Game-Creative Learning in Programming Courses Over 15 Years

Emília Pietriková, Norbert Ádám and Anton Baláž

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 04200 Košice, Slovakia
E-mail: emilia.pietrikova@tuke.sk, norbert.adam@tuke.sk, anton.balaz@tuke.sk

Abstract: This article highlights the impact of game creation on an introductory programming course (CS0, CS1) combined with problem-based learning. It introduces and defines the Game-Creative Learning (GCL) approach, which gradually transforms traditional 'learning from scratch' methods into an innovative framework where students develop their own computer games while learning a new programming language. Over 15 years, this approach has engaged over 10,000 students – novice programmers, addressing educational challenges and leading to higher course quality, increased motivation, and better student engagement. Beginning with simple tasks like playing with Karel the Robot, students' progress to creating their robots and ultimately developing 2D games. Throughout the course, scores and levels of engagement are assessed and tracked in a competitive environment. This approach aligns with Bloom's taxonomy by guiding students progressively through its cognitive levels.

Keywords: Game-Creative Learning; Computer Science Education; Programming Education; Bloom's Taxonomy; Courseware; Higher Education; Problem-Based Learning; Novice Programmers; Karel the Robot

1 Introduction

Gamification is not a new concept. As early as the 17th Century, J. A. Comenius called for educational innovations, including the introduction of games into education [1]. In the last two decades, with the advent of new technologies, the popularity and importance of computer games significantly increased, which also affected students [2]. Today's students are not the breed their educators used to be. M. Prensky in his well-known publication [3] states that today's students are digital natives while their teachers are digital immigrants. His study is a powerful message to teachers, a call to integrate modern information technology into academic courses as their natural component.

We know the industry demands skilled programmers. However, today's students differ from those for whom standard programming courses were originally designed. These courses typically begin with 'Hello world!' programs, move through basic I/O formatting and simple mathematical operations, and culminate in challenging programming concepts. Such an approach is often unattractive to modern students. Even if the educational process uses computers, the practical outputs of the course usually consist only of a simple screen with a few numbers. Students consider the idea behind these numbers to be uninteresting and the practical application of these to be unrealistic.

This paper describes the innovations applied to an introductory programming course (CS1 or CS0) implemented gradually over 15 years. This process has impacted over 10,000 novice programmers over the course of 15 years. As stated in [4], there is no need for innovation if the process is working well, so our work relies on the successful implementation of game-based solutions of multiple studies like [5] or [6]. Reflecting [7], we designed an educational system fitting the needs, goals, and interests of students wanting to learn programming.

We named this approach *Game-Creative Learning (GCL)* as it is a subset of problem-based learning (PBL), influenced by elements of gamification and game-based learning (GBL) [8]. PBL has been popular mainly in the last two decades [9]. Originally, PBL was popularized by [10], following their research concerning students' reasoning abilities. The approach arose from the desire to give students an opportunity to apply practical and theoretical knowledge to problems within their trade, replicating features of real-life application contexts. This means a clear move away from the previous approach. To date, PBL is considered a relatively stable learning approach, delineated by specific ways of implementation [11]. Since then, many studies have been published – most report primary concerns linked to PBL, for instance the prescriptive model by [12], proving that the presence of the task model results in spending much more time on discussing the problem situation and on planning and selecting the activities to be performed.

Recently, gamification has risen due to many influential studies, including [13] or [14]. One of its goals is to create close ties between the users and the environment to increase its popularity [15]. In education, the environment is a set of tools to increase student engagement, i.e., one of our primary motivations. An example of this is the use of an educational MMORPG called CMX, which has demonstrated improvements in students' learning and motivation in programming [16].

The following study is dedicated to programming teachers intending to transform their practices interactively to raise skilled programmers. The hypothesis: *If we use GBL and PBL to innovate our educational approach used in academic courses while preserving their original content and aims, we will increase the knowledge and engagement of our students.* In line with [7], we build an effective learning environment to train good programmers and achieve everyday engagement.

2 Game-Creative Learning

We coined the term *Game-Creative Learning* to denote learning by creating a custom computer game in three stages – see Fig. 1. In *the first stage*, students play a game to learn elementary programming concepts and become familiar with the domain. This approach was inspired by the GBL approach [8], except that this game serves as a model for the games the students would create later (third stage).

The second stage includes recreating the game the students previously played, following the lecturer's instructions or a guide. This involves a gradual handover of the knowledge of the programming language required to master the specific problem. Following this approach, students remain motivated and don't get overburdened by the complexity of the programming language and the problem itself. It also allows us to dig deeper into specific language concepts. This approach was inspired by PBL [9].

The third stage represents a slight transition to developing a new, more complex game and specific language concepts. The aim is to address a nontrivial problem using an industrial programming language. The intention is to reuse what students already know but in different problem sets and to introduce new and more complex issues. In this stage, students also have enough space to be creative and to recognize the associations with specific parts of the second stage.

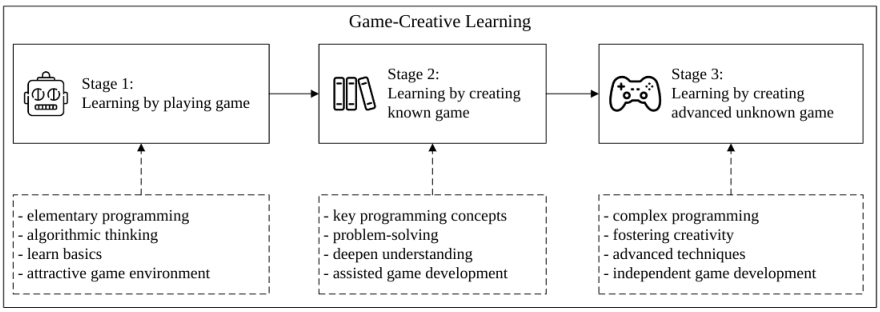

Figure 1

Game-Creative Learning: General idea vs application in CS0/CS1

An essential element of this approach is the learning environment. It includes practical guidelines for the second stage and advice on tasks for the third stage. We developed e-learning materials as part of this environment, clearly outlining learning outcomes and practical objectives [17]. Additionally, we provided detailed problem descriptions and instructions for solving them. We also included additional tasks and resources.

Given the complexity of our learning environment, the GCL approach is further supported by version control and automated assessment tools. However, evaluating the correctness of the entire solution, including the quality of the code,

requires a robust mechanism. In our experiment, we assess student assignments through an automated tool and checks by tutors, depending on the problem's complexity. Despite the significant number of studies on automated code assessment, our approach is the closest to [19].

Throughout the term, we track the score of the students, which encourages students to be active and competitive. As a result, the particular game assignments often end in multiple attractive custom solutions complemented by graphics or additional game features.

3 Course Innovation: Experiment

Our introductory programming course (CS1 or CS0) is run during the second term of our undergraduate program. This mandatory course is based on a sophisticated learning environment focused on procedural programming. Every year, it is attended by hundreds of students (having mostly no prior programming experience)[1]: 2009-946, 2010-1124, 2011-789, 2012-802, 2013-497, 2014-503, 2015-670, 2016-603, 2017-636, 2018-606, 2019-688, 2020-757, 2021-706, 2022-693, 2023-565, 2024-777. The course curriculum does not differ substantially from the curricula of standard academic courses, though specific topics have been reorganized and adjusted to the innovations.

When planning the innovations, the first task was to select a suitable programming language. Based on surveys [20] and [21], we had to choose either an educational language as Karel the Robot [22], or an industrial one as C. Educational languages are simpler to understand and to learn algorithmic thinking, but usually, impracticable. On the other hand, industrial languages can motivate students, though starting with them means beginning with a steep learning curve.

In our experiment, we decided to get the best of both worlds – to use the educational pseudo-language Karel the Robot as a C library. Based on [23], we did not use the language in an isolated fashion. Considering the entire curriculum, we allowed the students to follow the C syntax rules from the very beginning, even though they worked in the environment of the robot [24].

A further step was task selection. For beginners, we preferred to use a sole but more complex task instead of multiple but simpler tasks. According to [20], we used micro-worlds as a physical metaphor for various algorithmic problems since it is easier to understand the robot's movements within the world's limits than to understand cycles using vectors or Fibonacci sequences.

[1] The fluctuation in student headcount is influenced by demographic changes and other factors that are not directly related to the educational content or delivery.

In the 15 years, we have gradually applied the innovations in smaller steps. Initially, we divided the course into two stages: Playing with and then creating Karel the Robot. Later, we added a third stage focused on creating another game. Additionally, we enhanced the learning environment by developing tutorials, implementing a version control system, and automated assignment assessment.

As a result, the course consisted of three stages, each using different features of PBL, GBL, and gamification, according to the proposed *GCL* approach:

1) Using Karel the Robot – algorithms served as games.
2) Implementing students' robot – custom (game) library.
3) Implementing Sokoban – advanced programming of 2D game.

3.1 Stage I: Playing the Robot

Based on the idea of associating industrial requirements with education, we implemented the robot as a C library in the first stage. This allowed us to adjust the programming language to the specified requirements, i.e., suppress any unwanted language features and replace them with custom macros, procedures, and types. So, students could focus on introductory programming elements, namely brackets, semicolons, or function calls.

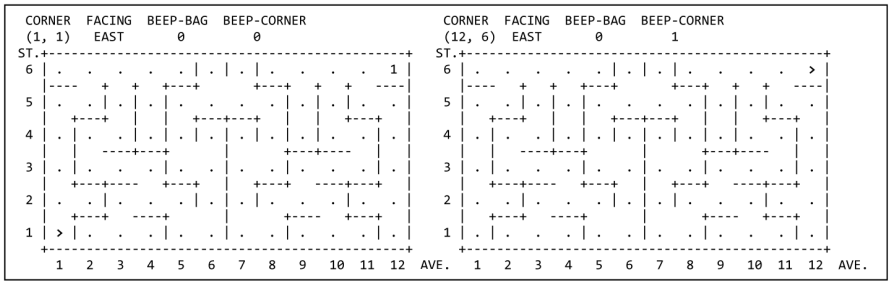

Figure 2
Environment of Karel the Robot: Example of initial and final situation

Karel the Robot inhabits a 2D game environment consisting of avenues and streets blocked by walls and corners containing objects called beepers – see Fig. 2. The world is inhabited only by a single robot that can move only forward, turn 90 degrees to the left, pick up a beeper from its current position (if there is any), or place a beeper at its current position (if it has any). The robot carries the beepers in its bag and can detect whether it is empty. The robot can also determine the direction it stands in (one of the four cardinal directions) and whether there is a beeper at the current corner (robot's current position).

Students create simple programs for the robot, through which they solve problems in a 2D environment. Since they use the library, playing with the robot is subject to C syntax. Playing the game, students learn elementary language statements and

the basics of flow control and cycles. All programs include the **karel.h** header file and are compiled as C programs. Robot commands are represented by calls to library functions like **void turn_on("world.kw")**, **void step()**, **void pick_beeper()**, **int facing_east()**, or **int beepers_in_bag()**.

Students can create new commands using custom functions in C. However, they do not use data (variables) – this allows for hiding specific topics, namely types or I/O formatting. We introduce them in the second stage. Instead, the library provides the **loop(number)** function.

Three works inspired the design of Karel and the command names, originally implemented in Pascal [25], Java [26], and a pseudo-language [21]. Fig. 2 shows an example of the environment. In the example, the task is to find a beeper by navigating through a maze. The situation on the left represents the initial state, while the situation on the right represents the final state. Naturally, the solution should be adaptable to other similar world maps. Fig. 3 shows such a solution, comparing the same program written in C, a Pascal library, and Karel's original pseudo-language.

```
#include <karel.h>

void turn_right(){
 loop(3){
   turn_left();
 }
}

int main(){
 turn_on("maze.kw");

 while(no_beepers_present()){
   if(right_is_clear()){
    turn_right();
   }
   else{
    while(front_is_blocked())
      turn_left();
    }
   step();
 }

 turn_off();

 return 0;
}
```

```
PROGRAM
stairs(INPUT,OUTPUT,SITUATION,REPORT);
%INCLUDE 'KAREL:KAREL(PASCAL)'

procedure turnright;
  begin
    turnleft;
    turnleft;
    turnleft
  end

begin
  turnon;
  while next_to_a_beeper do
    begin
      if right_is_clear then
        turnright
      else
        while front_is_blocked do
          begin
            turnleft;
          end;
      move
    end;
  turnoff
end
```

```
BEGINNING-OF-PROGRAM
 DEFINE-NEW-INSTRUCTION turnright AS
   ITERATE 3 TIMES
     turnleft;

 BEGINNING-OF-EXECUTION
   WHILE not-next-to-a-beeper DO
     BEGIN
       IF right-is-clear
         THEN turnright
       ELSE
         WHILE front-is-blocked DO
           turnleft;
       move
     END;
   turnoff
 END-OF-EXECUTION
END-OF-PROGRAM
```

Figure 3

Comparison of a program in C, Pascal, and pseudo-language of Karel the Robot

In the first stage of the course, we achieve two goals:

- Students learn elementary programming concepts and train their algorithmic thinking.
- Moreover, they work with attractive game outputs from the beginning.

3.2 Stage II: Creating the Robot

The second stage of the course benefits from the first one, in which the students use the library as a black box. In this stage, the domain is already known, and the problem is already understood. So, the goal is to create a custom robot library (approx. 500 LoC), following the detailed instructions. The students' job is to implement every robot command, in which they deal with I/O formatting, variables, arrays, structures, and pointers. This way, the students transition from playing games to creating them.

In the second stage of the course, we achieve two goals:

- Students deepen their understanding of key programming concepts.
- They enhance problem-solving skills through the development of a functional robot library.

To address advanced problems, namely working with dynamic data structures or memory allocation, and to support independent work, we added a Sokoban game add-on, described in the following section.

3.3 Stage III: Creating a Game

In the last stage, students create their own 2D computer game (approx. 650 LoC). Sokoban is a simple logic game representing the third GCL stage. It covers advanced topics, dynamic memory management and pointers, practically reflected in the individual game levels. Although students have guidelines to support them, detailed steps are not provided at this stage. Students can rely only on the practical hints provided.

Technically, Sokoban is similar to Karel, they both use a 2D world, but it is a real game. Since we consider students gaming natives, we believe this supports their motivation. To manage display controls, we decided to incorporate the Curses programming library [27]. Fig. 4 depicts one game level.

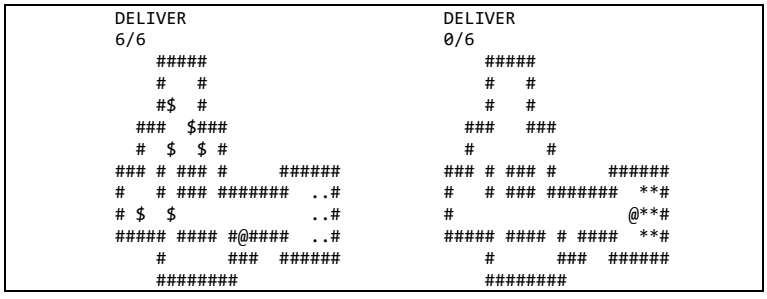

Figure 4
Sokoban game level: Example of initial and final situation

The symbols have the following meaning:

- `#` A wall through which the player cannot pass,
- `$` A box that player can push,
- `@` The player,
- `.` The target position of the box,
- `*` A box placed on a target position,
- `SPACE` The floor on which the player can move,
- `DELIVER` Boxes yet to be shifted to target positions.

In addition to learning the advanced aspects of procedural programming, the third stage provides sufficient space for student creativity, usually manifested as new game elements or visual graphics. Fig. 5 depicts one of these solutions. In addition to these items, tracking the score of students supports competition.

Figure 5
A graphical representation of Sokoban

In the third stage of the course, we achieve three goals:

- Students apply advanced programming techniques to create a 2D game.
- They exercise creativity by experimenting with game and visual designs.
- They gain confidence by independently developing software projects.

3.4 Assessment

During the 15 years, we developed tools for automated assessment of student assignments. The assessment is combined with the lecturer's checks to evaluate the quality of students' code [18, 19]. Today, the course incorporates four assignments, each linked to a particular course stage, while the implementation of

Sokoban is split into two assignments: The implementation of the Sokoban environment and the development of the individual game levels.

Students push their code in a versioning system. The automated assessment tool connects to Git [28], and generates regular feedback. With this, students have a chance to fix potential errors and improve their overall score before reaching the deadline. It enhances student engagement throughout the term, which is one of our main goals. Another goal is the early detection of potential shortcomings. Our intention is not to assess a black box at the end of the term but to identify the specific problems as early as possible [4].

4 Measurement and Evaluation

We use three metrics to evaluate our experiment: student activity/engagement, knowledge retention, and student opinion. Analyzing the results of the 15 years, we see that the GCL approach is a success.

4.1 Student Activity & Engagement

The first metric focuses on identifying student engagement, one of the GCL goals. In particular, it tracked the course web sessions, the committed code (Git), and problem discussion entries.

One-term results are in Fig. 6. The upper and lower charts show student activity throughout the term. In both graphs, several peaks indicate typical student behavior: The work intensity increases with the approaching deadline. Initial exposure to new tools like Git caused peaks in activity during Assignment 1. Ideally, there should be no peaks near the deadlines, as students are expected to commit their assignments well in advance. However, on a weekly scale, the differences in the curves are negligible.

4.2 Knowledge Retention

The second metric incorporates student score comparison. The chart shown in Fig. 7 captures the score rate density in the period ranging from 2009 to 2024. We aim to determine to what extent our knowledge-increase expectations have been met.

The results revealed that a significant number of students achieved borderline scores. However, comparing the particular box plots, a higher score is evident every second year. Therefore, we state that GCL leads to an improvement in the knowledge acquisition process.

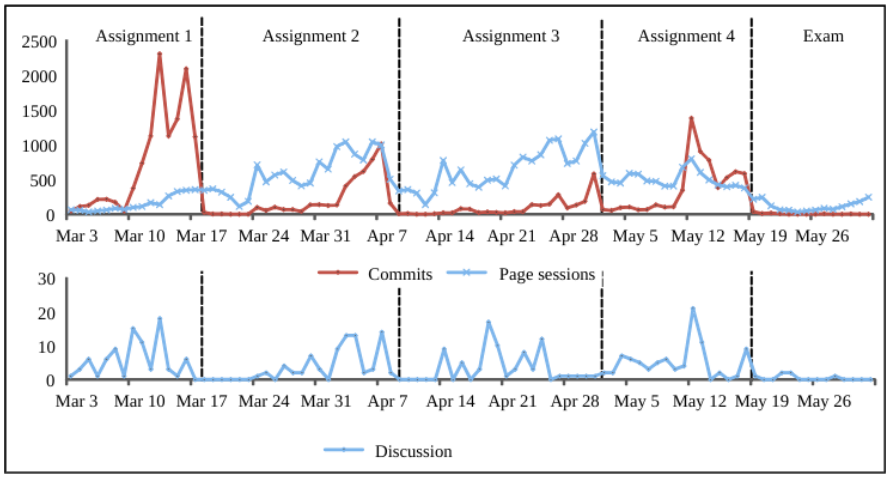

Figure 6

Student engagement data

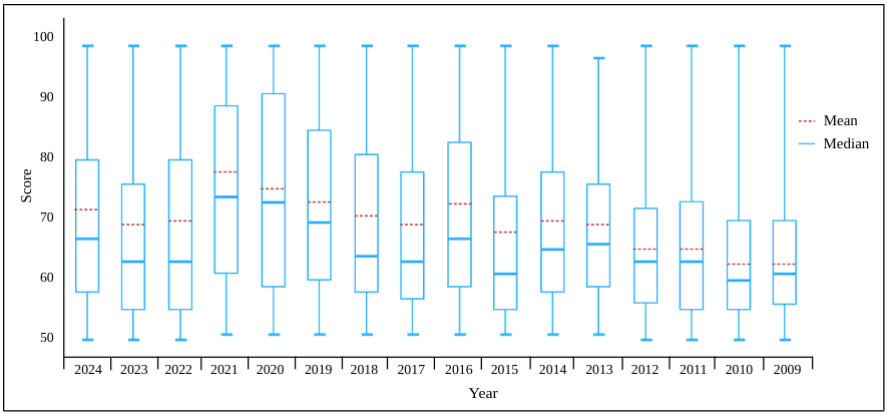

Figure 7

Student score over a 15-year period

A detailed examination of the score rates revealed some shortcomings of the last stage. These included the fact that most discussion entries appeared on the pages related to the Sokoban assignments (the game and the levels) and the lectures dedicated to the related topics of modular programming, multi-dimensional arrays, and pointers. Since the problem could lie either in the third stage of the course or in the third stage of the GCL, we did another experiment with an object-oriented programming course (with the participation of 170 students). The course's curriculum differed, but the GCL approach remained the same.

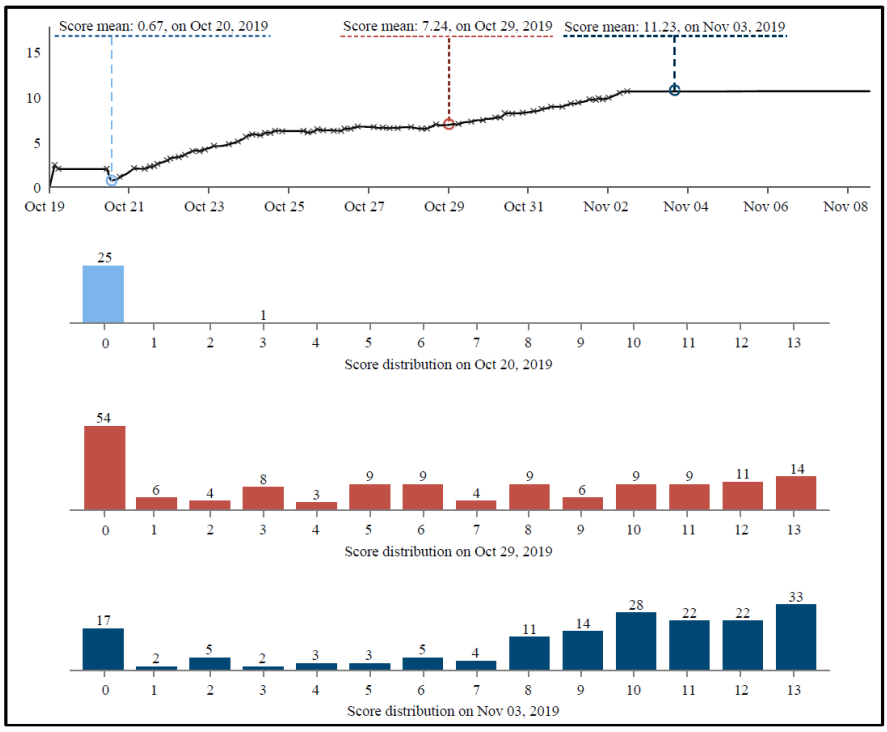

Figure 8
Tracking student scores during one assignment period

The results of the student progress in the assignment period are shown in Fig. 8. Almost 24% of students engaged in this experiment was unsuccessful (reaching a score below 8). On the contrary, the mean score of successful students was 10.99 out of 13. This indicates that students are receptive to the education process and improve gradually throughout the term. Due to this, we can consider GCL as a factor contributing to knowledge increase.

4.3 Students' Opinions

The last metric focused on the students. Incorporation student feedback into course design is essential for continuous improvement. Reflecting approach in [29], we regularly gather and analyze student feedback. The aim is to collect the students' opinions, reflecting their satisfaction and any suggestions for further course improvements. In particular, we found two opinions to be relevant:

- *I think I understand procedural programming concepts*,
- *Karel helped me understand basic programming concepts*.

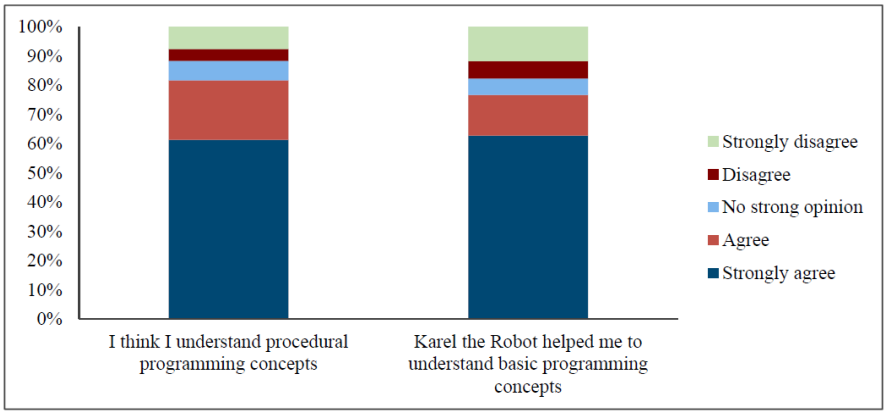

Figure 9
Measurement of students' personal opinion

The chart depicted in Fig. 9 shows that Karel significantly contributed to mastering procedural programming.

According to [3], student satisfaction is influenced by several factors. This is why we are interested in their personal opinion. Here are some examples of the most typical opinions:

- Describe your experience with Karel:
  - I like that it visualized things, which got me into programming (2010);
  - It clarified some issues I previously didn't understand (2016);
  - Finally, I saw theory in practice, and it was fun because it was a game (2020);
- Explain your first experience with this course:
  - It was my first game (answer provided by many respondents);
  - I have always wanted to create my own computer game, and I plan to add my elements, including shooting or special beepers (2012);
  - I like this form of learning; I was looking forward to every new week (2015);
  - I like the tutorials describing various issues and their solutions (2019).

4.4 Threats to Validity

All achieved results might be distorted by possible threats, which are challenging to eliminate. In our study, we identified four key threats: External web visitors, inconsistent commit behavior, off-platform discussions, and student dropout. We explain them closely in the following section.

Since the course web is public, it is likely that the session counts also include random visitors. So, we combined the first measurement with tracking both commits and the discussion entries.

The commit rates could be higher if all students gradually committed their solutions. A few students tended to commit their solutions only once, upon finishing the assignment, even if they were active almost daily. This was caused by the lack of experience using the version control system. Better support in using these tools could eliminate the problem.

Discussion rates could also be higher. This is caused by the fact that students often lead discussions beyond the course environment using social media. Even though the first threat might cause the student engagement figures to be lower than reported, the other two might have the opposite effect.

The statistical decrease in student headcount during the term, common in environments with large student populations, can distort the overall results of student score tracking. More sensitivity to realistic input measurement techniques could yield a more accurate result.

5 Integrating Bloom's Taxonomy

When classifying educational objectives, we concentrated on the cognitive process instead of knowledge [30]. We believe our work contributes to the achievement of the cognitive objective, the higher application, aligned with the statement above that *In computing, we might hypothesize that learning objectives are what we might term a higher application*. This is why we focused on higher application when classifying educational objectives. Fig. 10 shows a revised model for computing, incorporating the capstone level of higher application, relying on the interpretation categories of Bloom's taxonomy, as stated in [31].

The model shows that the third stage of GCL enhances the Remember, Understand, Apply, and Analyze levels, which reflects learning elementary concepts. Students can identify particular constructs of C in a piece of code, recognize the concept code dealing with the subject matter (Remember), translate algorithms from one form to another, and explain a concept or an algorithm (Understand). They can apply a known process or algorithm to a familiar problem

(Apply) and break down a programming task into its components and organize them to achieve an overall logic objective (Analyze) [32]. So, students use the game to think, though they can still not solve complex tasks.

The second stage enhances the Analyze, Evaluate, and Create levels, reflecting the implementation of a solution to the complex task. Students can break a task down and synthesize the components (Apply), determine whether the code meets the

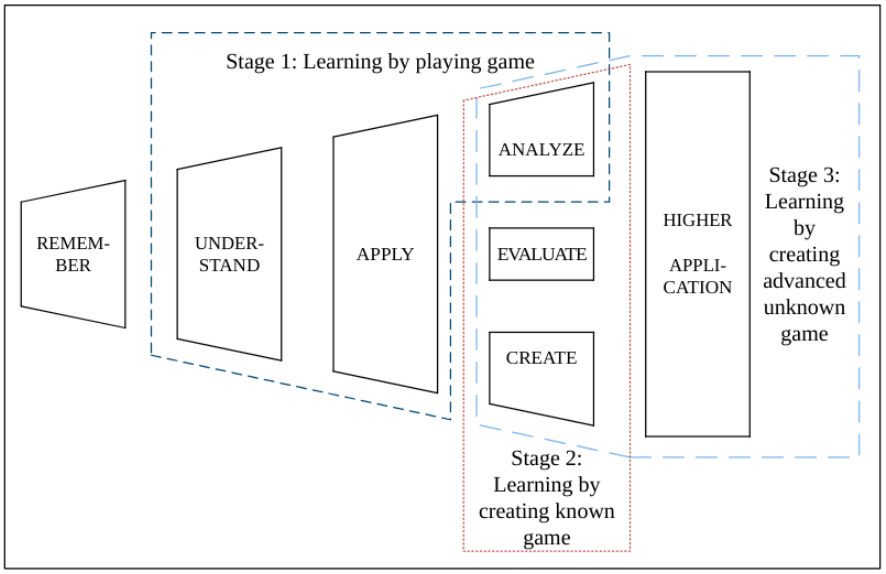

Figure 10

Integrating Bloom's taxonomy with the Game-Creative Learning approach

requirements, evaluate its quality and define an appropriate testing strategy (Evaluate), and propose new alternative algorithms to solve a complex task (Create). Nevertheless, they can still not use complex libraries and software technologies flexibly.

The third stage enhances the role of the higher application on top of the Analyze, Evaluate, and Create levels. Every student reaching this level can analyze, design, implement, and test software solutions on their own and use advanced software libraries, tools, and technologies effectively. Thus, students are moving through all levels of Bloom's taxonomy – from learning by playing games to learning by creating games.

Conclusions

This study describes *Game-Creative Learning* (GCL) as a unique educational approach, combining elements of gamification [15], game-based learning [8], and problem-based learning [11]. The proposed hypothesis states that the innovation of the educational approach in an academic course increases student knowledge.

Additionally, it enhances student engagement, preserving the original content and aims of the course. This hypothesis was verified in multiple ways.

The GCL approach has demonstrated success in enhancing student engagement and knowledge acquisition over 15 years, engaging over 10,000 students. By progressively guiding students through increasingly complex programming tasks, GCL has led to measurable improvements in student performance, with higher engagement levels and better retention of programming concepts.

Formally, the course continuity has improved since the respective topics build on each other, and students solve more realistic tasks. In addition, they work in an attractive environment from the very first lecture and are motivated by using a real-world, industrial programming language.

Regarding student opinions, particularly among those who are gaming natives, questionnaires showed that most students find this approach and the current course curriculum motivational. They claim it helped them understand fundamental aspects of programming more effectively. Additionally, all measurement results have been carefully recorded in charts, confirming the hypothesis.

Our course has two primary features: The attractive environment and many participating students. The way we introduced the innovations in the course is supported by most of the literature concerning gamification and game-based learning. In contrast with [33], we show that GCL enables *low-cost robots* (Karel) to help students to explore more sophisticated methods instead of turning out to be dead ends. Following [4], who shared their message in the 1990s, we believe that appropriate changes to *first-year courses* are the most significant in improving the quality of the higher education system.

The success of GCL in programming education suggests its potential application across various disciplines, offering a model for integrating gamification and problem-based learning into broader educational contexts. As educators seek to adapt to the evolving needs of today's students, the GCL approach presents a compelling case for reimagining traditional curricula to foster deeper learning and sustained student interest.

Similar to the model introduced by [30], when we classified the educational objectives and incorporated a higher application of Bloom's taxonomy, as interpreted by [31], we focused on the cognitive process, not knowledge. We emphasized cognitive aspects by progressively challenging students with more complex tasks, fostering their analytical and problem-solving skills [34] as they advanced through the course stages.

We believe we also addressed the following issues regarding the quality of e-learning, as identified by [13]:

- E-learning efficiency – combining online guidelines, discussion, and automated assessment into a complex learning environment;

- Student response – immediate adjustment to particular changes and the overall engagement;
- Importance of support for students – open discussions and detailed task instructions;
- Positive investment return – increased student knowledge and better student engagement.

Educators and institutions are encouraged to embrace the potential of GCL, ensuring that future generations of students are not only engaged but also better prepared for the challenges of the digital age.

The development of our approach continues, focused on assessment: Automated code functionality assessment vs. code quality evaluation. These issues are inspired by the techniques of creating programmer knowledge profiles, as seen in [35] and [36]. The aim is to shift from student engagement to education [7] and training good programmers [17].

Acknowledgement

This work was supported by project Kega No. 015TUKE-4/2024 "Modern Methods and Education Forms in the Cybersecurity Education" and project VEGA No. 1/0630/22 "Lowering Programmers' Cognitive Load Using Context-Dependent Dialogs".

References

[1] Comenius, J. A. (1630) Schola Ludus (School by Play) Comenius

[2] Palova, D. and Vejacka, M. (2022) Implementation of gamification principles into higher education. European Journal of Educational Research, 11(2):763-779, DOI: 10.12973/eu-jer.11.2.763

[3] Prensky, M. (2001) Digital natives, digital immigrants. On the Horizon, 9(5):1-6, DOI: 10.1108/10748120110424816

[4] Brown, S. and Race, P. (2021) University Teaching in Focus: A Learning-centred Approach, chapter Using effective assessment and feedback to promote learning, pp. 74-91, Routledge, 2 edition

[5] Tareque, M. H. et al. (2024) You Hacked My Program! Teaching Cybersecurity using Game-based Learning. In 26th Western Canadian Conference on Computing Education (WCCCE), pp. 1-7, ACM. DOI: 10.1145/3660650.3660672

[6] Schildgen, J. and Rosin, J. (2022) Game-based learning of sql injections. In International Workshop on Data Systems Education, pp. 22-25, ACM. DOI: 10.1145/3531072.3535321

[7] Alhazbi, S. and Halabi, O. (2018) Flipping introductory programming class: potentials, challenges, and research gaps. In International Conference

on Education Technology and Computers (ICETC), pp. 27-32, ACM. DOI: 10.1145/3290511.3290552

[8] Kuk, K. V., Milentijevic, I. Z., Randelovic, D. M. et al. (2017) The Design of the Personal Enemy - MIMLeBot as an Intelligent Agent in a Game-based Learning Environment. In Acta Polytechnica Hungarica, 14(4):121-139, DOI: 10.12700/APH.14.4.2017.4.7

[9] Toth, P., Horvath, K. and Keri, K. (2021) Development Level of Engineering Students' Inductive Thinking. In Acta Polytechnica Hungarica, 18(5):107-129, DOI: 10.12700/APH.18.5.2021.5.8

[10] Barrows, H. S. and Tamblyn, R. M. (1980) Problem-based Learning: An Approach to Medical Education. Springer, New York, 1 edition

[11] Servant-Miklos, V. F. (2019) A revolution in its own right: How maastricht university reinvented problem-based learning. Health Professions Education, 5(4):283-293, DOI: 10.1016/j.hpe.2018.12.005

[12] Poornima, S. and Pushpalatha, M. (2020) A survey on various applications of prescriptive analytics. International Journal of Intelligent Networks, 1:76-84, DOI: 10.1016/j.ijin.2020.07.001

[13] Szabo, C. M., Bartal, O. and Nagy, B. (2021) The Methods and IT-Tools Used in Higher Education Assessed in the Characteristics and Attitude of Gen Z. In Acta Polytechnica Hungarica, 18(1):121-140, DOI: 10.12700/APH.18.1.2021.1.8

[14] Pinter, R. et al. (2020) Enhancing Higher Education Student Class Attendance through Gamification. In Acta Polytechnica Hungarica, 17(2):13-33, DOI: 10.12700/APH.17.2.2020.2.2

[15] Palmquist, A. (2021) 'Gamification was not the problem': A case study exploring factors affect teachers approvement of gamification. In International Academic Mindtrek Conference, pages 106–116. ACM. DOI: 10.1145/3464327.3464347

[16] Malliarakis, C. et al. (2017) CMX: The Effects of an Educational MMORPG on Learning and Teaching Computer Programming. In IEEE Transactions on Learning Technologies, 10(2):219-235, DOI: 10.1109/TLT.2016.255666

[17] Binas, M. et al. (2014) Useful recommendations for successful implementation of programming courses. In International Conference on Emerging eLearning Technologies and Applications (ICETA), pp. 397-401, IEEE, DOI: 10.1109/ICETA.2014.7107618

[18] Biro, P. et al. (2022) A New Method to Increase Feedback for Programming Tasks During Automatic Evaluation Test Case Annotations in ProgCont System. In Acta Polytechnica Hungarica, 19(9):103-116, DOI: 10.12700/APH.19.9.2022.9.6

[19] Juhar, J. Et al. (2015) Towards automated assessment in game-creative programming courses. In International Conference on Emerging eLearning Technologies and Applications (ICETA) pp. 307-312, IEEE. DOI: 10.1109/ICETA.2015.7558505

[20] Pears, A. et al. (2007) A survey of literature on the teaching of introductory programming. ACM SIGCSE Bulletin, 39(4):204-223, DOI: 10.1145/1345375.1345441

[21] Perera, P. et al. (2021) A systematic mapping of introductory programming languages for novice learners. IEEE Access, 9:88121-88136, DOI: 10.1109/ACCESS.2021.308956

[22] Pattis, R. E. et al. (1995) Karel the Robot: A Gentle Introduction to the Art of Programming. John Wiley & Sons, New York, 2 edition

[23] Ayalew, Y. (2021) Procedural programming experience as a predictor of success in object-oriented programming. In International Technology, Education and Development Conference (INTED) pp. 4092-4098, IATED. DOI: 10.21125/inted.2021.0835

[24] Somlyai, L. and Vamossy, Z. (2024) Improved RGB-D Camera-based SLAM System for Mobil Robots. In Acta Polytechnica Hungarica, 21(8):107-124, DOI: 10.12700/APH.21.8.2024.8.6

[25] Untch, R. H. (1990) Teaching programming using the Karel the Robot paradigm realized with a conventional language

[26] Roberts, E. (2005) Karel the Robot learns Java

[27] Raymond, E. S. and Ben-Halim, Z. M. (1993) Writing programs with ncurses

[28] Torvalds, L. and Hamano, J. C. (2005) Git

[29] Farkas, G. et al. (2023) Quality Improvement in Education, based on Student Feedback. In Acta Polytechnica Hungarica, 20(6):215-228, DOI: 10.12700/APH.20.6.2023.6.12

[30] Johnson, C. G. and Fuller, U. (2006) Is Bloom's taxonomy appropriate for computer science? In Baltic Sea Conference on Computing Education Research, pp. 120-123, DOI: 10.1145/1315803.1315825

[31] Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., and Robbins, P. (2008) Bloom's taxonomy for CS assessment. In Conference on Australasian Computing Education, pp. 155-161, DOI: 10.1145/1345443.1345438

[32] Masapanta-Carrion, S. and Velazquez-Iturbide, J. A. (2018) A systematic review of the use of bloom's taxonomy in computer science education. In ACM Technical Symposium on Computer Science Education (SIGCSE), pp. 441-446, DOI: 10.1145/3159450.3159491

[33] Ruf, A., Muhling, A., and Hubwieser, P. (2014) Scratch vs. karel – impact on learning outcomes and motivation. In ACM International Conference Proceeding Series: Workshop in Primary and Secondary Computing Education (WIPSCE), pp. 50-59, DOI: 10.1145/2670757.2670772

[34] Kovari, A. (2020) Study of Algorithmic Problem-Solving and Executive Function. In Acta Polytechnica Hungarica, 17(9):241-256, DOI: 10.12700/APH.17.9.2020.9.13

[35] Aragao, M. et al. (2019) Scaling up a Programmers' Profile Tool. In Symposium on Languages, Applications and Technologies (SLATE), pp. 11:1--11:8, Dagstuhl Publishing. DOI: 10.4230/OASIcs.SLATE.2019.11

[36] Pietriková, E., and Chodarev, S. (2015) Profile-driven source code exploration. In Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 929-934, IEEE. DOI: 10.15439/2015F238