Smart Learning Environments

RESEARCH Open Access

G4D - a treasure hunt game for novice programmers to learn debugging

Akhila Sri Manasa Venigalla* and Sridhar Chimalakonda

* Correspondence: cs19d504@iittp.ac.in
Research in Intelligent Software & Human Analytics (RISHA) Lab, Department of Computer Science and Engineering, Indian Institute of Technology, Tirupati, India

Abstract

Visual Programming Environments (VPEs) are predominantly being used to teach programming concepts through interactive games with interesting narratives. Games have been developed to teach basic concepts of programming such as deriving logic, writing code, debugging the code and so on. Debugging code is one of the most important activities that can improve the skill of tackling a problem. In programming, one needs to identify the correct location of an error and fix it, which is usually learned through experience. Games have been developed to teach debugging to novice programmers. Syntactical errors occur frequently in the early stages of programming. The existing debugging games aim to support users in debugging the logic of the problem, but do not target on correcting the code snippets based on syntax. To address this challenge of providing syntactical support, we propose a treasure hunt based debugging game, in which users pass through various levels of the game by debugging code snippets written in C language. We have evaluated G4D based on MEEGA+ model, with 20 volunteers, having different programming backgrounds. The results of the user survey indicate that G4D has a good quality level and about 75% of the volunteers have either strongly agreed or agreed to recommend G4D to their colleagues.

Keywords: Learning technologies, Visual programming environments, Novice programmers, Debugging games

Introduction

Computers and computer based technologies are impacting a broad spectrum of our society. They are the motivating factor of advances in areas as diverse as education, engineering, medicine and basic sciences. It has been observed that integrating technology with education has a positive impact on the attitudes of teachers and students (Christensen, 2002). Online learning has proven to be effective in enhancing student learning outcomes (Nguyen, 2015). Various techniques are being integrated in classrooms to make learning interactive, interesting and to increase retention capacity of learners. Many domains such as medicine, construction and design of machines and buildings, computer programming and so on use technology enhanced tools to support better conceptual and experimental understanding of students (Cook et al., 2011; Li, Yi, Chi, Wang, & Chan, 2018; Teng, Chen, & Chen, 2018). With the intervention of

intelligent tutoring systems and visual programming environments, learning programming has become simple and accessible. Games allow an interactive, creative and entertaining experience and hence could be used in many introductory programming courses. It has been observed that games increase motivation among students towards learning various concepts, specifically in the area of computing (Barnes, Powell, Chan, & Lipford, 2008). Several games are being developed to support learning of various computing aspects such as programming in a visual and interactive manner. It has also been observed that such games help in creating robust and deeper conceptual understanding among the students (Eagle & Barnes, 2009).

Debugging is considered as one of the most important phases and a fundamental skill of programming (Beller, Spruit, Spinellis, & Zaidman, 2018; Ko & Myers, 2005). It is also regarded as a skill that is challenging to learn and teach (McCauley et al., 2008). Debugging comes with practice and hence novice programmers face difficulties in identifying the errors and fixing them (Gould, 1975; McCauley et al., 2008). Also, novice programmers take significantly more time in correcting the errors than professional developers as they lack the idea of strategies that could be used to debug (Chiu & Huang, 2015). Studies also report that good programming understanding doesn't always imply effective debugging skill and hence suggest emphasizing on debugging in programming courses (Ahmadzadeh, Elliman, & Higgins, 2005). It has been observed that students who take up debugging exercises spend less time debugging their programs than those who don't (Chmiel & Loui, 2004). Also, debugging improves problem solving skills among various age groups. Researchers have developed online tools and games to help programmers debug their code and to teach debugging (Luxton-Reilly, McMillan, Stevenson, Tempero, & Denny, 2018; Miljanovic & Bradbury, 2017).

Most of the existing games that teach various programming concepts, including debugging, target students in primary and middle school level (Lee, 2014; Miljanovic & Bradbury, 2017). Hence they focus more on correctness of the logic, rather than correctness of the syntax. However, most of the errors during programming occur due to invalid syntax (Ko & Myers, 2004; McCall & Kolling, 2014). Also, it has been observed that understanding syntax of a program significantly contributes in debugging the program, along with semantic understanding (Luxton-Reilly et al., 2018). Some compilers and IDEs support syntactical debugging by automatically correcting few errors, which are mostly the cases of missing characters such as semicolon in some programming languages. Other syntactic errors are displayed as error messages. Attempts are being made to enhance these syntactic error messages to support programmers with more specific information about the error and ways to resolve the error, but these enhancements have been observed to ineffectual in few cases (Denny, Luxton-Reilly, & Carpenter, 2014). Also, it has been observed that syntactic errors are primary con-cerns of majority of the programmers (Gould, 1975). Considering the importance of syntactic knowledge and the amount of time that might be spent by novice developers in resolving syntactic errors, it is necessary to make novice programmers aware of syntax of the program. This could help in saving time spent in resolving programming errors. Therefore, we propose the prototype of a debugging game, G4D, that allows users to debug a code snippet, by navigating through a set of clues. It provides users with various practice problems, with errors that occur frequently. This gives novice programmers an exposure to debugging

and contributes to their experience. This could help learners, who are partially aware of syntax, to improve their debugging efficiency.

The contributions of this paper are as follows:

A game aimed to teach debugging to novice programmers, specifically to debug syntactic errors.

The pedagogical approach and theories followed by the game.

An evaluation of the game based on player experience, usability and correctness of the game.

The remainder of this paper is structured as follows. Section 2 discusses the related work followed by Section 3, which focuses on design methodology of G4D. Section 4 talks about development of the game and presents a use case scenario of the game. We present the evaluation and user survey results in Section 5 and Section 6, respectively. Finally, we discuss the limitations in Section 7 and we conclude our argument along with future directions in Section 8.

Related work

With the increase in acceptance rates of online teaching and learning environments, several tools to teach programming and make learning interactive, interesting and creative using various technologies such as Mixed Reality, Artificial Intelligence and so on are being developed.

Scratch facilitates users to drag and drop blocks of code snippets, to perform actions in a visual environment. It aims to teach programming and problem solving to students of primary and middle school (Resnick et al., n.d.). Alice supports the learning of basic programming concepts through 3D visualizations. A storytelling extension has also been added to Alice, which enables users to create 3D animated stories through basic programming concepts (Cooper, Dann, & Pausch, 2000; Kelleher, Pausch, Pausch, & Kiesler, 2007). Greenfoot provides a custom defined environment with predefined functions to support animations, graphics, sound and so on, to learn Java programming language (Kolling, 2010).

Among various tools developed to support novice programmers, few tools focus on debugging. Delta debugging, introduced by Zeller identifies a minimal subset of \failure-inducing" changes that prevent the program from working correctly (Zeller, 1999). Ko et al. have proposed an interrogative tool, whyline, that facilitates users to debug by slicing the program (Ko & Myers, 2004). Lapidot et al. have integrated debugging with music in (Lapidot & Hazzan, 2005). They have given a faulty code, linked up with a well-known music bit, with lyrics and melody to students. The students had to debug the code until the song is rightly played. The CMeRun tool proposed by Etheredge, enables users to look into the code by displaying each statement as it gets executed and thus helps them in identifying the point of error (Etheredge, 2004). Luxton et al. have proposed an online debugging tool, LadeBug, that provides users with predefined exercises containing faulty code snippets. Users can debug the code snippets by identifying the point of error based on information provided, and also add checkpoints to the code (Luxton-Reilly et al., 2018).

Chiu et al. have observed that using games to debug effectively improves the programming concepts of students (Chiu & Huang, 2015). RoboBug game has been designed to support debugging of various programming languages. It provides a set of

debugging tools that the user has to select in a given time limit based on the error identified (Miljanovic & Bradbury, 2017). Gidget game involves users to correct multiple sets of pre-written statements at each level to ensure required navigation and communication among characters in game (Lee, 2014).

Gidget uses basic English statements, and does not use the language syntax. It focuses on arriving at correct logic, not on correct syntax. RoboBug uses predefined debugging tools, where the user needn't use his/her skill of debugging. Although there are tools and games to help novice programmers learn debugging, to the best of our knowledge, tools lack creativity and the existing games mostly target students of primary and middle schools. G4D aims to help novice programmers, in the first year of their undergraduate course and those who have minimum knowledge on programming, in learning to debug the syntax, rather than the logic, through a creative treasure hunt themed game. The treasure hunt theme used to debug is another distinguishing factor of the game.

Design and methodology

Programmers generally consider locating the defect as one of the most time consuming and crucial tasks (Jones, Harrold, & Stasko, 2002). In the proposed game, we try to support novice programmers by encouraging them to isolate the chain and then correct the defect in the isolated chain.

A faulty code snippet is presented to the player. Clues to debug the code snippet are provided to the player when few boxes in the game are broken. The player has to identify the right place in the code snippet to insert these clues, which shall result in debugging the code. This might improve the skill of locating the defect. Also, the player has to identify appropriate clues as few faulty clues are also presented. These tasks shall thus enable players to practice isolating and correcting a defect.

We have developed a prototype version of G4D, a treasure hunt based debugging game to support novice programmers in debugging a given code snippet. The game deals with a girl, Veda, whose spaceship gets crashed on an alien planet. She can repair her spaceship and come out of the planet only after she finishes all levels of the game. Levels in the game refer to cities guarded by gates, on the planet and, to pass through each level, she will have to collect and fix few broken objects that can be used to repair different parts of the spaceship. Once, all the required objects are collected in the level, the city gates can be opened using a key that Veda finds in the city. The player takes the avatar of Veda and will be provided with few clues in each level to fix the object. These clues shall be placed in various boxes in the city. Few boxes also contain misleading clues, and some boxes might be empty. The key required to open the city gates is placed on one of the rocks in the city. When the player opens a box, a clue is prompted and added to the collection list. Once all the clues are collected, when player reaches crashed component of spaceship, faulty code snippet, with a numbered list of collected clues is displayed. The player should correctly map clue number to respective erroneous parts of the code, failing which, the player will again be navigated to spaceship, where he/she is given another chance to map the clue numbers.

The parts of spaceship to be fixed in each level are analogous to faulty code to be debugged, and objects required to fix these parts are analogous to steps to be followed to debug the faulty code. At each level, the player will be provided with 5 faulty code

snippets that are to be debugged. The clues provided to the player to fix the code represents the debugging steps. Hence, the player, in the course of game, would be able to learn and practice debugging various code snippets and be wary of the syntax.

We have designed G4D by following scaffolding approach to support the pedagogy of learning debugging, with considerable care taken towards cognitive load that could be induced on the users by adapting cognitive load theory.

Scaffolding to support pedagogical approach of learning to debug

Scaffolding technique has been observed to positively support development of educational games, specifically to support the pedagogy of learning programming (Jantan & Aljunid, 2012). Hence, we have also employed few scaffolding characteristics among the 10 characteristics that were observed to support games for programming (Jantan & Aljunid, 2012), in the design of G4D. The aim of G4D to ease learning to debug through serious game and the difficulties involved in learning debugging such as problem identification and solving also are observed to be inline with the advantages provided by scaffolding technique. Metaphor Scaffolding has been first introduced as an integrated approach to support learning by David Wood, Jerry Bruner and Gail Ross (Wood, Bruner, & Ross, 1976). The idea of integrating scaffolding techniques with learning is to initially mask the domain specific information and provide details on approaches and processes involved in the domain through metaphors, with eventual unmasking of the domain specific information (Guzdial, 1994). Scaffolding helps learners in accom-plishing tasks with minimal intervention from the teacher, thus enabling novices to solve problems that would have been beyond the reach with unassisted e orts (Wood et al., 1976). In G4D, the scaffolding technique is implemented by initially masking the details involved in debugging such as searching for the solution to correct a defect with the metaphor of searching for parts of the broken spaceship. Broken Spaceship to be repaired metaphor is implemented to design the game. The instructions provided in the game, based on various actions performed by users in the game support the scaffolding characteristics - Clarifies purpose, Appropriateness, Continuity and Keep Students on task. Users are also equipped with frequent message prompts that guide users on the next steps to be taken, thus implementing the scaffolding characteristic of providing Clear Direction. As the levels in the game progress, support provided to users in view of the number of clues and the message prompts related to clues is gradually reduced, thus adapting Transfer of responsibility and handover/takeover scaffolding characteristics.

Cognitive load theory in design of G4D

We have designed G4D based on cognitive load theory, that aims to make learning more efficient (Sweller, 1988). One of the main methods of cognitive load theory that has been followed in designing G4D is to break down a problem into parts and then integrate these partially completed blocks of the problem to arrive at a solution for the main problem. G4D facilitates users to debug a program by identifying and decoding the clues that correspond to statement level and thus support division of the problem. Clues found by users correspond to individual statements in program and debugging each statement based on these clues helps users to eventually debug the complete

program. The main storyline of G4D also stays inline with this divide and solve method of cognitive load theory, as it deals with repairing a broken spaceship part-by-part in each level. Figure 1 depicts the cognitive load theory method of solving a problem by breaking it into sub parts with respect to G4D. The program is divided into statements that are debugged with the help of corresponding clues found while playing G4D.

Incorporation of steps to debug in G4D environment

Effective Debugging is an essential skill for programming. Debugging refers to appropriately identifying the point of error and resolving it. It is one of the most common tasks that have to be dealt by programmers at all levels, from novice programmers to highly skilled programmers. G4D has been designed to support incorporation of three of the seven debugging steps mentioned by Andreas Zeller in (Zeller, 2009).

The seven steps mentioned by Andreas Zeller are as follows:

Step 1: Track problem in database.
Step 2: Reproduce Failure.
Step 3: Automate.
Step 4: Find possible infection origin.
Step 5: Focus on most likely origin.
Step 6: Isolate infection chain.
Step 7: Correct Defect.

The first three steps refer to real-time developers of an organization, where defects are reported as unforeseen problems by users or testing teams. They should hence be logged onto the database, reproduced and an automated test case should be developed.

However, for novice programmers, debugging starts from step 4, as the error would most likely be reported in the code by themselves, rather than testing teams, and hence,

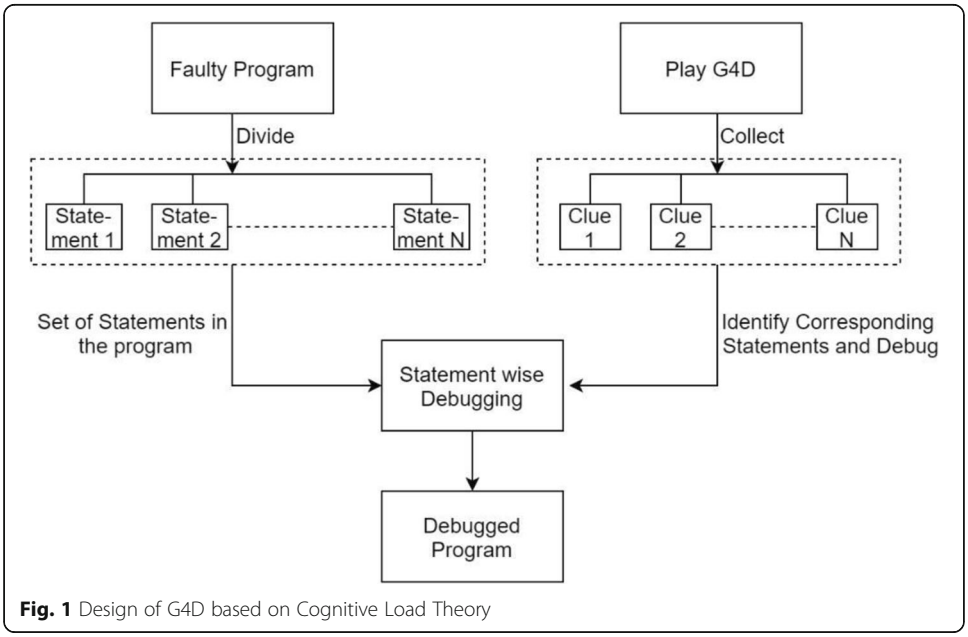

Fig. 1 Design of G4D based on Cognitive Load Theory

they might not have to track, reproduce the error or automate a test case. The scenario for which a program fails serves as a test case by itself.

The clues found by the player in the game, for a specific program are displayed adjacent to the faulty code snippet, with an aim to instigate player's thought process in focusing on the possible infection origin and most likely origin of infection. The player is required to select the appropriate position to fill in obtained clues in the code snippet. This requires identification of the line that causes defect, which is equivalent to identifying possible infection origin (Step 4), and the specific position in the identified line that is to be corrected, which is equivalent to finding the most likely origin of infection (Step 5). Furthermore, points are awarded to users for locating the appropriate position of each clue, which motivates players towards the infection chain isolation step (Step 6), as the player requires to isolate defective points before locating appropriate positions for the clues. Finally, the player is required to locate apt positions in a program for all the clues, thus correcting the defective program (Step 7). G4D aims to instigate the player's to follow these steps, that finally helps them in debugging the program and consequently qualifying various levels in the game, moving towards fixing the broken spaceship, in accordance with storyline of the game.

G4D also tracks time taken by player in the game and prompts the player with appropriate dialogues, that are aimed to provide directions to the player towards the next steps to be taken.

The existing literature has also specified certain criteria that are expected from serious games, to ensure that the advantages of learning through games is preserved. These criteria include interdisciplinary and social learning, facilitating learning out-comes with in game assessments, use of feedback systems that drive users towards success by ensuring transparent approaches" and so on (De Freitas, 2018). G4D uses the broken spaceship metaphor in the game to adapt the interdisciplinary learning aspect, rather than having pure programming. It also uses the background of an alien planet to retain players' interest and motivation. The players are notified about cor-rectness of identification of locations during the game, based on which, players could redo the identification of erroneous locations, thus facilitating learning through in-game assessment. The feedback is provided as dialogues to help player progress in the game based on time spent by the player, and points are given to the player on correct debugging to encourage the player. In case of incorrect identification, the player is redirected to the crashed spaceship to redo debugging of the code, with clues retained, ensuring constructive feedback mechanism.

Development and user scenario

Development

We have developed G4D, a treasure hunt game to learn debugging using Unity 3D game engine. Figure 2 shows the approach followed to develop the game.

We have used 3D Kit to define player features, model actions of player, import structures, layouts and functionalities of these structures. Scripts to define controls of each object in the game such as player, alien enemy animal and other structures have been written in CSharp programming language. Faulty Code snippets, with their corresponding correct code snippets, written in C language are defined. Clue sets required to debug these code snippets are also defined.

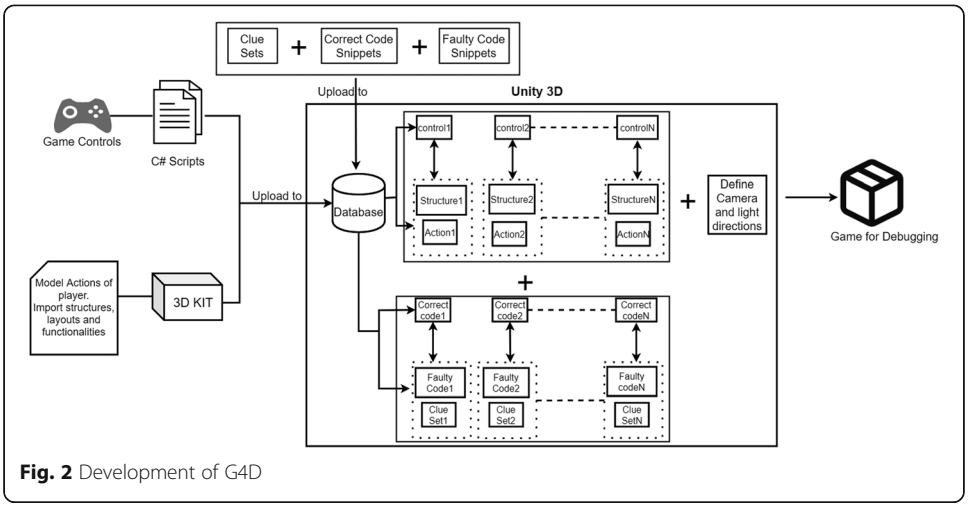

Fig. 2 Development of G4D

Clue sets that can help in debugging the code snippets, faulty code snippets to be debugged and correct code snippets are then uploaded onto the database. Functionalities and structures imported from 3D Kit and Scripts written in CSharp programming language are then uploaded to the database. Each structure is associated with predefined actions that can be performed. Each structure-action pair in the game is also associated with corresponding control features inside the database. Each faulty code snippet is associated with a clue set that is useful in debugging the snippet. This pair of faulty code snippet and clue set are then linked with the correct code snippet in the database. The camera angle and direction, and the light direction are defined with the help of Unity 3D. Unity 3D helps in integrating the scripts containing controls, written in CSharp programming language, with defined structures and actions.

A global timer is set to track time taken by player in the game at each point. If the player is observed to show minimum progress during a threshold time period, a dialogue is prompted to the player indicating the immediate next step to be taken by the player in the game.

User scenario

Consider Moksha, a novice programmer, who is enthusiastic in learning debugging aspect of programming. She considers to play G4D, to learn and practice debugging.

She starts to play the game after knowing the story-line, and takes up the character of Veda in the game. She lands on the screen of level 1, as shown in Fig. 3. [A] of Fig. 3 displays the number of lives, the player is left with. [B] of Fig. 3 is the player character, Veda. [C] of Fig. 3, refers to background, which is the alien planet. [D] of Fig. 3 is the crashed spaceship component that is to be repaired in this level. When Moksha reaches the spaceship, she is shown a code snippet that is to be debugged, as in Fig. 4. Moksha then moves forward in the game, to find boxes, as shown in [B] of Fig. 5 that contain clues to debug the given code snippet. She also comes across alien enemy animals that are capable of attacking, which results in degradation of the player's life. [A] of Fig. 6 depicts Veda killing the animal using the weapon provided. [B] of Fig. 6 shows the animal attacking Veda and [C] of Fig. 6 shows the loss of Veda's life as a consequence.

Fig. 3 Landing Scene of G4D

Once, all the clues and key to open the alien city doors are found, Moksha is prompted to debug the given code, after which the key gets activated, the door can be opened and Moksha can go to the next level. Key is depicted by [A] in Fig. 7 and the door is depicted by [B] in Fig. 7.

In course of the game, when Moksha finds a box, she breaks it with the provided weapon, to find clues that help in debugging the code snippet. She finds the first clue to debug, which is displayed as shown in [A] of Scene 1 of Fig. 8. When Moksha

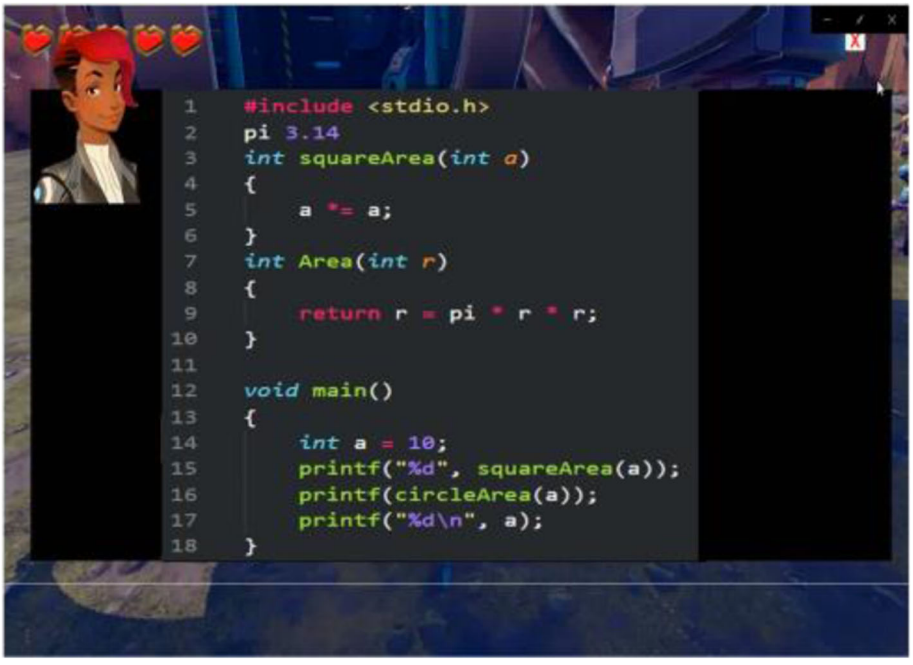

Fig. 4 Code Snippet to be debugged

Fig. 5 Box containing clues

reaches the crashed spaceship, the code snippet to be debugged, with clues collected till then is displayed, as shown in [A] of Scene 2 of Fig. 8. Moksha, then moves forward to find the other clues as shown in Fig. 9. She finds clue *return*, as shown in Scene 1 of Fig. 9, in a box, which is displayed along with the code snippet to be debugged, when Veda visits the crashed spaceship (Scene 2 of Fig. 9). She finds the other two clues *circle* and *%d* (Scene 3 and 5 of Fig. 9) and the same is then displayed alongside the code snippet, as shown in Scene 4 and Scene 6 of Fig. 9.

Once all the clues are found, Moksha goes back to crashed component of the spaceship to debug the code that can repair this component. She first identifies the location of clue in the code snippet by selecting the position on a specific line with mouse pointer. She is then displayed a blank as shown in Fig. 10, in which she can fill in the clues. She selects the lines and positions in which clues are to be inserted as shown in Scene 1 of Fig. 11, but as the selected line is incorrect (pointed by [A] of Scene 1 in Fig. 11), she looses some points in her score and is navigated back to the crashed spaceship with a dialogue as displayed in Scene 2 of Fig. 11. She selects the position of error, again, as shown in [A] of Scene 3 of Fig. 11. But, as one of these selected positions is

Fig. 6 Scenes depicting killing enemy animal and animal attacking Veda

Fig. 7 Alien city door and key to open it

incorrect (pointed by [A] in Scene 3 of Fig. 11), she looses some points in her score and is navigated back to the crashed spaceship with a prompt as displayed in Scene 4 of Fig. 11. She attempts again, identifies the right positions, and fills in the clues based on the numbers associated with each clue as shown in Scene 5 of Fig. 11. Since, these positions are incorrect, Veda's life gets deteriorated and she is navigated back to the crashed part of the spaceship as shown in Scene 6 of Fig. 11. Moksha then selects the appropriate location and enters correct positions (numbers associated with clues) after revisiting the code snippet as shown in Fig. 12. Since the selected are positions are correct, the key found earlier gets activated, and Veda can open the city gates, cross the city and move forward to next levels.

Evaluation

Instruments for evaluation

G4D has been developed as a treasure hunt based 3D game, with an aim to support and motivate users to learn and practice debugging of code snippets. Hence, it has been

Fig. 8 First clue to debug

Fig. 9 Finding clues

predominantly evaluated to assess its usability, correctness and player experience. Several existing games that have been designed with similar aim of supporting teaching and learning, have been evaluated based on similar user experience such as system interaction, learning content, terminology, appearance, user satisfaction, likeability and so on (Lytridis & Tsinakos, 2018; Tlili, Essalmi, & Jemni, 2016). Models such as EGameFlow (Fu, Su, & Yu, 2009), MEEGA and MEEGA+ (Petri, von Wangenheim, & Borgatto, 2016) have also been used to extensively evaluate educational games in various aspects of user experience. Considering the current scope of G4D, we observed that an adapted version of MEEGA+ would fit well for evaluating G4D (Petri et al., 2016). MEEGA+ has been introduced to evaluate quality of games for computing education by Petri et al. and deals broadly with two quality factors - usability and player experience. Thirteen dimensions for these two quality factors have been defined in the model as presented below. We have also included correctness criteria to be evaluated with respect to the clues provided in the game along with Player Experience and Usability.

Fig. 10 Selecting error position on a line

Fig. 11 Incorrect Debugging Scenarios

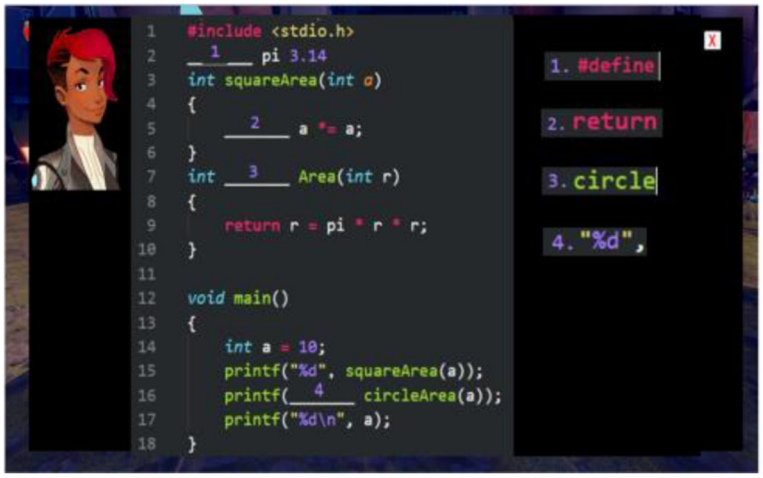

Fig. 12 Debugging code correctly

Player Experience -
Focused Attention
(FA) Fun (F)
Challenge (Ch)
Social interaction (SI)
Condence (Co)
Relevance (R)
Satisfaction (S)
Perceived Learning (PL)
User Error Protection (UEP)
Usability -

Learnability (L) Operability (O)
Asthetics (A)
Accessibility (Acc)
Correctness (C)

G4D has been designed as an o ine, single player 3D game, that can be played on a personal laptop or a desktop. Also, G4D has not been experimented in any course, but has been developed independent of the course, to address a wider range of audience. Considering these factors in the scope of G4D, we have omitted assessment of two dimensions - Social interaction (SI) and Perceived Learning (PL) as they deal with multi-player environments and introduction of the game in a course. Also, we have excluded User Error Protection (UEP) dimension as it does not come under the scope of G4D. Existing questionnaires related to these dimensions from various sources have been grouped and customized with an aim to improve the original MEEGA questionnaire in MEEGA+. The current version of G4D is a prototype version, in its initial stages, with only one level. Hence, we have excluded few questions that dealt with progress and experience in the game across multiple levels.

Participants

Thus, to evaluate G4D, we considered 20 volunteers in the age group of 17–25 years, 14 male and 6 female, with different programming backgrounds. This selection of volunteers with multiple programming languages is to obtain feedback about both usefulness and correctness of clues displayed to the users.

Procedure for evaluation

We performed a study with a 5-point Likert scale based questionnaire of 27 questions, of which, four questions dealt with demographics of the volunteers and the other 23 dealt with the dimensions mentioned above and correctness of clues. The volunteers were requested to play G4D and provide their reviews and suggestions by answering the questionnaire. Since we did not consider all the dimensions of MEEGA+ questionnaire, we have manually evaluated the survey results, following the evaluation methodology of MEEGA+. We have hence calculated the quality of G4D based on Item Response Theory (IRT) and have considered the Cronbach Alpha values mentioned in MEEGA+ for the two quality factors.

Results

The demographics of volunteers represented in Table 1 infers that about 50% of the volunteers have zero years of prior programming experience. Also, about 55% of the volunteers play games either very often or often. The results of user survey are presented in Table 2, in terms of mean and standard deviation. We have then sorted

Table 1 Demographics of volunteers

	Frequency	Percent (%)	Cummulative (%)
Age			
17	3	15	15
18	4	20	35
19	2	10	45
20	6	30	75
21–25	5	25	100
Gender			
Male	14	70	70
Female	6	30	100
Prior Programming Experiencein years			
0	10	50	50
1	2	10	60
2–3	6	30	90
4–5	2	10	100
Frequency of playing games			
Very Often	5	25	25
Often	6	30	55
Neutral	7	35	90
Sometimes	1	5	95
Never	1	5	100

Table 2 Adapted MEEGA+ questionnaire

Variable	Question	Mean	SD
A	G4D design is attractive (interface, graphics, cards, boards, etc.)	4	0.648
	(1 = strongly disagree, 5 = strongly agree)		
L	I needed to learn a few things before I could play G4D.	3.45	0.887
	(1 = strongly disagree, 5 = strongly agree)		
L	Learning to play G4D was easy for me.	4.2	0.767
	(1 = strongly disagree, 5 = strongly agree)		
L	I think that most people would learn to play G4D very quickly.	4.15	0.745
	(1 = strongly disagree, 5 = strongly agree)		
O	I think that G4D is easy to play	4.1	0.718
	(1 = strongly disagree, 5 = strongly agree)		
O	G4D rules are clear and easy to understand.	3.95	0.759
	(1 = strongly disagree, 5 = strongly agree)		
Acc	The fonts (size and style) used in G4D are easy to read.	3.7	0.923
	(1 = strongly disagree, 5 = strongly agree)		
Co	When I first looked at G4D, I had the impression that it would be easy for me	4.15	0.875
	(1 = strongly disagree, 5 = strongly agree)		
Ch	G4D is appropriately challenging for me.	3.5	1.27
	(1 = strongly disagree, 5 = strongly agree)		
Ch	G4D provides new challenges (o ers new obstacles, situations or variations)		
	at an appropriate pace. (1 = strongly disagree, 5 = strongly agree)	3.85	0.745
Ch	G4D does not become monotonous as it progresses (repetitive or boring tasks).	3.75	0.966
	(1 = strongly disagree, 5 = strongly agree)		
S	Completing G4D tasks gave me a satisfying feeling of accomplishment.	3.8	0.951
	(1 = strongly disagree, 5 = strongly agree)		
S	It is due to my personal e ort that I managed to advance in G4D.	3.35	1.089
	(1 = strongly disagree, 5 = strongly agree)		
S	I feel satisfied with the things that I learned from G4D	3.85	1.039
	(1 = strongly disagree, 5 = strongly agree)		
S	I would recommend G4D to my colleagues.	3.9	1.071
	(1 = strongly disagree, 5 = strongly agree)		
F	I had fun with G4D.	3.75	0.91
	(1 = strongly disagree, 5 = strongly agree)		
F	Something happened during the game (game elements,		
	competition, etc.) which made me smile. (1 = strongly disagree, 5 = strongly agree)	3.85	0.988
FA	There was something interesting at the beginning		
	of the game that captured my attention.	3.85	0.875
	(1 = strongly disagree, 5 = strongly agree)		
FA	I was so involved in my gaming task that I lost track of time.	3.4	1.046
	(1 = strongly disagree, 5 = strongly agree)		
FA	I forgot about my immediate surroundings while playing G4D.	3.5	0.827

Table 2 Adapted MEEGA+ questionnaire *(Continued)*

Variable	Question	Mean	SD
	(1 = strongly disagree, 5 = strongly agree)		
R	The game contents are relevant to my interests.	3.75	1.118
	(1 = strongly disagree, 5 = strongly agree)		
C	The clues given in G4D are correct with respect to the code snippet provided.	4.25	0.716
	(1 = strongly disagree, 5 = strongly agree)		

out the means of all Player Experience related questions and those of Usability related questions. According to MEEGA+, the Cronbach Alpha value for Player Experience based dimensions is 0.856 and that of Usability based dimensions is 0.930. Applying Cronbach Alpha values to means of the dimensions and then normalizing the value by multiplying it with 10, resulted in quality score of 63.07. According to MEEGA+, games with quality score value less than 42.5 are considered to be of low quality, between 42.5 to 65 are considered to be of good quality and those with value greater than 65 are considered to be of excellent quality. Thus, we observe that, based on the survey results, G4D has a good quality level. Based on this result, one of the description of G4D quality is that the game frequently presents moments of fun among the users and is observed to be relevant to users' interests. Also, it has been stated that games with Good Quality level provide moderate attention to players, however, they do not make users forget about their surroundings. The mean and standard deviation of the questionnaire also indicates that G4D provides apt clues in most scenarios. The mean and standard deviation plots for the dimensions are represented in Fig. 13.

Few participants have also suggested to add more complex clues. Participants also suggested and commented:

"A map could be arranged so that we can identify the boxes in the map".

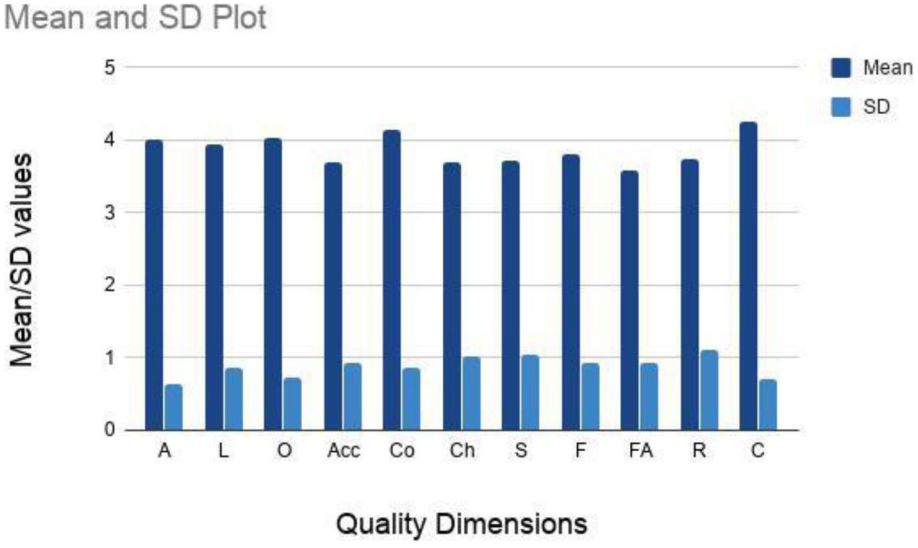

Fig. 13 Result of Questionnaire in terms of Mean and Standard Deviation of MEEGA+ quality dimensions

"Adding more number of levels could be more useful".

Discussion and limitations

The game uses the storyline of a spaceship crashed on an alien planet, and the player has to find objects of the spaceship and eventually fix it through a series of levels. Once, the spaceship is fixed, the player will succeed in leaving the alien planet. We designed the objects to be fixed and fixing the objects to be analogous to faulty code snippets and debugging the code snippets respectively. It might thus motivate users to play the game and help them to practice debugging. The current version of G4D, being a basic prototype version, comprises of only one level, with one faulty code snippet to be debugged. While we intended to build the game to support programmers novice to various programming languages, we have limited our scope only to C language.

Code snippets presented at each level are static and written in C language alone. Debugged code snippets are evaluated by comparing with pre-loaded code snippets, restricting the players to make changes in code and run these snippets. However, this being a treasure hunt based game, players are expected to only use pre-fixed code snippets. In the future versions, a compiler could be added to support dynamic code snippets as well. Programming constructs and concepts used in code snippets are narrow and limited.

Though the idea of G4D is to support learning to debug through a treasure hunt based game, the current version displays clues as text in the dialogues, which is not tightly coupled to the story of the game and could also induce cognitive load on the users. The clues in the game could be presented in a better way through game elements, that could improve the usability and player experience of the game.

The scaffolding techniques adapted in the design of prototype version of G4D do not consider two of the 10 identified characteristics - Teacher Support and Internalization. We plan to integrate other characteristics in the future versions of the game. Also, other techniques that support pedagogy of learning programming could be explored to identify the best fit technique.

Currently, the game has been evaluated based on a modified MEEGA+ model that omitted three dimensions Social Interaction (SI), User Error Protection (UEP) and Perceived Learning (PL), considering the present scope of G4D and the evaluation environment of the game. Updating the scope of the game and evaluation environment to include these three dimensions could provide better idea on usability and player experience of G4D. Other evaluation methods that include both quantitative and qualitative approaches such as assessing performance of participants before and after using G4D or assessing performance of two teams, where one team uses G4D and the other team does not use G4D before attempting to debug a program could also be explored to arrive at better results.

Conclusion and future work

In this paper, we presented the prototype version of a treasure hunt game that can be played on personal computers of users. Debugging is considered as one of the important tasks of programming (Beller et al., 2018; Ko & Myers, 2005). Identifying the location of bug and resolving it is considered as an important task of debugging (Zeller, 2009). The proposed game could help novice programmers to learn how to debug

various code snippets written in C language. It displays the code snippet in C language and thus also makes users wary of debugging a code syntactically, which could be helpful in real-time programming. Also, G4D provides directions to users as dialogues when required, by intelligently tracking the time spent by users at a particular point in the game. The user survey conducted based on MEEGA+ model with 20 volunteers revealed that G4D has a good quality level. Also, mean value of 3.9 indicates that a majority of the volunteers were willing to recommend G4D to their colleagues.

The game can further be extended to support various other programming languages such as java, python and so on with minor changes in code snippets being stored. We plan to increase the number of levels in the game and the complexity of code snippets provided at each level. We also plan to extend the game to support multiplayer and online modes. Increasing the number of dialogues and extending game plot could also increase the motivation to play the game. We plan to extend the game to address complex concepts of programming such as recursion, pointers and also to provide code snippets that deal with the basic concepts in depth. We further plan to increase the number of code snippets to be solved in each level and to introduce levels which have code snippets that deal with the amalgamation of many basic concepts. In future versions, we intend to add a map that displays layout of the alien city as suggested by participants in the survey.

Considering the loosely coupled nature of the process of debugging with G4D storyline, we plan to improve G4D to provide clues for debugging as elements in the game. Clues could be provided as parts of the broken spaceship such as screws, parts of the engine, landing gear, reaction control thrusters, speed brake and so on, with clues related to programming, embossed on these parts. Also, the code snippet to be debugged could be depicted equivalent to repairing the broken spaceship, with lines of code embossed on various parts of the space ship. The player would then be required to identify parts of the spaceship that failed, locate specific points of failure in each part, map identified points of failure to obtained spaceship parts and then place the parts in appropriate positions, that could repair the spaceship. This could reduce the cognitive load that would be induced on players due to the current loosely coupled nature of the game, which could further motivate the users towards playing the game and consequently learning debugging.

Also, G4D could be introduced as an exercise to a course in the first year of under graduation, that involves introduction to programming for evaluation of the game. This change in evaluation environment could facilitate in including Perceived Learning (PL) dimension of the MEEGA+ model. Evaluation could also be enhanced to include pre and post tests, before and after using G4D, that would be capable of providing quantitative results based on the students' performance. We also plan to explore qualitative and quantitative evaluation mechanisms in the future.

Acknowledgements
We thank our under graduate student *Deep Ghadiyali* for helping us in developing the G4D game.

Authors' contributions
Both the authors have equally contributed to the manuscript. While *AV* has contributed more in terms of implementation of the idea, SC has contributed more in terms of the idea. The author(s) read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The game and the results of evaluation are available from the corresponding author on reasonable request.

Competing interests
There are no competing interests.

Received: 4 May 2020 Accepted: 6 August 2020
Published online: 26 August 2020

References

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. *Acm Sigcse Bulletin*, *37*, 84–88.
Barnes, T., Powell, E., Chan, A., & Lipford, H. (2008). Game2learn: Improving the motivation of cs1 students. In *Proceedings of the 3rd international conference on game development in computer science education*, (pp. 1–5).
Beller, M., Spruit, N., Spinellis, D., & Zaidman, A. (2018). On the dichotomy of debugging behavior among programmers. In *2018 ieee/acm 40th international conference on software engineering (icse)*, (pp. 572–583).
Chiu, C.-F., & Huang, H.-Y. (2015). Guided debugging practices of game based programming for novice programmers. *International Journal of Information and Education Technology*, *5*(5), 343.
Chmiel, R., & Loui, M. C. (2004). Debugging: From novice to expert. *Acm Sigcse Bulletin*, *36*, 17–21.
Christensen, R. (2002). Effects of technology integration education on the attitudes of teachers and students. *Journal of Research on Technology in Education*, *34*(4), 411–433.
Cook, D. A., Hatala, R., Brydges, R., Zendejas, B., Szostek, J. H., Wang, A. T., & Hamstra, S. J. (2011). Technology-enhanced simulation for health professions education: A systematic review and meta-analysis. *Jama*, *306*(9), 978–988.
Cooper, S., Dann, W., & Pausch, R. (2000). Alice: A 3d tool for introductory pro-gramming concepts. *Journal of Computing Sciences in Colleges*, *15*(5), 107–116.
De Freitas, S. (2018). Are games effective learning tools? A review of educational games. *Journal of Educational Technology & Society*, *21*(2), 74–84.
Denny, P., Luxton-Reilly, A., & Carpenter, D. (2014). Enhancing syntax error mes-sages appears ineffectual. In *Proceedings of the 2014 conference on innovation & technology in computer science education*, (pp. 273–278).
Eagle, M., & Barnes, T. (2009). Experimental evaluation of an educational game for improved learning in introductory computing. *ACM SIGCSE Bulletin*, *41*(1), 321–325.
Etheredge, J. (2004). Cmerun: Program logic debugging courseware for cs1/cs2 students. *Acm Sigcse Bulletin*, *36*(1), 22–25.
Fu, F.-L., Su, R.-C., & Yu, S.-C. (2009). Egameflow: A scale to measure learners' enjoyment of e-learning games. *Computers in Education*, *52*(1), 101–112.
Gould, J. D. (1975). Some psychological evidence on how people debug computer programs. *International Journal of Man-Machine Studies*, *7*(2), 151–182.
Guzdial, M. (1994). Software realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments*, *4*(1), 001–044.
Jantan, S. R., & Aljunid, S. A. (2012). An experimental evaluation of scaffolded educational games design for programming. In *2012 ieee conference on open systems*, (pp. 1–6).
Jones, J. A., Harrold, M. J., & Stasko, J. (2002). Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on software engineering*. Icse 2002, (pp. 467–477).
Kelleher, C., Pausch, R., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the sigchi conference on human factors in computing systems*, (pp. 1455–1464).
Ko, A. J., & Myers, B. A. (2004). Designing the whyline: A debugging interface for asking questions about program behavior. In *Proceedings of the sigchi conference on human factors in computing systems*, (pp. 151–158).
Ko, A. J., & Myers, B. A. (2005). A framework and methodology for studying the causes of software errors in programming systems. *Journal of Visual Languages and Computing*, *16*(1-2), 41–84.
Kolling, M. (2010). The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 14.
Lapidot, T., & Hazzan, O. (2005). Song debugging: Merging content and pedagogy in computer science education. *ACM SIGC SE Bulletin*, *37*(4), 79–83.
Lee, M. J. (2014). Gidget: An online debugging game for learning and engagement in computing education. In *2014 ieee symposium on visual languages and human-centric computing (vl/hcc)*, (pp. 193–194).
Li, X., Yi, W., Chi, H.-L., Wang, X., & Chan, A. P. (2018). A critical review of virtual and augmented reality (vr/ar) applications in construction safety. *Automation in Construction*, *86*, 150–162.
Luxton-Reilly, A., McMillan, E., Stevenson, E., Tempero, E., & Denny, P. (2018). Ladebug: An online tool to help novice programmers improve their debugging skills. In *Proceedings of the 23rd annual acm conference on innovation and technology in computer science education*, (pp. 159–164).
Lytridis, C., & Tsinakos, A. (2018). Evaluation of the ARtutor augmented reality educational platform in tertiary education. *Smart Learning Environments*, *5*(1), 6.
McCall, D., & Kolling, M. (2014). Meaningful categorisation of novice programmer errors. In *2014 ieee frontiers in education conference (e) proceedings*, (pp. 1–8).
McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67–92.
Miljanovic, M. A., & Bradbury, J. S. (2017). Robobug: A serious game for learning debugging techniques. In *Proceedings of the 2017 acm conference on international computing education research*, (pp. 93–100).
Nguyen, T. (2015). The effectiveness of online learning: Beyond no significant dif-ference and future horizons. *MERLOT Journal of Online Learning and Teaching*, *11*(2), 309–319.
Petri, G., von Wangenheim, C. G., & Borgatto, A. F. (2016). Meega+: An evolution of a model for the evaluation of educational games. *INCoD/GQS*, *3*, 1–40.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (n.d.). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.
Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, *12*(2), 257–285.
Teng, C.-H., Chen, J.-Y., & Chen, Z.-H. (2018). Impact of augmented reality on programming language learning: Efficiency and perception. *Journal of Educational Computing Research*, *56*(2), 254–271.
Tlili, A., Essalmi, F., & Jemni, M. (2016). Improving learning computer architecture through an educational mobile game. *Smart Learning Environments*, *3*(1), 7.
Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, *17*(2), 89–100.
Zeller, A. (1999). Yesterday, my program worked. Today, it does not. Why? *ACM Sigsoft Software Engineering Notes*, *24*, 253–267.
Zeller, A. (2009). *Why programs fail: A guide to systematic debugging*, (p. 20). Elsevier.

Publisher's Note