120th ASEE Annual Conference & Exposition
FRANKLY, WE DO GIVE A D*MN
June 23-26, 2013

Paper ID #6968

From Robots to Gorillas: Computer Programming for Engineers

Dr. Dileepan Joseph P.Eng., University of Alberta

Dileepan Joseph received the Bachelor of Science degree in Computer Engineering from the University of Manitoba, Winnipeg, MB, Canada, in 1997 and the Doctor of Philosophy degree in Engineering Science from the University of Oxford, Oxford, UK, in 2003. Since 2004, Dr. Joseph has been with the Faculty of Engineering at the University of Alberta, Edmonton, AB, Canada, where he has specialized in the team teaching of computer programming and where he has developed a research program in electronic imaging. From 2012 to 2013, Dr. Joseph was a Visiting Professor at McGill University, Montreal, QC, Canada.

From Robots to *Gorillas*: Computer Programming for Engineers

Abstract

From 2009 to 2012, the author contributed to the team teaching of computer programming to thousands of first-year engineers at the University of Alberta. Despite a complex scenario, his main objectives were to improve both the course material and his teaching evaluations. Through productive collaborations with colleagues and administrators, he succeeded via a transition from procedural C++, with a virtual robot called Karel, to MATLAB, with a video game called *Gorillas*. These two versions of the course are compared and contrasted, with a focus on the author's own contributions. Furthermore, the pedagogical approach is compared and contrasted with that of relevant literature. As with the state of the art, the work argues in favour of teaching introductory programming using MATLAB. Unlike the state of the art, the proposed approach exploits video game design and iterative and incremental development. Effectiveness of the contributions are demonstrated through student, peer, and self assessments.

1. Introduction

At the University of Alberta, all 1st year engineers take a 12-week course, called ENCMP 100 Computer Programming for Engineers, offered by the Department of Electrical and Computer Engineering (ECE). This course, which has consolidated lecture notes, lab assignments, and exams, is taught by 5 lecture instructors, 1 lab instructor, and 20 teaching assistants (TAs) to about 800 students per year. In the last 4 years, during which the author was a lecture instructor, the course has undergone major changes, prompted by student dissatisfaction as expressed in 4th year exit surveys. This paper describes strategies taken in this complex scenario to improve the course material taught to students and to improve the author's teaching evaluations.

In particular, the paper compares two versions of the course. The first, in 2008–10, taught procedural C++ with a virtual robot called Karel. The second, in 2010–12, taught MATLAB with a 2-player game called *Gorillas*. Karel is a pedagogical tool, first introduced by Pattis[1] in 1981, and updated by Bergin et al.[2] in 1997. It continues to be used in introductory programming courses, such as in CS106A Programming Methodology[3] at Stanford University. *Gorillas* in MATLAB is based on *Gorillas*[4], a 1991 game by IBM distributed with MS-DOS 5.

Video game play has long been used for teaching purposes. As Marques et al.[5] describe recently, this extends to the teaching of programming. Doss et al.[6] briefly discuss how video game development can also be used for the same purpose. However, none of the 6 examples reviewed involve MATLAB. Herniter and Scott[7], and recently Azemi and Pauley[8], present advantages of introducing computer programming to engineers through MATLAB, either alone or in conjunction with C or C++. Azemi and Pauley briefly mention the design of simple games, e.g., tic-tac-toe, but focus on a robotics project and its challenges. That project required teaching of both MATLAB and C/C++ and the authors conclude by recommending against it.

This work validates changes made, despite strong differences of opinion, in 1st year engineering to go from procedural C++ with virtual robots (Karel) to MATLAB with game development (*Gorillas*). Moreover, the new course introduces important software engineering concepts, such

as iterative and incremental development (IID)[9], with programming. As Reichlmayr[10] explains, at the Rochester Institute of Technology, IID is taught to sophomore computer science and ECE students, who have already learned programming. The approach proposed here has been used successfully, as evidenced by student, peer, and self assessments, to teach a large and diverse group of Civil, Chemical, Electrical & Computer, and Mechanical Engineering students.

2. Procedural C++ with Robots

Although once taught by the current Dean of Engineering, ENCMP 100 lecture sections were taught exclusively by 2 Faculty Service Officers (FSOs) in the years preceding the 2008–9 academic year. According to the University, FSOs are "academic staff who assist and collaborate with faculty members in teaching and research." Starting 2008–9, the ECE Department Chair began to assign tenured/tenure-track (TTT) faculty, in particular those registered as professional engineers, to teach ENCMP 100. At the same time, Karel the Robot, a pedagogical tool by Pattis[1], was introduced as a stepping stone to procedural C++ programming.

In a recent interview, the Department Chair explained why he changed the teaching assignments. As with TTT faculty, student enjoyment of FSO teaching varied from instructor to instructor and year to year. However, the Chair had more options with TTT faculty, who were more numerous and more flexible. FSOs were hired for their specialized skills, which made it difficult for the Chair to reassign teaching duties if needed. Moreover, as ENCMP 100 was the only ECE course taken by 1st years, the Chair wanted to offer students a more diverse and representative group of instructors. Professional engineers were preferred for accreditation purposes.

Concurrent to the above departmental initiatives, the Faculty of Engineering mandated a task force in 2008 to review ENCMP 100. Chaired by an Associate Dean, the committee comprised 4 tenured faculty, i.e., a representative from each department nominated by its chair, as well as the senior FSO, ex officio, who was still teaching the course (the junior FSO had left the university). In its final report[11], the task force recommended that the course focus on "algorithmic thinking," with Pattis' approach given as an exemplar. The committee also recommended that the course involve MATLAB programming, either in conjunction with procedural C++ or alone.

2.1. "Algorithmic Thinking"

The task force defined algorithmic thinking[11] as "the ability to systematically break a complex task or problem into a sequence of potentially iterative but well defined simple steps. These skills have wide application across the engineering discipline in tackling project execution and system design, but are particularly necessary for writing programs that must be executed by computers that have no innate ability to interpret ambiguity." Members of the committee consulted faculty at other universities with 1st year programming courses. The success of Pattis' approach at Stanford prompted the committee to recommend adoption of Karel by ENCMP 100.

Bergin et al.'s implementation[2] of Karel, called Karel++, was adapted by the senior FSO for use in ENCMP 100. Because the course focused on procedural C++ (C programming with C++ input/output streams), the object-oriented nature of Karel++ was hidden via preprocessor macros. Students were provided a Visual C++ starter project, which included a precompiled Karel++ library, and a C++ stub file for task programming. The project was configured to load Karel's

world from a text file. Students were also provided a Windows application with which world files could be created and edited graphically. Overall, the implementation worked well.

Fig. 1 illustrates a problem from the 1st lab assignment given to students, after the 1st week of lectures. Karel is a virtual robot who lives in a Cartesian world composed of east-west streets and north-south avenues. Intersections are called corners. Karel can move directly from one corner to the next, unless there is a wall section between them. Corners, and Karel's beeper bag, may initially contain zero or more beepers. Karel can transfer a beeper from the corner he is on to his bag or vice versa. In Bergin et al.'s words[2], "a *situation* is an exact description of what the world looks like" and "a *task* is something that we want a robot to do." Fig. 1 depicts Karel sitting in his house. His task is to fetch the newspaper (pick up the beeper), return to his armchair (his initial position and orientation), and read the newspaper (put down the beeper).

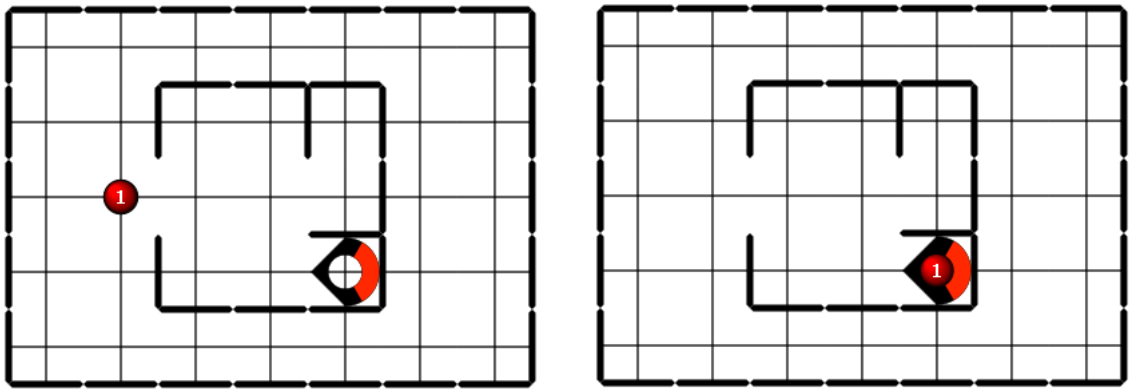

Fig. 1: Initial (left) and final (right) situations for a Karel task, from McEvoy and Iglinski[12].

An advantage of the Karel programming language is its simplicity. For instance, there are no variables. In terms of robot motion, there are only 2 primitive instructions. With `move()`, Karel takes one step in the direction he faces. With `turnLeft()`, he rotates 90° counter clockwise. New instructions (`void` functions) can be created to `turnRight()` and `turnAround()`. Together with primitive instructions `pickBeeper()` and `putBeeper()`, one can complete the task illustrated above. An incorrect program could send Karel into a wall, which causes an error shutdown. Therefore, another advantage of Karel programming is its visual nature.

Karel also has primitive senses, represented by 8 predicates (`bool` functions). For example, `frontIsClear()` returns `true` if and only if the robot is not facing a wall. New predicates may be created, and the NOT operator (`!`) is allowed. Selection (branching) and repetition (looping) is possible via `if`, `else`, `loop` (a simplified `for`), and `while` statements. These were all taught in the first 2 weeks (4 lectures) of the course, supplemented by 2 lab assignments. Notwithstanding the occasional return to Karel, the rest of the course taught procedural C++. In line with the task force guidelines, about 20% of the lectures and labs involved Karel programming.

In addition to teaching a section, and writing a share of exam questions, the author developed and led a formal Karel programming contest, based on an informal one that was previously developed by the senior FSO and lab instructor. Its goals were to encourage creativity and offer personal interaction in what was otherwise a very structured course with large classes. There were 2 categories. In the "limited" one, students were restricted to the Karel programming

language as taught. In the “unlimited” one, any C/C++ programming construct could be used with the Karel platform. Students could enter individually or in pairs, and were allowed to get advice from instructors and TAs. The contest was judged by lecture and lab instructors.

Table 1 lists the entries from Winter Term (Jan. to Apr.) contests of 2009 and 2010. In 2009, there were 20 contestants out of 674 students (3.0%). In 2010, there were 19 contestants, mostly in pairs, out of 717 students (2.6%). The course was also taught by the senior FSO in the Fall Terms (Sep. to Dec.) to 50–100 students, who also had contests. Contests were not for credit but for prizes. Instructors, and spectators who attended the contest presentations, were impressed by the quality of the entries, especially in 2010. As indicated, more students opted for the unlimited category, where the richer instruction set facilitated expression of their creativity. Moreover, games were popular, comprising 41% of all entries and 59% of unlimited entries.

Table 1: Summary of individual/team entries for Karel programming contests (Winter Term).

2009	Title	Game
Limited	Magic Squares	No
	Find the Cheese	No
	The Labyrinth Old School	No
	Karel: Search and Rescue	No
	Left Hand Rule	No
	The Karel Cube	No
Unlimited	Karel’s BeeHaviour	No
	Follow the Leader	No
	The Karel Clock	No
	BotBall	Yes
	Battleship	Yes
	Karel-Pong	Yes
	ENG-LISH	No
	Vingt-et-Un	Yes
	Karel Sudoku Solver	No
	Karel Graphing	No
	The Fox and the Hounds	Yes
	Eight Gates of Bill	Yes
	A Gift for Karel	No
	Avoid the Mines – Karel Edition	Yes

2010	Title	Game
Limited	Karel: “Rescue Project” Solver	No
	DNA – The Coding of Life –	No
	Maze	No
	Teletype Machine	No
Unlimited	Tic-Tac-Toe	Yes
	Tetris	Yes
	Conway’s Game of Life	Yes
	Destroyer of Linear Systems	No
	Karel of Duty	Yes
	Karel Bomberman	Yes
	Triage	Yes
	Karel the Word Bot	No

2.2. Programming Language

Back in 2008, there was considerable debate over what programming language(s) should be taught in ENCMP 100. It involved the highest level of the faculty administration because the 1st year program is common to all 4 departments and there was significant disagreement. The primary mission of the Associate Dean’s task force was to reach an acceptable decision. In the end, a decision was made to switch to MATLAB, effective the 2010–11 academic year. Given the size of the faculty, the vigour of the debate, and the extensive consultation done internally and externally, a brief explanation may be of benefit to instructors and administrators.

As explained recently by the ECE Department Chair, the Mechanical and Chemical Engineering representatives on the task force were strong proponents of MATLAB. The Civil Engineering

representative was open minded. The ECE representative, who was not an instructor of the course, defended procedural C++, the status quo in terms of language. Having been a party to internal discussions at the time, the author can state that the ECE representative was simply reflecting the preference of most ECE faculty who voiced an opinion. At one extreme, ECE faculty felt that MATLAB was merely a "glorified calculator" and therefore unsuitable for teaching programming. Many agreed with this somewhat. A few disagreed.

After considering multiple languages, the task force recommended keeping procedural C++ but making other changes to address student dissatisfaction, expressed most clearly in 4th year exit surveys. This initial report was rejected by the faculty's Academic Planning Committee (APC). Returning to work, the task force consulted the lead instructor of a 1st year programming course, which used only MATLAB, at Georgia Tech University. His experience featured in the final report[11], which provided 2 options. Although a hybrid course with both procedural C++ and MATLAB was recommended, APC chose the MATLAB only option. The ECE Associate Chair for Undergraduate Studies, who sat on APC, was impressed by the textbooks available to teach programming with MATLAB. Plans were made to teach ECE students C/C++ in 2nd year.

Despite the lack of consensus over continuing with procedural C++, "because most of the Faculty's nine undergraduate programs don't tend to utilize [it] in subsequent courses," there was consensus[11] that "a common core level of programming instruction be present in our programs… given the cross-cutting impact of computing on the engineering profession." There was also consensus that it was important to emphasize "algorithmic thinking" and that the Karel approach was good for this purpose. As no MATLAB implementation of Karel existed, the senior FSO, who had just adapted Karel++ for the procedural C++ course, which began in Sep. 2008, was given time to adapt it for the new MATLAB course, which would begin in Sep. 2010.

3. MATLAB with *Gorillas*

Programmers talk about top-down design and bottom-up implementation. A large 1st year course like ENCMP 100 is shaped by similar forces. On the one hand are recommendations from a multi-department task force, overseen by the faculty's powerful APC. On the other hand are a group of instructors assigned by the ECE Department Chair to actually develop and teach the course. Of relevance, the senior FSO opted, in 2010, for a University-wide retirement incentive for which he was eligible, and 3 new instructors, all tenured faculty, joined the teaching team. Owing to a tenure application, the author was excused from the development effort.

The team of 4 lecture instructors, excluding the author, and 1 lab instructor decided to retire Karel. Most were unfamiliar with Karel and, although the FSO had successfully completed MATLAB wrappers for Karel++, they had not been beta tested. The team also had good pedagogical reasons. First, they wanted to avoid syllabus duplication, where concepts taught with one programming language were re-taught with another. Second, a MATLAB course enabled visual examples easily enough without Karel. In retrospect, the author agrees.

However, the new course was short 2–3 weeks of lecture material, as all instructors, including the author, discovered during the semester. To fill the gap, the author compared and contrasted the new course with the old one. What Karel had provided, via the programming contest, was a way to unify multiple knowledge parts into a single creative whole. Also, having mentored Karel

contestants, the author understood the practical advantages of the IID model[9] of programming over the waterfall model being taught, and knew that 1st year engineers could appreciate it.

With Karel, over 40% of student-led contest entries were games (Table 1). Therefore, the author decided to teach students how to program a game in 6 versions using the IID model. Each time a syllabus part was completed, the game was revisited and an improved version created. Entitled *Gorillas* in MATLAB, the game is based on *Gorillas*[4] (Fig. 2), a 1991 game by IBM. In this artillery game, which requires angle and velocity inputs each turn, 2 gorillas throw exploding bananas at each other atop a city skyline. The computer can play either or both players.

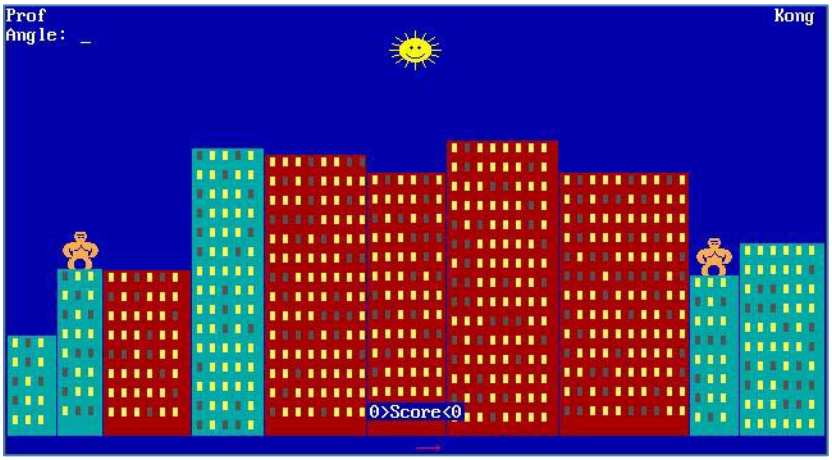

Fig. 2: Screenshot of a *Gorillas* remake, programmed by Moly[13].

3.1. Method of Development

Unlike the waterfall model, which is linear, the IID model puts requirements specification, implementation (coding), and verification (testing) in a circular loop. Each iteration of the loop results in a usable prototype, which evolves incrementally. *Gorillas* in MATLAB is developed in 6 iterations (Fig. 3), each corresponding to a separate lecture. Depending on the way a version is taught, e.g., with or without usage of the prepared code, less or more of the lecture time may be used, as the instructor prefers. IID has a long history, according to Larman and Basili[9], although it is now taught[10] with agile software development, which is relatively modern.

In addition to the MATLAB code, which is available under a permissive free software licence, 30 PowerPoint slides are provided to introduce/conclude each version and the whole project. There are 3–4 slides per version. Table 2 gives bullet points verbatim from the Requirements and Implementation slides of each version. Each version also has a Testing slide that includes a representative example of testing, which is done in class by the instructor. Versions 1 and 6 also have Analysis slides, with figures and formulas that apply Newtonian mechanics.

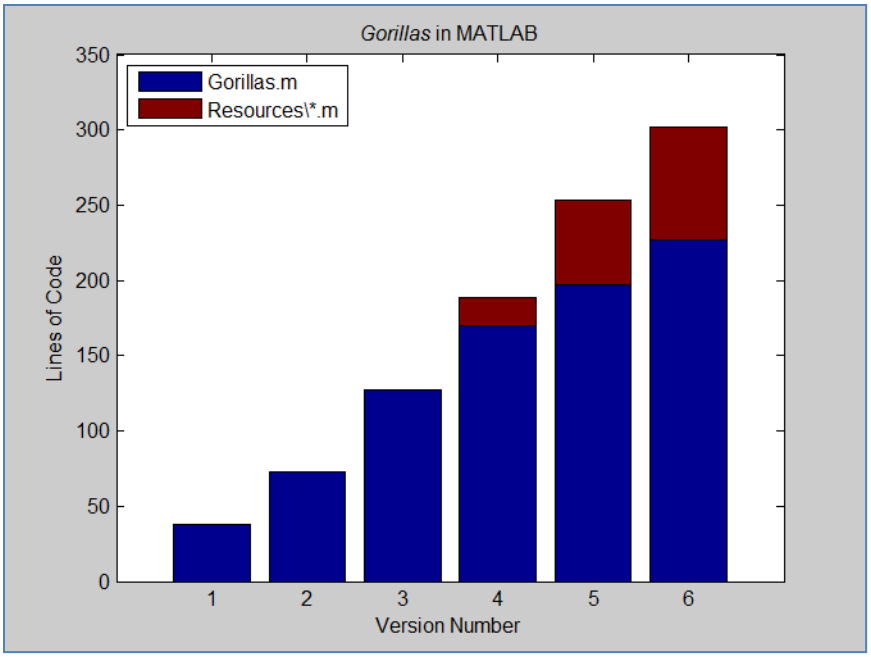

Fig. 3: Program size, excluding comments and blank lines, vs. iteration.

Table 2: Bullet points from lecture slides presented before/after a game version is developed in class.

Version	Requirements	Implementation
1. Basics and Plotting	• The game involves a stage and two gorillas, both of whom simply are human users. • The stage is randomly generated each time, as are the positions of the two gorillas. • Each gorilla hurls a bomb-like banana, starting with the gorilla facing east. • A gorilla loses if a banana splatters at its feet; the other gorilla wins (unless both lose).	• Game action takes place in a **figure** window with **input** typed in the command window. • The stage and bomb trajectories are **plot**ted; **text** annotations indicate gorilla positions. • Given angle and velocity, **round**ed, banana motion is computed from Newton's laws. • A banana is assumed to splatter where its motion **plot** intersects the stage "**stairs**".
2. Selection and Repetition	• Continue the game, i.e., give each gorilla a turn to hurl a banana, until a gorilla wins. • Animate hurled bananas and detect when they hit the stage to stop their motion. • Because they are bombs, show on the stage where previous bananas have splattered. • Detect whether a bomb lands at a gorilla's feet to determine and announce a win.	• A **for** loop, used to give each gorilla a turn, is nested in a **while** loop, used to repeat play. • Banana motion is animated with a **while** loop; **find** and **if** statements detect stage hitting. • Vectors track the x-y coordinates of splatters, which are indicated on the stage by a **plot**. • Two relational expressions, **if**/**elseif** and **break** statements, and a **logical** variable are used to identify a win and **end** the nested loops.

3. Functions and Structures	• Improve code readability, maintainability, and extensibility, i.e., reduce complexity. • Introduce Kong, a computer player who will automatically hurl a banana given a turn. • Allow User versus User, User versus Kong, Kong versus User, and Kong versus Kong modes, where User is a human player. • When one gorilla wins, give the option for another game to be played if desired.	• A long script is refactored into a short primary **function** and five short sub-**function**s; related variables are grouped into **struct**ures. • Employing **rand** to pick parameters, a sixth sub-**function** adds the computer player. • Using a **menu** and **switch**es, two **function**s are edited to assign User and/or Kong to gorillas. • Via a **while** loop and **menu**, multiple games are enabled by also editing two **function**s.
4. Files, Strings, and Cells	• Enable each gorilla to "talk" to its adversary, e.g., to hurl taunts along with bananas; the dialog may be in text form, as in comics. • Prioritizing personality over intelligence, make Kong, the computer player, more interesting even if his strategy is poor. • Acknowledge sources of inspiration and material incorporated from others; credit should be given where credit is due.	• Gorilla talk is read from a file into a **cell** array of word-wrapped strings (nested **cell** arrays); **rand**omly-chosen dialog is plotted as **text**. • An IMDb page is read with **urlread** and parsed with **strtok** to obtain Grimlock quotes, which define Kong's character after some edits. • User-defined **function**s are re-used to display acknowledgements in a **figure** window; credit is given to *Gorillas* and *The Transformers*.
5. Images and Sounds	• Instead of text symbols, illustrate gorillas during the game using suitable images. • Prior to each game, display a title screen, e.g., a gorilla-versus-gorilla face-off. • Incorporate sound effects for all significant game events, e.g., the title screen. • Update acknowledgements, as needed, to ensure external material is credited.	• Gorilla clip art, found online, is included using **imread**, **image**, and basic image processing. • A title screen is created in the **figure** window using **subplot**, **image**, and **axis** commands. • After preprocessing, sounds are **play**ed with a **global** variable and the **audioplayer** function. • Credit is given to *Nintendo* and *MathWorks*, original sources of the images and sounds.
6. Software Engineering	• Engineer a MATLAB game, like *Gorillas*, with an endearing computer player called Kong. • Make Kong smart enough to pose a challenge but not too smart (to keep him interesting). • Introduce Prof, a computer player who is hard to beat, like a final boss; include a personality befitting this "main antagonist in the story". • Incorporate acknowledgements, as needed, to ensure external material is credited.	• *Gorillas* in MATLAB (V5) is engineered using iterative and incremental development. • Kong is reprogrammed to compute accurate velocities, for **rand**omly-chosen angles, but opponent distance is perturbed via **randn**. • Prof, easily added via code modularity and reusability, professes words of wisdom while frustrating opponents with precise bombs. • Gandhi is acknowledged with apologies.

Words are emphasized, as shown in Table 2, to reinforce MATLAB vocabulary (bolded words) and link to external resources (underlined words). For example, with Version 4, students are shown how to scan an Internet Movie Database (IMDb) page, having movie and TV scripts, to extract all quotations from a *Transformers* character (an endearing robot named Grimlock, who becomes the voice of the first computer player—a gorilla named Kong). Because the project is released freely for educational purposes, such usage is permitted by copyright law.

Fig. 4 shows screenshots from Version 6 of the game. As shown in Table 2, each version has a title, and the last version is called Software Engineering. This is because it is taught, in the last class, after software engineering concepts are introduced in the last part of the course. Students recognize that the waterfall model has not been used, which leads to a discussion of the IID model. Other concepts, such as modularity and reusability, are exploited to introduce another computer player, the final boss, to the game. As shown in Fig. 4, he is hard to beat.

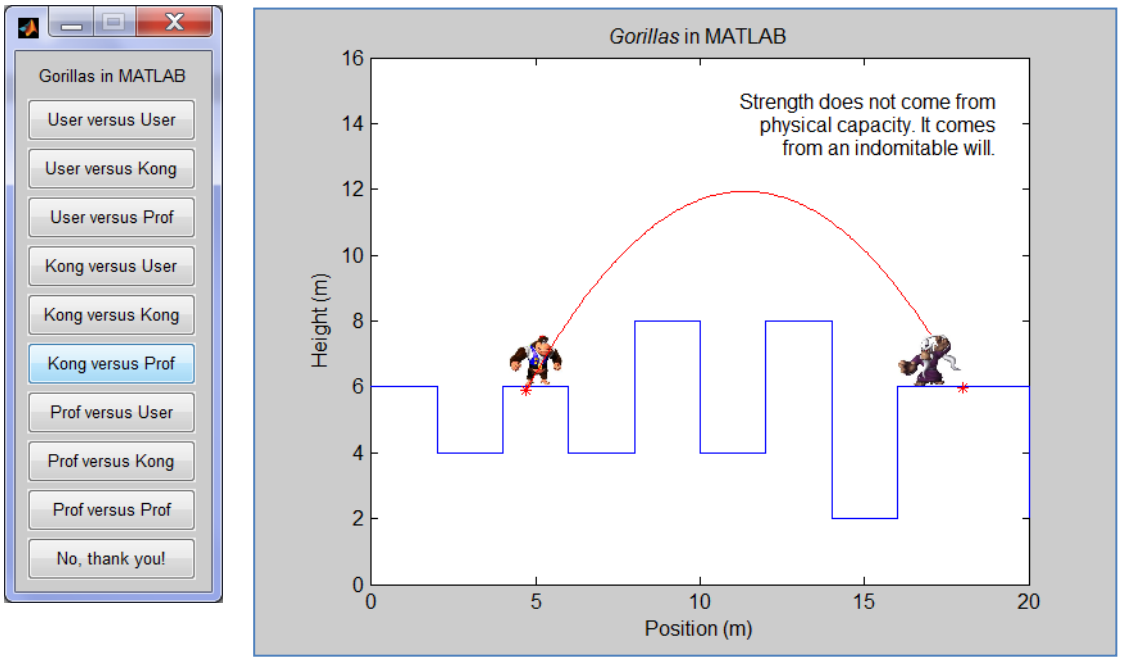

Fig. 4: Screenshots from *Gorillas* in MATLAB (Version 6), programmed by the author.

3.2. Syllabus Covered

ENCMP 100 is composed of 24 lectures over 12 weeks, 2 lectures of which are used for review prior to the midterm and final exams. The remaining 11 weeks may be divided into a syllabus of 6 parts, where the first 5 take about 2 weeks each and the last takes 1 week. Each version of *Gorillas* in MATLAB is taught at the end of each part. Because the lecture notes, comprising 567 PowerPoint slides created by other instructors, are not publically available, Table 3 correlates the syllabus with sections from the course textbook. The table also lists the MATLAB vocabulary (keywords, functions, and operators) introduced with each version, which is a representative subset of the vocabulary taught in the course and by the textbook sections.

Table 3: MATLAB vocabulary and Attaway[14] sections introduced with each program version.

Version	MATLAB (R2012)	Attaway (2/E)
1. Basics and Plotting	`axis`, `ceil`, `clc`, `clear`, `close`, `cosd`, `disp`, `figure`, `hold`, `input`, `length`, `plot`, `rand`, `round`, `sind`, `size`, `stairs`, `text`, `title`, `xlabel`, `ylabel`	1.1–1.6, 5.2, 2.1–2.5
	algebraic and colon operators	
2. Selection and Repetition	`abs`, `break`, `case`, `clf`, `elseif`, `end`, `false`, `find`, `for`, `if`, `pause`, `switch`, `true`, `while`	3.1, 5.4, 3.2–3.7, 4.1–4.3, 5.1, 5.5–5.6
	logical and relational operators	
3. Functions and Structures	`else`, `floor`, `function`, `isequal`, `menu`, `num2str`, `otherwise`, `struct`	2.7, 5.3, 6.1–6.5, 8.2
	dot operator	
4. Files, Strings, and Cells	`cell`, `fclose`, `feof`, `fgetl`, `fopen`, `fprintf`, `isempty`, `strcmp`, `strtok`, `strtrim`, `urlread`	2.6, 7.1–7.4, 8.1, 9.1–9.3
5. Images and Sounds	`audioplayer`, `global`, `image`, `imread`, `load`, `play`, `playblocking`, `save`, `subplot`	14.1–14.2
6. Software Engineering	`erfinv`, `randn`, `sqrt`, `tand`	

The textbook is divided into 2 parts: (1) Introduction to Programming using MATLAB, which has 10 chapters; and (2) Advanced Topics for Problem Solving with MATLAB, which has 5 chapters. *Gorillas* in MATLAB correlates very well with all sections of Chapters 1–9 of Part 1 and 2 sections of Part 2. There are 5 significant differences. First, a few sections, e.g., (2.6) Introduction to File Input/Output (Load/Save), are placed with full chapters, e.g., (9) Advanced File Input and Output. Second, an entire chapter, i.e., (5) Vectorized Code, is redistributed, e.g., (5.4) Logical Vectors is placed after (3.1) Relational Expressions. Third, different functions are occasionally used for the same basic purpose. For example, in (9.2) Writing and Reading Spreadsheet Files, the textbook introduces `xlswrite` and `xlsread` as advanced file operations. In Version 4, *Gorillas* in MATLAB introduces `urlread` instead for the same purpose.

Compared to the textbook, and slides by the other instructors, the most significant difference with the *Gorillas* in MATLAB approach is the way in which structures and cells are introduced. Instead of taking them together as heterogeneous data structures, structures are introduced after functions for encapsulation of related data and related processing, respectively. Cells are introduced with strings because, with string arrays, there is a desire for numerical indexing but also a desire for variable-length strings, requirements that are elegantly implemented with cell arrays. Hence, (8.1) Cell Arrays and (8.2) Structures are separated in the syllabus.

Unlike the course, the textbook does not intend to introduce software engineering. For this part, the other instructors edited slides previously made by the senior FSO for the procedural C++ course. Because this material did not touch IID or refactoring, the author's slides (Table 2) link to corresponding Wikipedia pages, which were deemed sufficient for in-class discussion.

4. Evaluation and Reflection

Over its history, the evolution of ENCMP 100 has involved many stakeholders: administrators, instructors, and students. The impact of the transition from procedural C++ (2008–10), featuring

Karel the Robot, to MATLAB (2010–12), featuring IID with *Gorillas*, is best assessed with a mixed approach. This includes quantitative and qualitative results from student, peer, and self assessments. Of relevance, the Faculty of Engineering[15] and the University of Alberta[16] themselves recognize mixed approaches to teaching assessment.

4.1. Student Assessments

In terms of quantitative results, the student rating of instructor excellence is of paramount importance internally. For example, the Faculty of Engineering holds[15] that "Promotion to the rank of Professor cannot be granted to individuals whose overall… Instructor Excellence median rating… is less than 3.5 out of 5 in three or more of the preceding five years." This rating is obtained from a mandatory in-class survey, conducted by administrative staff, toward the end of each course. Students do not know exactly when the survey will take place. On a scale of 1 to 5, where 1 represents "strongly disagree" and 5 represents "strongly agree," the rating is the interpolated median response to the statement "overall, this instructor was excellent."

Table 4 gives the student ratings of instructor excellence for all courses taught by the author since 2006. The author never demanded attendance. With ENCMP 100, all course material was available online and the author had an 8 AM section. In 2012, there was a major snowfall the morning of the survey. Ratings of other instructors are private, but there is another reason to focus on the author's ratings – it factors out instructor variability. He was the only one to teach both versions of the Winter Term course, which differs from the Fall Term one.

Table 4: Student ratings of instructor excellence for all the author's courses since Sep. 2006.

Year	Course	Class	Sample	Rating
2011–12	ENCMP 100 Computer Programming for Engineers	183	50	4.5
	EE 231 Numerical Analysis for Elec. and Comp. Eng.	74	48	4.7
	ECE 541 Digital Signal Processing	18	14	4.7
2010–11	ENCMP 100 Computer Programming for Engineers	155	66	4.7
	EE 231 Numerical Analysis for Elec. and Comp. Eng.	80	48	4.0
	ECE 541 Digital Signal Processing	30	23	4.7
2009–10	ENCMP 100 Computer Programming for Engineers	163	87	4.3
	EE 231 Numerical Analysis for Elec. and Comp. Eng.	77	55	4.7
	ECE 541 Digital Signal Processing	40	31	4.5
2008–9	ENCMP 100 Computer Programming for Engineers	153	65	4.4
	ECE 541 Digital Signal Processing	27	22	4.8
2007–8	EE 231 Numerical Analysis for Elec. and Comp. Eng.	43	22	4.7
	EE 317 Electromagnetics for Computer Engineers	3	n/a	n/a
	ECE 541 Digital Signal Processing	20	17	4.8
2006–7	EE 231 Numerical Analysis for Elec. and Comp. Eng.	50	20	4.9
	EE 317 Electromagnetics for Computer Engineers	8	8	*5.0
	ECE 541 Digital Signal Processing	28	26	4.6

*Unofficial survey (fewer than ten students).

With a consolidated course, instructor-specific ratings emphasize the individual's role, unlike course-specific ratings. In 2008–10 and 2010–12, students were told that the author's main contributions to course development were the Karel programming contest and *Gorillas* in MATLAB, respectively. With the transition, the author's ratings rose from 4.3–4.4, subpar for

him, to 4.5–4.7. Typically, 4.3 and 4.7 are the median and top quartile ratings in the faculty for all undergraduate courses (4XX and below): large or small; consolidated or not; with labs or without; and 1st, 2nd, 3rd, or 4th year. A 4.5 or higher rating is possible if and only if 50% of students or more "strongly agree" that "overall, this instructor was excellent."

There is room for improvement, of course. In addition to responding to the multiple-choice survey, some students left comments. The majority of these comments were critical of the consolidated midterm exams, which proved harder in 2012 than in 2011, and in 2010 than in 2009. These multiple-choice exams were written by all lecture instructors. Each year, a new exam was written because previous ones were released to students for practice. Multiple-choice exams are difficult to get right. When the author introduced multiple choice to EE 231 in 2011, his rating experienced a significant dip – the 4.0 outlier. He edited the exam based on statistical feedback and his rating recovered the following year. Since 2012, the ENCMP 100 team has agreed to stop releasing prior exams but instead to maintain and update two banks of similar exam questions, one for practice and one for assessment. Because *Gorillas* in MATLAB was used by all sections for the first time in 2011–12, no exams questions were based on it.

The impact of the 2010 changes to the 1st year course on 4th year exit surveys will begin to be known in 2014. Nevertheless, as he expressed in a recent interview, the ECE Department Chair is optimistic. Student representatives are no longer complaining to him about the course. Indeed, as evidenced by a course entry survey (Table 5), antagonism toward the course on its 1st day, from a diverse group of students, has declined from 20% to 15% over the past 3 years.

Table 5: Answers to selected questions from an entry survey taken in all course sections.

Question	Answer	2010		2011		2012	
I intend to enter the following Engineering department:	Civil and Environmental	169	28%	144	22%	157	24%
	Chemical and Materials	175	29%	206	32%	184	28%
	Electrical and Computer	120	20%	135	21%	131	20%
	Mechanical	146	24%	166	25%	188	28%
	Total	610	100%	651	100%	660	100%
My level of interest in learning about programming is:	Very interested	112	18%	90	13%	93	13%
	Interested	212	33%	243	35%	282	40%
	Neutral	185	29%	231	34%	215	31%
	Prefer not to	97	15%	96	14%	93	13%
	Strongly against it	30	5%	26	4%	16	2%
	Total	636	100%	686	100%	699	100%

4.2. Peer and Self Assessments

In 2010–11, *Gorillas* in MATLAB was taught only in the author's ENCMP 100 section, while it was being initially developed. Other instructors filled the 2–3 week gap in the course material their own way. After seeing it, when classes were over, other instructors decided to adopt the material for their own sections. Once further development was complete, *Gorillas* in MATLAB was taught to 768 students in the Winter Term of 2012. In addition to them adopting it, and offering positive oral feedback, written comments from peers ranged from simple expressions of gratitude to "That is a very nice effort from you to make the course more interesting."

Having taught both versions of ENCMP 100, i.e., procedural C++ with robots and MATLAB with *Gorillas*, the author is able to compare them. Karel aside, the senior FSO spent 8 years (2000–8) developing the procedural C++ course, with contributions from the junior FSO and lab instructor. Whereas all material was very well prepared, 1st year engineering students, 80% of whom did not intend to pursue ECE, could not appreciate it enough because, as the task force put it[11], procedural C++ "requires a certain minimum skill level and overhead to use effectively that can be frustrating to students." In contrast, *Gorillas* aside, the lecture and lab material prepared for the MATLAB course is wanting of polish. However, the language is intrinsically easier to learn, like Karel++, and students use it in other courses, even in 1st year engineering.

Karel++ was appealing because of its simplicity. Although not for credit, 50% more students in the author's section, totalling 6.1% of his class, participated in the programming contest in 2010 than in 2009. Participants were diverse. Winners of the unlimited category in 2010 went on to study Civil and Mechanical Engineering. Participants also valued the mentorship they received, some asking for letters of reference years later. Contest participation helped 1 student win a major national scholarship shortly after finishing the course. With time, it may have been possible to change the format of the contest to benefit more students, e.g., making it for credit and involving TAs as judges. However, Karel was retired in the transition to MATLAB.

Gorillas in MATLAB impacted several hundred students quickly, because of its teacher-centred approach and its adoption by peers. The contribution also resulted in what the author believes are significant improvements. It unified the syllabus much better than did Karel, who was barely employed in class after the first 2 weeks (the contest was held out of class). It introduced important software engineering concepts, like refactoring and IID, together with programming. As a video game with computer players, it provided a direct opportunity to discuss the use of software to emulate personality and intelligence, which students found engaging. Finally, thanks to IID, it provided all students, not just those who participated in a contest, a taste of how to develop a relatively large program over a relatively long period of time.

5. Conclusion

Gorillas in MATLAB is available for download from the author's website. This includes source code, data files, and PowerPoint slides for 6 lectures. Comprising up to 25% of a 12-week course at the University of Alberta, the material is used to teach introductory computer programming. For the remaining 75% of the course, a complementary syllabus is provided in this paper, based on all sections of the first 9 chapters of a textbook by Attaway[14], plus additional material from Attaway and other sources. The syllabus is divided into 6 parts. At the end of each part, a version of *Gorillas* in MATLAB is taught. This provides a unifying framework for the course.

Relative to a previous course on procedural C++ with Karel the Robot, students expressed their preference for the current course by giving the author a significantly higher instructor excellence rating, the most important quantitative student evaluation in the faculty. Contributions of faculty and departmental administrators, as well as other instructors, to course development were explained. Karel also played a part, as a previous programming contest influenced the author's contributions. Despite a complex political process, top-down and bottom-up forces combined to yield an improved course, as evidenced by student, peer, and self assessments.

As with recent literature, this work supports the teaching of introductory computer programming with MATLAB instead of C/C++, which is topical. Unlike the literature reviewed, it provides a detailed account of how the teaching of programming is facilitated by the development of a complex video game. For teaching purposes in general, the design of video games, as opposed to the playing of video games, is also topical. Finally, *Gorillas* in MATLAB is distinguished by the introduction of IID, an important software engineering concept, to a 1st year programming course, one that includes Civil, Chemical, and Mechanical Engineering students.

Acknowledgements

In addition to the colleagues and administrators mentioned above, the author is grateful to the ENCMP 100 students whom he has taught over the years for their valuable feedback. Finally, this paper is dedicated to the memory of Professor Nelson Durdle (1943–2013), Director of Computer Engineering at the University of Alberta from 2007 to 2009.

References

[1] Richard Pattis, Jim Roberts, and Mark Stehlik, *Karel The Robot: A Gentle Introduction to the Art of Programming 2/E*, Wiley, 1994.
[2] Joseph Bergin, Mark Stehlik, Jim Roberts, and Richard Pattis, *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*, Wiley, 1997.
[3] Eric Roberts, "Karel the Robot learns Java," Department of Computer Science, Stanford University, 2005.
[4] Wikipedia Contributors, "*Gorillas* (video game)," en.wikipedia.org/wiki/Gorillas_(video_game), 2012.
[5] Bradley Marques, Stephen Levitt, and Ken Nixon, "Video Games as a Medium for Software Education," *Proceedings of the IEEE International Games Innovation Conference*, 2012.
[6] Kathlyn Doss, Valerie Juarez, Daniel Vincent, Peggy Doerschuk, and Jiangjiang Liu, "Work in Progress – A Survey of Popular Game Creation Platforms Used for Computing Education," *Proceedings of the ASEE/IEEE Frontiers in Education Conference*, 2011.
[7] Marc Herniter and David Scott, "Teaching Programming Skills with MATLAB," *Proceedings of the ASEE Annual Conference and Exposition*, 2001.
[8] Asad Azemi and Laura Pauley, "Teaching the Introductory Computer Programming Course for Engineers Using Matlab," *Proceedings of the ASEE/IEEE Frontiers in Education Conference*, 2008.
[9] Craig Larman and Victor Basili, "Iterative and Incremental Development: A Brief History," *Computer*, vol. 36, no. 6, 2003.
[10] Thomas Reichlmayr, "The Agile Approach in an Undergraduate Software Engineering Course Project," *Proceedings of the ASEE/IEEE Frontiers in Education Conference*, 2003.
[11] Steven Dew, Jos Derksen, Peter Steffler, Nelson Durdle, Carlos Lange, and Paul Iglinski, "Report of the ENCMP 100 Task Force," Faculty of Engineering, University of Alberta, 2008.
[12] Sarah McEvoy and Paul Iglinski, "ENCMP 100 Computer Programming for Engineers: Assignment 1," Faculty of Engineering, University of Alberta, 2010.
[13] Moly, "Gorillas.bas," www.kongregate.com/games/Moly/gorillas-bas, 2008.
[14] Stormy Attaway, *MATLAB: A Practical Introduction to Programming and Problem Solving 2/E*, Elsevier, 2011.
[15] Engineering Faculty Council, "Faculty Evaluation Standards and Procedures," Faculty of Engineering, University of Alberta, 2010.
[16] M. Anne Naeth, *Teaching Resource Manual*, University Teaching Services, University of Alberta, 2003.