Chapter

Software Design Considerations for Mathematics in Mobile Games

Katherine Smith, Yuzhong Shen and Anthony Dean

Abstract

A software system has been designed and developed to allow for the display, symbolic manipulation, and player entry of mathematics expressions in mobile games. Display, manipulation, and entry of mathematical expressions are traditionally difficult tasks. Increased limitations on screen space and user input when developing for mobile devices only exacerbate these difficulties. The developed software system balances considerations for ease of use and user interaction with the desire for players to be able to enter answers in a way that is more flexible and interactive than multiple choice, which is the dominant method of interactions in serious games. The software system uses a modular design to separate symbolic math software components from math display components to facilitate reuse of the software system. Additionally, the system displays mathematical expression in a way that is consistent with textbook and handwritten notations to ease the transition between the game and the classroom. Finally, the system provides affordances for natural user interaction to promote usability and engagement. This system has been used to develop a mobile game designed to help students master introductory calculus and physics at the undergraduate level.

Keywords: game design, serious games, user interaction, STEM education, human factors

1. Introduction

There are many interventions in higher education that can be implemented through mobile games. Presenting these interventions using mobile applications is supported by the fact that 92% of higher education students own at least two mobile devices [1]. Further, characteristics of mobile technology are converging with well-known pedagogical principles as they simultaneously evolve to become more personalized, learner centered, and ubiquitous [2]. Research into mobile learning effectiveness shows positive results. Additionally, incorporating mobile devices into mathematics courses specifically leads to enhanced student perception of learning mathematics [3], which has been shown to be a stronger indicator of positive outcomes than previous academic success [4].

Traditionally, individuals needed to learn systems with unique languages, such as LaTeX or MathML, in order to produce readable mathematical content electronically. Further, text input on mobile devices is difficult and cumbersome [5] and students identify hindrances typing or entering answers on mobile devices [6]. For mobile games in mathematics and most other STEM courses, it is necessary to display and manipulate mathematical expressions and formulas. In mathematics,

even entering a simple expression can require many more screen taps than one would expect which can lead to student frustration [7]. This is an issue that must be overcome as student perception of the ease of use and usefulness of mobile learning is correlated with the likelihood that they will use it in their coursework [8]. Particularly in mathematics, difficulties include display, manipulation and user input of mathematical expressions and use of symbolic methods to determine whether or not an answer is correct [9]. Complicating this further is the difficulty of balancing the limited interface with small screen size in order to provide a positive user experience [10].

Finally, there have been many concerns raised by educators and instructional designers that indicate the importance of focusing on curricular goals as the primary focus during the development of applications for mathematics education [11, 12]. Focusing on the curriculum means not only teaching the right content but teaching and displaying the content in the right way so that the knowledge students obtain from a mobile application translates into the classroom. One way this can be facilitated is by using a system that allows for the correct display, manipulation, and input of mathematical expressions and objects. While there are many computer-aided algebra systems available, many of these are either proprietary or require some method of coding the input to get the desired result making them difficult to embed in mobile applications or difficult for students to use [13].

To embed mathematics directly into mobile games, the authors have created a modular system that focuses on allowing students to easily, but freely, input expressions that are comparable with expressions in their textbooks while keeping the number of screen taps low. This software system incorporates simplicity and interactivity which have been identified as key contributors to overall usability in user interface design [10]. In addition, the system is implemented in two modules. The first is a set of classes that allow for symbolic manipulation. The second is a set of classes that aid in the manipulation and display of the expressions inside a game engine. By incorporating these classes in a game engine, other mobile applications developed in the same game engine can incorporate the work relatively easily. By keeping the symbolic classes separate, the system can be ported to a different game engine by only modifying the display classes.

The remainder of this chapter is organized as follows: Section 2 explores related work, Section 3 details the methods used to develop the system, Section 4 provides results from implementing the system in an educational STEM game, and Section 5 concludes the chapter and discusses recommendations for future work.

2. Related work

There are several examples of mobile applications that help students practice mathematical concepts up to calculus. While most of these are not games, reviewing them shows what has been accomplished in mobile mathematics display.

One example of a mobile application with practice problems in math including topics up to differential equations and multivariable calculus is Khan Academy [14]. Khan Academy is a free repository for content but has been adding questions to their website and mobile application. Another example is IXL Math [15]. This application, which requires a paid subscription, includes practice problems up to calculus. Additionally, there are numerous examples of mathematics in games for content at and below the level of basic algebra.

The previously mentioned applications tend to limit the keyboard input by problem type. If a problem does not require trigonometric functions, then these input keys will not be available. Recently, Nakamura and Nakahara developed an

interface for mathematics input that limits the number of screen taps [7]. They accomplished this by opening a submenu when a key is touched. The user then flicks up, down, left or right to insert the corresponding symbol.

The only game to the best of our knowledge that incorporates math beyond algebra is a game previously developed by the authors to cover topics in precalculus [13]. This game employed external Python libraries to generate images of the expressions needed in the game. When this communication was done at run time, there was a noticeable delay between the function call and display of the expression. Additionally, interaction was limited because the expressions were generated as images and it is almost impossible to detect user interactions with mathematical symbols embedded in images. Finally, it was necessary to distribute the core components of a Python installation with the game which increased the package size, making mobile deployment infeasible. Generating the images in advance eliminated the delay and decreased the package size but removed the ability to randomly generate problems in real time.

3. Methods

The purpose for developing this software system is to create a solution that allows for display, symbolic computation, and answer entry for introductory, undergraduate STEM courses including first courses in calculus and physics. Typical introductory calculus problems include derivatives and integrals of various functions such as polynomial, rational, and trigonometric functions. By knowing the most prevalent types of functions, templates were created for the terms of each function type. The template functions can be added by the player and then modified to construct a complete answer to a problem. While this system does not address every problem type in undergraduate STEM courses, the methodology provides a flexible model that can handle a range of common problems. Additionally, the model can be extended by developers to cover additional problem types as needed.

The modular system developed has the following characteristics:

1. Using a consistent software design that supports both expression display and symbolic manipulation and comparison.

2. Controlling spacing and sizing to optimize both overall size, for readability, and relative size of expression components, for understandability.

3. Providing affordances to support natural user interaction. That is, displaying objects in the interface in a way that makes it clear how a user can interact with them.

For this system, a major goal was to provide a user experience that allowed for more variability in question answer formatting than multiple choice or matching. To accomplish this, it was necessary to develop a system that could display a problem, perform symbolic manipulations to automatically generate the correct answer, read in the user's answer, and compare it to the correct answer.

3.1 Software design for manipulation and display

Two parallel sets of classes were developed. Since the current mobile application was developed in Unity [16, 17], the classes handling the display needed to be

closely linked to game objects and their properties. However, the classes handling the symbolic mathematics are only dependent on mathematics itself. Therefore, the symbolic math classes could be well separated from the display classes to facilitate reuse of the symbolic classes on different development platforms.

3.1.1 Development of symbolic math classes

Since many operations in mathematics are binary operations, mathematical expressions can be represented as a binary expression tree (**Figure 1a**). Because the operations of multiplication and addition are both associative, limiting those operations to two children was not necessary. This restriction was eliminated to make checking of expressions easier by allowing more than two terms that are added or multiplied to be stored in a list (**Figure 1b**). There are two types of nodes. Internal nodes are operations while leaf nodes are single terms. To emulate this in code, a set of three abstract classes was developed (**Figure 2**). These three classes are intentionally kept very general. Abstract classes allow methods to be defined without implementation. Every MathObject can be simplified, differentiated, and integrated, but the process for each operation varies for different types of mathematical objects and therefore must be defined in the classes derived from MathObject. Therefore, the MathObject class only provides these common functions that are applicable to expressions and their individual terms. The MathTerm class assumes that each term has a lead coefficient, exponent, and argument coefficient and that there is some convention for sorting these terms which will assist with symbolic comparison of two expressions. The expression class assumes that an expression is a list of MathObjects (either Expressions or MathTerms) that will be associated by some binary mathematical operation.

An example of a class deriving from MathTerm would be a TrigTerm. A generic trigonometric term can be written as follows:

$$a\sin^{b}(cx) \tag{1}$$

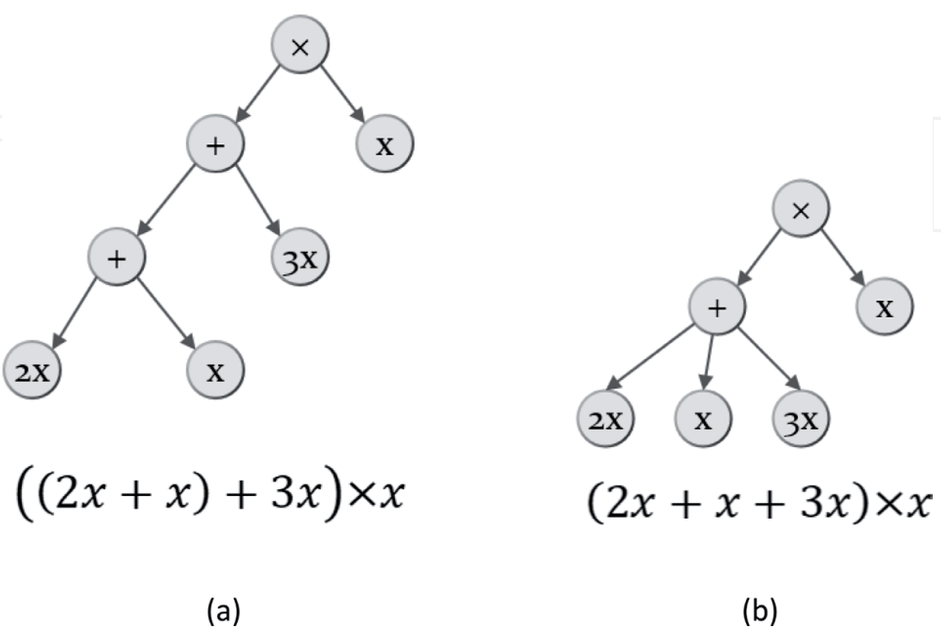


Figure 1.
(a) Traditional binary expression tree with equivalent mathematical expression. (b) Modified expression tree used in this work.

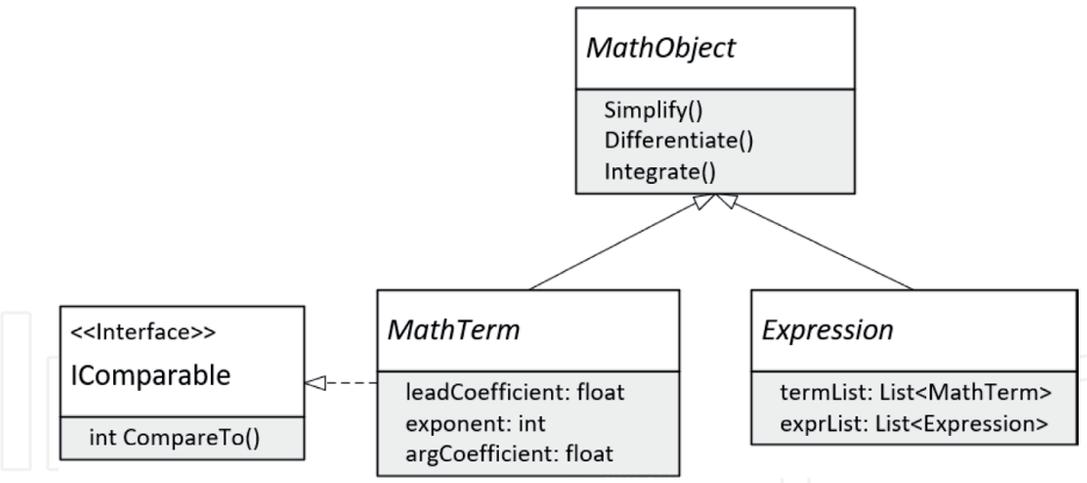


Figure 2.
Class diagram for abstract MathObject class and two derived classes, which have derived classes that are not shown in this class diagram.

In Eq. (1), *a* corresponds to the *leadCoefficient*, *b* corresponds to the exponent, and *c* corresponds to the *argCoefficient*. Additionally, an enumerated variable listing the allowable trigonometric functions is required.

Using just those four variables, methods to simplify, differentiate, and integrate a TrigTerm can be defined. For example, the differentiate function has a switch statement in which each case is determined by the trigonometric function type. The result of the differentiate method is an expression that is equivalent to the derivative of the term. Other examples of classes deriving from the MathTerm class are polynomial, rational, logarithmic, exponential, and so on.

An example of a class that can be derived from the expression class is a sum class. This class includes methods for simplifying sums by combining terms that are considered like terms. Polymorphism is utilized to allow use of various methods to determine which terms are like based on the type of each term. Additionally, the class includes overloaded operators that allow addition and multiplication of two sums or a sum and an individual term. This allows the developer to use more common notation to combine expressions. Since many mathematical expressions are handled through individual MathTerm objects, the only other expression class required so far has been a product class. Of course, differences and quotients can be represented using sums and products, so these are naturally included.

In addition to being able to express longer equations by combining individual terms through sums (differences) and products (quotients), the next level of complexity would be composite functions, i.e., functions formed by substituting one function into another function. While these are not handled in the current implementation, they would be an easy extension. Once the actual function substitution is handled, derivatives can be expressed as a product using the chain rule. Since the chain rule is recursive, only a few additional functions would need to be defined to achieve a complete implementation for composite functions.

3.1.2 Development of display classes

Once the issue of symbolic storage and manipulation is handled, the next step is to determine a way for the expressions to be displayed to the user in a manner that is consistent with textbook and handwritten notation. To preserve flexibility, a set of paired classes was designed for each class derived from MathTerm (**Figure 3**). This way if the code is ported to a different tool, the symbolic part can be easily reused.

The symbolic classes could be reused directly while the display classes would need to be rewritten for the target game engine.

An abstract TermController class was designed to handle the connection between the symbolic MathTerm class and the rendered game object. The TermController assumes that each term is composed of a list of component game objects that each have an assigned type, such as coefficient, variable, exponent, parenthesis, and so on. Additionally, each term is a part of a list of MathTerms and has an operator that connects it to other elements in the list as well as a button to delete the term. Finally, the TermController contains a list of modifiable components. This is a subset of the list of component game objects and indicates which components can be modified by the player. When the term is used for player input, each of the item in this list can be modified. For example, operators can be changed from plus to minus or times. Additionally, coefficient values can be changed. When the term is fixed as part of a problem statement, these objects are fixed as well and cannot be modified.

The TermController also provides virtual functions for updating the term values and controlling the spacing for display. These functions should be overridden by each subclass to handle cases for each term type. Since each TermController is associated with a MathTerm of a given type, a generic class inheriting from the TermController class is used as a bridge between the abstract term controller class and the derived classes. Generic classes have one or more type parameters. The generic class inheriting from the abstract TermController class has a single type parameter that must be a derived class of MathTerm. For clarity, we will refer to this class as TermController<T>. This type parameter is used to declare the myTerm variable which indicates the MathTerm that is associated with the TermController. This also allows for additional methods that are required for all TermController objects regardless of the type of the associated MathTerm, such as setting whether the term can be modified and deleting the term.

Like the derived classes from the MathTerm class, derived classes of the TermController<T> class specified were developed for each function type to handle unique aspects of display associated with that function type. An example of one of

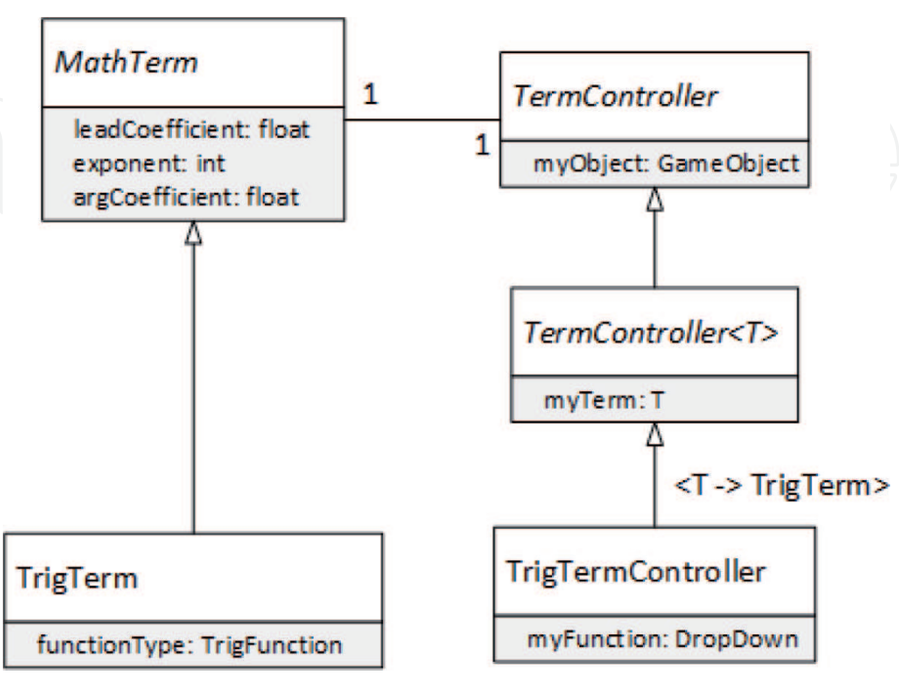


Figure 3.
Class diagram showing symbolic and display class.

these derived classes is TrigTermController which is associated with a TrigTerm by inheriting from TermController<TrigTerm>. This class has Text component variables for the lead coefficient, argument coefficient, and exponent and a DropDown component variable for the trigonometric function. These are modifiable components. Additionally, the TrigTermController class provides methods to control spacing within the term based on space available and methods for visually updating the modifiable components as well as updating the associated TrigTerm. Similar classes derived from the TermController<T> class have been developed for each class derived from MathTerm.

3.2 Adaptive sizing for display

On mobile devices, screen size is limited and therefore sizing and positioning for maximum readability without losing information is important. In mathematical expressions, this has implications at two levels. At the individual term level, the sizing and placement of parts of the expression convey meaning. The simplest examples include exponents and fractions such as y^6 and $\frac{1}{4}$, respectively. At the expression level, keeping sizing consistent for parts of terms, rather than just terms, enhances readability and understandability. For example, in an expression containing a trigonometric term with an exponent and a polynomial term, the exponents should be approximately the same size.

Methods for screen space sizing and positioning make it easy to control the sizing of terms relative to each other and the sizing of their parts relative to term size. This means that it is easy to make a trigonometric term and a polynomial term the same size overall. However, that means that their corresponding parts will not be the same size. To address this, custom methods were developed at the TermController level as well as the overall expression level to ensure consistency.

At the individual term level, it was important to establish a consistent rule for the sizing relationship between components. A static dictionary was created at the TermController level to dictate the relative widths of coefficients, variables, exponents, and so on. Another static variable was created and used to set the aspect ratio of each character. The aspect ratio allowed for the height of any character to be calculated given its width, or vice versa. These values were manipulated by the developers to come up with constant values that resulted in the display and adaptive sizing being visually consistent with a textbook presentation.

To size components adaptively based on the space available, the width of a term was calculated by summing the theoretical widths of all its components, considering that numeric terms have more than one digit. This overall term width is used to calculate a standard unit of width by dividing the maximum space the term can occupy by the overall width. Next, the height of the term is calculated by dividing the width unit by the character aspect ratio. Finally, it is determined whether the width or height is the limiting factor and the entire term is scaled accordingly by redefining the width unit if necessary. All calculations are based in screen space and it is assumed that each term can occupy a box with a given width and height. This overall box will be sized to control the ratio of the size of this term to other terms in the expression.

Once all terms are sized individually, they need to be scaled relative to other terms in the same expression. To accomplish this, the width units for the controllers are compared to find the controller that is using the smallest width. This width is then used as the standard width to resize each term based on its overall length and the available space. As a result, each term is scaled by the ratio of the standard width for the entire expression divided by the width unit for the individual term. To control vertical position, the height of the term with the smallest height is used as

the height unit for all terms. The result is an expression that scales to fill the available space as terms of different types are added and removed.

3.3 Affordances for interaction

To keep the focus on the content being presented, interactions need to be intuitive and unencumbered. As previously mentioned, any input on a mobile device can frustrate the user if not well implemented. In the present case, this was handled by trading off between flexibility and ease of interaction. There are only certain parts of the terms and expressions that the user can modify. For example, in a polynomial term with a single variable such as $3x^5$, it does not matter whether the variable is an x or a y. The meaning of the term is the same if the coefficient and the exponent are the same. This means that instead of having three parts to modify, the user only has two. In the display, parts that can be modified are shown using a different color, while static pieces are shown in black.

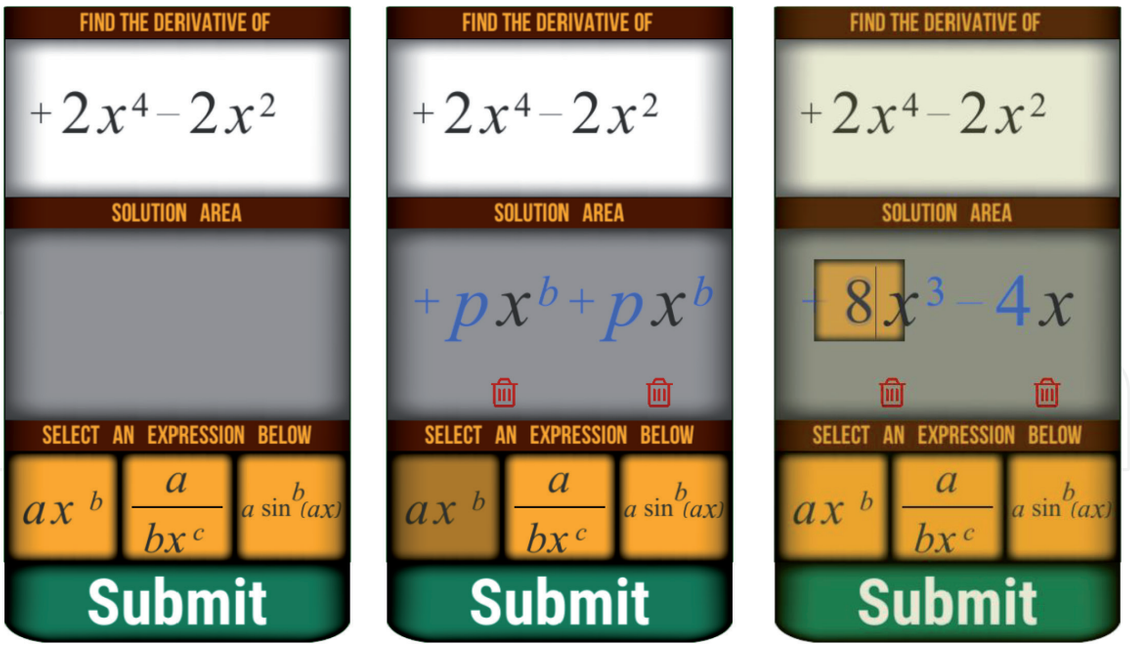
Another example of limiting flexibility to ease interactivity is in the use of predefined derived classes of MathTerm. This makes sense since mathematics instruction is usually broken down by function type. For example, it is common in calculus to learn polynomial differentiation and then move on to other function types. In the current implementation of the input system, the user is provided with buttons that allow them to add whole terms of a variety of types to their expressions. Then, they can tap various parts of the expression to modify their values. If we consider the entry of the term $3\cos^2(4x)$ using a traditional input system on a mobile device it would take at least 14 screen taps including switching between text, numeric, and symbolic keyboards at least four times. In the present system, entry of this term would take nine screen taps and require no switching between keyboards as a tap on the 3 will bring up a numeric only keyboard and a tap on the cosine functions will bring up a list of trigonometric functions. By knowing which part of the expression is being edited, the number of options can be greatly reduced from a full keyboard layout with ~30 key options to a smaller keyboard with ~10 options. Limiting the number of options reduces the time the user spends searching for the right key.

4. Results

The system developed has been implemented in a mobile game for calculus and physics. This game covers topics including derivatives, integrals, and function behavior. All problems are randomly generated at run time. In addition, the answer is determined and then compared to the user's answer.

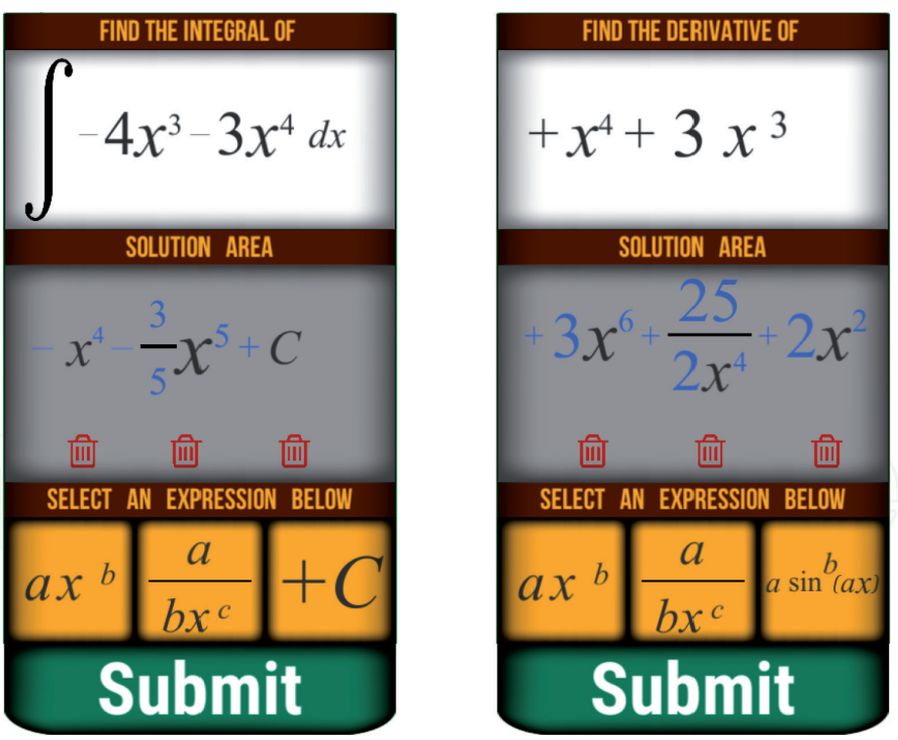
Figure 4 shows how a player would complete a problem asking them to find the derivative of a polynomial function. In **Figure 4a**, the initial problem is shown, and the user can use any of the three buttons to insert a polynomial, rational, or trigonometric term, respectively. In **Figure 4b**, the player has inserted two polynomial terms using the polynomial term button. Initially, the coefficients and exponents are not set to values and the player can tap to edit them. Finally, in **Figure 4c**, the player has entered the correct coefficients and exponents and tapped to change the middle operator to a minus. Once the player taps submit, their answer will be checked symbolically. They will be told whether they are correct and be shown the correct answer to the question.

Figure 5 shows (a) an integration problem and (b) an example of how different types of terms scale together so that the sizing of all components is consistent. In **Figure 5a**, the player is asked to find the integral of an expression containing two



(a) (b) (c)

Figure 4.
Derivative problem showing (a) problem statement, (b) after player tapped to include a polynomial term, and (c) after player tapped to add additional term, change a sign and enter coefficients and exponents.



(a) (b)

Figure 5.
More examples. (a) Integral question with fractional coefficients and (b) derivative question showing terms scaled as additional terms are added.

polynomial terms. The player has entered two polynomial terms and a constant as their answer. The lead coefficient is converted from a whole number to a fraction by tapping and holding. Then the numerator and denominator can be manipulated individually.

5. Conclusions and future work

A system has been designed and developed that allows mathematical expressions to be embedded in mobile games. This system uses a consistent software design to support symbolic manipulation and display of expressions. This design facilitates reuse even if a different game engine is used by clearly separating the classes used for manipulation from those that implement display. Additionally, spacing and sizing of expression components are controlled to enhance readability and understanding. Finally, affordances are provided to support user interaction. Expressions are constructed by combining template terms and manipulating the components of those terms to provide balance between the desire for players to have the freedom to enter a variety of answers and the complexity, in both manipulation and user interaction, introduced by allowing answer entry to be truly freeform.

There are several directions that can be addressed in future work. First, additional classes can be developed to provide the ability to solve problems with different function types. Also, when the player is presented with a question, the question is on a separate panel which removes the player from game play. It would be better to incorporate the problems more diegetically and incorporate them directly into the scene of the game. Finally, additional affordances for interaction can be included that make interactions even more natural. For example, some players may not find it intuitive to tap and hold to convert a whole number to a fraction. It would be interesting to study this mode of interaction with alternatives, such as swiping up or down, to determine which is most intuitive and provides the best outcomes.

Acknowledgements

This work was made possible through the Office of Naval Research STEM under ONR GRANT11899718.

Conflict of interest

The authors declare no conflict of interest.

Author details

Katherine Smith*, Yuzhong Shen and Anthony Dean
Old Dominion University, Norfolk, Virginia, United States of America

*Address all correspondence to: k3smith@odu.edu

References

[1] Brooks DC. ECAR Study of Undergraduate Students and Information Technology. Louisville, CO: EDUCASE; 2016

[2] Sharples M, Taylor J, Vavoula G. Towards a theory of mobile learning. In: Proceedings of mLearn. 2005

[3] Baya'a NF, Wajeeh DM. Learning mathematics in an authentic mobile environment: The perceptions of students. International Journal of Interactive Mobile Technologies. 2009;**3**:6-14

[4] Lizzio A, Wilson K, Simons R. University students' perceptions of the learning environment and academic outcomes: Implications for theory and practice. Studies in Higher Education. 2002;**27**(1):27-52

[5] Elias T. Universal instructional design principles for mobile learning. The International Review of Research in Open and Distributed Learning. 2011;**12**(2):143-156

[6] Gikas J, Grant MM. Mobile computing devices in higher education: Student perspectives on learning with cellphones, smartphones and social media. The Internet and Higher Education. 2013;**19**:18-26

[7] Nakamura Y, Nakahara T. A new mathematics input interface with flick operation for mobile devices. MSOR Connections. 2016;**15**(2):76-82

[8] Cheon J, Lee S, Crooks S, Song J. An investigation of mobile learning readiness in higher education based on the theory of planned behavior. Computers & Education. 2012;**3**:1054-1064

[9] Watson J, Angus SD. Does regular online testing enhance student learning? Evidence from a large first-year quantitative methods course. In: The Quantitative Analysis of Teaching and Learning in Higher Education in Business, Economics and Commerce: Forum Proceedings. Parkville, VIC, Australia: University of Melbourne; 2008

[10] Lee D, Moon J, Kim YJ, Mun YY. Antecedents and consequences of mobile phone usability: Linking simplicity and interactivity to satisfaction, trust, and brand loyalty. Information and Management. 2015;**52**(3):295-304

[11] Rubin A. Technology Meets Math Education: Envisioning a Practical Future Forum on the Future of Technology in Education. Technical Education Research Centers (TERC); 1999

[12] Williams DL, Boone R, Kingley KV. Eacher beliefs about educational software: A delphi study. Journal of Research on Technology in Education. 2004;**36**(3):213-229

[13] Smith K, Shull J, Dean A, Shen Y, Michaeli J. SiGMA: A software framework for integrating advanced mathematical capabilities in serious game development. Advances in Engineering Software. 2016;**100**:319-325

[14] IXL Learning. IXL Math Online Math Practice [Online]. 2019. Available from: https://www.ixl.com/math [Accessed: 20 May 2019]

[15] Khan Academy. Khan Academy [Online]. 2019. Available from: https://www.khanacademy.org [Accessed: 20 May 2019]

[16] Unity Technologies. Unity [Online]. 2019. Available from: https://unity.com/ [Accessed: 20 May 2019]

[17] Wu W-H, Wu Y-CJ, Chen C-Y, Kao H-Y, Lin C-H, Huang S-H. Review of trends from mobile learning studies: A meta-analysis. Computers & Education. 2012;**59**(2):817-827