical Symposium, Norfolk, VA, March 3-7, 2004, pp. 301-305.

APPENDICES

Appendix A: Auburn University J2ME Development Guide

Auburn University J2ME Development Guide

Nischita R. Meda

David A. Umphress

Department of Computer Science & Software Engineering

Auburn University, AL 36849

{medanis, umphrda}@auburn.edu

1	Introduction.....	52
1.1	Purpose.....	52
1.2	Definitions, Acronyms and Abbreviations	52
1.3	References.....	52
2	J2ME Development and Auburn University	53
2.1	J2ME Development	53
2.2	Tools – JGRASP	54
2.3	AU OK?	54
3	Need for Standards.....	54
3.1	Form Factor	54
3.2	Screen Size and Format	54
3.3	Limited Color and Sound Support	55
3.4	Limited Application Size	55
3.5	Heap Space	55
3.6	High Response Time.....	55
3.7	Interruptibility	55
4	Application Guidelines	55
4.1	Device Independence	56
4.2	Sound, Vibration and Lights	56
4.3	Screen Format	57
4.4	World-Aware Applications	57
4.5	Simultaneous Key Presses	59
4.6	Key Events	60
4.7	Avoid Key Names	60
4.8	User’s Time.....	60
4.9	High Scores List.....	61
4.10	Limited Memory	61
4.11	Limited Processing Power	61
4.12	Move Computation to the Server	62
4.13	Limited Bandwidth	62
4.14	Limited Storage	62
4.15	Measuring Program Execution Speed	62
4.16	Libraries	62
4.17	Use of Resources	63
4.18	Device Debugging	63
4.19	Object Oriented Design vs. Size Optimization	63
5	Application Standards.....	64
5.1	Fundamental Standards	64
5.2	System Specific Standards	65
5.3	Usability Standards	65

5.4	User Interface Standards	66
5.4.1	General User Interface Aspects	66
5.4.2	Generic Feature Screen	67
5.4.3	Settings.....	67
5.4.4	Instructions and About.....	67
5.4.5	Game Screen (for games).....	67
5.5	Delivery of the Application	68

1.0 Introduction

1.1 Purpose

This manual presents standards on user interface look and feel, human-computer interaction guidelines, security requirements, reliability guidelines, performance requirements, and content restrictions for developers who wish to write MIDlets for J2ME MIDP devices. These standards are mainly targeted towards Auburn University students writing applications for the various wireless engineering courses being offered. Students who wish to receive Auburn OK certification for their applications are expected to adhere to these standards and guidelines.

Since game applications require more or different standards than regular applications we will also discuss them when required.

1.2 Definitions, Acronyms and Abbreviations

J2ME Java 2 Platform, Micro Edition. A version of Java environment suitable for use on more restricted devices such as mobile phone handsets and PDAs.

MIDlet A Java application designed to use the MIDP libraries.

CLDC Connected Limited Device Configuration. It defines used Java language and virtual machine features. CLDC contains java.lang, java.io, java.util, and javax.microedition.io

MIDP Mobile Information Device Profile. A set of Java class libraries used by J2ME on mobile handsets and PDAs built on top of CLDC. It contains javax.microedition.lcdui, javax.microedition.io.HttpConnection, javax.microedition.rms, javax.microedition.midlet, java.util.Timer, java.util.TimerTask, java.lang.IllegalStateException

RMS Record Management Store

JAD files Java Application Descriptor files

JAR files Java Archive files

PDA Personal Digital Assistant. A small handheld computer such as a Palm Pilot.

CBA Command Button Area. A toolbar controlled by the soft-keys.

UI User Interface.

AI Artificial intelligence. In games, AI refers specifically to the logic governing the behavior of computer-controlled opponents.

1.3 References

- [1] Designing single player mobile games. Available at: http://www.forum.nokia.com/main/1,6566,21,00.html?fsrParam=1-3-/main/1,6566,21_10,00.html&fileID=3601. Accessed September 9, 2003.
- [2] MIDP and Game UI. Available at: http://www.forum.nokia.com/main/1,6566,21,00.html?fsrParam=1-3-/main/1,6566,21_10,00.html&fileID=2867. Accessed September 11, 2003.
- [3] Developer Platform 1.0 for Series 40: Usability Guidelines for J2ME Games. Available at: http://www.forum.nokia.com/main/1,6566,1_0_10,00.html. Accessed November 20, 2003.

- [4] Introduction to Mobile Game Development. Available at: http://www.forum.nokia.com/main/1,6566,040,00.html?fsrParam=3-3-/html_reader/main/1,,2768,00.html&fileID=2768. Accessed November 6, 2003.
- [5] Developer Checklist for J2ME Applications. Available at: http://www.forum.nokia.com/main/0,6566,22_10,00.html. Accessed November 19, 2003.
- [6] Developer Checklist for J2ME Games. Available at: http://www.forum.nokia.com/main/0,6566,22_10,00.html. Accessed November 19, 2003.
- [7] Nokia OK MIDP Application Requirements. Available at: http://www.forum.nokia.com/main/1,6566,00_25_10_35_20,00.html. Accessed November 19, 2003.
- [8] Nokia OK MIDP Applications Requirements for Games. Available at: http://www.forum.nokia.com/main/1,6566,00_25_10_35_20,00.html. Accessed November 19, 2003.
- [9] Guidelines for Game Developers using Nokia MIDP Devices. Available at: http://www.forum.nokia.com/main/1,6566,21,00.html?fsrParam=1-3-/main/1,6566,21_10,00.html&fileID=2822. Accessed August 20, 2003.
- [10] Efficient MIDP Programming. Available at: http://www.forum.nokia.com/main/1,6566,1_0_10,00.html. Accessed November 16, 2003.
- [11] Brief Introduction to MIDP Programming. Available at: <http://www.forum.nokia.com/main/1,6566,21,00.html?fsrParam=1-3-/main/0,6566,21,00.html&fileID=2637>. Accessed November 16, 2003.
- [12] Eric Giguere. Writing World-Aware J2ME Applications. Technical Articles and Tips January 29, 2000; Release 1.0. Available at: <http://developers.sun.com/techtips/mobility/midp/ttips/worldaware/>. Accessed September 30, 2003.
- [13] Phone Scoop Glossary. Available at: <http://www.phonescoop.com/glossary/term.php?gid=4>. Accessed October 1, 2003.
- [14] James Keogh. J2ME: The Complete Reference; Published by McGraw-Hill Companies, 2003.
- [15] Chris Preimesberger. Building Mobile Game Applications for Fun and Profit. Available at: <http://www.devx.com/Nokia/Article/6218> Accessed March 2004.
- [16] Kay Neuenhofen. Designing and Writing Java Action Games for Small Devices. <http://developers.sun.com/techtips/mobility/blueprints/articles/game/> Accessed April 2004.
- [17] Eric Giguere. Optimizing J2ME Application Size. J2ME Tech Tips, Feb 26, 2002. <http://java.sun.com/developer/J2METechTips/2002/tt0226.html> Accessed January 2004.

2.0 J2ME Development and Auburn University

2.1 J2ME Development

TBD

2.2 *Tools – JGRASP*

TBD

2.3 *AU OK?*

TBD

3.0 Need for Standards

In this document we outline the standards that all the J2ME applications must follow for Auburn OK certification. By standards we don't mean any numbers that should be met. These are simply answers of yes or no to things that are deemed necessary.

There are some limitations to the medium that we are addressing i.e. mobile phones, so we will need the standards to overcome these limitations and to set some expectations. Here are some of those limitations.

3.1 *Form Factor* [13]

Mobile phones come in different physical styles (form factors) which can be classified as follows:

Block: This is also called the candy-bar style (for thinner phones) and is the most basic style. The whole phone is one big chunk with no moving parts apart from the buttons and maybe the antenna. Key guard feature is usually employed to prevent accidental pressing of keys. There is a special combination to lock and unlock the buttons.

Flip: This is identical to block type but there is a thin flip or cover that flips open. This flip covers the keys so there are no unwanted key presses when the phone is not in use. In some phones the flip also covers the display of the phone. These kinds of phones have features like Active Flip where calls can be answered and ended by opening and closing the flip. In voice activated phones this feature can be used to make calls by opening the flip and saying a name.

Folder: Also known as clamshell phones, they consist of two halves, connected by a hinge. The phone folds close when not in use. The top half usually contains the speaker, and the display or battery, with the bottom half containing the remaining components. These phones also feature Active Flips.

3.2 *Screen Size and Format* [4], [15]

People don't like to carry big and heavy phones, so screen sizes are likely to remain small. But Screen sizes do vary from device to device and to develop an application that must run on more than one device the application must be optimized to each device.

Mobile phones have resolutions as low as 82 x 99 pixels in monochrome and up to 176 x 208 in 256 colors. Writing applications to suit such varied target devices usually means developing for the lowest common denominator and making adjustments for specific devices. If different versions cannot be developed for the low-end and high-end devices then it means having crippled functionality for some devices.

3.3 Limited Color and Sound Support [4]

Although many phones that support 12-bit color are available now, most phones that are already in use are black and white.

Mobile phone applications have a limited ability to play sounds since J2ME specification does not require hardware manufacturers to support sound at all. MIDP 1.0 does not mandate sound support; neither does MIDP 2.0 which is also based on CLDC. But MIDI support and use of sounds is becoming a standard.

3.4 Limited Application Size [1], [4]

There are two limitations on application size: the amount of memory on Java-enabled phones available for MIDlets, and size permitted for a MIDlet. The actual limit depends on the handset and also on the operator's OTA policies. Program code, graphics, and sounds all take up space and contribute to the application's compiled size.

Another concern is the RMS used by the MIDlets. The MIDlet should make sure that there is enough RMS storage memory for the suite's use. With JSR-185 compliant devices, the maximum amount a suite can use is 30kB of RMS.

3.5 Heap Space [10]

Heap space is used to store graphic buffers, objects created, etc, when an application runs. Mobile phones do not have large amounts of heap memory. It can range from 200kB to 1 MB and is shared with other applications if the platform supports multitasking.

3.6 High Response Time

The time for a machine to respond after it receives a request is usually high (in seconds) in over the air network. It is not very important for most applications other than games.

3.7 Interruptibility [4]

Mobile phone applications can be interrupted by incoming calls or messages. So they should be able to pause and recover without any errors.

Writing mobile phone applications is not an easy task because of the large number of these limitations and all the things that have to be considered. Also since this is an emerging field when compared to writing PC applications, we give here some guidelines to help you understand this technology better. Also references [3], [9], and [10] are good for reading.

4.0 Application Guidelines

This section contains some general guidelines for efficient programming and tips on how to follow the standards. It is not mandatory to follow these guidelines. These are provided only as helpful information.

4.1 Device Independence [9]

Devices may differ in screen size, keyboard layout, and available APIs. These differences should be taken into account when writing applications for more than one device in order to write the most portable application.

The application should get the screen size from the system using methods *getHeight* and *getWidth* from class *Canvas* instead of hard coding the coordinates into the program.

<<Example Code: Use *getHeight()* and *getWidth()*>>

```
//Class: GameManager

public void paint(Graphics g)
{
drawBoard(g);
g.drawImage(diceImage, getWidth(), getHeight(), Graphics.BOTTOM | Graphics.RIGHT);
g.setFont(gameProp.pawnFont);
g.drawString(resources.getString(Resources.LABEL_SCORE), getWidth() - 6, getHeight() - 34,
Graphics.BOTTOM | Graphics.RIGHT);
}
```

Using the default MIDP classes instead of the vendor specific classes improves the portability of the application though it compromises some extra features provided by the vendor APIs.

4.2 Sound, Vibration and Lights [9], [11]

Go easy on the sound. Avoid annoying sounds: too loud, too high-pitched. A mobile application may be used anywhere and so the user should be given an option to mute/unmute the sound feature within the application. Also, it is better to avoid background music, if possible.

Same goes for the vibration feature, and it should be used much less than the sound feature as it can become annoying.

<<Example code: Check what sound features are available>>

```
CQPlayer makePlayer()
{
CQPlayer player;
//Create a Nokia Player if Nokia API is present,
//if not create an MMAPI Player if MMAPI is present,
//else create a dummy GameSounds Player
try
{
//This stmt throws an exception if Nokia API is not present
Class.forName("com.nokia.mid.sound.Sound");
//If we got here, then Nokia API is present
Class clas = Class.forName("NokiaPlayer");
System.out.println("Nokia API is present yooahoo");
player = (CQPlayer)(clas.newInstance());
}
catch(Exception ne)
```

```

{
//If no Nokia API then check for MMAPI
System.out.println("Nokia is not present oops");
try
{
//This stmt throws an exception if MMAPI is not present
Class.forName("javax.microedition.media.Manager");
Class.forName("javax.microedition.media.Player");
Class.forName("javax.microedition.media.MediaException");
Class.forName("javax.microedition.media.control.ToneControl");
//If we got here, then MMAPI is present
Class clas = Class.forName("MMAPIPlayer");
System.out.println("MMAPI is present ok");
player = (CQPlayer)(clas.newInstance());
}
catch(Exception me)
{
//Create a dummy player
System.out.println("MMAPI is also not present ooohh");
player = new CQPlayer();
}
}
return player;
}

```

Keeping the backlight on is very important for mobile games. For other applications it is left to the discretion of the developer. It must be mentioned here that depending on the amount of time the user is likely to use the application it is better to turn it off keeping in mind the battery usage. It can come back up once the user presses a key.

4.3 *Screen Format*

Console and PC screens are in a landscape format while mobile phone screens are often in square shape or they are higher than they are wide i.e. portrait format. Designers need to be aware of this when porting games or applications from other platforms.

Keeping in view the size of the screen, character sprites should not appear too big or too small. Usually their maximum width and height should be between 10 to 15 percent of the screens width and height.

4.4 *World-Aware Applications* [12]

Well written applications are both internationalized and localized. An application is internationalized, if it can correctly handle different encodings of character data. An application is localized, if it formats and interprets data (dates, times, time zones, currencies, messages and so on) according to rules specific to the user's locale (country and language). An application that is both internationalized and localized can be called as “world-aware”.

CLDC includes a very small subset of J2SE core classes. Therefore it can be difficult to write world-aware J2ME applications based on CLDC. Still it is not impossible if you know what's available and what's not.

The MIDP system property `microedition.locale` defines the current locale of the device. Its value can be retrieved using the method `System.getProperty()`. The MIDP specification allows a null value to be used in the system property but in some implementations it is never null. Instead of filling the requirements of all locales in a single JAR (compiled version of the MIDlet) file, it is better to create separate JARs for each locale.

<<Example code: Implementing Localization>>

```
//Class: CastleQuest

public CastleQuest()
{
    //Initialize Resources object which contains the strings
    //for user interface. It supports internationalization and
    //localization based on microedition.locale and microedition.encoding
    //Therefore, this resources object is used throughout the program.
    String locale = new String();
    try
    {
        locale = System.getProperty("microedition.locale");
    }
    catch(Exception e)
    {
        System.err.println("microedition.locale is not available");
    }
    player = makePlayer();

    resources = new Resources(locale);
    ...
}

//Class: Resources

public class Resources
{
    //Types of strings used: LABEL for short texts of commands, titles etc
    //and TEXT for descriptive text like instructions, about etc.

    private static short in = 0; //index for each string
    final static short LABEL_ABOUT = in++;
    final static short LABEL_BACK = in++;
    final static short LABEL_CANCEL = in++;
    final static short LABEL_CONTINUE = in++;
    final static short LABEL_EXIT = in++;
    final static short LABEL_GAMEOVER = in++;
    final static short LABEL_HIGHSCORES = in++;
    final static short LABEL_INSTRUCTIONS = in++;
    ...
}
```

```

final static short TEXT_INSTRUCTIONS = in++;
final static short TEXT_INVALIDMOVE = in++;

final static short NUM_IDS = in; //Number of string resources declared

private static Resources object = null;

private String[] strings;

Resources(String locale)
{
if(locale == null)
{
locale = "";
}
strings = stringsEnUS();
}

private String[] stringsEnUS()
{
//strings is assigned US English strings
String[] strArray = new String[NUM_IDS];

//Assign text to Labels
strArray[LABEL_ABOUT] = "About";
strArray[LABEL_BACK] = "Back";
strArray[LABEL_CANCEL] = "Cancel";
strArray[LABEL_CONTINUE] = "Continue";
strArray[LABEL_EXIT] = "Exit";
strArray[LABEL_GAMEOVER] = "Game Over";
strArray[LABEL_HIGHSCORES] = "High Scores";
strArray[LABEL_INSTRUCTIONS] = "Instructions";
...
}
String getString(int id)
{
if((id >= 0) && (id < strings.length))
{
return strings[id];
}
else
{
throw new IllegalArgumentException("id=" + id + " is out of bounds max=" + strings.length);
}
}
}

```

The CLDC system property `microedition.encoding` defines the default character encoding of the device. Others are `microedition.configuration`, `microedition.platform`, and `micoedition.profiles`. Again these can be retrieved using `System.getProperty()`.

4.5 *Simultaneous Key Presses* [9]

Many phones do not support multiple simultaneous key presses.

4.6 Key Events [9]

MIDP class Canvas lets key presses to be handled as low-level code events (“the 1 key was pressed”) or as abstract game actions (“Fire key was pressed”, this is determined by the manufacturer). MIDlets can determine mapping between game actions and keys by using `getGameAction (int keyCode)`. It is always better to use the abstract game actions unless there is a special requirement.

The problem with MIDP 1.0 specification is that it lets game actions to be converted to key codes and vice versa through the methods `getGameAction (int keyCode)` and `getKeyCode (int gameAction)`. `getKeyCode` method returns only one key code for any action. This will result in ambiguity if both “up” button and “2” button mean Up action and the method returns only one key based on the implementation. Whereas using the `getGameAction` correctly returns Up actions for both the keys.

<<Example: To show usage of `getGameAction`>>

```
//Class: GameManager

//Handle key presses for game play
protected void keyPressed(int keyCode)
{
    ...
    //Dice rolls on left or right game actions of the device if it is the player's turn
    else if(playerTurn && hasMoved && (getGameAction(keyCode) == LEFT ||
                                     getGameAction(keyCode) == RIGHT))
    {
        //Stop dice animation if dice is rolling
    }
}
```

4.7 Avoid Key Names [9]

Avoid associating hard-coded text with key codes. `getKeyName` returns different values in different MIDP devices. In order to write portable applications use `keyCode` values or `getGameAction` values instead of `getKeyName` strings.

4.8 User's Time

Do not waste the user's time. Allow them to skip the introduction. Do not require re-entry of data and provide short-cuts and reasonable default values.

<<Example: Dismiss splash screen on any key press>>

```
//Class: SplashScreen

public void keyPressed(int keyCode)
{
    dismiss();
}

...
private void dismiss()
{
}
```

```
timer.cancel();
midlet.splashScreenDone();
}
```

4.9 *High Scores List*

When implementing high scores list, tell the user what score he reached before asking for a name, and provide the previously entered name as default. Make it optional for the user to enter a name; do not force.

4.10 *Limited Memory* [2], [10], [14]

MIDlets use three types of memory: program memory, heap and persistent storage. MIDlets should be frugal in their memory use and handle outages carefully.

To reduce memory requirements, avoid the use of object types when you can use scalar types, and always use the minimum data type suited for storing data. The simplest way to reduce the size of an application is to remove code by simplifying the application and removing unnecessary features.

Usage of heap memory can be minimized by avoiding the instantiation of objects till their point of use, and also by using previously instantiated objects. After use these allocations should be marked for garbage collection by assigning null values to them. Catch `OutOfMemoryErrors` on all allocations, to avoid the device's default mechanism to display the memory error.

While using methods `totalMemory()` and `freeMemory()` of `java.lang.Runtime` class keep in mind that the application may not be able to use all of the memory reported by `freeMemory()`.

Obfuscation tools can help reduce the size of MIDlets by trimming the class files, removing unused methods and classes and renaming methods and variables. Another way to reduce the MIDlet JAR file size is to minimize the size of the resource files. This can be done by using as few images as possible and by using fewer colors in those images. Combine many images into one file whenever it's possible.

In MIDlets persistent storage is used for record stores accessed through `javax.microedition.rms.RecordStore` class. The key concern in using this type of memory should be to catch and gracefully handle any exceptions thrown by the methods of this class. Finally if there is too much data to save on the device, you can always store it on the server.

4.11 *Limited Processing Power* [2]

MIDlets should be optimized for efficiency, profiling sections of slow-running code to determine where bottlenecks exist. Profiling should be carried out on handset hardware, as emulators frequently do not replicate the exact behavior of devices. In particular, optimization of graphical operations may prove useful, through use of vendor proprietary

APIs, which allow better performance; use of clipping rectangles; and benchmarking of graphical performance on target devices.

4.12 Move Computation to the Server [14]

Avoid running computationally intensive tasks on the device. Move these tasks to the server. Deciding what and how much functionality to move to the server is tricky and depends on the application and also the device's connectivity.

4.13 Limited Bandwidth [2]

MIDlets should use bandwidth sparingly and occasionally. MIDlets must account for the fact that network connections may not be available at any given moment, and respond gracefully.

4.14 Limited Storage [2]

On-device storage using the **javax.microedition.rms** classes must be minimized. Possible alternatives include the bundling of static data (such as images or audio files) in the MIDlet JAR file, or the reading and writing of resources from networked storage (using the **javax.microedition.io** classes).

<<Example: Using RMS to store persistent information on phone>>

```
//Class: MyRMS  
//This class shows reading and writing of persistent data on the phone
```

4.15 Measuring Program Execution Speed [10]

Java programs usually spend most of the time executing the library code that the application calls. Therefore it is better to know the performance of the libraries (especially graphics libraries) by measuring how much time it takes to execute. Note that performance measurements not only differ from model to model but also in different versions of the same model.

Add the following extra lines to the program code to find out how long a call takes:

```
long startTime = System.currentTimeMillis();  
doSomething(); //code that you want to time  
long timeTaken = System.currentTimeMillis() - startTime;
```

This will give results in milliseconds. To avoid varying results due to garbage collection during the test call `System.gc()` before starting the test. Make sure to check the resolution of the phone's system clock; it may not have a millisecond precision.

4.16 Libraries [10]

It is good to use libraries of frequently needed functionality. But there might be a lot of functionality in the library that you might not use. So it is a good idea to remove the classes or some methods in those classes that you do not need and recompile the library to be included. This will result in a much efficient use of the JAR file space.

4.17 *Use of Resources* [10]

Make sure screens no longer needed are released for garbage collection. Delay the creation of rarely used screens and let them be garbage collected immediately after use. This will trade execution speed for some heap memory.

Be careful of memory leaks. These occur when a reference to an unwanted object is still present. Remember to set references to unneeded objects to null if they do not go out of scope quickly.

4.18 *Device Debugging* [16]

On-device debugging is painful when working with small devices. It is best to use an emulator like MIDP Wireless Toolkit (WTK for short) whenever possible, because an emulator gives you access to a console on which you can print messages with `System.out.println ()`. You can comment out this code to prevent it from taking up space before you ship the application, but it is still available for you to debug the next release.

WTK 2.0 is a device emulator with a switch in its Preferences settings that lets you monitor memory usage. But keep in mind that not all the objects that show up in the monitor are caused directly by your application. Examine your application design and think broadly about improving performance (especially graphics related) later on; however don't worry about the details immediately. Keep in mind that performance-tuned code is often much harder to debug than naïve code.

4.19 *Object Oriented Design vs. Size Optimization* [16], [17]

Note here that we say size optimization and not size. Only when the application crosses the size limits should one think about compromising OO-ness for optimization. OO-ness is compromised since most of the techniques described below contradict popular OO programming practices. Since we do not have any heuristics to support the 'less-OO design is good for mobile phones', the developer should decide how much of OO design can be sacrificed. Again, not every step mentioned here makes sense for every J2ME application.

1. The first step in reducing size is to remove unnecessary classes by pruning the application's functionality. Check to see if all the features are necessary and build the most minimal version of the application.
2. The second step is to look at inner classes or anonymous classes which commonly implement an event listener. Each class file has a certain amount of overhead associated with it. The most trivial class, with no variables or methods, will give a class file that is almost 200 bytes in size when compiled. Inner classes also bloat code, since the compiler must generate special variables and methods to allow an inner class to access its enclosing class' private information.
3. Collapse your application's inheritance hierarchies. This contradicts the practice of factoring common code into one or more abstract classes but it makes more sense to simplify the inheritance hierarchy.

4. Consider using public variables rather than accessor methods. This is technically bad programming practice, but saves a lot of space because in Java, implementation and use of accessor methods add to the size of the code.
5. Merge classes with common or close functionality or if the classes are very small.
6. The last step is to shorten the names of packages, classes, methods and data members used in the application. This seems silly but a class file holds a lot of symbolic information. Shortening the names of things will reduce the class file size. The savings won't be dramatic, but they can add up when spread over several classes. As we have already mentioned, a tool called obfuscator can do this. The primary purpose of an obfuscator is to 'hide' the code by making it nearly unreadable when decompiled. A side effect of this process is to shrink the size of the application by 10% to 30%.

5.0 Application Standards

This section classifies and enumerates the standards that have to be followed in order for an application to be certified as Auburn OK. These standards are a must (whenever they are applicable).

The list of standards will also help developers detect possible problems in their applications and acts as a checklist. When using this section as a checklist, write out a comment for each standard (using its serial number) which tells if the requirement is applicable to the application or if the application could/could not support the functionality. When your list of comments has all the standards ticked off, it means that your application is ready for Auburn OK certification.

5.1 Fundamental Standards [5], [6], [7]

1. The student must be a graduate or an undergraduate studying at Auburn University
2. The applications must not have any references to:
 - a. Violence against humans/animals
 - b. Gratuitous depiction of violence against animals/non-human characters
 - c. Sex, hard pornography, pedophilic imagery
 - d. Alcohol
 - e. Drugs and drug taking
 - f. Use of guns (handguns, rifles, knives, swords) except for fighting games based on martial arts/wrestling or fighting against "abstract" enemies (aliens, spaceships, etc.). These games must not include gratuitous depictions of their effect
 - g. Use of handguns/rifles except in sports context (e.g. Clay Pigeon shooting is okay)
 - h. Use of knives/axes/swords/guns in human v/s human combat
 - i. Racist imagery
 - j. Overtly political messages, contentious political/global events, terrorism.
 - k. Religious imagery
 - l. Gambling with money

- m. Stealing or robbing, e.g. bank robbery, where the player is perpetrating the crime
 - n. Criminality, where the player is the criminal
 - o. Swearing
 - p. Material targeted and marketed only for small children
 - q. References to tobacco
3. There must be no reference to Auburn University in the application
 4. The application must not harm any system applications
 5. The application must not harm any data stored in the system
 6. Sensitive data must not be stored (e.g., addresses, credit card numbers)
 7. Encryption (if supported) must be used for sensitive data (credit card numbers, passwords and addresses)
 8. Connections must always be agreed to by the user
 9. The installation procedures should work properly
 10. An installation guide and documentation must be provided explaining its use
 11. The application must support a clean uninstall (does not leave any unwanted data on the system)

5.2 *System Specific Standards* [7], [8]

12. Occasional tasks must be handled properly (e.g. receiving calls)
13. Exceptional tasks must be treated appropriately (e.g., for emergency conditions like low battery)
14. Tasks that cope with errors (e.g., caused by using the interruption of network connection during its use) must be considered and treated appropriately
15. The details reported in the JAD file must be consistent with the information reported within the application (e.g., information shown in the About session)
16. Speed must not compromise the application use and purpose (except for games)
17. During game interaction the following events must be handled to pause and save the game session (ready to be retrieved through the Continue functionality):
 - a. An incoming call is received
 - b. A soft key is selected and the screen displayed changes
 - c. The red soft key is selected
 - d. The application is exited and the user does not quit the current game session
18. The status of the application (e.g., the status of the settings: sound, vibra, etc.) must be saved and retrievable when the following events occur:
 - a. An incoming call is received
 - b. The red soft key is selected
 - c. The application is exited

5.3 *Usability Standards* [7], [8]

19. The first screen of the application must contain the title, and should lead to the menu of functions
20. The end-of-call button must always allow exiting from the application
21. The main functionalities must be accessed easily through a Main menu
22. Each functionality must work as described in the application documentation

23. The speed of performance must be acceptable (e.g. less than 10 seconds to show the next screen)
24. Environmental issues (for e.g. noisy environments, interaction with other people...) must not compromise the use of the application.
25. In the case where images do not substitute essential text, text must also be included
26. The labels for the soft keys (if defined by the developer) must be consistent with other applications on the system, or as defined by the manufacturer (e.g. on some systems right soft key refers to “negative” commands such as: Back, Quit, Exit, Cancel, Clear, Exit (exceptions justified by cultural aspects are accepted)
27. There must be no hidden features
28. Difficult terminology and acronyms must be avoided (except when the application is dealing with them)
29. Each screen must appear for the time necessary to read its information
30. The terminology must be correct and clear. There should be no grammatical mistakes, and/or truncated text
31. It should be possible to perform the main tasks without reading the Help/Instructions section
32. If sound/vibra functionality is/are used, the following requirements must be satisfied:
 - a. The current status of the settings should not affect the usability of the application (e.g., whether or not the sound is on)
 - b. Each sound (if any) must be distinctive and have a well-defined meaning (e.g., happy sound for passing a level, sad sound if the player dies)
 - c. Vibra functionality should be used for specific circumstances and more rarely than sounds

5.4 *User Interface Standards* [7], [8]

5.4.1 General User Interface Aspects

33. The layout of the application should allow reduced learning and retention time.
34. The consistency of the following features must be maintained throughout the application: terms, layout, soft keys, colors (or reverted colors), sounds
35. The screen must be completely used (except when zoomed out)
36. In a List, Command List or Choice Group the items order should decrease per frequency of use
37. If a list is formed by selectable elements, it must be clear how to pick them (e.g. highlight the selected item)
38. Forms fields must be logically grouped (e.g., name and surname together, then address city, etc.)
39. All of the main features of the application must be selectable in the Main menu. For games they include: Continue (if a game session has been paused), New game, Settings (if any), High scores (if implemented), Instructions, About, Exit

5.4.2 Generic Feature Screen

40. The header/title (consistent with the terminology used in the Main menu) must be the title/header of the screen
41. It must be possible to go back to the Main menu from a generic feature screen. This is applicable also for the Settings screen (if any), High scores (if implemented), Instructions, and About

5.4.3 Settings

42. If the application is stored under Games menu, requests for lights, sounds, and vibration must adhere to the Game settings
43. If the application is stored under Applications, requests for lights, sounds, and vibration must adhere to the global phone settings
44. The current status of each setting must be clear
45. Each setting must have separate enable/disable functionality (e.g., Vibra, Backlight and Sound)

5.4.4 Instructions and About

46. The About screen must contain information about the developer and other generic information about the application. There should be consistency with the application's details (JAD)
47. Instructions must be present, useful, and complete. They include: purpose of the application, rules(if any), and use of keys (this may have a separate section)

5.4.5 Game Screen (for games)

48. Every screen (except for playing screen for a game) must contain information about the screen or the point of operation in the application to give indications of the actions being performed
49. The current status of the game must be clear and feedback must be provided (e.g. number of lives available, current level, current score)
50. When an event occurs, immediate response must be provided (e.g. the player selects a game enabled key or the enemy moves)
51. The rule/function of each element of the game must be clear (e.g. how is the player and the enemies represented and how they interact)
52. When another screen is displayed, the current game session must be paused and it should be possible to continue it
53. When the user goes to Main menu the game must be paused and also allow the game to be continued
54. When the game is over feedback must be provided. This will include: warning that the game is over, total score (if any), who won (e.g. if the game is played between the device and the player) and other game dependent features (e.g. level)
55. If the player registers his/her name, the score and the name must be available in the High scores section
56. When a game is over the Continue element must not be present in the Main Menu

5.5 *Delivery of the Application* [7], [8]

- 57. Both JAD and JAR files are needed
- 58. The application should not have any inconsistency in JAD and JAR files
- 59. JAD + JAR file size should not exceed 30kB (this varies with handsets)
- 60. The user guide and/or feature list of the application must be provided
- 61. Documentation must be accurate
- 62. In client-server solutions, the client software must be provided. The server software or access over the Internet must be available

Appendix B: Castle Quest Source Code
(Available upon Request)

Appendix C: NSF Proposal
(Available upon Request)