Champlain Regional College – Saint-Lambert Campus

The Desrosiers Mechanics Teaching Tool

Integrative Project in Computer Science and Mathematics (420-204-RE)

Final Project Report

Start Date: May 14, 2014

Submission Date: May 20, 2014

By Gregory Desrosiers

Project Producer: Amin Ranj Bar

Assistants for Programmer: Marie Pupo & Emily O' Donnell

College Program Director: Pedro Cabrejo

Executive Producer: Don Shewan

Produced in association with Champlain Regional College (Sherbrooke, Quebec)

900 Riverside Drive, Saint-Lambert, Quebec, Canada, J4P 3P2

Table of Contents

Introduction 3

Project Overview

- Story 3
- System Objectives 4
- Project Constraints and Scope 4
- Tools and Methodologies used by Developers 5
- Critical Project Events 7

Project Design

- Analysis, Definition, and Problem Comprehension 8
- Specific Algorithms 9
- UML Design 11
- GUI Design 17

Methods of Evaluation 32

Results: System Quality

- Developer Perception 35
- Objective Measure 35
- Developer Evaluations 36

Project Management 38

Conclusion

- Success on the Tasks 41
- Overall Degree of Completion 42
- Quantity, Difficulty, Quality (Learning Experiences) 43
- Developer's Comments 44

Project Credits & Copyright 44

As-Built System Documentation 51

Installation Instructions 53

FAQ 53

General Reference Guide 56

Introduction

This is the final report of "The Desrosiers Mechanics Teaching Tool," a GUI-based educational tool developed as part of a CEGEP course called "Integrative Project in Computer Science and Mathematics." The course had emphasis on project development using Oracle's JAVA programming language, several tools, and knowledge from past courses in the Computer Science and Mathematics program.

The program consisted of courses in three different concentration departments: computer science, mathematics, and science (in particular, physics, but with at least one chemistry course). Only courses from mathematics or science departments were eligible to be used as the material mandated for the projects.

This report is split into different sections: the overview of this project, the project design, evaluation methods, results, project management, and project conclusion.

The overview of the project includes the story behind the project, what were the objectives in developing this project with regards to the story or problem discussed, the constraints and scope, the tools used in this project, and the overall project events that occurred during development.

In Project Design, there is a section giving out details of the problem this project was designed to solve; in particular, the analysis of the problem, the actual explanation of the problem, and what was to be understood about it. Customized UML diagrams are provided, as well as the actual GUI for all possible states of the program, with one state for GUI with animated graphics or text.

The Results section provides the perspective of the developer, the objective measure, and the developer evaluations. Project management provides a time line of the development phase, along with a development log that has been worked on during the entire development period. Finally, a project conclusion provides details on the success of the tasks, the overall degree of completion, the quantity, difficulty and quality, as well as the overall comments provided by the developer.

***NOTE*: Several sections of this report are incomplete. Having to write a very long report with no partner is something I am not completely used to; in addition, because of some exhaustion I had from the entire semester, I took some time to relax myself before this report was worked on. In future reports written by me, I will make sure to give myself ample time to write them so that way they will be complete by the deadline. There's also more proper organization, working with partners, and managing time and stress properly.**

Project Overview

Story

A student in their first semester of their new program at CEGEP is exposed to a physics course involving motion and forces for the first time. They try their best to learn new concepts of the course and exercise their knowledge as best as possible, but there may be a catch because of several different circumstances; trouble paying attention and listening, not understanding a very complicated problem, weak concrete understanding, or simply the degree of special needs, if the student has them. It always varies from student-to-student.

There are several resources available at Champlain College Saint-Lambert where the student can access and get help in succeeding this physics course. Tutoring, the Learning Centre, adapted tutoring, teacher discussions (that is, the

student sees the teacher during office hours), websites, and other books. Other colleges should offer aid as well, but this is not always the case. In fact, there are limits to the accessibility of those resources, and so under this serious circumstance, the student may run potentially into lower grades and possibly a failure of the course.

To add a little flexibility to this accessibility problem, a school project has been developed to assist the student, as long as they have access to a computer with Internet. This offline program has been designed in such a way that it aids the student in their learning process before tackling the most important elements in any physics courses; tests and the final exam. In fact, students from the International Baccalaureate Program, experienced physics or mathematics students, or even teachers, should use this program in assisting other students with reviewing not only concepts and mathematical work, but also enhance the understanding with carefully-developed simulations.

In general, the project is to help students who have limited access to other resources, use it for fun, or simply as an additional tool to practice their knowledge for upcoming quizzes, tests and exams.

System Objectives

Certain objectives have been put forward in developing this project:

1. Plan and integrate a graphical user interface, complete with buttons, graphical components, and interface management.
2. Implement animation for appropriate features of the program to be developed.
3. Create space-conserved and time-conserved algorithms in the project code.
4. Build features for the program that allows the user to access different learning phase environments such as simulations and exercises.
5. Create and implement subject material, including simulations, based on the course as much as possible.
6. Make the program user-friendly, including appropriate text, carefully-positioned GUI components, and possibly keyboard shortcuts.
7. Remove unnecessary code from the program to save memory space for installation and use in other computers later on. (as part of the finalization process)
8. The program has to be cheap and simple to run.

Some objectives are not mentioned here because these objectives are more emphasized on how the user will use the program, rather than both the usage of the program, and the objectives of the developer emphasizing the code.

Project Constraints and Scope

There were several project constraints discovered before and during development:

- Time limit (development had to be completed in two months)
- Assignments from other courses in the semester and how these assignments were worked on, since some assignments used approaches that took too long and thus harmed production of this program
- University preparation and application
- Tutoring and meetings with college supporters
- Interest in development (as a result of self-esteem, unnecessary compulsions, unable to handle pressure)
- Working alone (not as advantageous as originally thought of)
- Some linear and irrelevant development approaches (ex. Working on the performance utility first instead of the GUI and subject material)

- Overall health and feeling of developer

There were also project constraints at the end of development:

- Not all six simulations of the only motion and forces chapter, Chapter 2: Motion in Two and Three Dimensions, were implemented. Only two simulations were coded and added to the GUI tool.
- Only one exercise was added to Exercise Mode (Chapter 2 – Section 1). In other sections of Chapter 2, the only chapter added to this mode, the general GUI is shown with drawn text.
- No quizzes were introduced in Quiz Mode; only GUI components are presentable.
- Only one image was implemented in one of the lectures added to Lecture Mode.
- Extensive testing has not been done in the program, so there are potential cases for the program to crash, to display glitches, or several other bugs. The promise for an exception-free program has not been kept in the final version of the project.
- The Loading Utility Progress Label is not completely friendly because it should also show a percentage which would be a ratio of how many bytes need to be loaded and how many bytes have been loaded so far. However, there was no algorithm implemented in calculating the memory size of objects, especially inherited components, and so the label only displays the current stage and its associated message.
- The program was not tested on different operating systems to ensure portability.

Originally, the scope of the project was to include material from thirteen different chapters of Richard Wolfson's Essential University Physics, 2nd Edition. However, because of the serious time limit and various circumstances, the material was cut down to two chapters: one involving what a student should know when learning physics, and one involving motion in two or three dimensions.

That, in turn, was some material to be incorporated into the program in four different ways. It had to be introduced as a visual lecture with diagrams where the user has the freedom to go between pages while reading along to try to understand again what was misunderstood, and can understand the idea from both the diagram and its associated text. It also had to be introduced as some simulations where the user can see what's happening, following this principle, in action; values can be edited, and in some simulations they can be animated. A set of exercises was a requirement as well, because it allowed the user to test their knowledge of what they learned, check their answers, and figure out (with some help, if necessary) how to get the actual answer of the question. Finally, the program would have the ability to give the user a quiz with an optional time limit where they can test their knowledge without looking at the subject material or the answers prior to completion.

In the final version of the project, only two simulations are included: one involving vectors, and one involving uniform circular motion with integrated animation. As discussed before in project constraints, only one exercise based on vectors was added, and no quizzes were featured.

Tools and Methodologies used by Developers

Two important tools have been used in developing the program itself: the Kepler release (2013) of the Eclipse JAVA EE IDE for Web Developers, and NetBeans IDE 7.3.1. The subject material written down before implemented into the program was done using Microsoft Word Starter 2010. Several images have been added to the

program as well: the main logo of the program, and some others, was designed using Adobe Photoshop Elements 9, and the demo lecture image was designed using Microsoft Paint on Windows 7 Home Premium.

The entire program has been developed using the 45th update of JAVA SE Development Kit 7, or v1.7.0.45. No external libraries have been downloaded nor used in the development phase, so as long as the user has the latest version, or at least the version of JAVA being 1.7.0 or better, the program should still run on compatible operating systems.

Most of the code has been done on the Eclipse IDE, including package and source code file organization, comments, and checking for compilation and runtime errors. NetBeans IDE 7.3.1 has been used to place GUI components in desired spots; this was done first to make sure that the components was placed as desired, then the generated code has been put in place on Eclipse with a few modifications.

A few software methodologies have been followed as part of the program development.

It has been developed following the object-oriented programming paradigm using the rules specified by JAVA and how the code is compiled and run at runtime. The source code is split into multiple files in the subfolders of the source code folder and the entries in the source code not only interact each other, but are also used for specific components and implementation.

For pre-development and post-development, the Unified Modeling Language framework has been used; in particular, the diagrams for the data entries and objects have been designed to reveal rough names of the fields and methods. Specific arrows have been used to denote, for example, inheritance, interface implementation, where the data entry is based on the source code package, and both composition and aggregation.

Because the constraints were a serious issue during development of this program, the project involved more rapid application development. However, there are several inadequacies followed during development, because it was the first time this development approach was used.

This involved iterative and incremental development; after initial planning, raw builds of the main GUI and subcomponents have been made overtime. Rough testing was executed among the builds to make sure that the raw components were working properly. Some revision had to be done for the components to fix up most of the problems encountered, and there was also specific planning and changes to the overall features. Essentially, some features had to go through iterative and incremental development several times to make sure that it came out just right within the constraints there were during development.

There was only one prototype created, although several revisions have been saved during development because of the needs of backup. The subject material to implement in the program was not very emphasized as it should have. The technology used to develop the program had a low cost, but yet the program was somewhat developed as a prototype with additional features. However, the program developed is not production software. Ease-of-change and breaking development down to step-by-step was emphasized because the code has been split up into multiple files following the object-oriented programming paradigm, where whenever a compiler or runtime error occurs, it's easier to find the line where the program crashed. Finally, active user involvement was not practiced because the testing of the software was mostly done by the developer; when the raw program was shared on Facebook to call in for testers, there was no feedback from them.

Critical Project Events

Below is a point-form list of the setback events that occurred as development of this project was in progress:

- There were not enough details in the original assignment documentation (Phase 1) on how a report should be organized, what format it should be, and whether or not a PowerPoint is mandatory.
- I did not take the time to prepare myself with some knowledge of JAVA during the Christmas break because I was too absent-minded and decided to relax too much instead, such as playing Final Fantasy VII for a long time.
- I did not develop a consistent and realistic schedule to time frame development on this project, as well as work on my other subjects at the same time. The schedule written in the original documentation was not developed enough because the stages of development were too general, and thus it was completely ambiguous on whether the GUI would be worked on first, or developing the code that follows the properties of a Mechanics chapter.
- I did not work with a partner, because one of the errands I would have to do would be to go to my partner's house, and vice versa, which does bother my schedule. Since I do push myself too hard in trying to get all of my homework done, yet spend parts of my weekend relaxing from being worn out and tired over the week, I would have been in trouble. In addition, I did not want to run into stress moments with team effort and coordination like what happened in Electricity & Magnetism with my partner, even if we were both excellent with labs.
- I was not being realistic in my planning and how to develop the program, because I paid too much attention in creating a perfect application and not thinking about things that can happen unpredictably. I did not paid attention to my limits as well. I was exercising myself too hard as a result of getting 100% in Graphical Environment Programming.
- In addition, I wanted to push myself so hard on this project because then I would be able to get to the top more easily when I enroll at University of Waterloo. I wanted to become so successful in this course so that way I can do excellent at university; it helps with building both educational and professional experience for the co-operative system I will be placed in. Even so, I did had a psychological problem where I wanted to become better than other students, especially Simon Cochrane.
- This project was an exercise for me to work on a JAVA-based computer game later on. I would go through learning new packages and tools so that way I would be able to work on a game for the Google Android and sell it on the market. Hence, it would help me with my own self-esteem and skills in software engineering and game design, which is what I absolutely need to become a computer game producer and director. It would help the finances as well.
- During the Reading Week, I did not get feedback in advance so that way I can cut down most of the material I wanted to include and thus save time in finishing the project. Therefore, I invested too much time in writing the classes that were designed for the independent topics in all of the Mechanics chapters I wanted to work on.
- Between April 4 and April 7, there was supposed to be a lot of work done in order to catch up progression on this project. Unfortunately, it was taken over by French because I had to read a 15-page reading and understand as much as possible, since it was due for an exam on May 7th. I did had a lot of time at the beginning to prepare for this, but I had no choice except to work on it frantically because my French tutor assigned me to do so, and I wanted to listen to her. Because French is an extreme difficulty for me, it took me two full days to get the assignment done, as well as finishing off a few exercises in the exercise book *En avant la grammaire*.
- There were a series of out-of-project tests I had to complete in order to pass the corresponding courses. The most important course I had to pay attention to was Linear Algebra, because even though I worked on my

other courses sufficiently enough to get a pass, I needed to get at least 80% in the course in order to satisfy my prerequisites for Waterloo.

- Because of the Linear Algebra test taken place on April 14 as well as a French oral presentation for the volunteering project, the weekend of April 12 and 13 was lost in progress of this project too.
- Instead of more professional-like simulations, to save time, only simple animated simulations have been developed.
- I did not developed a more concise schedule after my mark for the original proposal was given, because I was not being realistic enough to see if it's specific and formal enough. There was a serious lack of communication between me and the professor, because I was not really aware that I had to be much more realistic in terms of planning a project. In addition, the professor clarified to me that some computer program design documents contain over 100 pages.
- On May 3rd, when I was supposed to work on my two only lectures, there were problems with my two simulations I had to fix, as well as some listeners, and so it took me several hours to fix them.
- On the weekend of May 10th, it was not realized that comments of the source code had to be put up. That only came on May 13th, when I was planning on working my Newsactivism final project and just realized that this project had to be worked on before the presentation in case the teacher wanted to see the code after the presentation. And thus, the source code has been worked on to include comments for organization.

Project Design

Analysis, Definition, and Problem Comprehension

Because the problem discussed in the story involves a student who is struggling to pass motion and forces, the kind of program that is to be developed is a full GUI educational tool. In order to succeed better in the course and potentially pass, they have to not only take the time to better understand the given course material, but also to exercise their knowledge and get some proper feedback. That way, when the student has their exam, they can solve the problems more correctly; that is, they spend time recuperating the required subject material and practicing it, as they are not allowed to have their notes in class.

To add more to the difficulty of this project, it is highly possible that the student may run into trouble understanding written material, even if diagrams are to go with it. As such, one possible way of learning the same material in a more direct approach is by using simulations where they can refer to the equations to see what happens under a given set of conditions. Animation would be incorporated to see how properties can change over time, as physics do involve the use of time in change of certain properties and quantities.

The definition of this problem is that the student needs a computer program to help them with their motion and forces course.

There are four specialized stages that the student would go through before taking an actual test or exam: reading and following lectures, viewing and manipulating simulations, doing exercises, and testing their knowledge with predefined quizzes. The student is not limited to following the stages in one path; the student is also given some freedom to what they can do to practice the material they are having trouble with, even if they would go through some material in one or more stages at least once. That way, if the student, for example, encounters a problem with vectors in doing an exercise, they can go back to their notes, concepts of the material, or do the associated simulation again to clarify the ideas of vectors. They can do so at their convenience and as many times as they like, provided that they are respecting the date for a quiz, a test, or a final exam.

Therefore, the program that resolves this problem consists of four different modes corresponding to the four stages of learning material before a full exam, all with integrated material from the course.

Specific Algorithms

In this project, only one specific algorithm has been used from external resources. It is the timestep algorithm for the animated simulation on uniform circular motion. This algorithm does not solve the problem that this program is designed for; it's only for one of the individual components; no specific algorithm has been used to solve the problem analyzed in the scenario.

It can be viewed from the following site: http://gafferongames.com/game-physics/fix-your-timestep/

Uniform circular motion involves the use of time steps to create the animation needed so that way it is approximated in certain circumstances in real life. Of course several different environmental factors are ignored, but the simulation has more emphasis on the general equation of radial acceleration for circular motion.

The simulation should be running at a consistent rate, even though sometimes the CPU may experience a heavy load from at least one thread and the performance is hampered. In addition, we have to take into account the different factors presented for each computer and try to have a timestep algorithm as generalized as possible so that way the changes in position of a vector, let's say, remain constant. Also, we want to try to sync the operations of the animation and its elapsed time to real time as much as possible. Thus, the timestep algorithm used in the animation for the circular motion simulation is based on the last block of code the author of the site provides.

The timestep algorithm is described as follows in psuedocode:

Initialize a float variable *currentRadialAcceleration* with the value of the simulation variable *tangentialVelocity* to the power of 2.0, divided by the value of the simulation variable *radiusOfMotion*.

Initialize a float variable *originalRadialAcceleration* with the parsed value from the text of the simulation control panel reference variable *tangentialVelocityTextField* to the power of 2.0, divided by the parsed value from the text of the simulation control panel reference variable *radiusTextField*.

Initialize a constant double variable *dt* with 0.01.

Initialize two double variables *t* and *accumulator* with 0.0.

Declare a double variable *currentTime*.

Assign the variable *currentTime* with the double value of the value returned from the method *nanoTime* of the System class, divided by 10^9.

While the thread is not flagged to be released,

> Initialize a double variable *newTime* with the double value of the value returned from the method *nanoTime* of the System class, divided by 10^9.
>
> Initialize a double variable *frameTime* with the difference of *newTime* and *currentTime*.

Assign the value of *newTime* to *currentTime.*

Add the value of *frameTime* to *accumulator* and assign the sum to *accumulator*.

While *accumulator* is greater than or equal to *dt*

> Assign the simulation variable *tangentialVelocity* with the sum of its value and the product of the simulation variable *tangentialAcceleration* and *dt*.
>
> Assign the simulation variable *angularDisplacement* with the difference of its value and the product of, the quotient of the simulation variables *tangentialVelocity* and *radiusOfMotion,* and *dt.*
>
> Initialize a float variable *factor* with 0.25.
>
> If the simulation variable *tangentialVelocity* is greater than or equal to zero
>
> > For *factor* between 0.25 and a value where *factor* times 64.0 is equal to or greater than the simulation variable *tangentialVelocity*, assign *factor* with its value plus 0.25.
>
> Else
>
> > Negate *factor*.
> >
> > For *factor* between -0.25 and a value where factor times 64.0 is equal to or less than the simulation variable *tangentialVelocity*, assign *factor* with its value minus 0.25.
>
> Assign the variable *x2* of the simulation reference variable *tangentialVelocityLine* with the quotient of the simulation variable *tangentialVelocity* and the absolute value of *factor*.
>
> Assign *currentRadialAcceleration* with the float value of the quotient of the simulation variable *tangentVelocity* raised to the power of 2, and the simulation variable *radiusOfMotion*.
>
> Set the text of the simulation control panel reference variable *calculatedInstantaneousRadialAccelerationLabel* with the formatted string of *currentRadialAcceleration,* returned from the simulation reference variable *radialAccelerationStringFormatter*.
>
> Assign *accumulator* with the difference of its value and *dt*.
>
> Assign *t* with the sum of its value and *dt*.

Assign the simulation variable *timeElapsed* with the sum of its value and *t*.

Set the text of the simulation control panel reference variable *calculatedAverageRadialAccelerationLabel* with the formatted string of the float value of the average of *originalRadialAcceleration* and *currentRadialAcceleration*, returned from the simulation reference variable *radialAcceleraitonStringFormatter*.

Repaint the viewing panel *simulationViewingPanel* of the reference variable *Application.simulationPanel*.

Assign 0.0 to *t*.

Try to put the thread to sleep for the value of the simulation variable *frameDuration*. (16 milliseconds)

Catch any exception while trying to put the thread to sleep and assign it to *ex*.

Print the stack trace from the reference variable *ex*.

The actual coding, where this psuedocode comes from, is in the source code file Chapter2Simulation6Task.java of the tasks.simulations.chapter2 package.

This timestep algorithm is not heavily based on the size of input, because when the animation is running or paused, the input is kept constant. Instead, it is heavily based on the overall performance of the CPU.

There are two different loops in the inner loop that changes the size of the tangential velocity vector, and tangential velocity would change based on the inputted tangential acceleration. The algorithm does not complete one iteration at a constant time because changes in thread operation and the nature of the CPU could cause one iteration to be temporarily shorter or longer. But the algorithm does not follow a generalized mathematical function either, because even though there is a partial or one iteration to complete when the Boolean variable *releaseThread* is true, the user can stop the animation at any time using the start/stop trigger button.

In addition, the timing algorithm adjusts itself to how many calculation iterations are to be completed before rendering based on how long it took to complete the previous iteration.

Therefore, the time complexity of this algorithm is O(1).

UML Design

***NOTE*: This section is incomplete.**

There is no UML documentation provided for those diagrams that are formatted the same way as the package and class summaries described by Oracle's online documentation. However, because the source code files for the program are contained in the directory this report is saved in, some documentation comments, which are used to generate the documentation, are available in the source code. All source code files contain a block of comment at the beginning holding the information regarding what the classes in the file are supposed to do, the date of writing them, my name, a short disclaimer for future use, and a copyright notice.

Here are a few UML diagrams provided as reflected upon this program:

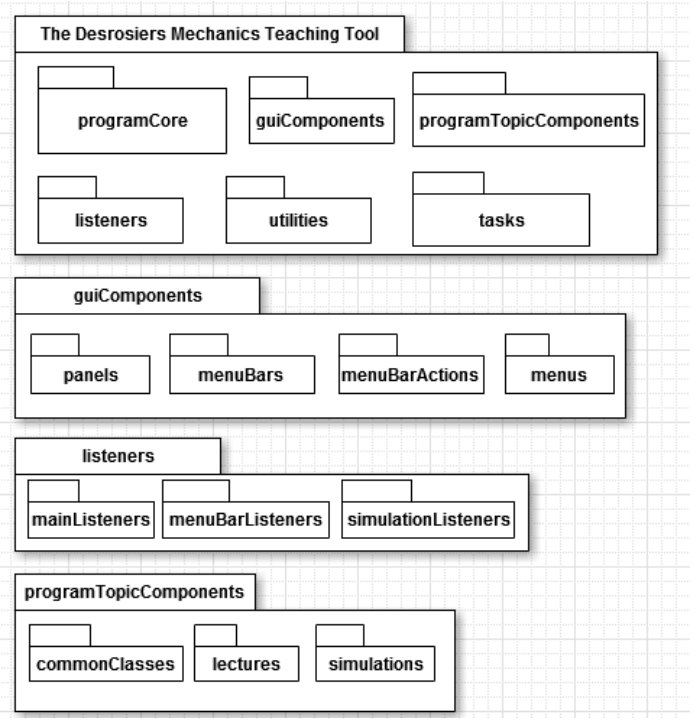

Fig. 1: Package Organization

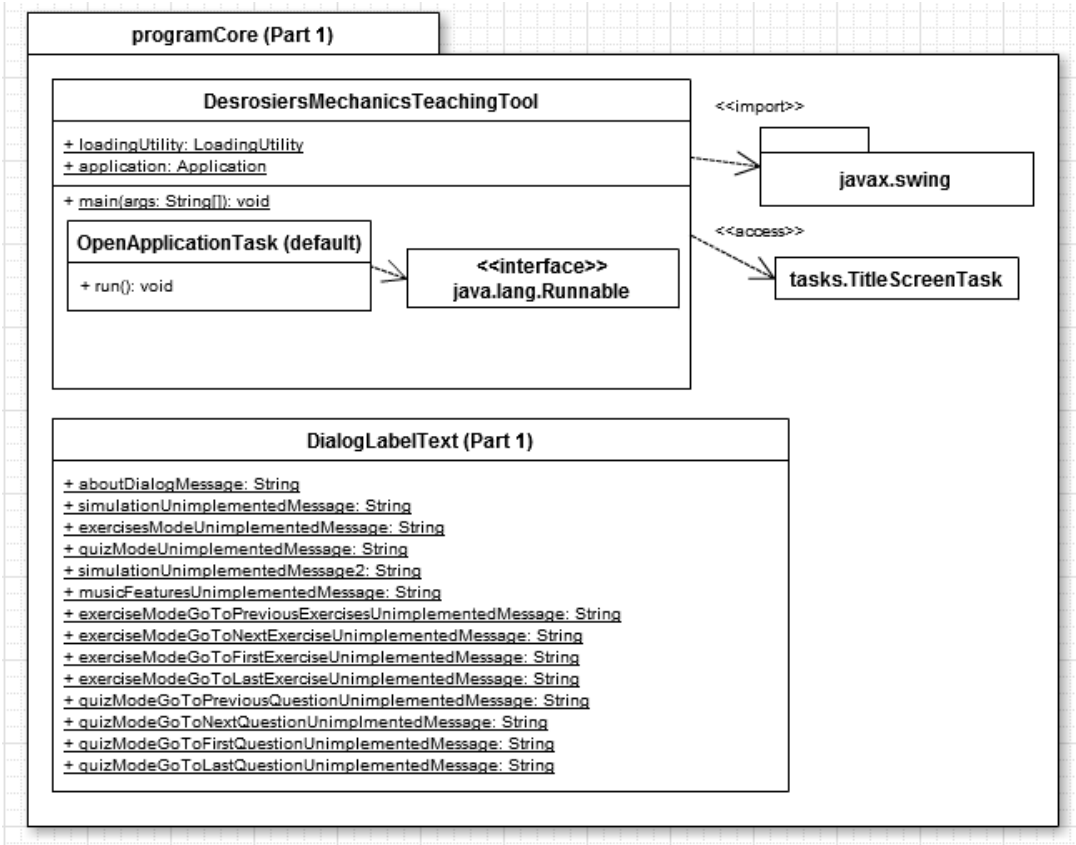

Fig. 2: programCore Package, Part 1

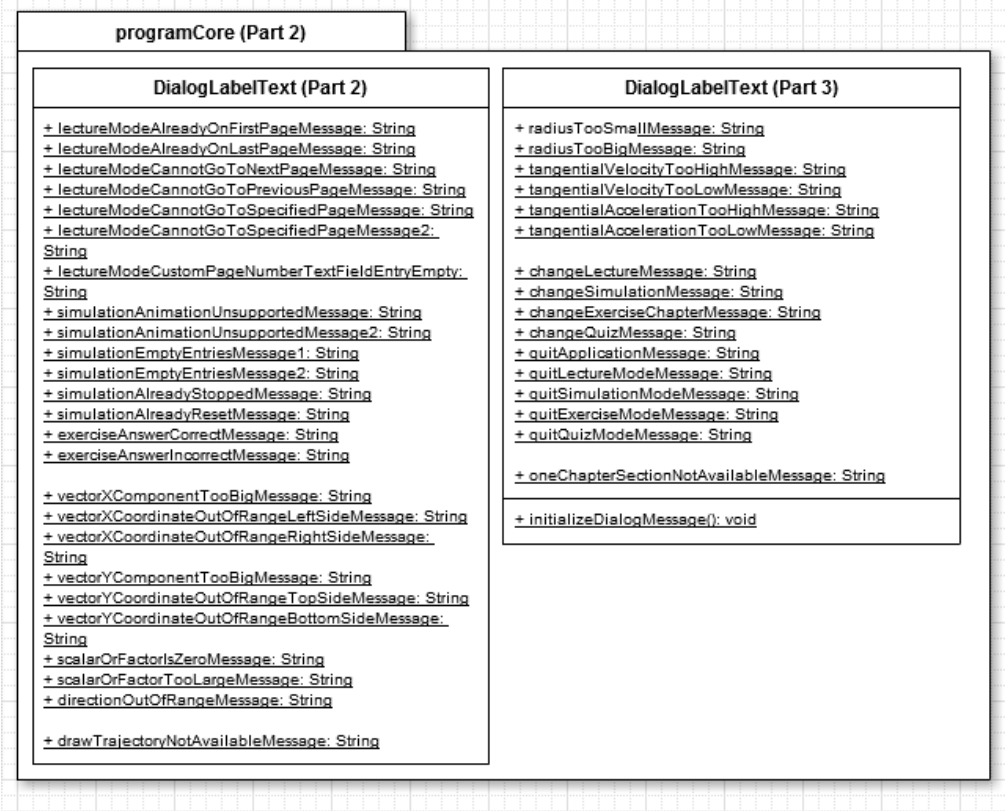

Fig. 3: programCore package, Part 2

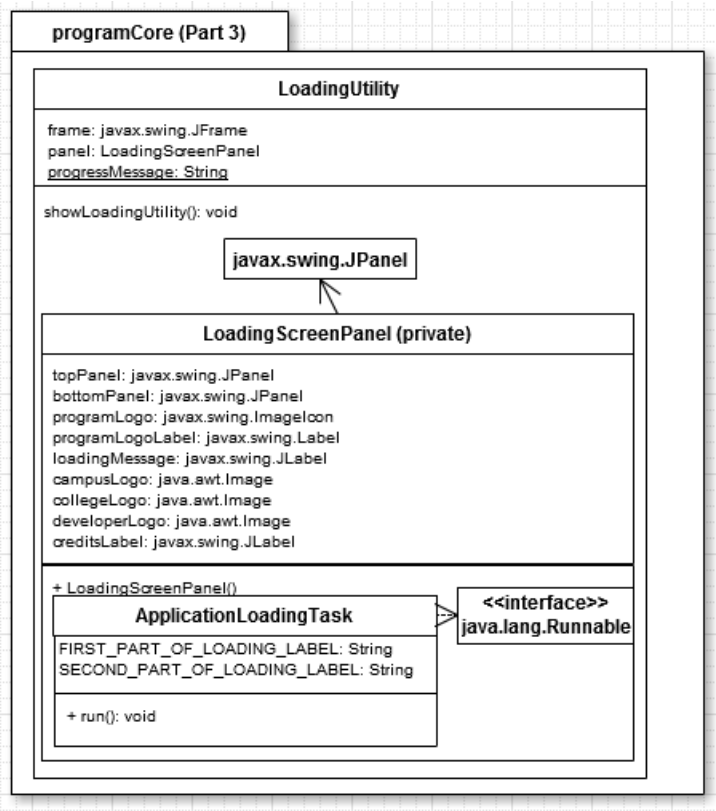

Fig. 4: programCore package, Part 3

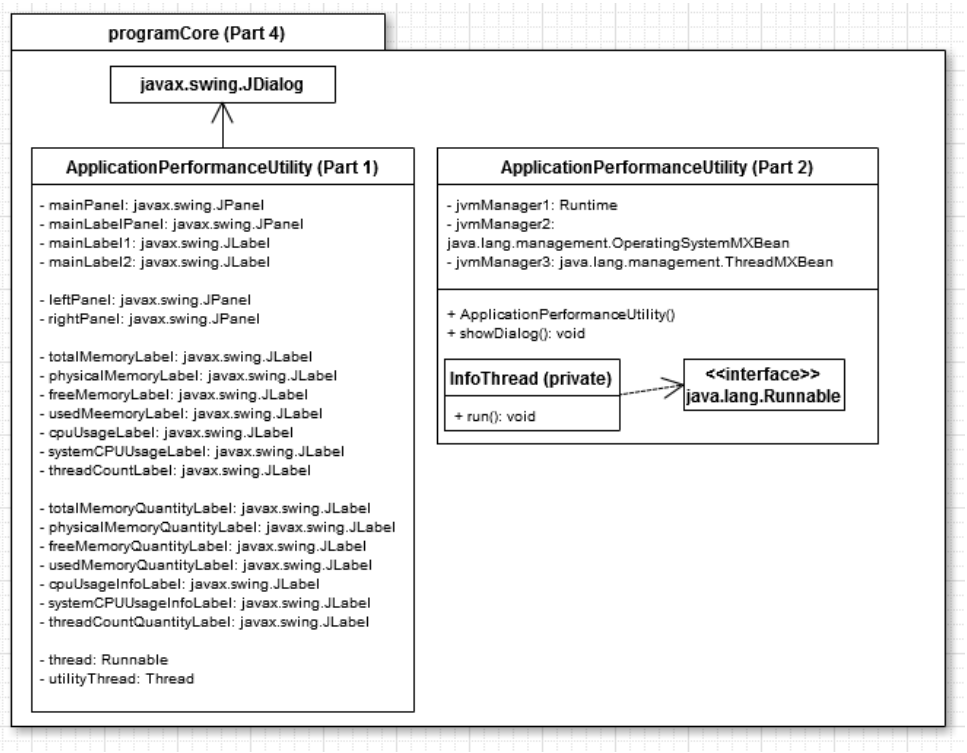

Fig. 5: programCore package, Part 4

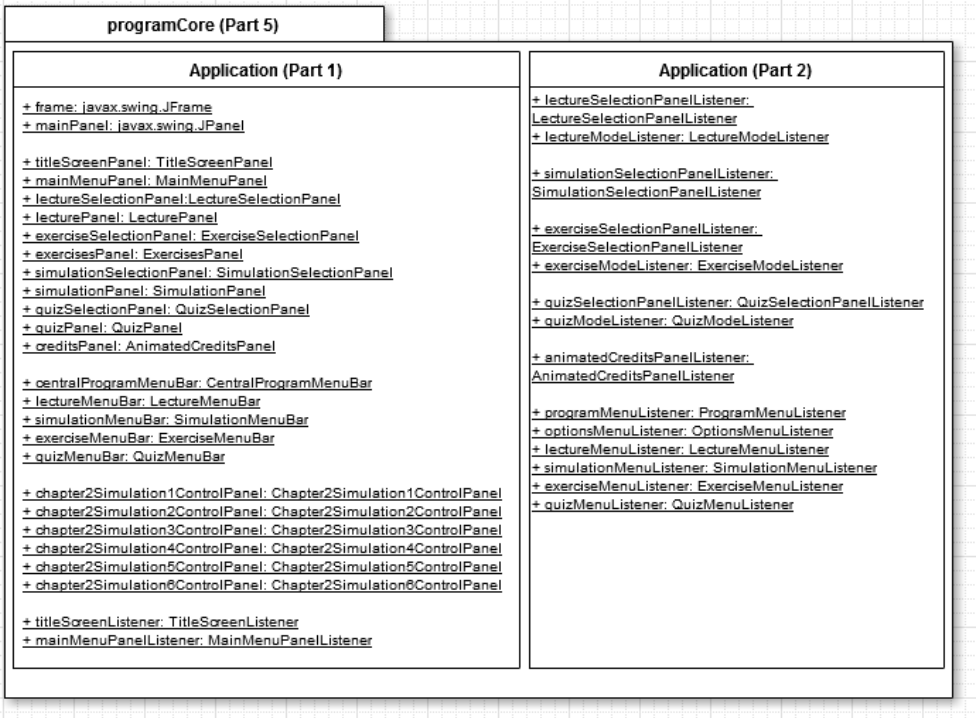

Fig. 6: programCore package, Part 5

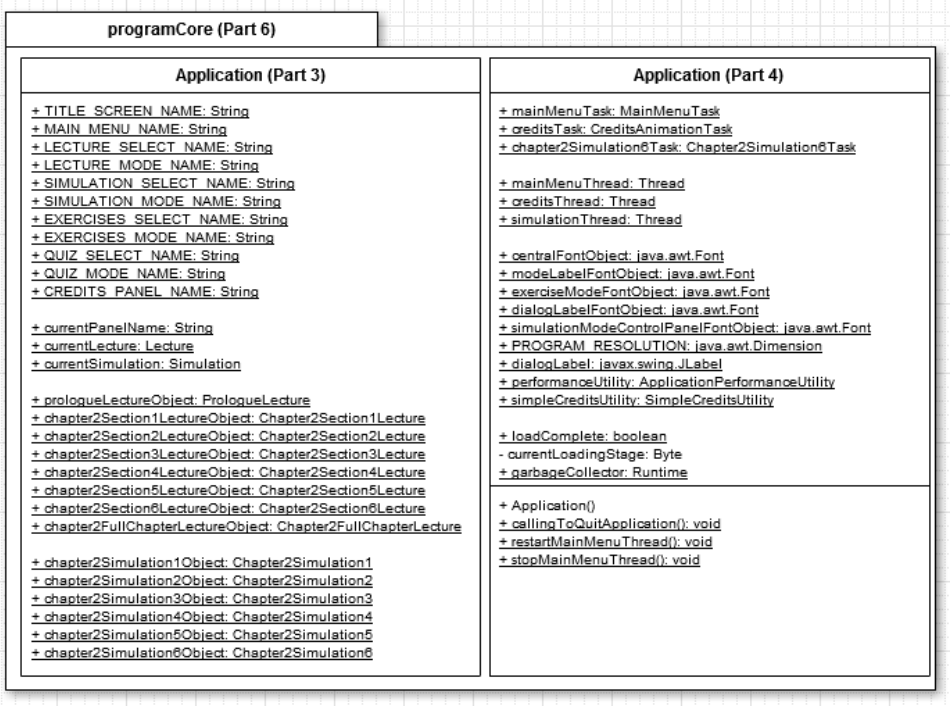

Fig. 7: programCore package, Part 6

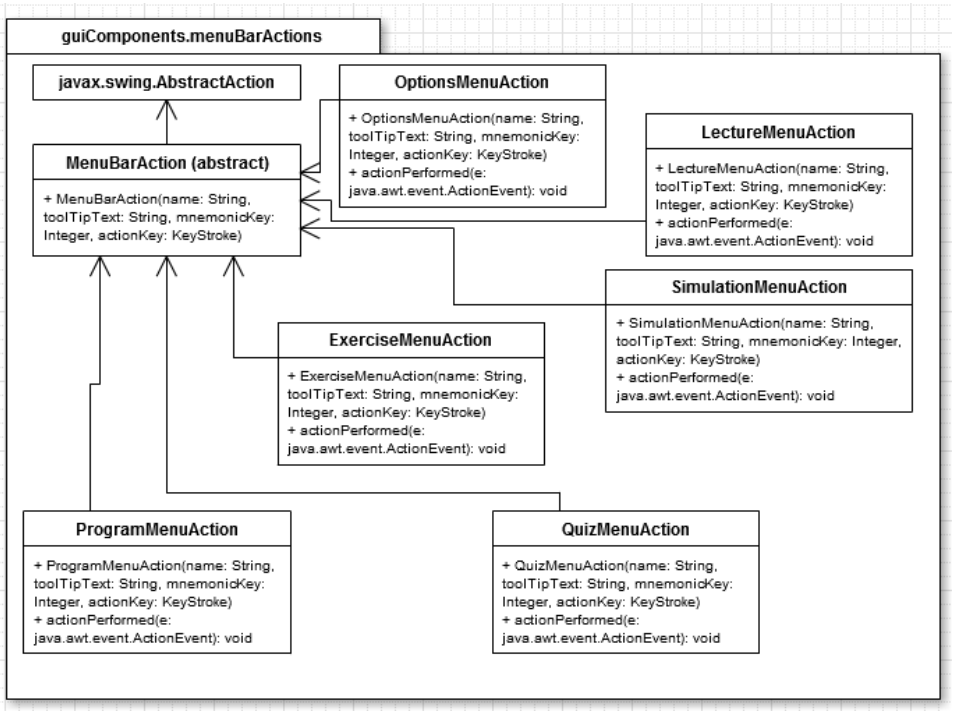

Fig. 8: guiComponents.menuBarActions package

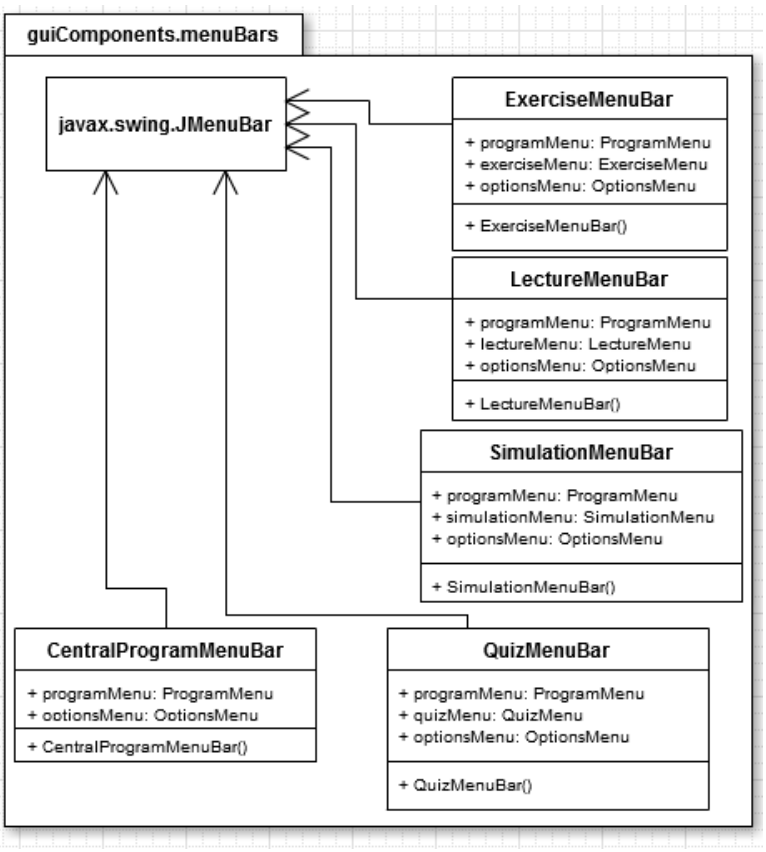

Fig. 9: guiComponents.menuBars package

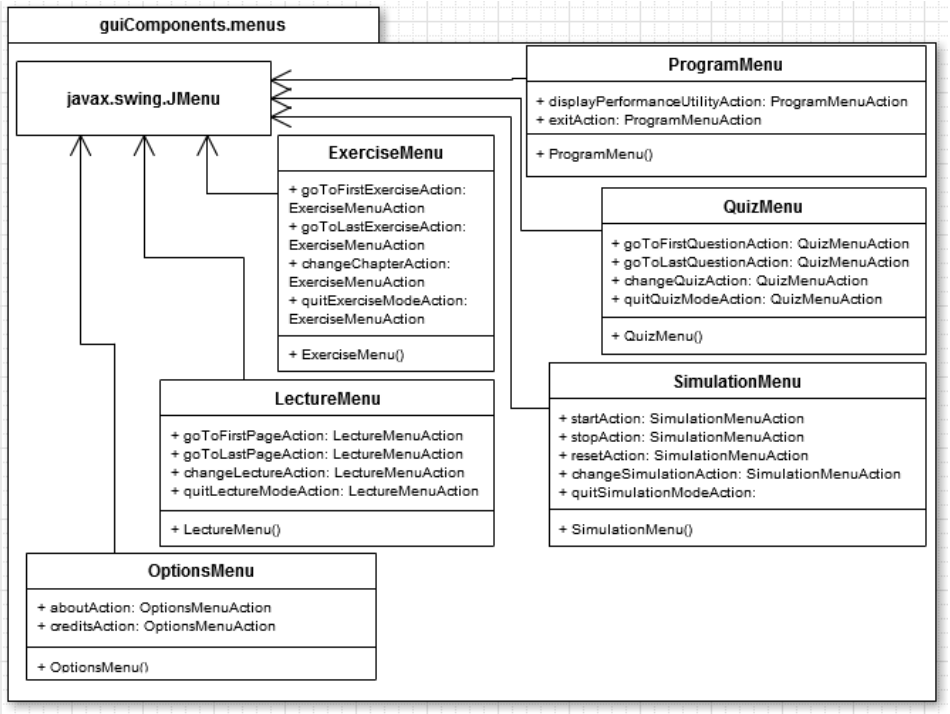

Fig. 10: guiComponents.menus package

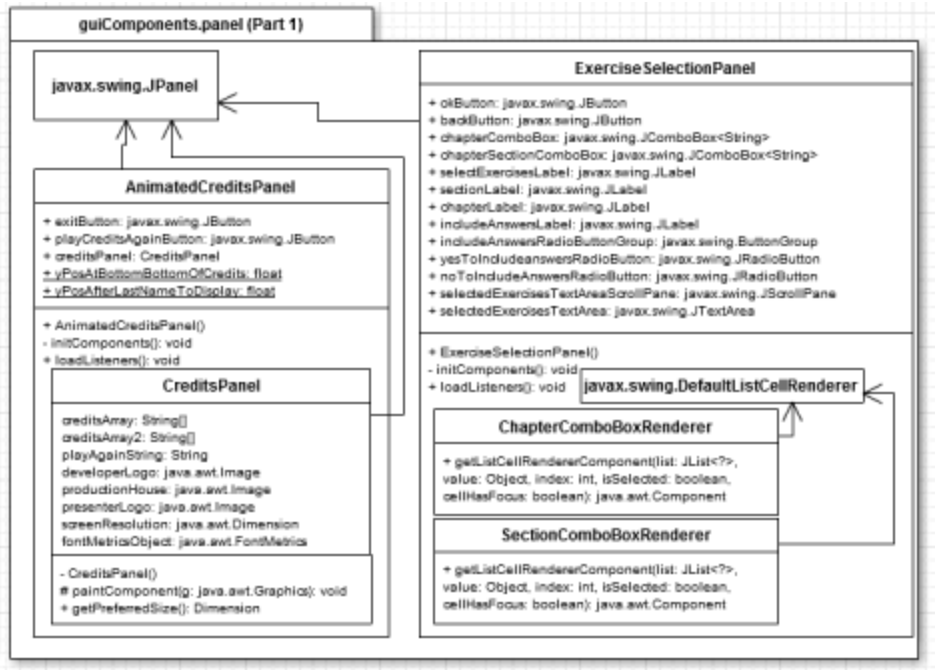

Fig. 11: guiComponents.panel package, Part 1

GUI Design

Shown here is the actual GUI design of the program; in each photo, there is an explanation to why the components are arranged this way.

Fig. 12: Splash Screen

It was originally expected that after the subject material is implemented in the program, it would take a long time for the program to load. It's not the case for efficient and quick PCs, but it's been developed anyway. The user usually has more attention with components that are on the center of the screen; in this case, the loading message is displayed close to the center, while the other graphics are displayed this way. Images and a copyright label are added, since some professional programs, like Adobe Photoshop Elements 9, often show their corporate logos, copyrights, and the staff involved.

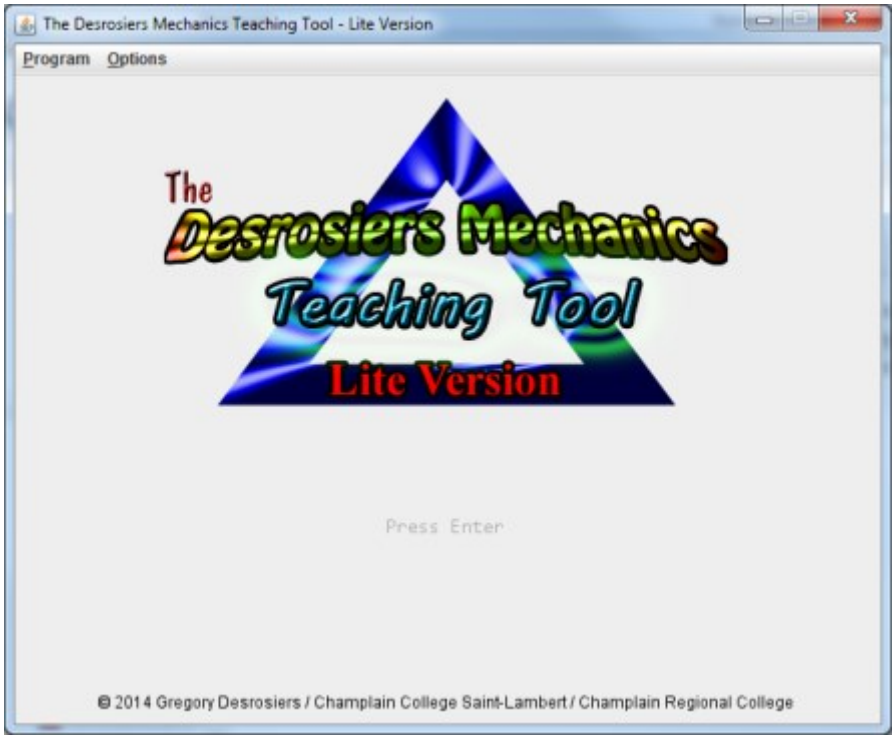

Fig. 13: Title Screen

Because the tool was originally proposed to have an interface somewhat like a video game, this animated title screen is added to the program. This is the usual way to present a title screen for a video game; there are other ways to arrange the screen, but the most common is center.

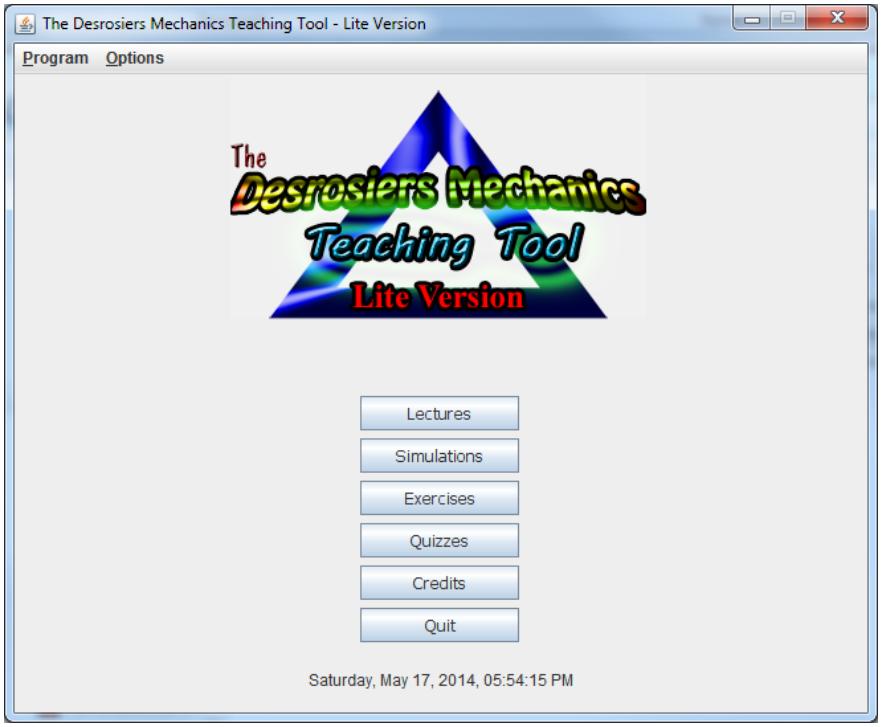

Fig. 14: Main Menu

It was proposed that the user has the flexibility to choose a mode at their convenience, as well as know what is the current date and time. So in this panel, six different buttons are placed as a centered column. These buttons are arranged this way because it is more organized than having a small table of buttons, or lay the buttons side-by-side horizontally.

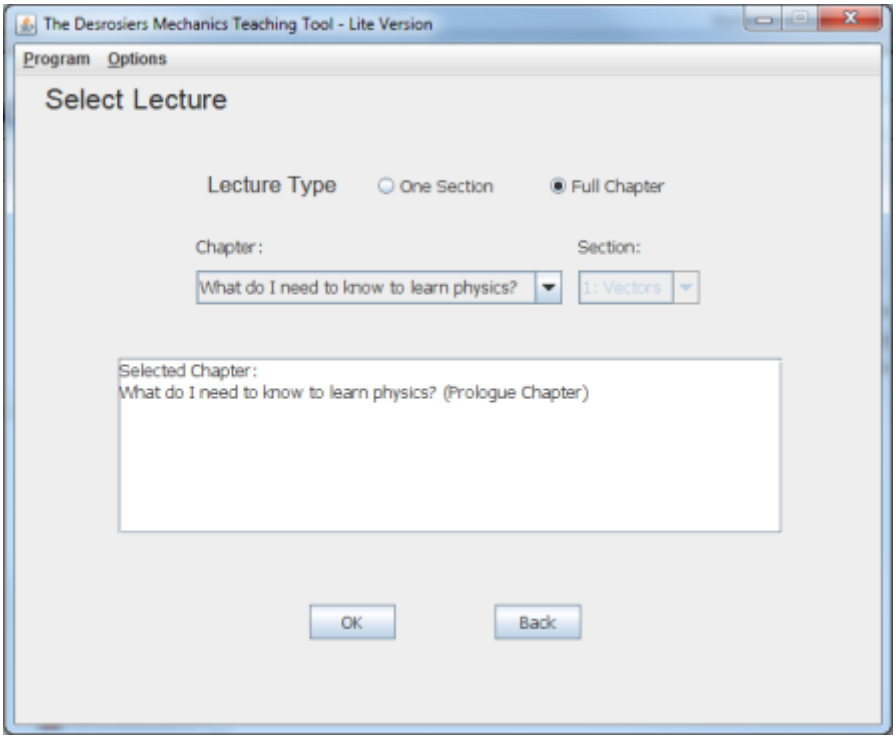

Fig. 15: Lecture Selection Screen

A lecture involves at least either a chapter, or a section, of physics material. The user can decide for themselves what chapter, or what chapter section, they want to read upon. There is a text area below the most relevant components to show what is the lecture selected before the user clicks on the OK Button. The user can also go back to the main menu using the Back Button.

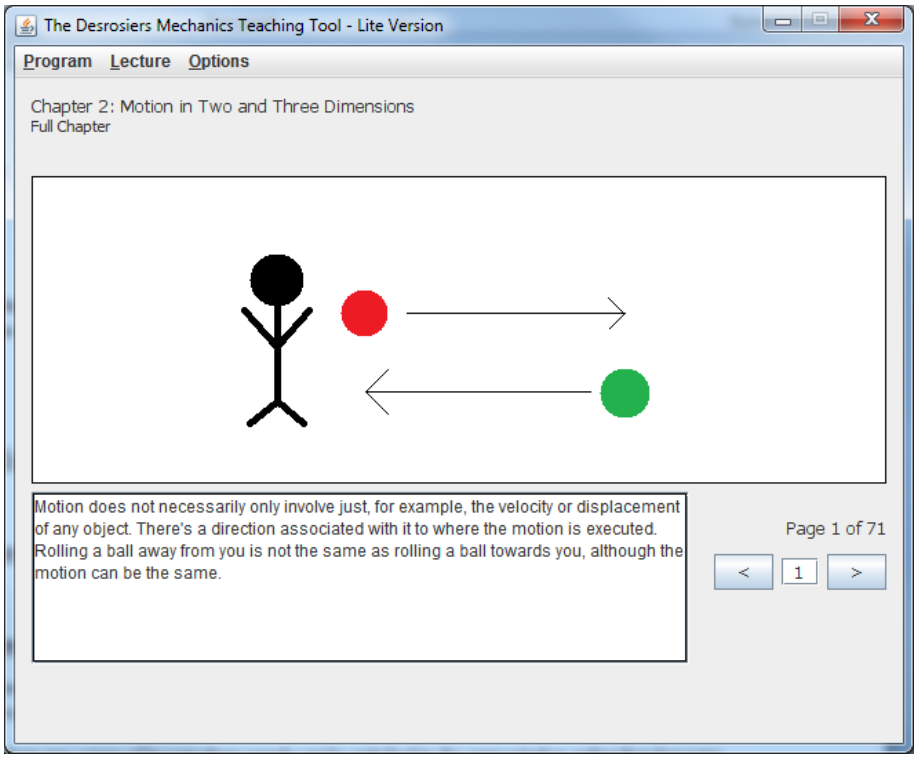

Fig. 16: Lecture Mode (with demo image)

The Lecture Mode has its GUI set up this way because the user tends to focus more of what's in the center rather than its surroundings, and it would want to see something that's graphical and not full of text, sort of like an in-game cutscene. As such, the viewing panel is played at about the center of the lecture mode panel, and the associated text is underneath it. For control, two buttons and a text field let the user roam through the pages of the lecture.

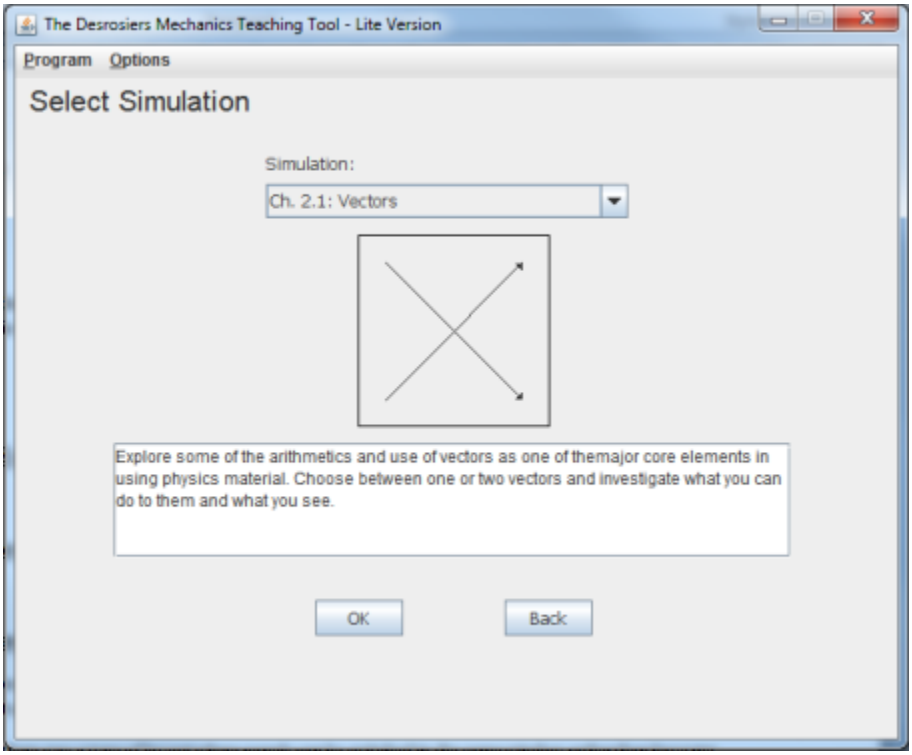

Fig. 17: Simulation Selection Screen

The user wants to not only have the freedom of selecting a simulation, but also to see what it looks like prior to running it. As such, a panel below the combo box shows the user a preview of the simulation as well as a simulation description through the text area directly below.

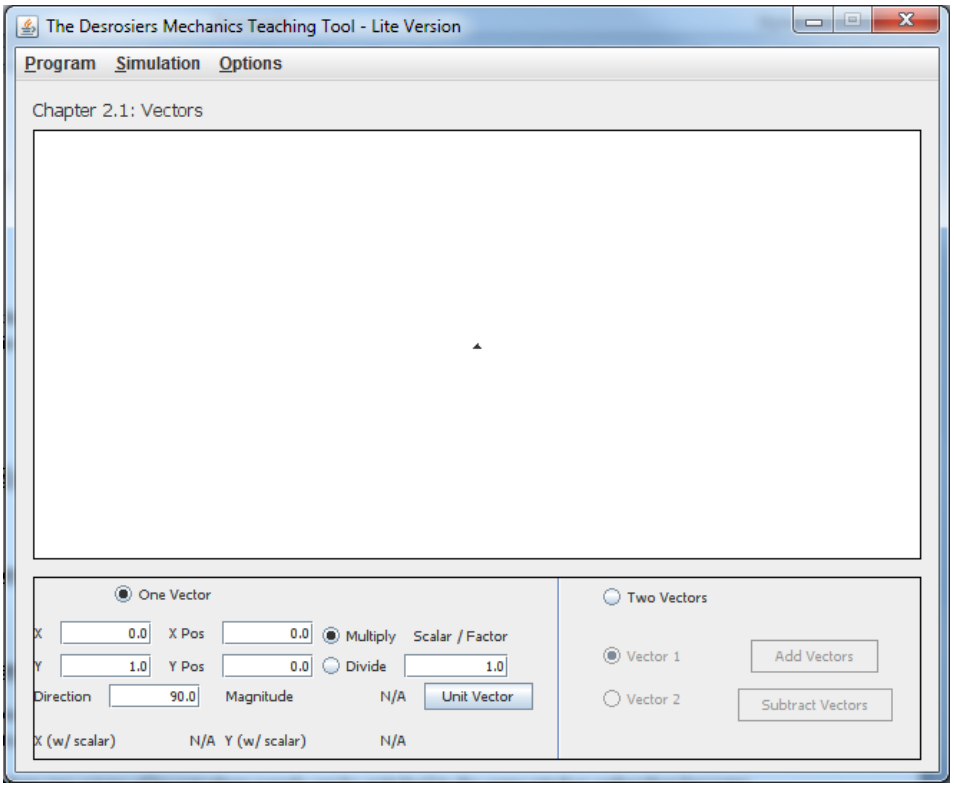

Fig. 18: Simulation 1 – Vectors

Following the principle of putting the most important component in the center, the control panel at the bottom is open to data entries that can be modified with the text fields and the Multiply or Divide radio buttons. This simulation is basically to represent the concepts of vectors by using one or two vectors as examples.

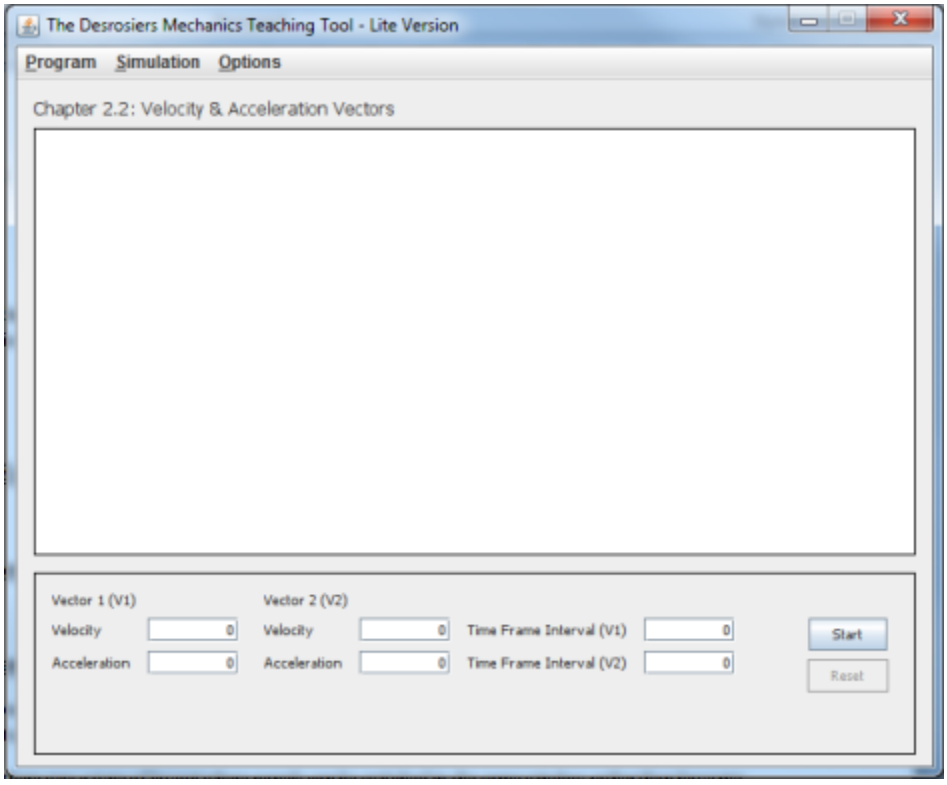

Fig. 19: Simulation 2 – Velocity & Acceleration Vectors

This unimplemented simulation was supposed to consist of two vectors, one above the other, starting from the left side of the viewing panel. After setting the fields for the one-dimensional vectors, when the user clicks the start button, there were to be two pairs of vectors painted on the screen. The acceleration vector is to move along the velocity vector overtime (during animation) so that way the user can see what's happening to the velocity vector.

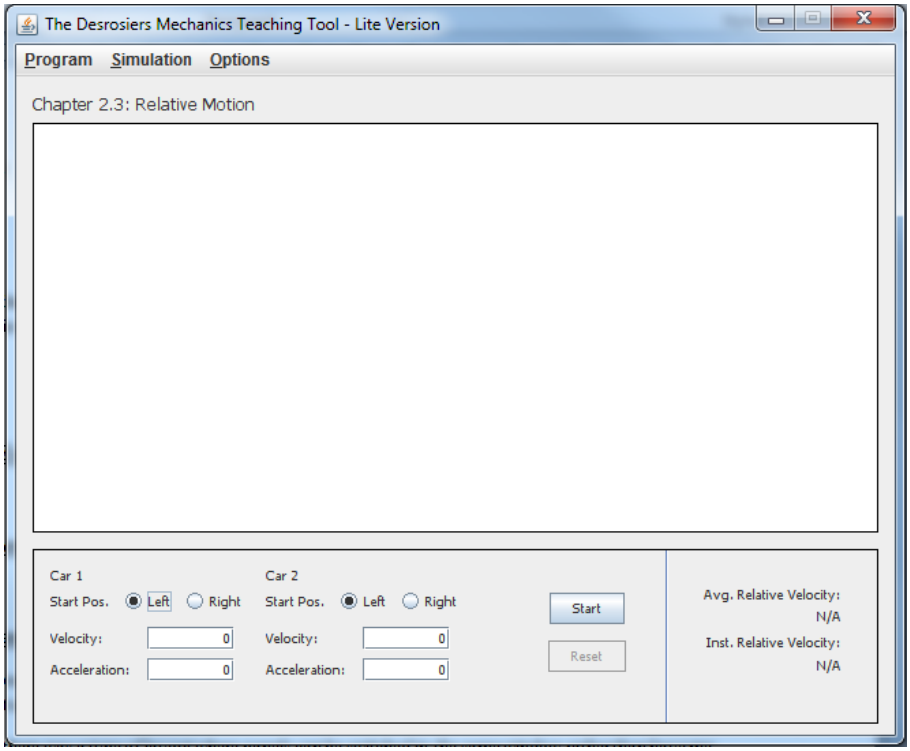

Fig. 20: Simulation 3 - Relative Motion

It was proposed to consist of two cars where the user can select the initial position, their corresponding velocity, and their corresponding acceleration. What would happen when the user clicks on the start button is, the cars move on their corresponding painted horizontal tracks in the viewing panel, and both the instantaneous and average relative velocities would be calculated and displayed through the use of two different labels.

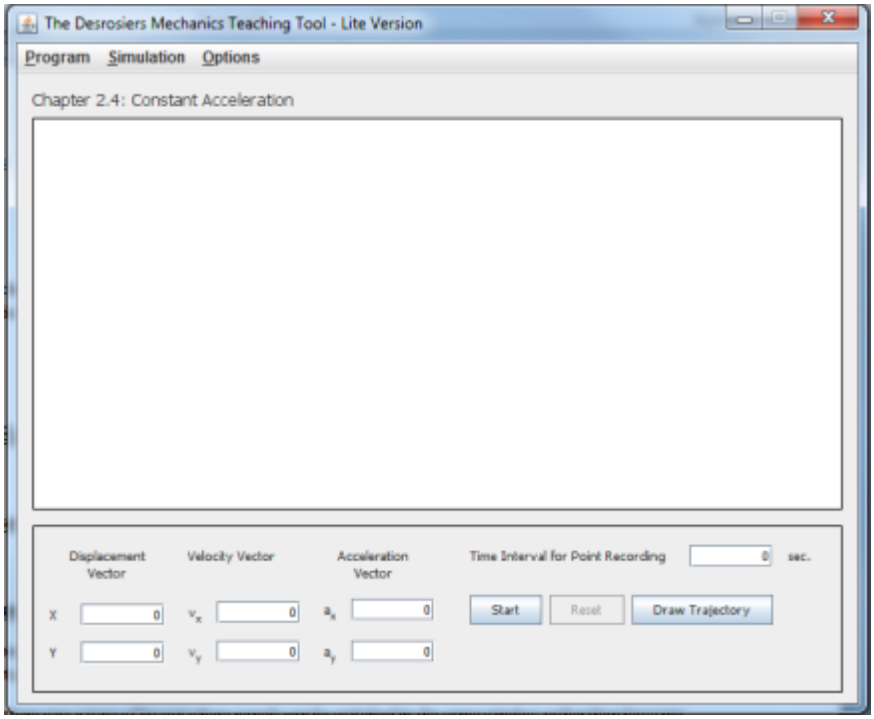

Fig. 21: Simulation 4 - Constant Acceleration

This is meant to be a two-dimensional simulation where the user sets three different vectors, and the time interval to record points for the user to view when the simulation stops. In the animation, the displacement vector is repainted with updated information based on the calculations from the time step and both the velocity and acceleration vectors.

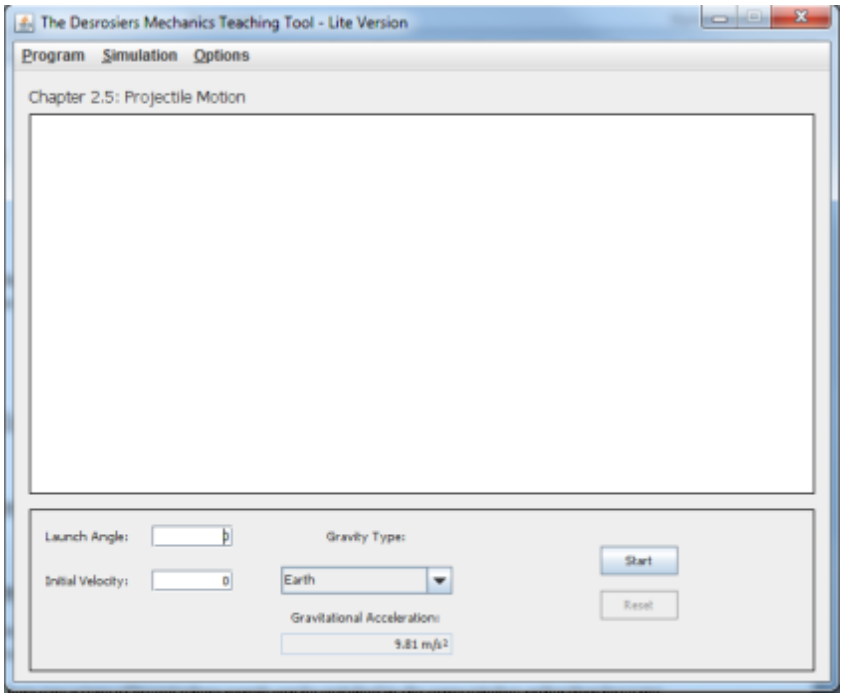

Fig. 22: Simulation 5 – Projectile Motion

This unimplemented simulation is meant to simulate projectile motion, where a circle (sort of like a soccer ball) of a constant radius starts off at the bottom left corner of the viewing panel, and the user sets three different variables: launch angle, the initial velocity at that angle, and what gravitational acceleration the projectile is under the influence of in flight. It's an animated simulation where the user can see the projectile's trajectory. If the projectile lands back down, the animation stops. It's the same thing when the projectile touches the right side of the viewing panel, only a dialog pane appears at the center of the screen.

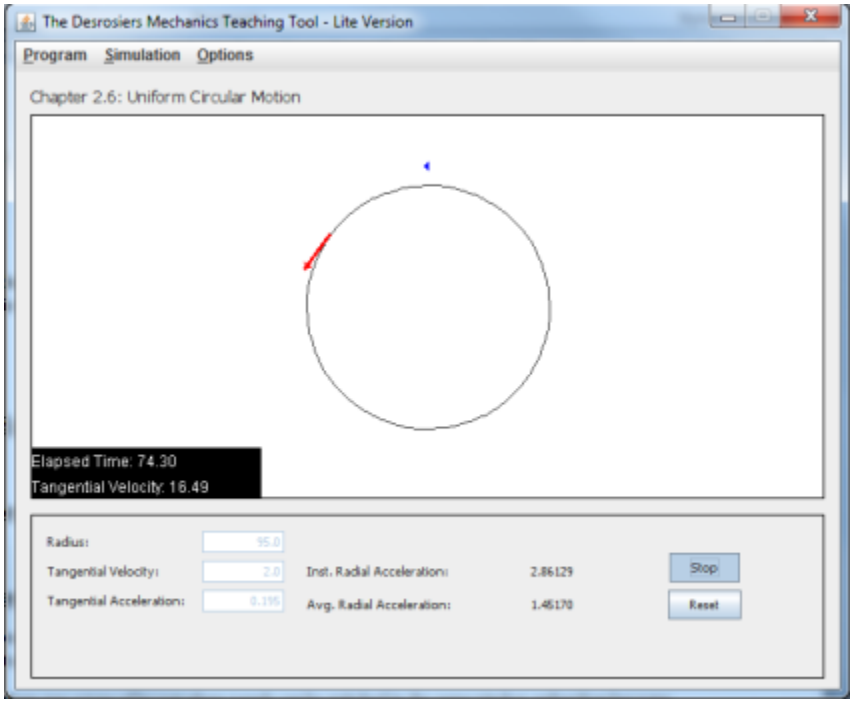

Fig. 23: Simulation 6 – Uniform Circular Motion

The user can experiment for themselves how radial acceleration is influenced by not only the radius of the centripetal motion, but also its current tangential velocity. The user can also add an acceleration to influence the velocity and therefore make the radial acceleration change overtime. This is an animated simulation where the user can see what's happening with the tangential velocity and both instantaneous and average radial acceleration. The red arrow tangent to the black outline is the tangential velocity, scaled down to let the user see. The blue arrow above the black outline is the tangential acceleration.

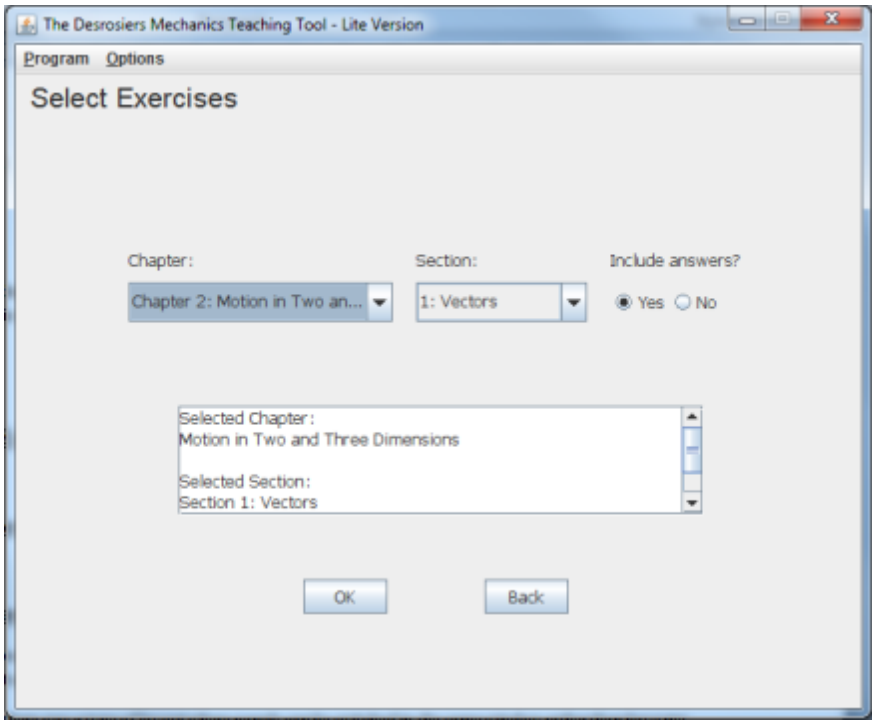

Fig. 24: Exercise Selection Screen

Again, the user must have the freedom to decide which exercises are desired, and whether or not answers are to be included. For this, two combo boxes with cell renderers (to display a tool tip text where the item highlighted has text too long to fit in the combo box menu), and a pair of radio buttons, are used for the program to load the appropriate exercises and set some components accordingly.

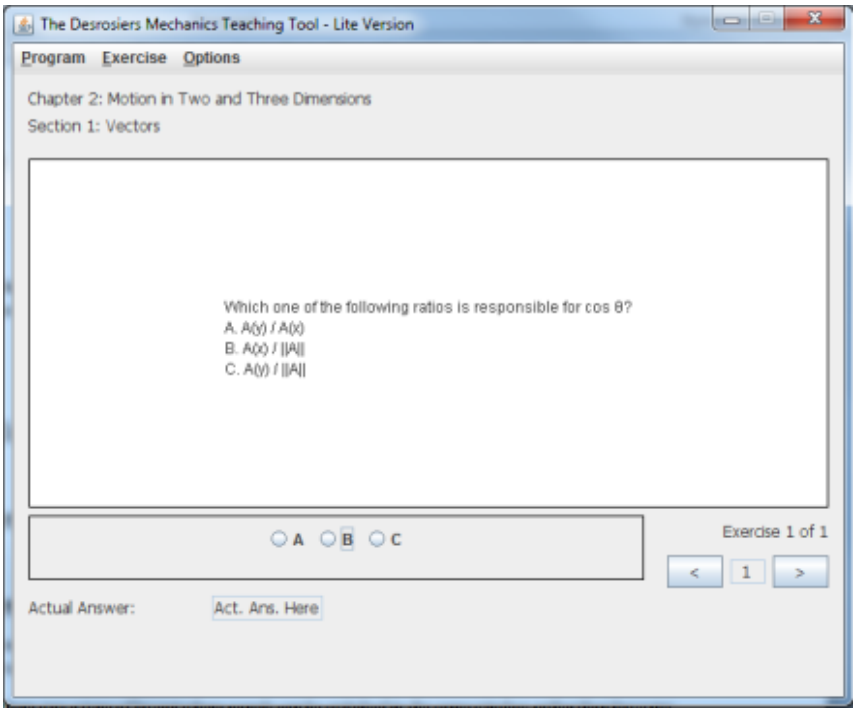

Fig. 25: Exercise Mode (with demo exercise)

The user wants to see the question directly in front of them, by the center. As such, this demo exercise is centered in the viewing panel. The answering components are centered as well in a generalized panel of their own, and it was intended that the user has the ability to go to the next, previous, or any exercise available.

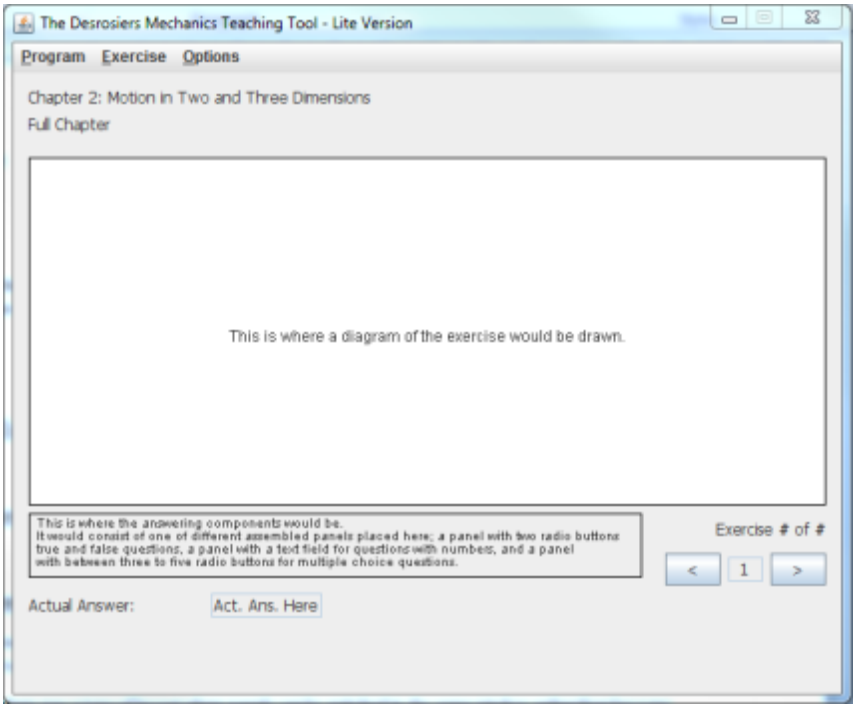

Fig. 26: Exercise Mode (not with demo exercise)

Because only one exercise is implemented in this version of the program, this is what the user will mostly see in Exercise Mode. He can take a look at the text and make a few assumptions to what the answering components would be like, but not the actual exercise question and diagram. There would be questions with just plain text and no diagram to display as well.

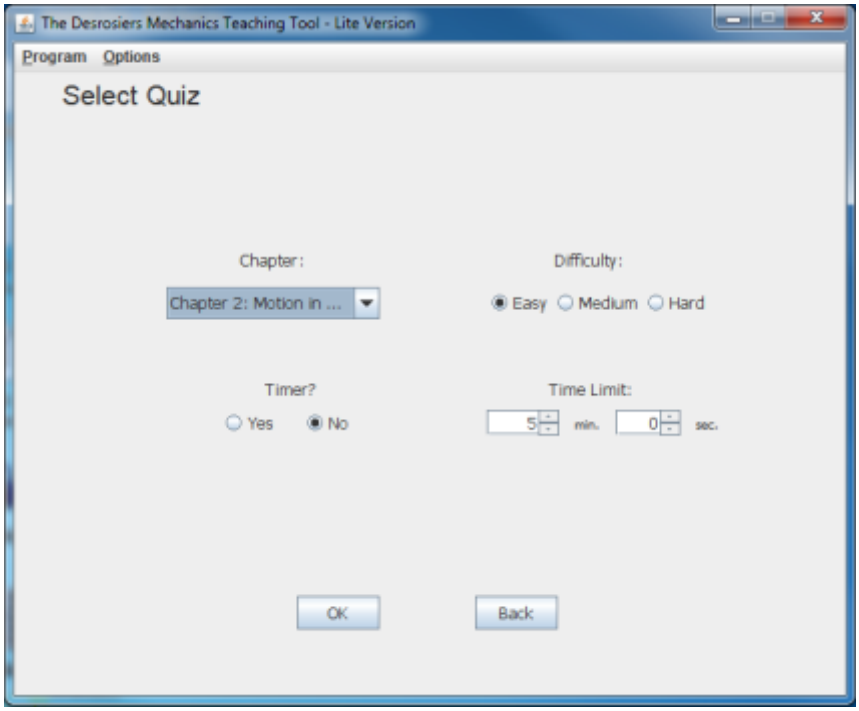

Fig. 27: Quiz Selection Screen

The user should have some flexibility to how they want to test themselves on their knowledge of a particular chapter

with questions that can be answered. There's also the importance of having a time limit. As such, there are four different things the user can set before the program loads a quiz under that set difficulty.

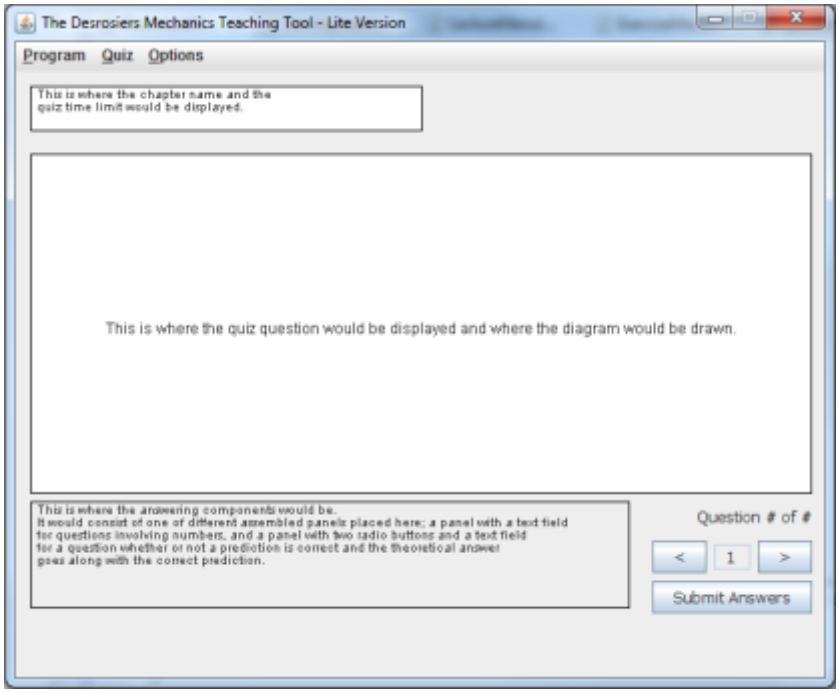

Fig. 28: Quiz Mode

No quizzes are implemented in this version of the program. However, the user can assess what a quiz would look like for a start. The time limit and chapter information would be displayed on the top left panel, the quiz question and its diagram would be seen at the center, and there would be answering components on the bottom left corner. It's either a text field, or a pair of radio buttons with an associated text field to give a quantitative explanation for a question involving a prediction. The user has the flexibility to go between questions, as well as submit the answers.

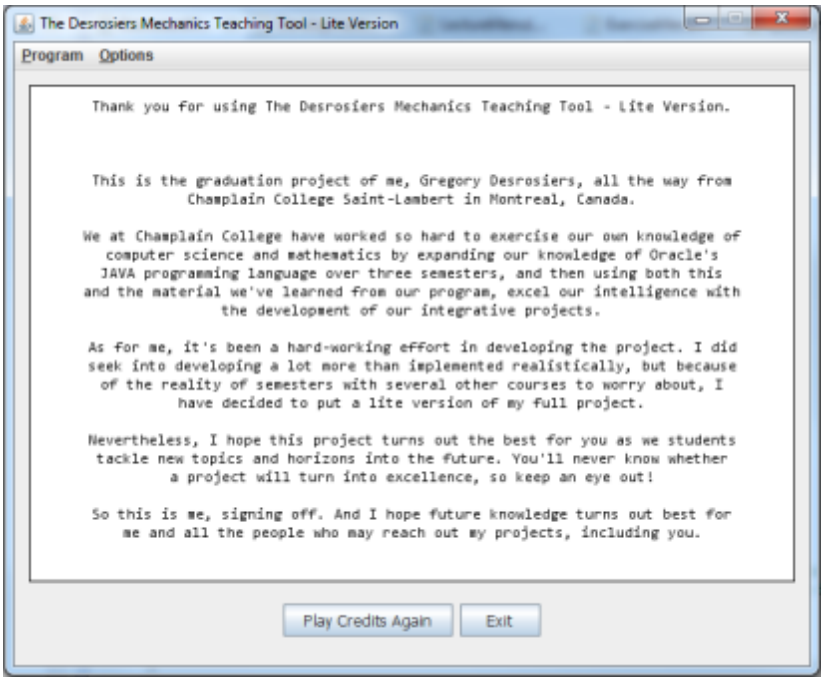

Fig. 29: Credits Mode

This mode consists of an animated credits sequence; much of the space in this panel is occupied by a custom panel that has an animation task associated with it. There are two buttons at the bottom for the user to either play the animation again, or to go back to the Main Menu.

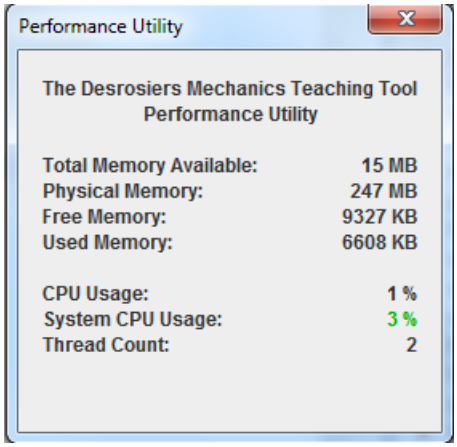

Fig. 30: Performance Utility

This utility was developed to make sure that the program ran efficiently, particularly with animation, and to detect possible signs that a memory error, or some animation lag, may occur. To make it easier to read, the left side labels are aligned to the left, while the right side labels are aligned to the right. There is a thread that runs in association with this utility, although it is discounted when counting how many simple threads are running in this program.

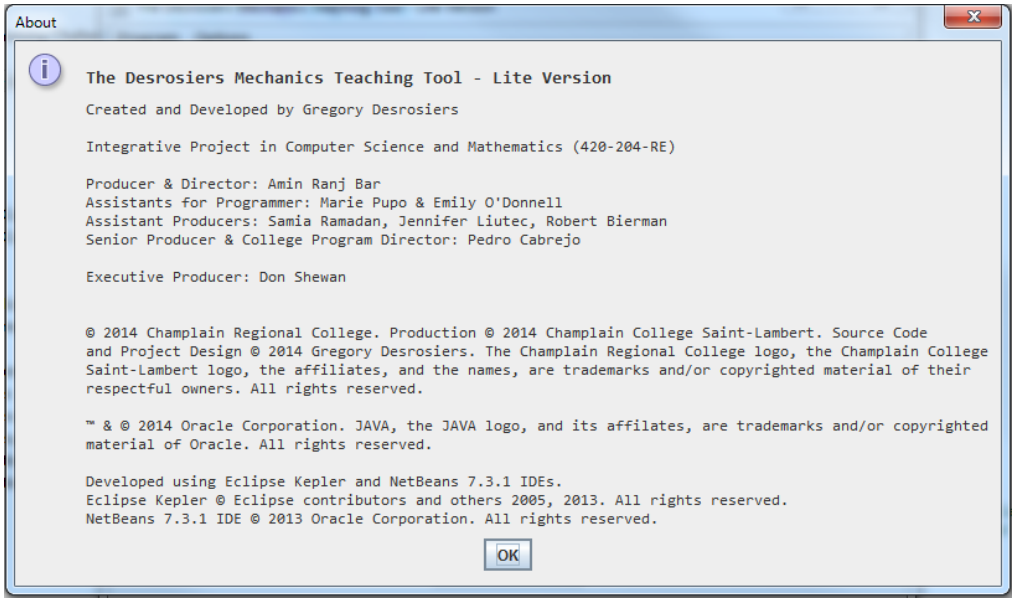

Fig. 31: About Dialog

A simple dialog to show quick information on who did what in this project, and what is the copyright information.

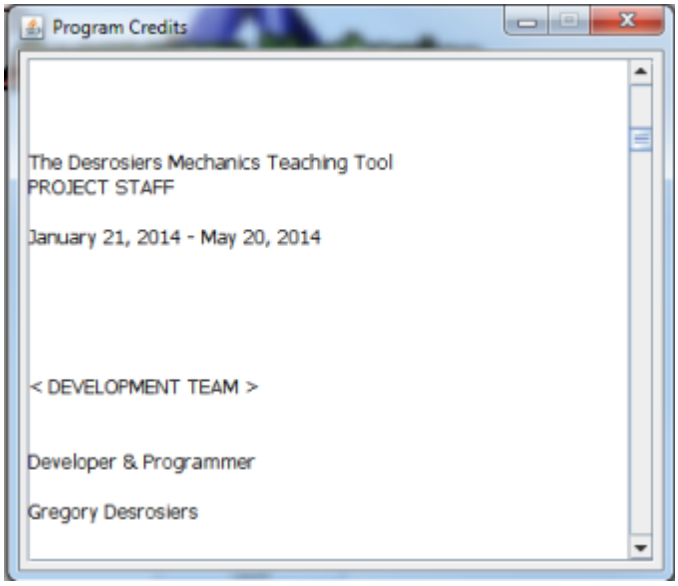

Fig. 32: Credits Utility

Even though there is an animated credits sequence, it's possible that the user can see something of interest but can't remember. This utility simply allows the user to take a look at not only all the people involved in this project, but to also satisfy their curiosity. Otherwise, it's for them to view who did what without any animation and more control; the user can view what they want from there by using the scroll bar.

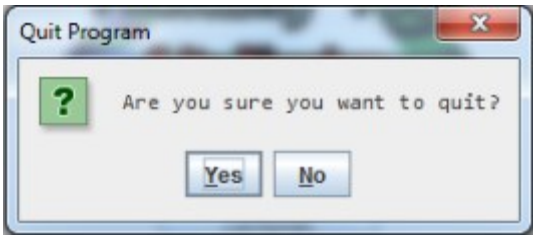

Fig. 33: Program Dialog

This is an example of a custom way to implement dialogs. Technically, they are from the static methods of JOptionPane, only that the message passed is actually an instance of JLabel with a new set font.

Methods of Evaluation

Discussed here in this section is what has been implemented in the final version of the project. Because of the way on how this project has been developed, some implementations involve components that are irrelevant to the material that was supposed to be incorporated into this program.

First of all, what I wanted to do was to put in a splash screen. I was expecting the program to take some time to load before it starts running, especially when all of the material to be used by this tool is to be loaded as well. Originally, I've learned that splash screens for JAVA programs would be images passed as a command prompt in the computer terminal when telling it to execute the program. The way on how I tackled that problem was, I developed an undecorated JFrame with graphics that has a task where an anonymous thread runs and calls the application constructor, and it updates a JLabel to show what the current stage of building the application is. It turned out to be somewhat useful, because the user can see what the program logo is like, who did what, what the copyright information is, and how far the program has loaded.

Next is the title screen. It involves painting an image and some text that is supposed to fade in and fade out over a time period. The way on how I did it was create a class called TitleScreenPanel, inherited from JPanel where it paints its own graphics, and the color of the "Press Enter" text is changed by the accessibility from a task called TitleScreenTask. The panel didn't came with a keyboard listener, because a keyboard listener does not work when it's added to a panel; it has to be loaded to the application frame instead. It turned out to be a good result, since it only had to be simple and user-friendly.

Then there is the general GUI. Since the components are different from panel to panel, and that the user should have instant access to a different panel the moment a certain action command is executed, there had to be a way to swap the panels without changing the window size or causing the program to only display a gray solid box. The way on how I did it was I created a JPanel called mainPanel, and set it to have a card layout; all major panels were to be added to mainPanel. Whenever a certain action command occurs, its associated listener loads the listener from mainPanel and calls the show method to show a different panel that's been added to the main panel, with its associated name. This turned out to be excellent because now the user can go between modes without having to use any external plugins that the program uses; except for the major plugin needed to run the program, swapping constructed Swing panels from one to the other is both memory and time efficient.

Because the panels have components placed in such a way where they are organized and user friendly, it is frantically impractical to code the placements of the components by hand. So what I did to design the panels was use a different IDE with a what-you-see-is-what-you-get editor called NetBeans IDE 7.3.1. I started off with the raw build of the GUI, I copied and pasted the source code from NetBeans to Eclipse, and made major adjustments such as changing the names of the variables and fixing several compiler errors Eclipse detected. It worked out as expected

because then when you are done with the important GUI build, you can make the code adaptable to Eclipse, code it with regards to the other elements of the program, and it works as it should.

For Lecture Mode, it has to keep track of what page the user is in for the associated listeners to respond appropriately, and to put pages together. What I did was, I created an abstract class called Lecture, eight overridden classes that are lectures, an abstract class called LecturePage as a generalized class for a lecture page, and three overridden classes that are lecture pages. The lecture pages for a lecture are held by the lecture object's array. The listeners of the Lecture Panel paint the graphics of the page through a method specified in the LecturePage class, and load the page's text to the description text area by simply calling the text area's setText method with text loaded on the lecture page. The text, in turn, is already assigned to the lecture page when the pages are created. Since the concepts of polymorphism are followed, when the user is in Lecture Mode and a page is destined for loading, the graphics and text come out as it should because the method to paint the graphics is called from the page object's actual type, and the listener has accessibility to the page's text. In addition, the kind of lecture pages to be loaded are isolated from one another in their own classes because certain pages involve an image. In this version, only one separate class is created that includes an image because only two chapters are featured in Lecture Mode. Essentially, Lecture Mode works out the way it should.

For the simulation with vectors, one important feature is having the viewing panel display the vectors with their properties based on the Cartesian coordinate system instead of JAVA's own coordinate system. What I did to make the vectors adaptable to the Cartesian coordinate system is simply find half the length and width of the viewing panel. The text fields for the initial points of the vector would be that for the X-coordinate, it is the vector's initial X position (in the JAVA coordinate system) minus half of the viewing panel's length (the variable holding this is *differenceBetweenJAVAAndCartesianOriginX* of the Simulation class), and for the Y-coordinate, it is the negation of the vector's initial Y position 9in the JAVA coordinate system) plus half of the viewing panel's height (the variable holding this is called *differenceBetweenJAVAAndCartesianOriginY* of the Simulation class). The result is, the positions of the initial points of the vectors are not as confusing to the user, since the user would mostly understand the Cartesian coordinate system.

In addition to this important feature, the vectors are moved closer to the center of the viewing panel when added together. To achieve this, what I did was I declared two double variables, *widthOfVectorSumRectangle* and *heightOfVectorSumRectangle*, and assign them with the sum of the scalar multiplication of both vectors' X and Y components respectively. If the first vector has a larger Y component than the second, its initial point is positioned to the bottom of the rectangle, and the second vector's initial point is position to the final point of the first vector. It's similar when the second vector has a larger Y component than the second, only that the second vector's initial position is placed at the bottom of the rectangle, and the first vector's initial position is where the second vector ends. It's somewhat an okay feature, because it is possible that the vectors can be large and thus the user won't be able to see what the sum of the vectors would look like.

For subtracting the vectors, there are two possible ways this operation can be done at the user's convenience. However, the general rule is that the vector that is negated for the operation has its initial point moved to the final point of the other vector, and then a Boolean flag that's checked primarily is assigned true, telling the program that the vectors are to be subtracted. In addition, the first vector is coloured red and the second vector is coloured blue to show that a subtraction operation is used than manipulating vectors as normal. It's not really useful because one or both vectors can go out of place and thus the user won't be able to see what the result looks like.

The second simulation has the important feature of using an approximate time step algorithm. It is discussed in the "Specific Algorithms" section, along with its psuedocode, but the advantage of using this algorithm is that elapsed time is approximately equal to real time, and the radial acceleration values are approximately correct when compared to theoretical values. There is still a discrepancy found because of the way on how floating-point numbers are handled in the computer, as well as the possibility that the value for the change in time, *dt*, is fixed and not

changed throughout the execution depending on the performance of the CPU operating on the simulation animation. However, the result is, because the calculated values in the program are mostly consistent, the user can still understand the idea of circular motion in both uniform and non-uniform cases.

Exercise and Quiz Modes didn't work out well, because there was a major shortage of time from the problems encountered during project development. Instead, there exist a few panels with painted text. Exercise Mode only has one exercise featured where the way on how the user gets a message on whether or not the answer is correct is by having an action listener associated with the answering components added to a custom panel.

For the Animated Credits Sequence, I wanted to display not only text, but also images. There was a setback presented, even though I wanted to use Graphics2D to paint the graphics; integer precision with the positioning of the images. The way on how I solved that problem was that, in my paintComponent method for the class AnimatedCreditsPanel, I created three different AffineTransform objects, translate them, scaled them, and used them as the canvases for the images to be painted. I wanted my credits to be a scroll-up type, so I have programmed that method to change the paint device's Y position with regards to the current position of the top of the credits, because the animation thread changes the Y position of the top of the credits. The result is a functional credits sequence where the user can watch the text and images move from the bottom of the viewing panel to the top. In addition, when the last text becomes no longer visible, or when any of the subframes open up as the credits are running, the credits stop and a different message appears to tell the user to click on "Play Credits Again" at their convenience.

As for the program performance utility, I wanted to display information regarding memory, what is the current CPU usage for the program, the system CPU usage, and how many threads are running in this program that only I know what they are. I developed a simple GUI with multiple labels, and a thread which gets information from three instances of three different classes which are referenced with polymorphism. The thread sets the labels on the right side with values returned from the methods of these three instances, except for the thread count; the thread count is derived by subtracting the total number of threads by how many daemon threads are there plus three. The result is, when the user is in the Main Menu, the utility displays not only the information that's updated less than 100 times a second, but two threads: one being the event dispatcher thread, and one for setting the label at the bottom of the Main Menu with new date and time information.

The About Dialog was intended to display some text in a formatted way. For this, I simply made some changes to a label in the Application class called dialogLabel, with text formatted in HTML, and passed it in JOptionPane's showMessageDialog method.

Finally, the simple Program Credits Utility lets the user see for themselves who did what with no animation involved. The text also had to be formatted as it is loaded from the same source files the Animation Credits Panel loads the text from when created. For this, in the class from where the utility is constructed with a text area and a scroll pane with a vertical scroll bar, there is a method that not only loads the text from the two text files regarding credits and copyright, but also formats them properly for display in the text area using two different loops.

Results: System Quality

Developer Perception

What I liked about doing this integrative project is that, I had the ability to exercise my past knowledge of JAVA into this project and learned a few new components.

I liked on how there was a type of layout where panels can be switched in the same window rather than have me program the selection listeners to remove and add the current panel and set the current index, which is quite cumbersome for a Swing frame. I liked what I did with the overall GUI development, the menu actions, the credits animation, and how I was able to successfully implement a time step algorithm based off one from a website where it allowed the elapsed time of the circular motion simulation to be approximately the same as real time. There's also keeping my program efficient and somewhat user-friendly; the general modes of the program and the selection screens are very organized, the program goes into a particular mode immediately after its selection screen, and for most of the time, the program doesn't crash. And, of course, I liked my implementation of the performance utility with estimated data the user can see for their computer-based curiosity.

The overall presentation of the program is somewhat decent enough; it could be more colorful and have some variety, but I put more emphasis on the GUI development and some implementation of the material, so I'm not completely worried about it. After all, an integrative project is not designed to be a treasure; it's simply to test the knowledge of both programming and what was learned in the entire college program.

I happen to have a few dislikes about this integrative project.

First, I dislike the overall GUI appearance when the user is in Exercise Mode with no exercises, and Quiz Mode, because even with text, the user won't be very appealed. Some text can be hard to read since it's small, and there really isn't that much implementation as I hoped for.

I even disliked the fact that I wasn't able to implement a lot of subject material into the program as I wished; if it wasn't for my ridiculous development approach and the way on how I executed my homework in other classes, and my errands including university, then development would be much smoother and reliable. Therefore, the program would be much better, presentable, and useful.

Finally, I should have taken some time to learn different libraries under the same programming language where this program would be more presentable, or more efficient to code. Instead, I disliked the fact that I wasn't able to put in much animation as I wanted to, and not be able to learn external JAVA plugin libraries such as LibGDX, where it could have been used on mobile phones, and it would let me learn on how to program an actual game.

Objective Measure

In the final development phase of this project, not much quality assessment has been made. However, the rough development of the tool and its major GUI components have been emphasized; in addition, I did some testing myself during development to make sure that the program functions as it should. Extensive testing was not commenced because the other students in the class were too busy with assignments of their own, especially with end-of-semester courses and exams.

Whenever I made serious changes to my program, I double-check to see if my implementation is correct and it comes out right by running the program and have it execute the part where I made the change. This has been done with most of the components developed into this program. In several occasions, I was able to spot errors with what I've recently programmed and made changes to the code for proper or easier execution.

In terms of the project itself, the following objectives, stated from System Objectives, have been completed:

- 1. Plan and integrate a graphical user interface, complete with buttons, graphical components, and interface management.
- 2. Implement animation for appropriate features of the program to be developed.
- 3. Create space-conserved and time-conserved algorithms in the project code.
- 4. Build features for the program that allows the user to access different learning phase environments such as simulations and exercises.
- 7. Remove unnecessary code from the program to save memory space for installation and use in other computers later on.
- 8. The program has to be cheap and simple to run.

The following objectives are incomplete and are still work in progress:

- 5. Create and implement subject material, including simulations, based on the course as much as possible.
- 6. Make the program user-friendly, including appropriate text, carefully-positioned GUI components, and possibly keyboard shortcuts.

This is because for Objective 5, there are diagrams missing for the lectures, four of the only six simulations of this version of the program are not programmed, and only one exercise has been added. There are no other exercises of the only motion and forces chapter programmed in the application, and there are no quizzes. As a result, because the user will see messages saying that features are unimplemented and see some incomplete GUI for Exercise and Quiz Modes, the requirement for a user-friendly program is not completely satisfied, but partially satisfied.

Developer Evaluations

There was a document associated with the prototype of the project where it contained several examples provided for simulation testing. It was not complete, and in addition, there was no feedback from other students in the course because there were too many constraints that forced them to not test this program at all.

The test data used for the simulations are as follows. In addition, there are columns with the actual values from the program and whether or not they are satisfactory.

Simulation 1 – Vectors

NOTE: There is no test data for adding and subtracting vectors because the information regarding the resulting vector is not displayed. In addition, the modes do function properly, but there are no evaluation sheets for the GUI itself. And there is no test information provided for manipulating vectors by their direction only; it has been tested at least a few times, but there is no absolute guarantee that the vectors will have their values correctly displayed when their directions are changed.

Ex. 1	Vector 1	Results satisfied?	Vector 2	Results Satisfied?
Input Data				
X Component	10.0	Yes (10.0)	55.5	Yes (55.5)

Y Component	25.0	Yes (25.0)	8.6	Yes (8.6)
Initial X Position	35.8	Yes (35.799988)	-300.0	Yes (-300.0)
Initial Y Position	62.5	Yes (62.5)	-140	Yes (-140.0)
Scalar / Factor	1.7 (multiply)	Yes (1.7)	1.56 (divide)	Yes (1.56)
Output Data				
X Comp. (scalar)	17	Yes (17.0)	35.577	Yes (35.577)
Y Comp. (scalar)	42.5	Yes (42.5)	5.513	Yes (5.513)
Direction	68.199°	Yes (68.199°)	8.808	Yes (8.808)
Magnitude	45.773	Yes (45.774)	36.002	Yes (36.002)

Ex. 2	Vector 1	Results satisfied?	Vector 2	Results Satisfied?
Input Data				
X Component	-15.3	Yes (-15.3)	-5.0	Yes (-5.0)
Y Component	3.7	Yes (3.7)	-5.5	Yes (-5.5)
Initial X Position	210.0	Yes (210.0)	-300.0	Yes (-300.0)
Initial Y Position	-100.0	Yes (-100.0)	-140.0	Yes (-140.0)
Scalar / Factor	25.0 (multiply)	Yes (25.0)	1.56 (divide)	Yes (1.56)
Output Data				
X Comp. (scalar)	-382.5	Yes (-382.5)	-3.205	Yes (-3.205)
Y Comp. (scalar)	92.5	Yes (92.5)	-3.526	Yes (-3.526)
Direction	166.405°	Yes (166.405°)	227.726°	Yes (227.726°)
Magnitude	393.526	Yes (393.526)	4.765	Yes (4.765)

Ex. 3	Vector 1	Results satisfied?	Vector 2	Results Satisfied?
Input Data				
X Component	86.3	Yes (86.3)	0.0	Yes (0.0)
Y Component	-95.97	Yes (-95.97)	5.0	Yes (5.0)
Initial X Position	300.0	Yes (300.0)	0	Yes (0.0)
Initial Y Position	-150.0	Yes (-150.0)	-145	Yes (-145.0)
Scalar / Factor	-4.0 (multiply)	Yes (-4.0)	55.5 (multiply)	Yes (55.5)
Output Data				
X Comp. (scalar)	-345.2	Yes (-345.2)	0.0	Yes (0.0)
Y Comp. (scalar)	383.88	Yes (383.88)	277.5	Yes (277.5)
Direction	311.963°	Yes (311.96°)	90°	Yes (90.0)
Magnitude	516.262	Yes (516.262)	277.5	Yes (277.5)

From these results, the simulation is thus correct with somewhat accurate information.

Simulation 2 – Uniform Circular Motion

	Radius	Tan. Velocity	Tan. Accel.	Elapsed Time	Theoretical Inst. Radial Acceleration	Actual Inst. Radial Acceleration	Theoretical Avg. Radial Acceleration	Actual Avg. Radial Acceleration
Ex. 1	60.0	25.0	4.0	11.84	87.266	87.269	48.841	48.842
Ex. 2	100.0	43.0	59.0	10.15	4119.714	4119.755	2069.102	2069.123
Ex. 3	75.0	-75.0	0.01	31.41	74.373	74.378	74.687	74.689

Ex. 4	96.5	-100.0	-4.575	6.16	170.266	170.271	136.946	136.949
Ex. 5	78.45	-500.0	100	14.56	11649.917	11649.918	7418.330	7418.331

From these results, the data retrieved from the simulation are approximately correct with what it should be, although there are a few discrepancies mostly from the way on how floating-point numbers are generated through a computer.

Project Management

To keep track of my progression and to ensure that tasks were completed on appropriate dates, a schedule, in a form of a table, has been constructed. Unfortunately, this is not a proper schedule, because it was only created in early April, and I was not ready yet to work as a professional programmer. As such, estimated planned dates have been given in such a way where it is somewhat realistic and thus I would program the task until then, or after the planned date.

As a lesson, a more proper schedule will be made next time a project such as this will be forwarded for development.

Task	Planned Date	Actual Date	Status	Assigned Person	Notes
Project Conception	January 24, 2014	January 24, 2014	Complete	Gregory	Four different conceptions written out; first one (education tool) selected.
Project Proposal	February 20, 2014	February 20, 2014	Complete	Gregory	Not enough training to make it look professional. Mark: 80%
Project Presentation	February 21, 2014	February 21, 2014	Failed	Gregory	PowerPoint presentation not created because of too much detail in my proposal. Mark: 60%
Raw Package and Source Code File Creations	February 25, 2014	February 27, 2014	Complete	Gregory	It was a good start in organizing the source code files for easier location when debugging. I was expecting for the code to be more completely organized before the project was presented.
Material Programming (on the chapters)	March 11, 2014	March 31, 2014	Complete	Gregory	All the chapters instead of just one chapter had the corresponding equations coded.
Rough Menu Bar Development	March 13, 2014	March 28, 2014	Complete	Gregory	Five different and appropriate menu bars

					are implemented in the program, corresponding to the user's current mode.

Mode Development (NetBeans)	March 16, 2014	April 20, 2014	Complete	Gregory	Only one exercise, and no quizzes, were implemented in the final version of the program. However, the control panels for the six simulations intended for implementation are in place.
Listeners Development	March 21, 2014	May 6, 2014	Complete	Gregory	The listeners for the menus were not programmed until May.
Logo Design	March 23, 2014	April 19, 2014	Complete	Gregory	Created using Adobe Photoshop Elements 9.
Subject Writing (Planning on Lectures, Simumations, Exercises & Quizzes)	April 1, 2014	May 6, 2014	Failed	Gregory	I was not able to handle the pressure I had in writing the material, and thus I ended up writing less than what I expected.
Lecture Programming on Subject	April 8, 2014	May 5, 2014	Complete	Gregory	Only one image has been used, but at least two lectures complete with text are provided: one with two and three dimensional motion, and one on the co-requisites of physics.
Simulation Programming on Subject	April 13, 2014	April 29, 2014	WIP	Gregory	Only two simulations have been coded: the one with vectors, and the one on uniform circular motion.
Exercise Programming on Subject	April 19, 2014	Cancelled	WIP	Gregory	Out of time to work on developing exercises based on the subject.
Quiz Programming on Subject	April 21, 2014	Cancelled	Not commenced	Gregory	Out of time to work on developing quizzes based on the subject.
Debugging Tool Development	April 22, 2014	April 24, 2014	Complete	Gregory	Only a programming utility has been

					developed.
Program Testing & Debugging (Before Demo Presentation)	April 25, 2014	May 6, 2014	Failed	Gregory	The last few days before presentation was taken over by finalizing the code for the listeners and finishing the subject material implementation. In short, I was out of time.

Program Bug Fixes (Before Demo Presentation)	April 28, 2014	May 6, 2014	Failed	Gregory	During the demo presentation to the teacher, there was a bug discovered in the vector simulation: instead of displaying the unit vectors as it should, the button was locked into place and the vectors did not show up; a possible exception has occurred.
Final Refurbishments and Deletion of Unnecessary Code (incl. Comments, before demo presentation)	April 30, 2014	May 6, 2014	Completed	Gregory	All unnecessary classes (in particular, the subject material classes) have been deleted.
Project Presentation (To Teacher)	May 2, 2014	May 6, 2014 (5:00 PM)	Completed	Gregory	Overall presentation went very smoothly. There was confusion at first to when I was to present my project because the original date file was not completely organized.
Program Testing & Debugging (After Demo Presentation)	May 13, 2014	Cancelled	Failed	Gregory	This stage was not commenced. Because I was exhausted from having to spend over three months of heavy college homework, I did not progress much on the project as I should have.
Program Bug Fixes (After Demo Presentation)	May 13, 2014	Cancelled	Failed	Gregory	This stage was not commenced. No one gave me feedback on the bugs they discovered since all of us in the classroom had

					to worry about other assignments.
Final Refurbishments and Deletion of Unnecessary Code (incl. Comments, after demo presentation)	April 30, 2014	May 13, 2014	Completed	Gregory	A change had to be done to only one interface in the program code on May 17, 2014, but comments are put into place.
PowerPoint Build for Final Presentation	May 13, 2014	May 11, 2014	Completed	Gregory	A simple PowerPoint file from PowerPoint 2007 is created, with a chosen theme.
Final Presentation	May 13, 2014	May 13, 2014	Completed	Gregory	Phase II - Implementation: 90% Phase III – Presentation: 95%
Final Project Report	May 20, 2014	May 20, 2014	WIP	Gregory	There are several sections that are incomplete because of temporary exhaustion from semester work and therefore shortage of time.

There's also a development log available in the same directory as this report; you'll need a program to open .txt files to view it, such as Microsoft Notepad on Windows.

Conclusion

Success on the Tasks

The following assumed tasks are those that have been completed:

- Project Conception
- Project Proposal
- Raw Package and Source Code File Creations
- Material Programming (on chapters)
- Rough Menu Bar Development **[tested by developer]**
- Mode Development (NetBeans) **[tested by developer]**
- Listeners Development **[tested by developer]**
- Logo Design
- Debugging Tool Development **[tested by developer]**
- Final Refurbishments and Deletion of Unnecessary Code (before demo presentation) **[tested by developer]**
- Project Presentation (to teacher)
- Final Refurbishments and Deletion of Unnecessary Code (after demo presentation) **[tested by developer]**
- PowerPoint Build for Final Presentation
- Final Presentation

These tasks are those that are work in progress:

- Lecture Programming on Subject **[tested by developer]**
- Simulation Programming on Subject **[tested by developer]**
- Exercise Programming on Subject **[tested by developer]**

These tasks are those that have been failed:

- Project Presentation
- Subject Writing (Planning on Lectures, Simulations, Exercises & Quizzes)
- Program Testing & Debugging (Before Demo Presentation)
- Program Bug Fixes (Before Demo Presentation)
- Program Testing & Debugging (After Demo Presentation)
- Program Bug Fixes (After Demo Presentation)

The only task where work on it was never commenced is programming quizzes based on the subject material.

Overall Degree of Completion

Based on the result of developing this program, because there were too many constraints during development, and there were often problems regarding several other circumstances, unfortunately the problem, discussed in the "Story" subsection of the Project Overview, has not been solved.

However, because this integrative project course is an educational course and is meant more for learning project management and effort, along with following instructions and what resources to use, this project does not count as a failure.

From the perspective of seeing this as a CEGEP integrative project, it is still under construction, and it is also untested. Only a first version prototype GUI has been built, with prototype listeners, two prototype lectures and simulations, one exercise, and no quizzes; the subject material implementation is not complete.

From the perspective of seeing this as a real-world application, it is completely under construction with no prototype visible, and it is also untested. The thirteen chapters originally proposed have not been added to this program.

It was suggested in the revision of the original proposal that the full project could be worked over the summer. But because of the degree of the subject material to be integrated into the program and the extensive amount of programming necessary for functionality, organization and professional use, it would take more than eight hours a day, for about three months, to implement all the motion and forces material into the program. Therefore, the project is too inconsistent to be given to the Physics Department of Champlain College Saint-Lambert for them to use to assist students in learning motion and forces.

Quantity, Difficulty, Quality (Learning Experiences)

There were a lot of things learned over the development of this project as well as the course in general. What I'll discuss here is my overall opinions first, and then point out some things I've learned over the course and what I did to apply them in practice.

The overall difficulty of this project was a little hard. There were serious development problems I've encountered over the semester since I had to finish five additional courses and work on other errands. Essentially, the biggest problems I had with this project is having an unrealistic development approach, not managing my time, and pushing myself too hard at the start of the course. In fact, what I would look forward to, if I were to do another integrative project, is working with a partner, learning on how to time manage properly with a lot of tasks, how to work for more than 6 hours a day, especially if this were to be my only course, and eventually becoming a professional software developer. These things were unfortunately not learnt in advance, and thus I was unable to practice them properly.

In other words, what I learned from this integrative project is that future projects will always be difficult. Just as I think an algorithm would be easy to code, in reality, there will be obstacles I have to face when I become a professional software developer. That's because I cannot assess beforehand that this algorithm will execute something in the program properly. In addition, there would be hard-to-program algorithms, having to develop realistic time frames and development executions. I'll have constraints to worry about and it's better to work on work on more of what's important than having to work out the interfaces beforehand and hence messing some or more elements up. For example, if a project would take too long to develop, I would look at working on the project off my work hours, and thus it disrupts my schedule for weeknights and weekends; therefore, I would have a very stressful experience developing the project. In general, it takes a very long time to develop such a successful computer program.

For this project, I wanted take the difficult direction to try to make the project as complete as possible. But based on the problems encountered, there were certain things that I had to take it easy with. For instance, in the Vectors simulation, whenever the user clicks on the "Subtract Vectors" button and selects how the vectors are to be subtracted, the vectors, including the resultant vector the user cannot see since it is not painted, form some kind of box. I wanted to center this "box" so that way the vectors are still painted in operation of subtracting one from the other, but placed in the center so that way the user can see better what the resulting vector is. Because I was running out of time, I modified the Subtract Vectors button to simply reposition one vector to where the arrow of the other vector is pointing, then paint that vector in the opposite direction as it originally did.

In terms of quality, I have learned that much of the project needs to be professionally designed as much as possible. It always depends on what the client or customer asks, as well as the business model and what the managers say to software developers, however, because each business is different. It also has to do with the policies put forward; essentially quality will be dependent on several different cases. The project also had to be so user-friendly, because if not, it won't be that much useful to the user; it would be full of logical problems and realism problems. The quality of the project was important so much that I had to squeeze it in as one of the prime goals of this project development, with regards to the time frame and other constraints during development.

For this project, what I did for professional design and realism is not only introduced choices to the user where they can choose what to do and come back at their own convenience, but also added some usable features. These features consist of a title logo, some dialogs for warnings or confirmation, organized GUI components with confirmation from the teacher of this course, text filtering, tool tip texts, and consistent approaches with lectures and simulations. There was also the writing of the lectures themselves, as it should follow mostly from the perspective of a teacher; it isn't formal because writing had to be done fairly quickly, but I was able to write something in such a way where the user could follow along quite easily. Of course I was only able to put in one image for Lecture Mode, but it's enough

to add a little more to the quality of the project. However, even if quality was emphasized during development, because there were a lot of problems, the program is not with professional quality as it should be.

Overall, there were things that were learnt in this course, but have yet to be practiced properly, and hence I'll need some training to better practice them when I'm at work as a software engineer, a program analyst, or an engineering assistant.

Developer's Comments

I feel really unfortunate for not putting in enough efforts into this project as I hoped for. The reason why is because I have never given out a very realistic approach into this project. I was sort of out of control, because the original proposal asked for way too much work into this project. I wanted to excel my knowledge of JAVA and help out others by putting in all the chapters I originally proposed, except that I was not very flexible with my other homework and classes, and I had to work on other errands, especially university application and scholarship applications.

I did not work with a partner because of two things: one, I wanted to keep my intelligence up in this course and make my classmates shocked about what I can do. In fact, I was a little anxious that if I were to work with a partner, then my intelligence would get turned down, and thus since I had trouble with my self-esteem, I would feel discouraged and not integrated as I should be. Second, the semester before this course, I ran into a few stress points while working with my partner in Electricity & Magnetism, because my abstract thinking was so weak that I couldn't be as smart as him; I did run into some points in my labs where we figured out what we can do, but otherwise I was a little too dependent on my professor.

All in all, the most general psychological problem I could come up with is not being able to handle college homework realistically enough, as well as my own self-esteem and personal problems. I'm still glad that this course has been completed, but what I'll need to go through in the future is some intervention plan, work on my abstract thinking, and building better self-esteem, time management, and my real-world thinking.

Project Credits & Copyright

Because this project was also a graduation project, I decided to create my own credits roll consisting of all the people who supported me in my efforts over the past two and a half years, as well as who was involved in the development of this project.

< DEVELOPMENT TEAM >

Developer & Programmer

Gregory Desrosiers

Development Support

Simon Cochrane
William Leclerc
Olivier Carrière
Sean Marcoux
Chloé Ménard
Mugisha "Christine" Joyce Kakou
Alexandre Comeau-Vermeersch
Ege "Jason" Özer
Jonathan Luciux-André
Darrin Fong
Jeremy Brown
Simon Thompson
Tiémé-Frédéric Togola
Patrick Todd
Louis-Philippe Haché
Ho Man Chan

< PROJECT AID >

Assistants for Programmer

Marie Pupo
Emily O' Donnell

< EDUCATION SUPPORT >

College Program &
Extracurricular Education Professors

Norcene Webb
Gabriel Indurskis
David Rozon
Kerry Cruise
Rachel Morris
Panagiota "Pota" Kanavaros
Alison Clark
Srinath Baba
Daria Buczko-Hackett
Krista Elizabeth Smeltzer
Jade Préfontaine
Michèle Titcombe
Fouad Ajami
Sarmista Das
Craig Sinclair
Peter Varfalvy
Salvatore "Sonny" Carlo Ruffolo
Carmela Mancuso

Gabriel Flacks
Marc Dagenais
Marie-Hélène Burman
Benoit Larose
Mao Chambers

Extracurricular Students

Alexandre Meijer
Bruno Gosselin
Jared Grant
Suleman Latif
Brendan Lacasse
Nicolas Pinard
Jeffrey Marois-McKenzie
Matthew Ullyot
Alexander Weihmeyer-Hamel

Tutors

Mary Pennefather
Anne-Florence St-Laurent
Abigail Free (McGill)
Catherine Kozminski-Martin

In-Class Assistants

Devon Strueger
Brigita Ehrensperger
Robert Shaul
Christina Tricarico (Concordia)
Chris Kwok (McGill)

IT Technicians

Denise Menard
Salvatore D'Amico

Mentors (from 2011-2012 Mel Berish's WINGS Project)

Nicole Bourassa
(Riverside School Board - Heritage Regional High School)

Rami Chaouki
Caitlin Kelley
Megan Clarke
Sarah Ismail
Jennifer Tracey

Student Services Director

Dean Howie

Student Services Secretary

Celine Bourgeois

Academic Psychologist

Wanita Jones

Counselors

Veronica Wynne
Mel Berish

Financial Aid & SLAM Life Officer

Dave Persons

Special Needs Education Teacher & Coordinator
(Riverside School Board's ADAM's PACE Program)

Marina Bresba

< PROGRAMMER PERSONAL SUPPORT >

Natalie Pepiot
Roxanne Evelyn Brace
Jessica Myre
Rachel & Sonia Giulione

Roxanne Bougie
Benjamin "Ben" Lomon
Jasna Queenville
Nathalie Geukers
Charles Brasset-Mimeault
Conor Antenucci
Jonathan Ferreira
Olivia Ciampini
Marianne Blais
Kayla White
Sam Chaussé
Mélissa Brochet
Stephanie Tardif-Bennett
Alex Pace
James Brace
Philip Steve Auclair
Marino Soumas
Kristopher Koliakos
Bhavrik "Nicki" Nanda
Paula Fasano
Justin Hunt
Jordan McAran Bourque
Shane & Kellie Kiakas
Gabriel Thibault-Fredette
Stephanie Fredette
Alexandra Coutu
Kate & Sarah Poitier
Jessie & Carly Howarth
Marcel Ndayizamba
Stefany St-Arnaud
Stéphanie & Isabelle Richard
David Bastien
Marie-Pier Desbiens
Connor & Madison Hynes
Ian & James Hutt-Borelli
Giuliana Bevilacqua
Benjamin Daunoravicius
Bonnie Gauthier
Jonathan & Nicholas Frenette
Sabrina Langelier
Wasif Choudhury
Catherine Benson
Natasha Marois-Mckenzie
Amanda Wai
Anthony Kazazian
Valerie Nguyen
Daniel Diaz
Michelle Di Meo
Jennifer Coulombe

Sarah Vanwambeke
William MacCaul
Mathieu Drolet
Barbara Goodall
Allyson Allaire
Maryse Quintal
Milène Mallette
Laurence Panneton
Mariah & Rebecca MacCaul
Gabrielle Caron
Samantha & Ashley Hogan
Emily Demers
Roxanne Gareau
Shauna O'Brien
Anne Ouillet
Allyson Allaire
Marley Chase
Jean-David Parent
Corine Tellier
Sarah Trépanier-Chicoine
Bonnie Gauthier
Trevor Wo
Alex Milton
Ophelie Lacasse
Jean-Michel Beaudoin
Cameron Levins
Carly Comtois
Alyssa-Rubylee Gardner
Michelle Steben
David Phibes
Bruna Sunye
Alexandra Smith

< SPECIAL THANKS >

Programmer Family Members & Friends

François Desrosiers
Ann Roy
Corina Wind
Jacques Caron
Quentin Desrosiers
Abraham Wind
Kelsey-Hope Patrick

Education Members

Dale Huston
Riverside School Board
Members of the Champlain Saint-Lambert - George Wallace Library
Champlain College Saint-Lambert - Physics Department
All Champlain Regional College Staff

International People and Corporations

Richard Wolfson
Pearson Education, Inc.
Eclipse Foundation
Original Sun Microsystems Team for JAVA
JAVA Creator James Gosling
Oracle Corporation
Rovio Entertainment, Ltd. (Mikael Hed, Nikolas Hed, Peter Vesterbacka)
Google, Inc.
Marc Zuckerberg and Facebook, Inc.
Stack Exchange, Inc.
w3schools.com
coderanch.com

Ministère de l'Éducation, du Loisir, et du Sport du Québec

< MANAGEMENT TEAM >

Academic Advisor

Dayle Lesperance

Assistant Producers

Samia Ramadan
Jennifer Liutec
Robert Bierman

Producer & Director

Amin Ranj Bar

Senior Producer & College Program Director

Pedro Cabrejo

Executive Producer

Don Shewan

This project is a tribute to

Linda McGirr, Computer Science Department Director & Professor

This program has been developed using Oracle's JAVA 1.7.0 programming language. (JAVA Development Kit v1.7.0_45)

As-Built System Documentation

The Desrosiers Mechanics Teaching Tool is a JAVA Swing application; all the major application components, resources, and listeners are referenced in a class called Application, belonging to the programCore package. The entire application is launched from a class called DesrosiersMechanicsTeachingTool, also in the programCore package. It creates an instance of a JAVA Swing frame called LoadingUtility, which loads the actual application itself but in a way where the user can see the progress made to load it up.

There are 20 packages, 94 source code files in all, which consists of one interface, four abstract classes, and 89 source code file classes. Several resources come from external files in a directory called "Resource Files" that is not compressed, and therefore the files should not be modified because the program will display the text written as not intended, or it will crash because the file is corrupted or missing.

Here are the features of the application that have been programmed, in a technical sense:

- Six different types of menu buttons as extended instances of an abstract class called MenuBarAction, which in turn extends AbstractAction from the javax.swing package: ExerciseMenuAction for the menu associated with Exercise Mode, LectureMenuAction for the menu associated with Lecture Mode, QuizMenuAction for the menu associated with Quiz Mode, SimulationMenuAction for the menu associated with Simulation Mode, ProgramMenuAction for the Program Menu in all menu bars of this program, and OptionsMenuAction for the Options Menu in all menu bars of this program.
- Five menu bars corresponding to where the user is: CentralProgramMenuBar when the user is not in one of the four modes (Lecture, Simulation, Exercise, Quiz), LectureMenuBar for Lecture Mode,

SimulationMenuBar for Simulation Mode, ExerciseMenuBar for Exercise Mode, and QuizMenuBar for Quiz Menu Bar.

- Six drop-down menus for the five menu bars of the program: ProgramMenu and OptionsMenu for all the time the user is running the full application, LectureMenu for Lecture Mode, SimulationMenu for Simulation Mode, ExerciseMenu for Exercise Mode, and QuizMenu for Quiz Mode.
- The user roams through the modes using eleven different panels added to a main panel of card layout: the MainMenuPanel for the Main Menu; LectureSelectionPanel, SimulationSelectionPanel, ExerciseSelectionPanel, and QuizSelectionPanel to select a lecture, a simulation, a series of chapter exercises, and to initialize a quiz with options respectfully; LecturePanel for Lecture Mode; SimulationPanel, with a generalized panel for simulation controls, for Simulation Mode; ExercisesPanel for Exercise Mode; QuizPanel for Quiz Mode; TitleScreenPanel for the Title Screen; and AnimatedCreditsPanel for Credits Mode to where the user watches the animated credits.
- Two complete simulations, one on vectors (Chapter2Simulation1 with Chapter2Simulation1ControlPanel as its control panel, Chapter2Simulation1ActionListener for all action event handling of that control panel, and two document listeners, Chapter2SimulationControlPanelDocumentListener1 and Chapter2SimulationControlPanelDocumentListener2, for the text fields of the control panel), and one on uniform circular motion with animation (Chapter2Simulation6 with Chapter2Simulation6ControlPanel as its control panel, Chapter2Simulation6ActionListener for all action event handling of that control panel, the same document listeners as the first simulation, and Chapter2Simulation6Task as its animation task.).
- Four unimplemented simulations with featured GUI (Chapter2Simulation2, Chapter2Simulation3, Chapter2Simulation4, and Chapter2Simulation5, all with preview graphics and visible control panels. You can't do much with them except look at them, and assess what the full simulations would be like.)
- 10 panel listeners (except Simulation Mode which has a listener of its own depending on the control panel used) for the panels the user will explore around while in the program. They create a link between two panels that the user needs to get from one screen to the next. (AnimatedCreditsPanelListener for manipulating Credits Mode, ExerciseModeListener for the important buttons, ExerciseSelectionPanelListener for selecting exercises, LectureModeListener for important functions in a lecture, LectureSelectionPanelListener for selecting a lecture, MainMenuPanelListener for the Main Menu, QuizModeListener for Quiz Mode, QuizSelectionPanelListener for selecting a quiz, SimulationScreenPanelListener for selecting a simulation, and TitleScreenListener for accepting the Enter button when the user sees the Title Screen
- Six listeners for each of the six different menus to be used by five menu bars (ProgramMenuListener, OptionsMenuListener, LectureMenuListener, SimulationMenuListener, ExerciseMenuListener, QuizMenuListener)
- Eight different lectures, one with an image included; a prologue lecture for what the user should know when taking physics, and seven others for Chapter 2 and its sections respectfully. (PrologueLecture, Chapter2Section1Lecture, Chapter2Section2Lecture, Chapter2Section3Lecture, Chapter2Section4Lecture, Chapter2Section5Lecture, Chapter2Section6Listener, and Chapter2FullChapterLecture)
- Animation for the animated credits, the main menu (current date and time information, as it is provided at the bottom of the Main Menu), the Title Screen (fade in and fade out through change of transparency, or alpha), and the simulation on uniform circular motion
- A custom-produced splash screen as an undecorated JFrame (LoadingUtility)
- All dialog text stored as a class separate from all the other classes (DialogLabelText)
- A performance utility (ApplicationPerformanceUtility) and a credits utility (SimpleCreditsUtility).

Installation Instructions

***NOTE*: The instructions have not been tested with different computers to make sure that the program runs on different hardware and operating systems. Use the instructions at your own risk.**

The program does not have an installation utility of its own; what you simply need to do are a few steps. However, there are a few system requirements you must satisfy, as this requires the JAVA plugin:

- Operating System: Microsoft Windows (XP or later), Mac OS X, Linux, Solaris
- Internet connection (you cannot buy the plugin from a store)
- Mouse and Keyboard Input (Touchpads also work. You do not need to have a mouse with a scroll wheel since the program does not use the mouse wheel as input.)
- Preferably 130 MB disk space (minimum; according to Oracle, you need a minimum of 124 MB disk space. The remaining space will be used by the program, even though it only requires less than 3.3 MB of memory.)
- A Pentium 2 266 MHz or faster processor (according to Oracle)
- 128 MB of RAM (minimum, according to Oracle)

The JAVA plugin you must install has to be v.1.7.0 Update 45 or later since the program has been developed using the 45th update of the JAVA SE Development Kit 7.

There are no formal steps to discuss, except here are a few simple steps to install the program.

1. Install JAVA by going to http://www.java.com, clicking on the Free Download JAVA button, and following instructions
2. After downloading the project file, which is a compressed folder of extension .ZIP, extract all the files from it using Windows Explorer or Finder, if you have Microsoft Windows or Mac OS X, or an extraction tool, to a separate directory on your hard drive. It should be somewhere in your Documents folder.
3. Go to the folder called "The Desrosiers Mechanics Teaching Tool – Lite Version (Program)" from that directory.

FAQ

Here are some of the common questions users may ask about this program:

Q: How do I start the program from the directory?

A: Assuming you have the right version of JAVA installed, double-click on the JAR file from the file explorer (in Microsoft Windows, it is the Windows Explorer), then wait for the program to load.

Q: Is there a splash screen associated with the program?

A: Yes, there is. When you start running the program, a splash screen appears at the center of the screen, showing that the program is loading. There is a label in the center of the splash screen that changes as the application is loading, showing how much progress has been done to load it.

Q: In the directory where the program is located, Microsoft Windows, or my operating system, identifies the program as an unknown file. Why is that?

A: The program is a single JAR file that the only way the operating system can recognize it is by installing Oracle's JAVA plugin on their JAVA website.

Q: What programming language was used to develop this application?

A: Oracle's JAVA. It's one of the world's most portable programming languages around, on the basis that as soon as you install a JAVA plugin or a development kit, you can run the associated JAVA ready-to-run files (.class or .jar files) on any system compatible with the plugin. It's also the programming language for apps developed specifically for Google's Android OS on mobile phones.

Q: Where did you get the images, used in the program, from?

A: Only the logos for the college campus and the college board were retrieved online using Google Images. The program logo was designed on Adobe Photoshop Elements 9, my (Gregory Desrosiers, the programmer) own logo is an edit of a photo of a city bus I took back in December 2010, and the image used in one page of the lecture for the first section of Chapter 2 was created using Microsoft Paint.

Q: Why, sometimes, whenever I am selecting a simulation, the preview image is not displayed properly?

A: The preview image is generated directly in the code, instead of through an image, by using resources from the former JAVA Abstract Windows Toolkit, where because it is outdated from modern graphics, it is prone to bugs.

Q: There are 57 slides in the chapter "What do I need to know to learn physics" and 71 slides in Chapter 2 in Lecture Mode, yet only the first slide in Chapter 2 Section 1 has a diagram displayed. Why do the other slides don't have a diagram?

A: There were plans to create images for the lectures, but time has ran short and thus I had to finish everything else with the program. A single image has been rendered so that way when I did my formal presentation of this program to my course classmates, it was of better quality.

Q: I can only see one exercise in Exercise Mode when I select Chapter 2 Section 1. Why is that?

A: There was a plan to put in different exercises for all sections of Chapter 2, except that because I had to worry about my other homework and I was running out of time, I decided to let the exercises go so that I can focus more on Lecture Mode and finish off fixing the buttons and menus.

Q: I do see a lot of names in both the Animated Credits and the Credits Utility; would you please explain to me why you decided to do that instead of focusing more on the class material?

A: I did an animated credits sequence and a utility because I wanted to give credit to my classmates, my teachers, some of the past people I've worked with over the two years and eight months I spent at Champlain College Saint-Lambert pursuing a DEC, as well as the management staff. Specifically, the professors in the campus' Computer Science faculty, one academic advisor, and even the campus director.

Q: On the Performance Utility, there are a few cases where the number of threads is two instead of one. How is this true?

A: First off, a thread is a computer task assigned to a special chip called the central processing unit, or CPU. The CPU is responsible for all executions, or processing or production, for a task to be finished. A task is always in binary because the CPU works with pulses of direct current to do certain operations; for example, add 1 to 5. This binary can be expanded for the CPU to execute multiple tasks, or harder tasks, as well.

When you are in the Title Screen, the Main Menu, Credits, or Simulation Mode where the animation is running, the reason why you see two threads is two things. There is a task responsible for the animation of the simulation, as an animated simulation is created from a constant stream of changing properties of objects and frame rendering; essentially, it's changing the position of, for example, a vector, and then calling the CPU to paint the new graphics. The other task is actually part of the graphical user interface this program is built upon; if you click on a button, a radio button, a menu button, or some other component, it fires an event denoting that a component has been changed or clicked. Listeners are responsible for receiving the event and executing the event accordingly. There has to be a way for the listeners to run at all times so that way when an event is triggered, it doesn't need to wait for it to be processed. So a task keeps the listeners up, ready to receive an action that will be executed by the time a component fires an event.

Q: Why does this application only require 3.18 MB?

A: This is counting the resource files and the code in the program that access the resources provided in your JAVA plugin. Anytime you run a JAVA program, it has to access the specific resources installed in your hard drive in order for it to run properly. Plus, there are not that many images to load, so that in turn reduces memory consumption.

Q: Is there a way to change the rendering scheme for the program, such as rendering hints, fonts and antialiasing?

A: No, there isn't. There were plans for it, but was taken out of development because it would take too much of the time I had to finish what I got. The only way to change it is to modify the source code, which for a user with no or little programming experience is tedious.

General Reference Guide

Key Strokes

There are several shortcuts you can take when you are running the program so that way you don't have to click the component to do its intended action. Some actions are available only if you are in a certain mode.

Certain commands have different functions, but only the one based on the mode you're in will be done.

NOTE: Menu commands are shown in correspondence with their action buttons.

Lecture Mode

Lecture Menu Access – ALT + P

Go to First Page – CTRL + F

Go to Last Page – CTRL + G

Change Lecture – CTRL + C

Quit Lecture Mode – CTRL + Q

Simulation Mode

Simulation Menu Access – ALT + S

Start Animated Simulation - CTRL + S

Stop Animated Simulation – CTRL + O

Reset Animated Simulation – CTRL + E

Change Simulation – CTRL + C

Quit Simulation Mode – CTRL + Q

Exercise Mode

Exercise Mode Access – ALT + E

Go to First Exercise – CTRL + G

Go to Last Exercise – CTRL + I

Change Chapter – CTRL + C

Quit Exercise Mode – CTRL + Q

Quiz Mode

Quiz Menu Access – ALT + Q

Go to First Question – CTRL + F

Go to Last Question – CTRL + L

Change Quiz – CTRL + C

Quit Quiz Mode – CTRL + Q

General Key Commands

Yes Button in Confirmation Dialogs – ALT + Y

No Button in Dialogs Dialogs – ALT + N

Program Menu Access – ALT + P

Display Performance Utility – CTRL + D

Exit Program – CTRL + X

Options Menu Access – ALT + O

About Dialog – CTRL + B

Credits Utility – CTRL + R