Experiments on PR-Based Gamification

Author: Please provide author information

Abstract

In this article we document some experiments on teaching a class on a Master Degree Program using a different perspective on gamification. Instead of winning badges or getting achievements, students earn classification points. This allows them to work as hard as they are willing, having in mind their current classification and how far they can reach. In the specific experiment we are reporting, students can earn points with pull requests to a common class project. We describe the details, extrapolate on different ideas for implementing this in other classes, and conclude with the pros and cons of such approach for student evaluation.

2012 ACM Subject Classification Social and professional topics → Computer science education

Keywords and phrases computer education, gamification on class, GIT

Digital Object Identifier 10.4230/OASIcs.ICPEC.2016.23

Category short

1 Introduction

Computer Science teaching is a complex task, not just on the first years, but also in advanced courses, like Master Degrees. Usually, teachers tend to use hybrid evaluation methodologies, comprised of theoretical and practical parts. Both types require different strategies for engaging students. In this contribution we are not dealing with theoretical evaluation, but only on how to engage students on practicing writing code.

A common approach for evaluating the student competences on writing code is to ask them to develop some kind of solution. Sadly, for some courses, that is not easy. For instance, when the solution that makes more sense to request to students is a large application, and just a part is related to the goals of the curricular unity. To remedy this problem, teachers require students to work in group, dividing the work load. This leads to other kind of issues, like non balanced competences among the group elements, making it hard to properly evaluate each one independently.

In this document we expose an experiment in a Master Degree on [Hidden for blind review], at the [Hidden for blind review], and especially in the course of "Game Engines," where the students learn how to structure and build from scratch the basic functionalities for a game engine.

This document is structured as follows: Section 2 discusses on different gamification approaches already discussed by other authors; Section 3 explains the methodology used in our experiment; Section 4 does an analysis of the experiment, pointing out for different areas where this kind of gamification can lead; finally on Section 5 we draw some conclusions.

2 Related Work

Gamification can be defined as the use of game elements in non-game contexts [5]. In practice, game elements can be considered as a conceptual toolkit which can be applied in a system in order to motivate certain behaviors. Among all relevant game elements, the classic PBL triad (Points, Badges, and Leader-boards) is often adopted in most of the gamification projects to reward/notify individuals upon the accomplishment of specific tasks. The primary reason for

their use is its simplicity (common sense in understanding their rules) and its effectiveness to achieve early motivation.

2.1 Motivation

Gamification is no longer a buzzword. Nowadays, gamification plays a central role and is being considered as the solution for many issues related with lack of retention and demotivation in education and business domains. In order to successfully apply gamification it is crucial to understand the level and type of motivation that we want individuals to achieve using our system.

Motivation can be defined as the core element which drives individuals to accomplish a task. Beyond the amount of motivation that is necessary to achieve in order to be able to complete a specific task, it is important to distinguish and manipulate their two major flavors: *extrinsic* and *intrinsic motivation* [21].

Extrinsic motivation is gained by pursuing tangible rewards acquired after successfully doing an activity (e.g., rankings, levels, points, badges, awards, financial incentives) [18]. Intrinsic motivation is acquired by doing an activity for the own purpose of the individuals inherent satisfaction and, most of the time, as an opportunity to learn or recycle their potentials. In this context, it is important to foster cooperation, competition, self-esteem, ego, sense of belonging and love [18].

Most of the time, intrinsic motivation offers long-term and high-quality behavioral engagement, whereas extrinsic motivation is relatively less effective. Nevertheless, extrinsic motivation is still necessary in scenarios where the activity is itself neither engaging nor rewarding to the individual. In this case, external rewards are delivered to foster participation. Yet it should not be overused, since excessive external rewards might impair individuals spontaneous interest in the activity [4].

Based on these facts, the balance between both types of motivation should be encouraged and applied in the gamification construction process. Although advised, this balance is very difficult to achieve. The majority of the gamified systems can involve users in the first weeks, but then users lose motivation either due to the game mechanics' routine, or sometimes due to lack of transparency in the application's point system. This routine or lack of confidence leads the user to feel discredited and to abandon the application. Thus, the perfect symbiosis of these two types of motivation perpetuates the users' confidence, leading them to be able to complete the tasks in order to be rewarded and, at a later stage, to achieve an inner satisfaction that goes beyond any reward and is more related to self-improvement and the desire to be better [5].

2.2 Gamification in the Computer Programming domain

Gamification is being widely used in education and business contexts. In the education realm, gamification is often used to facilitate the learning process of tedious and/or complex subjects. One of these subjects is computer programming.

It is a fact that learning how to code is a complex process [11, 20]. Early systems started by trying to mimic traditional teaching methodologies to the virtual world with few success. In fact, those systems were characterized by the excessive focus on the language syntax and the availability of programming exercises without full coverage of the course curricula, unbalanced in terms of complexity and not suitable for heterogeneous classes with students with different profiles. In the beginning of the century, most systems included automatic evaluation systems to foster practice. However, we continue to assist to students' lack of

motivation and the consequent poor grades and dropouts. In the last decade, gamification began to be used to circumvent this scenario and several programming learning platforms appeared.

These platforms can be grouped in three major categories:

- Massive Open Online Courses (MOOC),
- Competition Systems, and
- Serious Games.

The next subsections describes the most notable examples of the three categories. Other categories were left out such as code playgrounds[1] and sandboxes. The main reason for discarding those categories is the fact that their main focus is not learning and for their lack of game elements.

2.2.1 Massive Open Online Courses

Currently, there is a large number of open online programming courses available on the web such as Coursera[2], Udacity[3], edX[4], Codecademy[5], Khan Academy[6], and Free Code Camp[7]. Most of theses platforms offer a wide variety of learning material from top universities' courses, with the possibility, at the end of the course, of getting paid certificates.

These platforms typically use gamification to attract learners, adopting elements which go beyond the PBL triad such as different levels, progress indicators and experience points. For example, Khan Academy uses badges and progress tracking to engage students to enlist and complete courses. Codecademy uses levels to organizes lessons and progress indicators to notify learns of their current learning status.

In the context of educational institutions, the most widely used approach to create gamified open online courses is to rely on an Learning Management System (LMS) with game mechanics. LMSs were created to deliver course contents, and collect assignments of the students. However, many of them evolved to provide more engaging environments resorting to gamification. Some of the most notable examples are Academy LMS[8], Moodle[9], and Matrix[10] which include badges, achievements, points, and leader-boards. Moodle also has several plugins which offer a variety of other gamification elements [16].

One of the most relevant examples is Enki [16] which is a tool that blends learning with assessment and gamification,. In Enki, the content is presented in several forms as well as delivering programming assignments with automatic feedback, while allowing anyone to create courses freely.

2.2.2 Competition Systems

It is part of the human condition to be competitive. Despite some dangers associated with learning based competition systems [12], it has proven to be an effective technique to

[1] CodePen (`https://codepen.io`), CodeSandbox (`https://codesandbox.io/`), PlayCode (`https://playcode.io/`), JSFiddle (`https://jsfiddle.net/`), SoloLearn (`https://www.sololearn.com/`).
[2] Available at `https://www.coursera.org/`.
[3] Available at `https://www.udacity.com/`.
[4] Available at `https://www.edx.org/`.
[5] Available at `https://www.codecademy.com/`.
[6] Available at `https://www.khanacademy.org/`.
[7] Available at `https://www.freecodecamp.org/`.
[8] Available at `https://academy-lms.com/`.
[9] Available at `https://moodle.org/`.
[10] Available at `https://www.matrixlms.com/`.

stimulate students to exceed their capabilities [14, 3]. In this realm, programming contests are becoming more popular among learners [23]. This section presents some platforms and tools that use competition as a way to engage students in learning programming.

The most notable example of competition systems are the programming contests which can be defined as environments with a set of competitive activities where individuals or teams strive to find solutions for a specific set of problems. In these contests, the evaluation of submissions are handled by special components (often called automatic judges) allowing participants to make as many submissions as they want during competition, and learn from their mistakes. In this way, "losers" actually win knowledge which may compensate the negative effects typically associated with competitive learning activities [19].

The most popular automatic judges are DOMJudge [8], PC^2 [1], PKU JudgeOnline [17], and Mooshak [13]. Despite being used primarily to support programming contests, currently the majority of them are capable of providing rich and immediate feedback to students, allowing the motivation by using timed challenges and leader-boards [9].

There are several platforms that can be used to compete or train as part of the recruitment process for top technology companies. Some notable examples are HackerRank[11], CodeSignal[12], Codility[13], HackerEarth[14] Assessments, TestDome[15], HireVue[16], DevSkiller[17], iMocha[18] and CodinGame For Work[19]. Usually so-called as Tech Recruiting Platform or Remote Online Code Testing, these platforms offer PBL facilities and include contests which can assume different time frames (opened indefinitely, in which challenges are proposed every week/day/month, or run for a limited amount of time – 48 hours). These contests enable users to increase/decrease their rating, win medals (which are given to top solutions), and cash prizes or, even, jobs at top technology companies.

2.2.3 Serious games

A serious game is a game designed for a primary purpose other than pure entertainment [6]. Currently, serious games are applied in several domains including healthcare, well-being, advertisement, and education. In the later, serious games are used to mitigate the difficulties found when learning a specific subject, either because it is complex or boring. In this scope, serious games are being applied in computer programming learning in order to explain programming concepts or to foster the practice of skills that, for some reason, are harder to assimilate.

Many studies were made aiming to identify requirements for an educational game to promote the learning of problem-solving techniques in introductory programming [24].

A survey [15] including 49 serious games analysed the most common features, and those which may increase the game's adoption. In this scope, the importance of fostering collaboration and competition within the game and the game's coverage based on the ACM reference curriculum for Fundamental Concepts in Programming were the most cited missing

[11] Available at `https://www.hackerrank.com/`.
[12] Available at `https://codesignal.com/`.
[13] Available at `https://www.codility.com/`.
[14] Available at `https://www.hackerearth.com/`.
[15] Available at `https://www.testdome.com/`.
[16] Available at `https://www.hirevue.com/`.
[17] Available at `https://devskiller.com/`.
[18] Available at `https://www.imocha.com.my/`.
[19] Available at `https://www.codingame.com/`.

features. Other conclusion of this study was the weak attention to the adoption of the best accessibility and inclusion practices.

A recent survey [22] analysed 41 games and founded that most of them emphasize mechanics and dynamics rather than aesthetics, making in general, the games tiring and boring. Other weak point was that the majority of the games analysed are not directed linked to the undergraduate level and the poor coverage of the items in the ACM curriculum. As a main conclusion of this study, authors recommended considering the student's previous skills and knowledge and providing automatic and rich feedback when specific error events occurs.

As some notable examples, selected without any criteria, are NoBug's SnackBar, Wu's Castle and Robocode:

- NoBug's SnackBar [24] is a serious game to support the learning of basic computer programming. It is a blocks-based environment including also resources that allow the teacher to follow the student's progress and customize in-game tasks.
- Wu's Castle [7] aims to teach basic programming concepts such as loops and arrays by presenting the learner with multiple-choice questions or fill-in blanks. The feedback consists of displaying an avatar executing the code.
- Robocode [2, 10] challenges the student to develop a complete software agent capable of defeating all the opponents in the arena in order to practice object-oriented programming and structured programming.

3 Gamification Methodology

As described previously, the experiment was performed on a Master Degree course, with 15 students. The topic of the course is game engine development, and the main goals are to understand the different components of a game engine, including but not limited to managing the input from the player, perform the rendering tasks (sprites, cameras, models), simulate physics (namely collision) and play sound.

In order to allow students to understand these components and how they interact together, one of the objectives is to have students developing modules, that can be put together, and implement a simple game. Given the students background, we chose to use the C# programming language, and the MonoGame[20] framework.

As to students motivation, badges or achievements, by themselves, is not enough for engagement. Most of our students are working, as the master degree runs in a after-work basis. Their time for studies is quite limited, and therefore, their main goal is to complete the degree, and if possible, with an interesting grade. With this in mind, we decided to award points, in a way students can accumulate them. At the end, the amount of points the students are able to gather is their final grade[21] (of course, with a possible truncation on the maximum grade).

Having in mind this should be the gamification approach to keep students engagement, the tasks that will be rewarded are pull requests to a GIT repository, solving previously defined issues. In this specific situation, a private repository from GitLab[22] was used.

During the classes concepts are discussed and a basic structure of the code needed to handle the discussed functionalities are added. At the end (or after the class), the teacher

[20] `https://www.monogame.net/`
[21] In Portugal, student grades are in the interval 0–20.
[22] `https://gitlab.com`

opens a set of issues in the GitLab interface. Each issue is a functionality needed for the common class project, and a amount of points is defined as the reward for that task.

To guarantee students do not choose all the simple tasks for themselves, after a task being assigned, the student is unable to get another task until his current assignment is complete. To complete an assignment the student needs to develop the code and prepare a pull request (named *merge request* in GitLab). The teacher validates the code, suggesting fixes, or accepting the solution. Every time a pull request is merged, the amount of points defined in the task are credited to the student.

After one task is done, the teacher can create new issues to complete the code from a previous student with new features.

4 Discussion and Evaluation

While in the previous section the gamification methodology was presented, in this section we will discuss the pros and cons found during the experiment.

4.1 Weaknesses

Unfortunately there are some drawbacks on adopting this methodology. We present some of them now:

1. As expected, this approach requires more work from the teacher. While teachers already need to tutor their students when performing any other project, this kind of task requires more effort, not just because that each student will be working on a different problem, but also because from time to time a new set of issues needs to be published, allowing students to continue their work.
   At the same time, if the class is too big, it might be not feasible to have some many parallel tasks at once, forcing some students to be idle, waiting for others to submit their pull requests, in order to be able to get their own assignments.
2. The way students solve a specific task might not be the best one for other features that will be coded on top of that. While the teacher might do code validation, and request some changes, to force a specific approach for solving a problem might not be desired — it is useful to let the students understand what path to go, and deal with the problems that might arise from there. Thus, sometimes, creating new issues that require to deal with code developed by a previous student might require some extra work. In this situations, our approach was to create intermediate tasks to transform the code in a way it gets suitable for the next task.

4.2 Strengths

During the experiment we noticed some advantages on this methodology:

1. Students are able to work as much as they like, and at each step, they know exactly their current grade. While some students might be comfortable with a low but positive grade, other students will keep fighting to get a higher grade. As the effort to raise their score is manageable, and known *a priori*, they can decide which tasks they want to perform, and choose their own path.
2. Students with low coding skill usually struggle to develop a complex project. If the teacher is able to propose issues of different complexity levels, with a low point reward, good students will not opt for those tasks, as they would need to do many more simple tasks to get to a high final grade. But, for students with low coding skills, these are an

opportunity to gather some points. They might need to solve a larger number of issues, but their hard work will be rewarded with a positive grade. This is also an approach to make these students work more time, and therefore, learning better coding skills.

3. Making students to develop tasks on already existing code forces them to read other people code. This is an important part of the learning process of coding. If the first tasks are easier to implement, as they are to be developed on top of the code written during a class, and therefore, explained, as new tasks are proposed students are required to look to code from their colleagues.
4. While the first detected weakness is regrading bad code practice or not so versatile implementations, when a new issue for code refactoring is opened, most of the times the students that performed the original code are willing to take it, and refactor their own code, as they see that as an interesting challenge.

5 Conclusions

When this experiment was conducted, the main idea was not to report about it, but to give a boost on the teaching methodology for this specific course. Therefore, unfortunately there was no evaluation on the approach by the students, and given we could not reproduce this methodology last year, given the global pandemic, currently we do not have a proper evaluation by the students.

Nevertheless, during the experiment there was two main types of students: some of the students struggled to get a positive evaluation, and stopped working; but some other found the approach interesting and got interesting grades. From this last group, not all students were experienced programmers. Some of them achieved a good grade very quickly, performing more complex tasks. Some others, with more difficulties, were able to achieve their grades by preparing simpler pull requests, but with a higher number of solved issues.

One of the drawbacks of this approach was the lack of a system to keep track of the students score. At that time, the class used a plain public GitLab server (although with a private repository). Nevertheless, as GitLab is freely available, one could install it locally, and it would be possible to develop a plugin to allow this gamification to be performed automatically, without the need for the teacher to keep track of which pull requests were solved, and their rewards.

References

1 Samir E. Ashoo, Troy Boudreau, and Douglas A. Lane. CSUS Programming Contest Control System (PC2), 2018. URL: `http://pc2.ecs.csus.edu/` [cited September 2020].

2 Esmail Bonakdarian and Laurie White. Robocode throughout the curriculum. *J. Comput. Sci. Coll.*, 19(3):311–313, jan 2004.

3 Juan C. Burguillo. Using game theory and competition-based learning to stimulate student motivation and performance. *Computers & Education*, 55(2):566–575, 2010. `doi:10.1016/j.compedu.2010.02.018`.

4 D. Coon and J.O. Mitterer. *Introduction to Psychology: Gateways to Mind and Behavior with Concept Maps and Reviews*. MindTap Course List Series. Cengage Learning, 2012. URL: `https://books.google.sm/books?id=EYwjCQAAQBAJ`.

5 Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, page 9–15, New York, NY, USA, 2011. Association for Computing Machinery. `doi:10.1145/2181037.2181040`.

289 **6** Damien Djaouti, Julian Alvarez, and Jean-Pierre Jessel. Classifying serious games: the g/p/s
290 model. *Handbook of Research on Improving Learning and Motivation through Educational*
291 *Games: Multidisciplinary Approaches*, 01 2011. doi:10.4018/978-1-60960-495-0.ch006.

292 **7** Michael Eagle and Tiffany Barnes. Wu's castle: teaching arrays and loops in a game. In
293 *Proceedings of the 13th annual conference on Innovation and technology in computer science*
294 *education*, ITiCSE '08, pages 245–249. ACM, 2008. URL: http://doi.acm.org/10.1145/
295 1384271.1384337, doi:http://doi.acm.org/10.1145/1384271.1384337.

296 **8** Jaap Eldering, Thijs Kinkhorst, and Peter van de Warken. DOMjudge-programming contest
297 jury system, 2011. URL: https://www.domjudge.org/ [cited September 2020].

298 **9** Pedro Guerreiro and Katerina Georgouli. Enhancing elementary programming courses using
299 e-learning with a competitive attitude. *International Journal of Internet Education*, 10, 01
300 2008.

301 **10** Ken Hartness. Robocode: Using games to teach artificial intelligence. *J. Comput. Sci. Coll.*,
302 19(4):287–291, April 2004.

303 **11** Tony Jenkins. On the difficulty of learning to program. In *Proceedings of the 3rd Annual*
304 *Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages
305 53–58, 2002.

306 **12** Alfie Kohn. *No contest: The case against competition*. Houghton Mifflin Harcourt, 1992.

307 **13** José Paulo Leal and Fernando Silva. Mooshak: A Web-based multi-site programming contest
308 system. *Software: Practice and Experience*, 33(6):567–581, 2003. doi:10.1002/spe.522.

309 **14** José Paulo Leal and Fernando Silva. Using Mooshak as a Competitive Learning Tool. In *The*
310 *2008 Competitive Learning Symposium*, 2008.

311 **15** Michael A. Miljanovic and Jeremy S. Bradbury. A review of serious games for programming. In
312 Stefan Göbel, Augusto Garcia-Agundez, Thomas Tregel, Minhua Ma, Jannicke Baalsrud Hauge,
313 Manuel Oliveira, Tim Marsh, and Polona Caserman, editors, *Serious Games*, pages 204–216,
314 Cham, 2018. Springer International Publishing.

315 **16** José Carlos Paiva, José Paulo Leal, and Ricardo Alexandre Queirós. Enki: A pedagogical
316 services aggregator for learning programming languages. In *Proceedings of the 2016 ACM*
317 *Conference on Innovation and Technology in Computer Science Education*, pages 332–337.
318 ACM, 2016. doi:10.1145/2899415.2899441.

319 **17** Xu Pengcheng, Ying Fuchen, and Xie Di. PKU JudgeOnline, 2013. URL: http://poj.org/
320 [cited September 2020].

321 **18** Kalliopi Rapti. Increasing motivation through gamification in e-learning. *The Journal for Open*
322 *and Distance Education and Educational Technology*, 7, 06 2016. doi:10.12681/icodl.640.

323 **19** Miguel A. Revilla, Shahriar Manzoor, and Rujia Liu. Competitive learning in informatics:
324 The UVa online judge experience. *Olympiads in Informatics*, 2:131–148, 2008.

325 **20** Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming:
326 A review and discussion. *Computer science education*, 13(2):137–172, 2003. doi:10.1076/
327 csed.13.2.137.14200.

328 **21** Richard M. Ryan and Edward L. Deci. Intrinsic and extrinsic motivations: Classic definitions
329 and new directions. *Contemporary Educational Psychology*, 25(1):54 – 67, 2000. URL:
330 http://www.sciencedirect.com/science/article/pii/S0361476X99910202, doi:https://
331 doi.org/10.1006/ceps.1999.1020.

332 **22** M. Shahid, A. Wajid, K. U. Haq, I. Saleem, and A. H. Shujja. A review of gamification
333 for learning programming fundamental. In *2019 International Conference on Innovative*
334 *Computing (ICIC)*, pages 1–8, 2019. doi:10.1109/ICIC48496.2019.8966685.

335 **23** Andrew Trotman and Chris Handley. Programming contest strategy. *Computers & Education*,
336 50(3):821–837, 2008. doi:10.1016/j.compedu.2006.08.008.

337 **24** Adilson Vahldick, Paulo Roberto Farah, Maria José Marcelino, and António José Mendes.
338 A blocks-based serious game to support introductory computer programming in under-
339 graduate education. *Computers in Human Behavior Reports*, 2:100037, 2020. URL:

http://www.sciencedirect.com/science/article/pii/S2451958820300373, doi:https://doi.org/10.1016/j.chbr.2020.100037.