# aMazeChallenge: An interactive multiplayer game for learning to code

***Nicos Kasenides***
*University of Central Lancashire—Cyprus Campus (UCLan Cyprus)*
*Larnaca, Cyprus* *nkasenides@uclan.ac.uk*

***Nearchos Paspallis***
*University of Central Lancashire—Cyprus Campus (UCLan Cyprus)*
*Larnaca, Cyprus* *npaspallis@uclan.ac.uk*

**Abstract**

The accelerating growth of technology in the last decades has led to an ever-increasing demand for computing professionals. At the same time, the number of computing graduates around the world grows at a slower pace, resulting to a bottleneck in the supply for industry vacancies. One of the many reasons for this is a common notion that programming is a tedious and daunting task that is relatively unrewarding. Despite that, extracurricular events such as Hour of Code and Code Week have been successful in attracting more young people to computing, and educational programming games hosted during these events have become very popular. In this paper, we report our experience with developing a multiplayer educational game called aMazeChallenge which aims to teach programming to students in a gameful environment. In our game, players must program an avatar to escape a virtual maze arena using simple instructions in a block-based programming language. aMazeChallenge utilizes public cloud infrastructure to enable code execution at the backend while players can participate using an Android client. Through aMazeChallenge, our objective is to engage students in a fun, competitive environment where they can learn the basic concepts of programming. Our preliminary results show that students enjoyed playing aMazeChallenge and that the game increased their awareness of programming concepts as well as of the significance of computing.

**Keywords:** computer science education, computational thinking, gamification

## 1. Introduction

The infiltration of technology in our everyday lives has led to the rapid expansion of the digital realm. Since the invention of the personal computer in the 1970s, technology has experienced unprecedented growth. In turn, this has increased the demand for computing professionals — such as software engineers, software architects and analysts. Unfortunately, the supply of such professionals is falling short of the demand at an increasing rate due to various factors. Even as we bear witness to the prosperity brought forth by the understanding and evolution of computerized systems, many secondary education students do not consider a degree in this field. For example, figures from the National Science Foundation in the United States indicate that there are about 125,000 job openings related to Computing and only about 40,000 graduates to fill these positions [18]. This illustrates the large gap that exists between the demand and supply of computing graduates. In our digital age, this shortage of software specialists has created an ever-widening gap which must be addressed if we are to keep advancing our technology.

In recent years, the focus of the computing community has been to emphasize the importance of programming, mainly to younger audiences, through limited-time extracurricular events. Two such examples are the Hour of Code [14, 21] in the United States and the Code Week [11, 16] in Europe. Such events have been successful partly because their educational value is intertwined

with entertainment – something the traditional teaching methods do not always prioritize. While the success of these events is undoubted, their impact is limited by the fact that they are designed to be held during off-times —such as on weekends or after the normal teaching hours— and for limited periods. To complement the positive impact of these events, other student engagement options are often utilized, many of which are related to educational games.

Educational programming games, such as CodeCombat [1] and CodeMonkey [3] are fun environments where students of various ages can learn to code while playing. Such games rarely require any prior experience in programming and can thus be relevant to a wide range of audiences, in both primary and secondary education. With *motivation* being an important factor affecting the decision of students to learn programming [12], games must be entertaining enough to retain their interest to play and learn. Games like CodeMonkey and CodeCombat have traditionally been focused on *single-player gameplay*, asking students to solve progressively more difficult challenges or puzzles by defining code.

Other games, such as Robocode [10, 19] utilize *multiplayer gameplay*, either in the form of co-operative or competitive setups. We are motivated by the success of such games, which tend to be quite immersive. However, competitive multiplayer games like Robocode require significant prior programming knowledge which is a drawback when it comes to attracting new learners. Instead, we aimed to create a competitive multiplayer online game which allows players to learn the basics of programming quickly and then compete in a virtual environment, without requiring any prior programming knowledge.

In this paper, we report our experience with developing an educational game with which students can learn programming in an entertaining way, hopefully leading them to take an interest in further study in computing. We present *aMazeChallenge* [2], an interactive multiplayer online game for learning to code by playing. In its simplest form, aMazeChallenge is a maze-solving game in which players program their avatars to escape a virtual maze using simple instructions written in the block-based programming language Blockly [9].

The objective is to engage students in a fun, competitive environment where they can learn the basic components of a program. We describe the technology and underlying software system used to realize aMazeChallenge — our approach utilizes Google's App Engine to provide cloud-based backend services that make in-game actions such as player movements possible. The game's front-end is implemented as an Android application, which allows players to see how their avatar moves in the game and to write their maze-solving code using Blockly [9]. Furthermore, we describe additional features, such as code compilation, verification and processing, tutorials, personalization, and a custom designer tool for defining new levels.

We assess the effectiveness of aMazeChallenge using feedback collected from high school students who participated in a demonstration session. Our preliminary results help us assess whether or not our approach has potential to have a positive impact on a student's view of programming and affect their technical skills and mindset. From these results, we see that students enjoyed playing our game and became more aware of programming. Finally, we discuss the potential uses of similar technology to enable other types of educational games.

The rest of this paper is organized as follows: Section 2 outlines the objectives of this project, especially from an educational standpoint. Then, Section 3 describes our approach, the technology we used and our implementation strategy. In section 4 we discuss our preliminary evaluation and results and we share our experiences. We explore related works in section 5. Finally, we close with conclusions in section 6.

## 2. Objectives

At the highest level, using aMazeChallenge we aim to challenge misconceptions by youth that programming is a dull and daunting task. Thus a main objective is to show that programming can be fun and entertaining, while still being rewarding. In terms of teaching programming concepts through aMazeChallenge, our objectives are for the players to develop skills to:

- Read and understand existing code. It is almost impossible even for experienced programmers to write code without reading and understanding examples first. We target this objective by creating a smooth transition between real-life actions to code statements (*e.g.*, moving forward, turning left or right, etc.) during the tutorials.
- Improve or fix existing code. As the players' skills progress, they should be able to identify syntactic and semantic errors in code and be able fix them. We target this using a *training mode*, in which players are asked to incrementally improve their code to solve more complex levels.
- Write code from scratch. As the learners advance, they should become confident about their programming skills and able to define their own code from scratch. Our code editor allows players to write code without any minimal limitations.
- Improve code efficiency and clarity. Although not as important for beginners, being able to write clean and efficient code is a significant skill in software development. aMazeChallenge rewards players who create more efficient code, because these players are ultimately able to solve levels faster and thus beat other players.

By practicing the above skills, players develop critical thinking and problem-solving skills that are applicable in multiple domains. An important step in the learning process is to make players realize that they are —perhaps subconsciously— using several problem-solving techniques for everyday tasks. Furthermore, existing knowledge from other STEM subjects that are more prominent in secondary education may help the learning process.

An additional objective with aMazeChallenge was to create a gameful environment that features engaging graphics, relevant sound effects, and interesting game-play, as these are important in attracting young audiences to the game.

## 3. Methodology

In this section we describe our approach in terms of game features, as well as the software architecture of the aMazeChallenge.

### 3.1. Game rules and mechanics

aMazeChallenge is a turn-based, multiplayer online maze-solving game in which players compete against each other with the goal of exiting the maze. The first player to escape the maze wins the round. The game features highly dynamic levels with various types of objects and each player controls an in-game avatar which they can program to escape the maze and thus complete the challenge.

The players can program the avatar by utilizing specific functions —in the form of Blockly elements— which enable their avatar to move and navigate the maze. The available actions of a robot are '*moveForward*', '*turnClockwise*' and '*turnCounterClockwise*'. Meanwhile, the game arena randomly spawns several items that benefit or harm a player's avatar. Such items include: a) coins which give the player points, b) fruits, which regenerate health, c) traps, which slow down the player or cause them to miss some turns, etc. These items add an element of randomness and make the game more interesting, guiding players to define code to either

retrieve or avoid these objects. Furthermore, they encourage the players to utilize decision-making constructs to decide whether or not they want to approach or avoid certain types of objects.

The mazes are divided into cells, spread on a two-dimensional *grid*. We use two-dimensional arrays to store the state of each maze challenge, with the minimum size of mazes being 5x5 and the maximum being 30x30. To achieve the behavior needed for a maze, each cell in the grid has a separate state which indicates which of its sides has walls. Cells can have walls in their top, bottom, left and right sides. We implement this using a hexadecimal digit for each cell — for example, the digit E translates to 1110 in binary, which means that the top, bottom and left sides of the cell have walls, while the right side is left open. When a player attempts to move towards the walled side of a cell, their move is automatically rejected and their turn is lost. To avoid this, players can utilize other assistive blocks, such as '*canMoveForward*', '*canMoveLeft*', '*look*', '*getDirection*', to explore their surroundings before making an action. By default, all mazes are outlined with a wall to keep the players from falling off the grid.


**Fig. 1.** A screenshot of the aMazeChallenge app: this view shows a maze which includes reward and penalty items (left side), and the Blockly-based code editor (right side).

Players can gather points by collecting various types of coins and gifts which are spawned as in-game items. In addition, each avatar has a health status which can be increased by collecting fruits or decreased when stepping into bombs. Furthermore, other objects such as *speedhacks* or *traps* can affect the player's ability to move — speed-hacks increase the number of moves per turn to two for a small period while traps cause the player to miss several turns. The game can also spawn several *bombs*, which decrease the health of players who are in their vicinity when they explode. Bombs go through a sequence of stages before exploding, giving the players time to detect and avoid them. The number and type of objects generated by the game depend on

its *difficulty*. For example, *Easy* difficulty spawns mostly beneficial (also known as *Reward*) objects, while *Hard* difficulty spawns mostly damaging (also known as *Penalty*) objects. In between the two, *Medium* generates a balanced mix of reward/penalty objects. When a player's health status becomes zero, they are automatically ejected from the game but they can still start over by submitting new code.

If two players manage to complete the maze with the same number of turns, *e.g.*, because they used identical algorithms, they are ranked by points and then by health in descending order. The first player to submit plays first and the rest of the players take turns. Each player's code is executed once per turn and the game cycles through all of the players to execute their moves until all players have managed to exit the maze or have been eliminated. Figure 1 shows screenshots from the game's maze view and code editor, used to define the player's maze-solving logic.

### 3.2. Programming

To allow players to program their avatars, we utilize Blockly [9], a visual programming language developed by Google. Blockly has been utilized as an introductory programming language in a plethora of educational games. Block-based programming has become popular, primarily as part of Scratch [20], which is widely used in primary and secondary computing education [15].

We have created a Blockly library which includes commands for specific aMazeChallenge actions such as: "*moveForward*", "*turnClockwise*", "*look*", "*getDirection*", etc. We also define several custom data types to allow players to check their surroundings and navigate through the maze. For example, we define a "*Direction*" class, which can take the values "*North*", "*South*", "*East*" and "*West*". Players can thus receive useful information about their avatar's direction using the "*getDirection*" command, which returns a "*Direction*" type. Similarly, we define an "*Item*" type, which indicates the type of each of the in-game item — for example "*RewardItem*", "*PenaltyItem*" or "*NoItem*". Using the "*look*" command, players can determine whether or not there are in-game items in their vicinity and what they are. Additionally, we assist players with several other library functions such as "*compass*", which returns the direction the exit is relative to the player and "*onMove*", which enable the player to determine the direction they would be facing after making a certain move.

In terms of code structure, players are required to write their code within two functions, named "*Initialize*" and "*Run*". The initialization function is executed once at the start of the player's first turn and allows them to initialize variables or set up data structures. On the other hand, the "*Run*" function is executed in every turn and returns the type of move the player's code has resulted to. Statements and blocks defined outside these two function blocks are ignored. In addition to our custom aMazeChallenge blocks, we enable several default blocks provided by Blockly which offer standard programming constructs — variable declarations, expressions, conditionals, loops, etc.

Right before submitting player code for execution, the system performs several static checks to ensure that the code is valid. For example, we check: a) that the player has inserted their code inside the two required functions, b) whether or not the player has added any code blocks, c) if the run function returns an in-game action and, d) for any syntactically correct but invalid code, especially in cases where the code may go into an infinite loop. By default, the system deals with long-running code by interrupting their execution if they do not terminate after one second of execution time. When such mistakes are detected, an error message will appear in the editor, warning the players to handle these errors before proceeding.

### 3.3. Code execution

As discussed, aMazeChallenge utilizes Blockly to allow players to program their avatars. To run such code, it must first be converted to code that the client (Android-based) and the server

(App-Engine-based) can execute.

The Blockly library contains several code generators for various programming languages including JavaScript, PHP, Python, etc. These generators convert user code given as blocks to code expressed in these languages. Each block contains a code definition, which defines how it will be converted into code for a specific programming language. For aMazeChallenge, we opted for JavaScript, which acts as an intermediary language which can be executed in Java-based code – which is itself supported by our frontend and backend systems.

The code execution chain involves the following steps: First, we translate Blockly code into JavaScript. The end-result is further processed to a specific format for aMazeChallenge. During this step we can look for and identify mistakes made by the players and warn them before proceeding further. Additionally, we add code to handle data persistence across rounds – effectively maintaining the player's session. The processed JavaScript code is then sent to the server as plain text and stored. When it is that player's turn to play, the server retrieves the corresponding code and executes it. Our server runs on a Java runtime and we utilize a Java-based JavaScript interpreter/engine called Rhino [4], provided by Mozilla. Using Rhino, we can run specific JavaScript functions, such as the player's defined *Initialize* and *Run* functions and get their returned values – in this case, we are mainly interested in the value returned by the Run function, which determines the *move* to be made. These functions are interpreted by Rhino and converted into their respective Java statements, which can be executed by our server's runtime environment. Lastly, to support global variables which can persist through player turns we have implemented a custom system for exchanging values of attributes between JavaScript and Java code. The pipeline of processes used for code compilation and execution is summarized in the pipeline shown in figure 2.

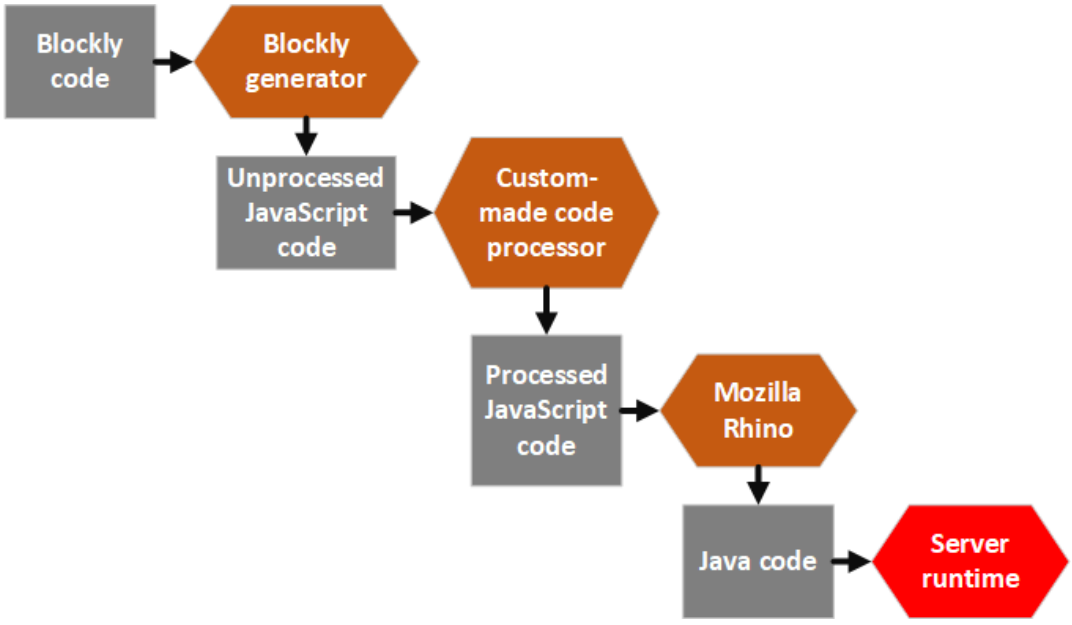


**Fig. 2.** The pipeline of processes used to generate, process and run code written by the players.

### 3.4. In-game features

Since aMazeChallenge is an educational game designed to introduce its players to programming fundamentals, we have created a *Learning* section which features a tutorial that introduces the player to a) the game and its rules, b) basic programming principles and, c) programming their

avatar to make in-game moves.

Upon completing the tutorial, players can personalize their avatar by selecting a custom image, avatar color and set their name and email. We collect this information so that we can track each player's progress in multiplayer mode. Then, players can proceed to write or edit code or load several code samples to see how maze-solvers can be written. For instance, we provide examples which utilize the left-wall-following and right-wall-following algorithms, random moving and one utilizing the compass function. In addition, players can also save their own code and load it at a later time. When the player exits the code editor, the code is automatically checked for syntax and logic errors before being compiled.

The aMazeChallenge app also features a training mode, allowing players to test their code locally on their device, before uploading it to the online server. We include several training scenarios for the players which get progressively more difficult. For instance, the first maze asks the players to walk in a straight line to complete the challenge — which means they only have to utilize the *moveForward* command. The second maze asks the player to walk straight and then make a turn at the corner to complete the challenge, thus requiring basic decision making to determine what to do in case they run into a wall. Subsequent training challenges become progressively more difficult and require the use of more complex code structures to solve.

After completing the above steps, players can *compete* with each other online by joining a challenge and then uploading their maze-solving code. The players' code is then executed by the server sequentially, as described in subsection 3.1 until all of the players have exited the maze or the challenge expires. When a player finishes an online game, they are asked to complete a questionnaire that is used to gather user feedback for the game.

Lastly, we enable players and game administrators to experiment with different types of mazes to make aMazeChallenge more interesting. We utilize a maze generator both as a standalone application and as an in-game feature to allow game administrators to create mazes for online gameplay and players to create custom mazes for training. Our maze generator allows us to create a large variety of maze types, such as a) single-solution mazes, b) multiple-solution mazes, c) sparsely-walled mazes, and d) empty mazes. Finally, the maze generator allows mazes with custom parameters such as game size and starting/finish positions in the generated maze, the difficulty, as well as the background color, wall color, and audio.

### 3.5. Architecture

Our approach is based on the client-server model, which allows for a distributed game to remain secure by carrying in-game operations on the server [17].

In this case, the client is an Android app that visualizes the state of the game by displaying the maze and the players' avatars in it. In addition, the client also allows the player to utilize Blockly with which they can write their code and identify common problems. When finished, players can join a game on the server, upload their code and watch it execute live, once the competition starts. To achieve this functionality, the client interfaces with the server via an HTTP-based API. This includes relevant commands, as shown in table 1.

The server receives the HTTP requests and decodes the commands and data coming from the clients *e.g.*, to join a game or make an in-game move. For this we utilize an API based on Web standards which exposes this functionality to the client devices. To realize the gameplay, the server receives the players' code, performs static code analysis, compiles it, runs it and then updates the state of the game accordingly. To update their own view of the game's state, the clients use a polling technique by requesting the current game state from the server once every second.

Game data, such as the game's state, player information and sessions are stored persistently on a cloud-based database, which is a separate component of the architecture. The server queries

| Endpoint | Functionality |
|---|---|
| /challenges | Retrieve a list of available challenges. |
| /join | Join an online challenge. |
| /submit-code | Used to submit player-defined code. |
| /game-state | Gets the state of the joined challenge. |
| /submit-questionnaire | Submits a questionnaire response. |
| /admin/add-challenge | Create a new challenge. |

**Table 1.** API endpoints implemented by aMazeChallenge.

the database when required, receives information and then encodes it in a specific format before sending it back to the client devices. In addition, when an action is made by the player, the server performs write operations on the database accordingly, *e.g.* to update their submitted code.

### 3.6. Infrastructure

We use cloud-based infrastructure to power our server-side. This enables us to: a) eliminate the need for server setup and maintenance, and b) create a scalable system that can support a large number of concurrent players. Specifically, aMazeChallenge is designed to utilize the Google Cloud Platform. We host our backend API on Google's App Engine, a Platform-as-a-Service product which allows our project's runtime to scale. The App Engine allows us to easily deploy our API using Java Servlets without having to manage any infrastructure such as servers, domain settings, etc. This simplifies the development process significantly. To persist game data, we utilize Google's Cloud Datastore, a NoSQL type of database with low latency and with support for high scalability. This enables us to scale to larger maze sizes or to a large number of players as needed, while still maintaining good performance. Lastly, App Engine and the Cloud Datastore are related products. The Cloud Datastore can be easily queried from within App Engine applications, which simplifies the development process.

## 4. Discussion

We tested and evaluated aMazeChallenge at an event organized at UCLan Cyprus, which was attended by 43 high school students. The average age of the students was 13.95 years old, with the minimum being 13 and the maximum being 16. Gender-wise, most (74%) of the participants were male, whereas only 26% were female. For the vast majority of students (83.7%), this was their first experience with programming. The students participated in a brief training which featured an introduction to what programming is and how it is relevant to many everyday tasks. During the training, the students were also presented with basic programming constructs such as variables, conditionals and loops and how these are related to different scenarios in a game.

The students were asked to answer two questionnaires — one before using aMazeChallenge and one after playing. The questionnaires included questions regarding their views about programming and how they rated their skills in programming and mathematics. In addition, they were presented with several programming questions that were based on the concepts taught during the mini-lecture. During the evaluation, the students were instructed to download the app, briefly explore the learning section and personalize their avatar. A special session was set up with a moderately difficult maze onto which the participants joined after writing their code in the editor. Upon joining the session, the students were able to see their avatar move within the maze according to the code they defined. After playing, the students answered several questions to rate aMazeChallenge and state which of its features they enjoyed the most. Our goal was to use these two sets of questionnaires to gather insight on if —and by how much— aMazeChallenge

affected the perception of the participants about programming. In addition, we aimed to evaluate the entertainment aspect of our game by gathering feedback from the students on whether they enjoyed the game and what they enjoyed the most.

Our preliminary results show that the participants felt relatively confident about their ability to learn programming. Asked to self-rate their abilities, participants responded with an average of 50%, with 0% being not competent at all and 100% being extremely competent. For female participants, this average was slightly higher, at 58%, and for male participants slightly less, averaging at 42%. In addition, students were asked to specify their reasons for taking an interest in programming. While 21% responded that they had no interest in programming, 40% said that they "Love Programming", 35% said that they were "Curious" about programming and 23% said that they were interested because it is linked to high employability. The rest of the participants (16%) said that they were already familiar with or good in programming. When asked if they would follow a programming career in the future, 46% of the participants responded with either "Likely" or "Very likely", while 9% responded "Not likely" and 12% with "Not likely at all". Female participants appear less likely to consider a computer science career than their male counterparts even though they feel more confident about learning programming. Around 57% of them said it was either very likely or likely to do so, whereas male participants appeared to be more interested (79%). According to the participants, the most enjoyable feature of aMazeChallenge was its graphics at 29%, followed by playing with others at 23% and being able to practice alone at 17%.

On the other hand some of the replies indicated that both aMazeChallenge and perhaps the evaluation approach needed further refinement and additional measurements. For example, a large group of participants (35%) answered that their programming skills were "Unchanged" or that they were "Confused" afterwards. 28% of the participants responded that their skills had increased after playing the game. We view this positively since we believe that the students were made aware of their true skills during the game session. Even though students faced several difficulties trying to understand how the game works and how programming can be used in a game, we believe that according to their responses, they are still likely to follow a programming career in the future. Of course aMazeChallenge is not their only exposure to programming and its benefits. While it cannot take full credit for this, we do hope that it has helped a number of students to give serious consideration to a programming career.

Whichever the impact of our game on the perception of students about programming, we have confirmed that the app we have developed works and can be utilized to power both educational and other types of online games. We have managed to successfully run aMazeChallenge on public cloud infrastructure at the Platform-as-a-Service (PaaS) layer. This gives the potential to multiplayer games like aMazeChallenge to be played by large numbers of players and feature more interesting and dynamic environments. We have also shown that custom runtimes can be created to run block-based code written by novice programmers. Using existing libraries and manual adjustments to the code, we were able to pass Block-based code through several stages and programming languages and ultimately execute it on our backend runtime. This makes it possible to create a plethora of game types that can be used in an educational context, with the potential of being Massively Multiplayer Online Games (MMOGs). Utilizing this type of games may be beneficial due to possible enhancements in educational experience over conventional games [7]. These usually manifest as social aspects either through collaborative or competitive gameplay, rendering them a promising future research direction.

## 5. Related work

Researchers and educators have utilized a variety of educational games to increase student engagement and motivation in general, and in programming specifically. In this section, we discuss related works.

IBM's Robocode [10, 19, 5] is one of many examples of educational programming games. Robocode is a game of tanks that must be programmed by their players to fight and defeat other tanks in a battle arena. Players can program their tanks using a robot API that exposes the possible functions of the robots. There are several tutorials provided online and the players can program the robots in two environments: Java and .NET. Robocode makes it possible to test robots in a training environment and to simulate various battle scenarios. Most importantly, players can play and compete against each other in a common arena. A survey conducted among 83 participants showed that 80% of them believed that their programming skills had increased after playing Robocode. It should be noted that as one of the more advanced examples of educational programming games, this focuses also on developing student skills in *Artificial Intelligence*.

The authors of [22] argue that "the 21st century sees a new group of younger and emerging generations who grow up with and are exposed to different devices". This exposure to technology makes students more perceptive to similar approaches in education. Because of this, the authors attempt to demonstrate that children who play games can benefit in terms of understanding programming concepts – such as sequences, iteration and decision making. They investigate the effectiveness of digital games using a game called "Program Pacman" over several workshops, during which the students had to program Pac-Man. Overall, the level of confidence of the students rose, as they reported feeling more confident with their programming skills.

Others have used similar approaches to "improve recruiting and retention in computer science through [...] game-based learning environments" [6]. Game2Learn [6] is a research lab that specializes in educational programming games and leverages them to counter the negative experiences of students related to computer science. The authors propose that games should be further utilized in computing curriculum and suggest that "iconic" (or visual) programming languages should be used. Lastly, the authors support that educational games should promote the use of writing correct code by featuring scenarios where the players must fix their code to fight off virtual bugs or monsters. Experimental results from multiple games created by Game2Learn have shown that game-based programming can be fun, engaging and satisfying for students.

More recently, other related games have appeared featuring the theme of escaping from a maze. Algotaurus [13] is an educational programming game in which the players control robots that exist in a "microworld" using a "mini-language". The objective is to escape a procedurally-generated labyrinth/maze by programming the robot with an algorithm. The authors argue that such games have the advantages of being self-taught once the basics of programming are introduced to the players — thus making them suitable for introductory workshops. Through experimentation, students learn how to write and execute code to see its effects. The authors describe their experience using Algotaurus to teach introductory programming to several audiences of varying ages. They found that several students found the tasks interesting as they would often skip breaks in order to continue and finish in-game tasks. Despite that, such approaches have limitations as several of the students felt frustrated after not being able to complete some tasks.

To eliminate negative experiences and make games as entertaining as possible, other studies have focused on the conceptualization and design of serious games. iPlus [8] is a design methodology which adds a pedagogical component to serious games. Using a meta-model, it allows a deeper understanding of the underlying concepts used to design serious games and enables their development using a formal modeling language. Furthermore, iPlus identifies crucial elements that must be taken into consideration when designing serious games, such as narratives, game rules and mechanics, gameplay, and more. The authors argue that these techniques lead to "*excellent engagement from end-users*".

## 6. Conclusion

Educational programming games often receive a mixed response from both students and educators. This is perhaps because they try to satisfy requirements stemming from both the education and entertainment sectors, which are not always aligned. Ideally, neither of the two should be sacrificed but, admittedly, it can be hard to find the right balance between the two. The success of such games possibly lies in creating a truly entertaining game that can captivate students while still being able to teach the necessary material. With new technology available, researchers may be able to develop a wider range of game types and game ideas. This will likely increase our chances of creating a successful game and thus have a bigger impact on students.

In our paper, we have shown that our approach has merit as it was built on top of a public cloud platform and enabled an online multiplayer game in a relatively short period of time and with minimal resources. Our approach details methods with which players can participate in programming games by learning and then writing code in a visual programming language which can be executed on the back-end after a series of processing steps. We reported our experience from a limited-scale, local event which was attended by several high school students. Feedback from the participants showed that aMazeChallenge made them feel more confident in their ability to program and showed that the majority of them were considering to follow a career in a computing-related subject.

In the future, we aim to expand our methods and tools to enable the development of a wider variety of game types. We also plan to expand aMazeChallenge to feature more levels and enable players to host smaller ad-hoc games where friends can play against each other. Finally, we aim to further evaluate our approach by organizing more training sessions and collecting further input.

## References


1. Codecombat - coding games to learn python and javascript. https://codecombat.com, 2019. Last accessed: 2021-04-09.
2. aMazeChallenge - App on Google Play. https://play.google.com/store/apps/details?id=org.inspirecenter.amazechallenge, 2020. Last accessed: 2021-04-09.
3. Coding for kids - game-based programming. https://www.codemonkey.com, 2019. Last accessed: 2021-04-09.
4. Rhino - mozilla MDN webdocs. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino, 2019. Last accessed: 2021-04-09.
5. Robocode. https://robocode.sourceforge.io, 2021. Last accessed: 2021-04-09.
6. T. Barnes, H. Richter, A. Chaffin, A. Godwin, E. Powell, T. Ralph, P. Matthews, and H. Jordan. Game2learn: A study of games as tools for learning introductory programming concepts. *Proceedings of the ACM SIGCSE*, 7, 2007.
7. P. Bawa, S. Lee Watson, and W. Watson. Motivation is a game: Massively multiplayer online games as agents of motivation in higher education. *Computers & Education*, 123:174–194, 2018.
8. M. Carrión, M. Santorum, J. Aguilar, and M. Peréz. iPlus methodology for requirements elicitation for serious games. *XXII Ibero-American Conference on Software Engineering, CIbSE 2019*, pages 434–447, 2019.
9. Google. Blockly - google developers. https://developers.google.com/blockly, 2021. Last accessed: 2021-04-09.
10. K. Hartness. Robocode: using games to teach artificial intelligence. *Journal of Computing Sciences in Colleges*, 19(4):287–291, 2004.
11. T. Kilamo, A. Nieminen, J. Lautamäki, T. Aho, J. Koskinen, J. Palviainen, and T. Mikkonen. Knowledge transfer in collaborative teams: experiences from a two-

week code camp. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 264–271. ACM, 2014.

12. E. Klopfer, S. Osterweil, K. Salen, et al. Moving learning games forward. *Cambridge, MA: The Education Arcade*, 2009.
13. A. Krajcsi, C. Csapodi, and E. Stettner. Algotaurus: an educational computer programming game for beginners. *Interactive Learning Environments*, pages 1–14, 2019.
14. J. Liu, H. Wimmer, and R. Rada. " hour of code": Can it change students' attitudes toward programming? *Journal of Information Technology Education: Innovations in Practice*, 15:53, 2016.
15. M. M. McGill and A. Decker. Tools, languages, and environments used in primary and secondary computing education. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 103–109, New York, NY, USA, 2020. Association for Computing Machinery.
16. J. Moreno-León and G. Robles. The europe code week (codeeu) initiative shaping the skills of future engineers. In *2015 IEEE global engineering education conference (EDUCON)*, pages 561–566. IEEE, 2015.
17. V. Nae, A. Iosup, and R. Prodan. Dynamic resource provisioning in massively multiplayer online games. *IEEE Transactions on Parallel and Distributed Systems*, 22(3):380–395, 2011.
18. National Science Foundation. Computer science degrees awarded. https://www.nsf.gov/statistics/nsf13327/pdf/tab33.pdf, 2020. Last accessed: 2021-04-09.
19. J. O'Kelly and J. P. Gibson. Robocode & problem-based learning: a non-prescriptive approach to teaching programming. In *ACM SIGCSE Bulletin*, volume 38, pages 217–221. ACM, 2006.
20. M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for all. *Commun. ACM*, 52:60–67, Nov. 2009.
21. C. Wilson. Hour of code—a record year for computer science. *ACM Inroads*, 6(1):22–22, 2015.
22. W. S. Yue and W. L. Wan. The effectiveness of digital game for introductory programming concepts. In *10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 421–425. IEEE, 2015.