Make your programs compete and watch them play in the Code Colosseum

Dario Ostuni
Department of Computer Science
University of Verona
Verona, Italy
dario.ostuni@univr.it

Edoardo Morassutto
Department of Electronics, Computer Science and Bioengineering, Politecnico di Milano
Milan, Italy
edoardo.morassutto@mail.polimi.it

Romeo Rizzi
Department of Computer Science
University of Verona
Verona, Italy
romeo.rizzi@univr.it

***Abstract*—Games have a role in many aspects of science, technology, and society. Also, games attract humans' interest and offer unique learning opportunities. Indeed, the role of games in education has a long tradition.**

In this paper we introduce *Code Colosseum*, a platform that takes competitive programming in the direction of games, instead of problems. By taking this direction, we aim to create a more engaging environment for students to compete in. The platform allows programs written by the contestants to compete in a real-time multiplayer game. The platform also allows to spectate the matches between the programs. The design and implementation of *Code Colosseum* has been kept as simple as possible, to facilitate participation, maintenance and setup.

To assess the effectiveness of the approach, we organized a tournament with 16 students, from both high schools and universities, as a pilot experience for *Code Colosseum*. In this tournament they created programs to play the *Royal Game of Ur*, a board racing game. The feedback from the students about the experience was positive, and the suggestions received will be implemented for future experiences.

***Index Terms*—Game-based learning, Computer science education, Competitive programming, Educational technology**

I. Introduction

With the rise of *STEM*[1] education, Computer Science has become one of the central subjects to be incorporated inside the curricula of secondary, and also primary, education systems around the world.

While the secondary education systems transit towards a more scientific-focused teaching of Computer Science, one way to give students an insight of it is through extracurricular activities [13]. One such example is competitive programming, which has a long history of being used as an educational tool [14]. Many competitions in this field have an interest in fostering Computer Science education, such as the *International Olympiad in Informatics* [1], which is more competition-oriented and for high-school students only, the *Kangourou of Informatics* [15], which is more didactic-oriented and also open to middle school students, and, recently, *Codeforces* [16], the leading web platform for programming contests.

[1]Science Technology Engineering Mathematics

978-1-6654-3886-5/21/$31.00

The role of competitive programming in Computer Science education has multiple facets: students not only get interested in various topics in Computer Science, but also group up in school-level, national, or even international communities. The social aspect value is confirmed by the attention it receives in web platforms such as the American *TopCoder* [2], the Indian *CodeChef* [3], the Japanese *AtCoder* [4] and the Italian *CMSocial* [5]. Such communities strengthen students' interest in competitive programming and Computer Science. They also offer a place where knowledge and competence is emphasized and where students which might not find it at their schools can seek for it.

While competitive programming fills an ever-expanding niche, many students might feel a bit discouraged, seeing it merely as a competition for the best of the best. Furthermore, competitive programming might not be as engaging for outsiders, because the knowledge requirement to appreciate it is quite high.

In this paper we present *Code Colosseum*, a platform to expand competitive programming using games as its primary feature. Using games as a mean to rise engagement and motivation has proved effective in Computer Science education [17].

There already exist other platforms that offer similar features. For instance, *CodingGame* [6] and *CodeCombat* [7] are web platforms that let students learn programming with games; here, their programs can be developed in the platform web environment and are run on the platform server. One of the main technical differences with *Code Colosseum* is that the students' programs are developed and run solely on their machines.

The paper is organized as follows: in Section II we give an overview of our motivations for proposing *Code Colosseum* and our goals for it, in Section III we provide details on the implementation of *Code Colosseum*, in Section IV we discuss the pilot experience we organized with *Code Colosseum* and in Section V we lay down our conclusions and future directions for the project.

II. Design, Motivations and Goals

We believe that creating a program (bot) that plays a game and letting it compete against other bots within a shared and

observable arena makes for an engaging environment. For games where optimal play is out of reach or where there is a well balanced component of luck, the matches can be interesting and instructive to observe and the competition offers opportunities for social interactions. When the outcomes are not entirely predictable in advance, people can discuss on the strengths and weaknesses of the bots and their strategies. This is even more so if the design and making of the bots is a team activity. Also, a match is engaging not only for insiders, who have a direct interaction with their peers and share with them the same language and mental space, but also for outsiders, who can be interested in just watching the matches, making enough sense of what is actually happening even without any prior Computer Science knowledge. In fact, this is very close to the idea behind *RoboCup* [18], which is a very successful competition where teams of high-school students assemble and program some robots to make them compete in a physical soccer-like game against those of other teams.

We propose *Code Colosseum*, a framework for the creation and deployment of real-time multiplayer games meant to be played by bots. The game could be a classic one, like checkers or any known card game, it could be a variant of a classic game, or it could be a completely new game. Both collaborative and competitive games are possible.

Code Colosseum is implemented as a client-server architecture, where the server manages the ongoing matches and the clients connect the players' bots to the server communicating over the net. Some of the main traits of *Code Colosseum* are:

- the server acts as the central and trusted hub for each match and has always complete information on the state of the game. To play a match, the clients need to connect only to the server;
- the server offers a lobby from where to create or join a match. When creating a match, one sets the number of players and how many of them should be covered by server-provided bots;
- the server can only manage a certain set of games. This set can be extended by implementing the rules and communication protocol for a new game;
- the player's bot can be any program, as such it can offer an environment from which a human might directly play or assist an AI playing it;
- all matches can be publicly spectated.

The fact that the player's bot runs on their machine has several and profound implications, among these:

- each participant can use their preferred programming languages, libraries, and tools without bearing on what is available on the server;
- during the matches, bots can make use of precomputed information and can leverage available hardware like GPUs;
- since each bot runs on the player's machine, standard debugging tools and techniques can be used to debug its logic and protocol implementation;
- bots do not add to the load of the server in terms of computing resources;
- it lifts the sandboxing requirement that would be otherwise needed for security reasons. This also helps in keeping a low complexity of design and implementation;
- since the client-server communication happens over the net, each participant needs a stable internet connection;
- the computational resources available to different players might wildly differ. The server cannot limit nor have knowledge of such resources;
- in team games, the server has no control over the intra-team communication. For instance, in 2v2 card games where no intra-team communication should be allowed the bots could privately communicate through other channels. Note however that *Code Colosseum* is still suitable for managing team games where the players of the same team can fully share their knowledge.

In our opinion, the following features of *Code Colosseum* are very important and require to be further explored and developed.

A. Visualization

Spectating the game might be a way to get or feel involved and to prove interest or participation in the competition. Even a team coach without technical preparation might offer suggestions, opinions or encouragement based on the matches they have seen. This sharing adds recognition and meaning while helping in promoting a positive environment of interest and participation. Also, for peers and classmates, spectating the game might be the first step to then getting involved. For these reasons, supporting the visualization of ongoing matches in an accessible way is an important feature that deserves further development. Unlike with usual competitive programming competitions, with games and in tournaments there is an unparalleled opportunity to obtain meaningful and immediately accessible visualizations that should certainly contribute in engaging and motivating all participants and spectators.

B. Tournaments

If the goal is to further foster STEM education then we must first comprehend the proper forms that make a proposal truly inclusive for a wider range of students. In particular, the social dimensions require to be addressed since they relate to profound motivations. Tournaments fill that space and offer a powerful opportunity to catalyze the interest of both contestants and the wider community that might gather around them in a high-school or university setting.

Also, tournaments bring a focus and a shared interest on matches and players. This is particularly true for elimination tournaments where the last matches are naturally followed by more spectators. Indeed, players have a natural curiosity to see how the best players perform and learn from their strategies. Also, players can group up in watching and commenting together the live matches. This offers further opportunities to socialize and exchange ideas and enthusiasm.

C. Simplicity

In the design and first implementation of the *Code Colosseum* platform we adopted a minimal approach striving for simplicity. The importance of this point cannot be underestimated. Our aim is to get a system that can be used with ease not only to play matches but also to develop a new game. In the long term, communities around this platform might form in high school or university settings. These communities will first be attracted by playing available games, but might later make an effort to produce their own games.

III. Implementation details

Code Colosseum is a framework to build and play multiplayer games over a network. It has a client-server structure and it is designed towards simplicity of setup and use. Both the server (*cocod*) and the client (*coco*) are written in Rust [19]. *Code Colosseum* is free software, released under the Mozilla Public License 2.0, and can be found on GitHub [8]. A graphical overview of the architecture can be found in Figure 1.

Both *cocod* and *coco* are written using the *Tokio* [9] asynchronous runtime. This allows the *cocod* server to be able to handle many concurrent connections efficiently, making it suitable for hosting hundreds of matches simultaneously.

Code Colosseum provides no user authentication by design, to keep its codebase and setup as simple as possible. The *cocod* server keeps a lobby of waiting-to-start and running matches. Anyone can create a new match for one of the supported games, and anyone can join a waiting-to-start match, although some limitations can be imposed, for instance by providing a password at the time of creation to restrict participation access. Once the number of players needed to start the match are reached, the match automatically begins and the players start playing. To create, list and join matches the tool used by the contestants is the *coco* client. All matches can also be spectated, even when they are already running or password protected.

It is possible to add a new game to *cocod* by expanding it. In order to do so, a *game manager* has to be written. This is a program that, given n bidirectional pipes (for the players) and an output only pipe (for the spectators), must implement the game logic and receive/send information from/to the players and the spectators accordingly. This minimal interface tries to lower the requirements to write a new *game manager*. Note however that this is not a trivial task: special attention is needed when handling multiple data streams simultaneously, as some kind of multi-threading or polling is needed, which, if managed incorrectly, can lead to problems such as deadlocks or synchronization errors. Moreover, even though both real-time and turn-based games are possible, the former are harder to implement, since they need to account for network latency.

The communication between the *coco* client and the *cocod* server is done using *WebSockets* [20] and a custom JSON-serialized protocol. Using *WebSockets* as the communication channel has several advantages over plain TCP channels:

- they can pass through HTTP proxies, which are common in schools and various institutions when the traffic is monitored and filtered;
- they are message-based rather than stream-based, which simplifies the handling of our message-based protocol;
- they can be put behind a standard HTTP reverse-proxy to enable connection security (using HTTPS) and traffic shaping;
- a *WebSocket* client can be instantiated inside a web browser, allowing for a web application to communicate directly with a *cocod* server.

When a contestant joins a match using the *coco* client with its bot, *coco* connects to the specified *cocod* server and waits until the match starts. When that happens, the *coco* client captures the *stdin* and *stdout*[2] channels of the bot, and virtually connects them to a bidirectional pipe of the *game manager* for that match. This is done using the previously-established *WebSocket* communication channel with the *cocod* server.

Spectators can join a match using the *coco* client at any time. The *cocod* server will send real-time updates of the game as generated by the *game manager*. If the spectator joins an already running match, then the *cocod* server will first send all game data from the beginning of the match. The *coco* client does not provide a native visualizer for games, hence a custom program that reads the game data and prints a human-readable representation of it is needed. It has to be specified to *coco*, which will send the game data to its *stdin* stream.

Both *coco* and *cocod* are CLI[3] programs. While for *cocod* this is not much of a problem, having a graphical client could be beneficial for tasks such as spectating a match. Note that *coco* allows the attachment of an arbitrary program for both playing and spectating games, opening the possibility of a GUI[4]. Since the communication channel is a *WebSocket* and most of the implementative burden lies on the server, a web client could be written to provide most of the functionalities of the *coco* client in a graphical and cross-platform way. Also, by providing additional layers, one could collect statistical information on the platform usage, like in *CMSocial* [21].

IV. Pilot Experience

We conducted a pilot experience that involved 16 students with a Computer Science background in a *double-elimination tournament* playing the *Royal Game of Ur*. The students were also familiar with the *Competitive Programming* field.

A. Royal Game of Ur

The *Royal Game of Ur* [22] is a strategy game where two players play against each other in a racing competition, moving their tokens in a board according to the result of four 2-faced dice. This game is of historical importance since it is one of the oldest known games (it was played in ancient Mesopotamia in the third millennium BCE), yet it is relatively unknown to most people.

[2] *standard input* and *standard output*.
[3] Command-Line Interface
[4] Graphical User Interface

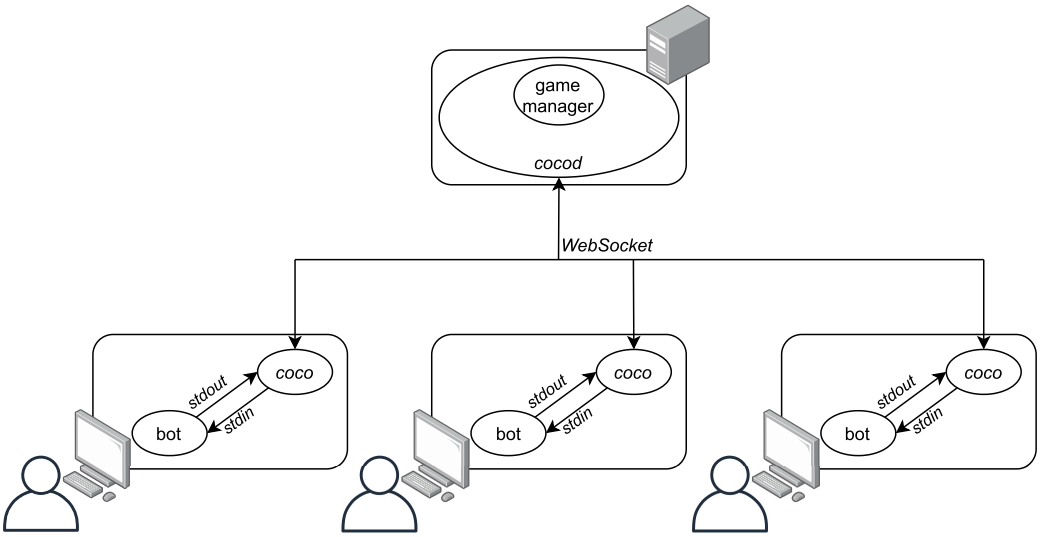

Fig. 1. *Code Colosseum* architecture: each contestant writes a bot that runs on their local machine along with an instance of *coco*. The bot and *coco* communicate through the *stdin* and *stdout* channels. The *coco* clients connect via *WebSockets* to *cocod*, which runs on the central server. Inside *cocod* an instance of the game manager is running.

We selected it as the pilot game for many reasons:

- the game rules are very simple to understand and quite straightforward to implement in code;
- the strategy component is nicely compensated by some luck, making it interesting to watch and partially bridging the gap between seasoned and novice competitors;
- other than the optimal strategy, there are many easier but well-performing strategies.

B. Double-elimination Tournament

In a double-elimination tournament there are 2 brackets: the winner bracket and the loser bracket. Initially all the players are in the winner bracket. They are moved to the loser bracket when they lose their first match, instead of being eliminated at their first defeat. This mechanism offers to every player a *second chance* to be the winner of the tournament. With a *second chance* players can learn from their mistakes, fixing their programs and improving their strategy.

This types of elimination tournaments have a number of matches that is a function linear in the number of players, keeping the total number of matches to a reasonable amount, especially when there are many players. Note that *Code Colosseum* only manages single matches, therefore the tournament structure is totally independent from it. In fact, we used *Challonge* [10] to keep track of the progress of the tournament.

C. Experience and Feedback

The participants and the organizers met in a *Discord* [11] server, and 14 out of the 16 registered players were present. The game was announced at the start of the tournament. At that point, participants were provided with the game rules and the game communication protocol. Players started implementing their bots using their favorite programming language and development environment. The bots would exchange text messages with *coco* following the game communication protocol. Players could prepare for the matches by either playing among them in friendly matches or against a server provided bot. After 90 minutes the first matches started, and up to two matches of the same round were held in parallel. Between rounds the participants were given some time for tweaking and fixing their programs (from 15 to 45 minutes, depending on the round).

Using an anonymous survey, at the end of the tournament, some feedback was collected from the participants and 13 of them answered. The questions were of three kinds: open, yes/no, and rating from 1 (least) to 5 (most).

Results [12] showed that none of the participants knew the game before the tournament, and all of them "enjoyed participating", rating it at least 3 with more than half of them rating it 5 (see Figure 2). All of them expressed the intent to participate in a second edition.

The game was pretty well accepted, 10 participants graded it as the chosen game at least 4. On the contrary nearly half of them would have liked more time for coding and debugging their programs; a couple of them preferred less time between the rounds for a nicer spectating experience. The students generally didn't find it hard to use the *coco* CLI client, probably due to their background.

Many participants asked for a web-based visualizer, with a database of the played matches. This would have been very useful for inspecting opponent's strategies with the added benefit of being more user-friendly and very platform-independent.

V. Conclusions and Future Works

This paper presented *Code Colosseum*, a platform to create and play real-time multiplayer games meant to be played by programs. We proposed *Code Colosseum* as a complementary educational tool, akin to competitive programming but based

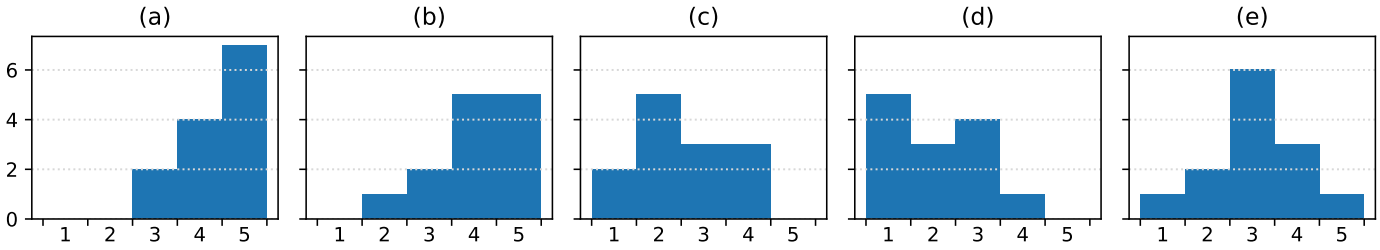

Fig. 2. Histograms of the feedback collected after the tournament from 13 participants. Answers were from 1 (least) to 5 (most). (a) *How fun was it to participate?* (b) *How much did you like the Royal Game of Ur as the game chosen for the tournament?* (c) *Should the tournament have lasted longer?* (d) *How difficult was it to use a terminal interface with respect to a graphical one?* (e) *In your opinion, would it have been interesting to be a spectator?*

on games instead of problems, to foster Computer Science education to a wider audience. We provided an implementation of the *Code Colosseum* concept with the *cocod* server and *coco* client and discussed their implementation details. We then described the pilot experience we organized, a *Royal Game of Ur* double-elimination tournament between 16 students. We then described and discussed the feedback received from the students about this experience.

From the students' feedback we can assess that the pilot experience was substantially positive: most of the students really enjoyed participating and all of them would participate again. The game made them explore concepts from game theory and statistics. The tournament has enthralled the participants to the game, so much so that some of them kept refining their bots for some weeks after the tournament took place.

In the future we plan to host a second tournament. Our commitment is to improve the experience by implementing the feedback we received from the participants. In particular, the following well-defined directions are indeed likely to have a positive impact:

- add a web interface for spectating the matches;
- add a replay functionality to re-watch past matches;
- rebalance the durations of the various phases of the tournament. In particular, by increasing the time before the first match.

Online resources

[1] *International olympiad in informatics*, https://ioinformatics.org/.
[2] *Topcoder*, https://www.topcoder.com/.
[3] *Codechef*, https://www.codechef.com/.
[4] *Atcoder*, https://atcoder.jp/.
[5] *Cmsocial*, https://training.olinfo.it/.
[6] *Codingame*, https://www.codingame.com/.
[7] *Codecombat*, https://codecombat.com/.
[8] *Code colosseum*, https://github.com/dariost/CodeColosseum.
[9] *Tokio*, https://tokio.rs/.
[10] *Challonge*, https://challonge.com/.
[11] *Discord*, https://discord.com/.
[12] *Feedback results*, https://docs.google.com/spreadsheets/d/1HPgLU-hNN3vGydThcqcRaFKLox5fkfFfWkNabBURkRM/.

References

[13] C. Bellettini, V. Lonati, D. Malchiodi, M. Monga, A. Morpurgo, M. Torelli, and L. Zecca, "Extracurricular activities for improving the perception of informatics in secondary schools," in *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, Springer, 2014, pp. 161–172.
[14] T. Verhoeff, "The role of competitions in education," *Future world: Educating for the 21st century*, pp. 1–10, 1997.
[15] V. Lonati, M. Monga, A. Morpurgo, and M. Torelli, "What's the fun in informatics? working to capture children and teachers into the pleasure of computing," in *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, Springer, 2011, pp. 213–224.
[16] M. Mirzayanov, O. Pavlova, P. Mavrin, R. Melnikov, A. Plotnikov, V. Parfenov, and A. Stankevich, "Codeforces as an educational platform for learning programming in digitalization," 2020.
[17] M. Papastergiou, "Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation," *Computers & education*, vol. 52, no. 1, pp. 1–12, 2009.
[18] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the first international conference on Autonomous agents*, 1997, pp. 340–347.
[19] N. D. Matsakis and F. S. Klock, "The rust language," *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 103–104, 2014.
[20] I. Fette and A. Melnikov, "The websocket protocol," 2011.
[21] W. Di Luigi, P. Fantozzi, L. Laura, G. Martini, E. Morassutto, D. Ostuni, G. Piccardo, and L. Versari, "Learning analytics in competitive programming training systems," in *2018 22nd International Conference Information Visualisation (IV)*, IEEE, 2018, pp. 321–325.
[22] I. L. Finkel, "On the rules for the royal game of ur," *Ancient Board Games in Perspective*, pp. 16–32, 2007.