Enhancing Student Learning and Engagement with Object-Oriented Block-Based Programming Tool

Xavier Jia Le Chin[1], Chee Kiat Seow[2], Yiyu Cai[3], Yongqing Zhu[4], Min Wang[5], and Qi Cao[2,*]

[1]School of Computing Science, University of Glasgow, Singapore
[2]School of Computing Science, University of Glasgow, Glasgow, Scotland, UK
[3]School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore
[4]School of Science and Technology, Singapore University of Social Sciences, Singapore
[5]School of Information Science and Technology, Guangdong University of Foreign Studies, Guangzhou, China
Email: 2002058@sit.singaporetech.edu.sg (X.J.L.C.); CheeKiat.Seow@glasgow.ac.uk (C.K.S.); myycai@ntu.edu.sg (Y.C.); yqzhu@suss.edu.sg (Y.Z.); wangmin@gdufs.edu.cn (M.W.); qi.cao@glasgow.ac.uk (Q.C.)
*Corresponding author
Manuscript received February 18, 2024; revised March 8, 2024; accepted April 8, 2024; published July 9, 2024

***Abstract*—Technologies have been adopted and integrated into various aspects of life, including education and learning. Gamified learning is effective to promote student learning and motivations. Object-Oriented Programming (OOP) is a fundamental topic taught to university students, and in certain countries, it is introduced as an enrichment course for younger students. It is beneficial to develop young students on the interests in software programming. Equipping younger students with essential programming skills will be helpful and better prepare them to excel in the digital age in the future. This research leverages the benefits of gamification to enhance student engagement by developing an educational game for teaching OOP. The developed educational visual programming tool simplifies complex OOP concepts and engagingly presents them through the medium of a game. It provides an alternative learning approach to visualizing and understanding OOP. The study yielded positive results, with improved test scores and a more consistent performance. It highlights the effectiveness of this approach in teaching OOP. The positive feedback from students regarding their enjoyment of gameplay also shows the potential of gamified learning in enhancing engagement.**

***Keywords*—object-oriented programming, gamified learning, block-based programming, interactive game building**

I. Introduction

It is useful to equip the younger generation with technical skills, including coding and technological literacy. Numerous benefits were observed in improving the technological fluency of children through software programming [1], e.g., algorithmic thinking or problem-solving skills. However, further research is required to incorporate acceptable educational technologies into teaching software programming. As such, there is a reasonable demand for a software programming tool suitable for children in the educational market.

There were initiatives in some countries to provide enrichment training in software programming fundamentals to adolescents in schools. For example, young students in Singapore were taught how to do procedural programming or block-based programming, which takes input data, processes it, and produces the corresponding output data. These initiatives promoted future careers in the Infocomm Technologies (ICT) industry to students. This enables individuals to adapt to the ever-evolving landscape of programming languages [2]. However, these initiatives usually do not cover popular paradigms like Object-Oriented Programming (OOP). OOP focuses on objects and their behaviors which works in a vastly different way. It allows software programmers to create various classes and objects which model the behaviors of data. It accentuates that a supplementary software tool on OOP learning could benefit such initiatives for nurturing the interests of young students in software programming.

It is crucial to adopt a pragmatic approach that focuses on the fundamental principles of programming. While logical reasoning, problem-solving, and computational thinking are already fostered through mainstream curriculum subjects [3], rudimentary programming concepts can be introduced to supplement the existing syllabus. However, the conventional methods of learning software programming often fall short in engaging students and promoting comprehension, which is frequently characterized as dull and repetitive. Lectures on different programming languages, syntax, and semantics are followed by laboratory classes where students simply replicate the steps provided by lecturers. This approach tends to prioritize memorization over comprehension, limiting the ability of students to grasp the subject beyond a superficial level of recalling information [4]. While explanation and logic are important aspects of learning, they do not necessarily facilitate a deep understanding of the topic. A higher level of abstraction is crucial for comprehending and applying programming concepts effectively [5]. Merely memorizing the concepts without truly understanding them can be detrimental to learning experiences.

Moreover, the current laboratory classes with one lecturer guiding approximately 50 students, may inadvertently contribute to a cycle of learned helplessness among students. This cycle occurs when students struggle to bridge the gap in understanding due to limited guidance and incompatible learning styles. Learned helplessness can be described as a state of powerlessness resulting from persistent failures, leading to apathy towards learning, particularly observed in the study of mathematics [6]. Moreover, learned helplessness in mathematics could signify a similar experience with software programming, as both involve problem-solving, logical thinking, and the use of algorithms to solve complex problems. Mathematics and programming share a logical foundation and are complementary [7]. The incorporation of software programming into mathematics education elevated the performance and interest of students [8, 9]. Therefore, a visual representation of the algorithms and programming in education could stimulate the interests of students, which

potentially reduces learned helplessness.

This research was driven by the significance and relevance of OOP in modern programming paradigms, as OOP holds a prominent position among the most widely used programming languages globally [10]. This research aims to address the aforementioned challenges by developing a novel gamified visual OOP tool to teach OOP concepts [5], prioritizing comprehension over memorization. By incorporating game design elements into a non-game context [11], the visual programming tool aims to effectively engage students and improve their understanding of abstract programming topics [12]. The main contributions of this research are as follows:

- A gamified visual OOP tool simplifies abstract OOP topics by providing interactive game elements that facilitate a hands-on approach to learning. It aims to offer students an engaging learning experience through experimentation, harnessing the immediate feedback loop present in games, which is lacking in conventional classroom learning.
- A level progression system, comprising of a tutorial stage, and various game objectives are to be implemented in the visual OOP tool to maintain student retention and motivation. It contributes to the development of innovative educational tools that address the issue of learned helplessness and promote effective programming education in the OOP course.

The organization of the remaining parts of this paper is as follows. Section II introduces the related works. Section III presents the proposed methodology. Section IV discusses the case study. Section V concludes the paper with future works.

II. Literature Review

Gamification is the application of game elements in non-gaming systems to make learning more engaging and interesting for students [13]. Gamification addresses the issues persistent in traditional education, namely the lack of real-time feedback and visualization. Serious games that incorporate learning activities are a form of gamification that is generally used in education [14, 15]. They serve as potent tools in persuasive technology to influence user behavior in alignment with intended core values.

OOP consists of complex concepts that are difficult for beginners due to the lack of real-time visual feedback, which may lead to learned helplessness [6]. Gamification is considered an effective methodology in teaching programming since it encourages creativity and critical thinking [16]. An example of such is the utilization of a serious game to break down complex cybersecurity concepts when teaching students [17], where experiments and results are presented from the perspectives of an educator. Incorporating basic gamification elements into a traditional introductory programming course demonstrates an increase in class engagement and attendance rate [18], which provides valuable insight into the preferred game mechanics of students and potential pitfalls. A game-based learning approach was also adopted to teach university students sorting algorithms of the Data Structures course [19], using a web portal to provide better accessibility to students.

However, learning games are not a replacement for teachers or existing school curricula but a supplementary product. Teachers play a critical role in ensuring that games align with the learning objectives and needs of students. As such, the educational game must be simple, possess clear rules, and have a defined objective to effectively fulfil its role as a supplementary tool for educators in their classrooms.

The plausibility of gamification is explored as a useful technique for learning unfamiliar concepts [20]. Game elements such as points, progress bars, leaderboards, and badges encourage students to engage with games by behaving as a form of conditional reinforcement, through mini-challenges and instant feedback systems. Instant feedback positively influences the learning and achievement of students, where immediate feedback helps students understand the subject better [21]. Although the efficacy of visual feedback is discussed in guiding and motivating Computing Science (CS) students to achieve success [22], it also possibly revealed that visual feedback without interactive functions only benefitted highly driven students.

A serious game was introduced in the form of a virtual museum to solve object-oriented problems [23]. A review was conducted on game designs where most games were for teaching procedural programming [24]. Another review discussed various serious games to teach programming courses [25]. But no games for teaching OOP were included, thus highlighting a niche in the market.

As a type of visual programming language, Block-Based Programming Languages (BBPL) enable users to visually drag and drop blocks to create executable programs, rather than writing software code using a traditional textual language. Various BBPL were analyzed based on the metrics of usability, teaching materials, programming language capabilities, syntaxes, paradigms, etc. [26]. Block-based serious games were introduced to teach introductory programming courses in Higher Education [27], with Scratch being the most popular. However, Scratch is classified as an object-based language, not an object-oriented language because it lacks classes and inheritance [28]. Therefore, it is observed that most BBPL lack support for OOP.

Support for inheritance is necessary for a language to become object-oriented, as such, most BBPL are not suited for teaching OOP. However, teaching OOP using a modified mainstream BBPL called Blockly was experimented on [29], which introduces code blocks relating to OOP concepts, omitting language-based syntax. The usage of code blocks enables students to focus on OOP principles without worrying about the specifics of a textual language, which has streamlined the learning experience. Examining another BBPL implementation named Alice, it could be observed that students achieved higher scores and had a preference towards OOP, but still misunderstood certain OOP concepts [30].

III. Proposed Methodology

From the literature review, it is observed that gamification is complex and requires a more nuanced approach in its implementation. The impact of gamification on students' learning and engagement depends on a variety of factors:

- Game Design: The game should be brief, enjoyable, and easy to play with clear learning objectives.
- Game Elements: The game should incorporate elements, such as points, badges, levels, etc.
- Interface and Controls: The game should utilize a simple

interface and control scheme.

- Teacher Involvement: The game should be designed as a supplementary tool in education. The teacher's role in guiding the students is still significant.
- Platform: The game should consider its accessibility to students and educators alike.

OOP is a popular programming paradigm, to equip students with the necessary knowledge for tomorrow's society. Hence, a carefully designed educational game that places emphasis on OOP concepts rather than programming syntaxes, could be introduced as a supplementary product to existing introductory programming courses. Moreover, the proposed methodology fulfils a niche, as the examination of existing products indicates a gap in the market.

It motivates the necessity to propose a methodology to create an OOP tool, with the inclusion of a block-based programming interface. The proposed methodology allows students to interact with entities within the game by utilizing a custom-made OOP-centric BBPL. It accounts for an immediate visual feedback system to promote active participation and attention retention as students can readily observe the output of their code and modify it accordingly.

Tower Defence (TD) has been selected as the genre for game creation due to its reputation for simplicity and accessibility. It incorporates a drag-and-drop control scheme, making it easy for players to engage with the game. This intuitive control system is advantageous as it closely resembles those found in BBPL, thereby reducing confusion to a minimum. A sample screenshot of the TD game is shown in Fig. 1, where players must prevent enemies from traversing from points A to B. By strategically placing turrets with offensive capabilities along the enemy path, players can effectively defend their stronghold. This straightforward gameplay, coupled with the drag and drop controls, ensures that the game is enjoyed by players of all skill levels.

Fig. 1. Sample screenshot of the Tower Defence game.

A. Overall Software Architecture

Unity Engine and C Sharp (C#) programming language were used in the game's development. Unity facilitates the distribution of games across various platforms, including a WebGL application, enabling direct play on web browsers without the need for installing any software on players' computers, thereby enhancing the game's accessibility.

The overall software architecture is shown in Fig. 2. The crux of the game is composed of three core components: **game map**, **enemies**, and **turrets**. They work in tandem using a Singleton Design Pattern, creating an engaging gameplay experience. The primary target audience for the proposed game is students. Two user stories are formulated and outlined from both educational and gaming perspectives:

1) As a student who is new to OOP, I want the game to be simple yet engaging and interactive, so that I can learn the concepts in a memorable way. This means that the game should have clear instructions, interesting graphics, and be easy to understand.
2) As a student who enjoys challenges while learning, I want the game to have different difficulty levels, so that I can progress at my own pace and feel a sense of accomplishment as I learn new concepts. It means that the game should have difficulty levels that gradually increase and provide measurable forms of success.

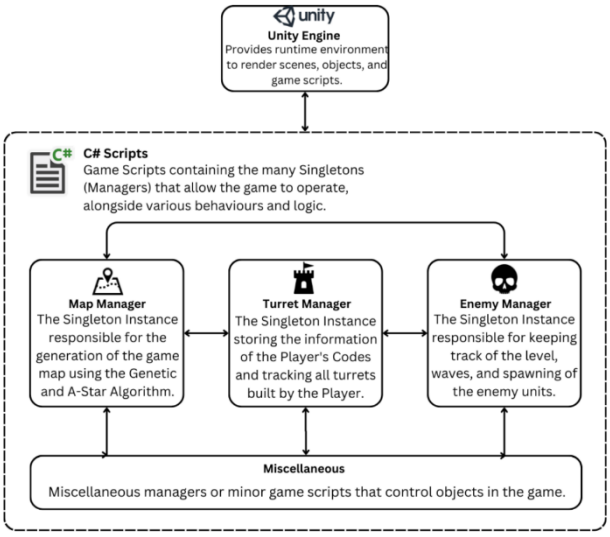

Fig. 2. High-level software architecture.

Game Map: As a crucial component of the OOP learning game, the map includes different obstacles and venues for the construction of a turret. Players must navigate around the obstacles, as it blocks the turrets' line of sight and provides cover for the enemies. These factors affect gameplay by providing different strategic options for players to take advantage of. Each map featured in each stage of the game is randomly generated in a three-step process as follows:

- Genetic Algorithm (GA) is used to iteratively modify a population of individual maps, selecting maps randomly from the population to be parents and produce the next generation of offsprings. Over successive generations, the population evolves toward an optimal map.
- Knight Piece Obstacle Generation is performed iteratively for each map derived from GA. As the grid-like design of the map draws similarity to a chessboard, obstacles were designed to be generated by selecting random points on the grid and calculating all the possible L-shaped moves that a knight piece can perform. Each move is marked as an obstacle on the game map.
- A start point and an end point will be placed on a random edge of the map for the A-star path generation. An A-star algorithm is adopted to generate the shortest path from the start to the end points, maneuvering around obstacles when possible. Obstacles are removed if there are no valid paths.

Enemies: The enemies act as the primary obstacles for players to overcome. Enemies come in several types, each with its own unique quirks. For example, some enemies have a high health pool, making them difficult to defeat. Other enemies are fast and agile, making them priority targets. Players can learn to adapt to enemy types and develop strategies to defeat them.

The appearance of the enemies differs in the different game scenes. However, as the pathing in TD games are linear, the enemies share similar behaviours and differ only in their traits. It means the same set of actions can be used for all enemies. Enemies are created with the two key components:

- Finite State Machine (FSM): It consists of a set of states, transitions between states, and actions performed when a transition occurs. It can reduce the development process time and provides a scalable solution for adding new states to enemies if necessary. The state diagram of the enemies in the game is shown in Fig. 3, where the state of each individual enemy is transited among five states according to various event triggers. The behaviors of the enemies in each state are inherited from the superclass in OOP programming.

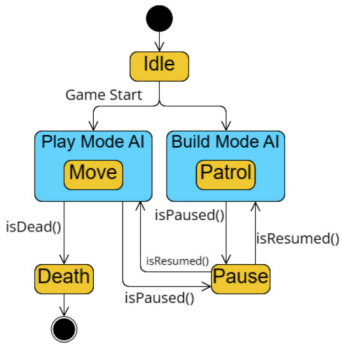

Fig. 3. Finite state machine diagram of enemies.

- Scriptable Objects: It allows for the creation of modular and reusable behaviour templates that can be shared between different enemy types. Different Scriptable Objects define their unique traits and behaviours.

Turrets: The turrets are the sole means of defence for players, allowing them to strategically place weapons to take out incoming enemies. Turrets possess varying functions, such as machine guns, rocket launchers, or laser cannons. Each can be coded to modify its behaviour by players.

Various functions and properties of the turrets are built upon OOP principles and utilise the template method pattern in its implementation. The Template Method Pattern allows subclasses to override specific functions in an algorithm without altering the overall method structure. It defines the skeleton of an algorithm in a base class. The pattern is built on the idea of constructing a base class template function including all steps of the algorithm but defers to subclasses to implement some or all of them. It enables subclasses to specify their implementation while still adhering to the structure specified in the superclass. This is the foundation upon which the custom BBPL was constructed, as code blocks share a common base while allowing for variety in block design. A high-level diagram of template method pattern is shown in Fig. 4, where the subclasses of *Shoot*, *Wait*, *Target,* and *Rotate* have unique functions, besides those inherited from their superclass of *Behaviour*.

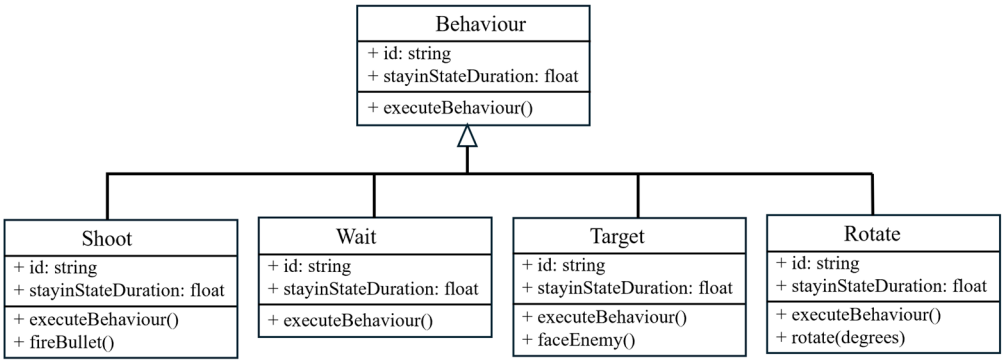

Fig. 4. Template method pattern diagram.

The Singleton Design Pattern is a creational design pattern that ensures only a singular instance of a class exists and provides a global point of access to that instance. The merits of utilizing a Singleton pattern include global access, reduced memory usage and resources, especially for web deployment using WebGL.

Several singletons are utilized to oversee various components of the game, e.g., the AI (Artificial Intelligence) Controller, Map Controller, and Game Controller. The Unified Modeling Language (UML) class diagram is shown in Fig. 5. The function of each key singleton mirrors the core components shown in Fig. 2 and is explained next:

- AI Controller: The AI of enemies in the game demonstrates autonomy. But it does not possess the ability to directly interact with game objects. This deliberate choice aims to prevent intricate interactions that could complicate code maintenance and debugging. Hence, the AI Controller is tasked with enabling this communication and acts as a central repository for other game objects to access information about each AI entity. Additionally, it oversees the creation of all AI units within the game.
- Map Controller: It serves a comparable role to the AI Controller. It oversees the GA map generation and acts as the intermediary between the player and the game. Its primary function is to facilitate interactions with the game map. It enables players to purchase, position, and sell

turrets on individual cells of the grid-like map. It also serves as the ground truth for the multiple turrets and houses the essential code blocks for the turrets to interpret and execute their functions.

- Game Controller: Unlike other singleton components, the Game Controller encapsulates crucial information of the game, including the game state, player's resources, and the progression between stages. It plays a pivotal role, coordinating with the other components and providing centralised control over the entire game.

B. Game Design

The developed game flowchart is shown in Fig. 6. The game is designed with two stages: a tutorial stage and a game stage. Within each stage there are two modes: code mode and play mode. To provide clarity for the relationships of the various stages and game modes, a brief UML class diagram is shown in Fig. 7. It is observed that each stage in the game contains only one set of game modes. Each game mode performs a different function, and players can toggle between the two modes at any time.

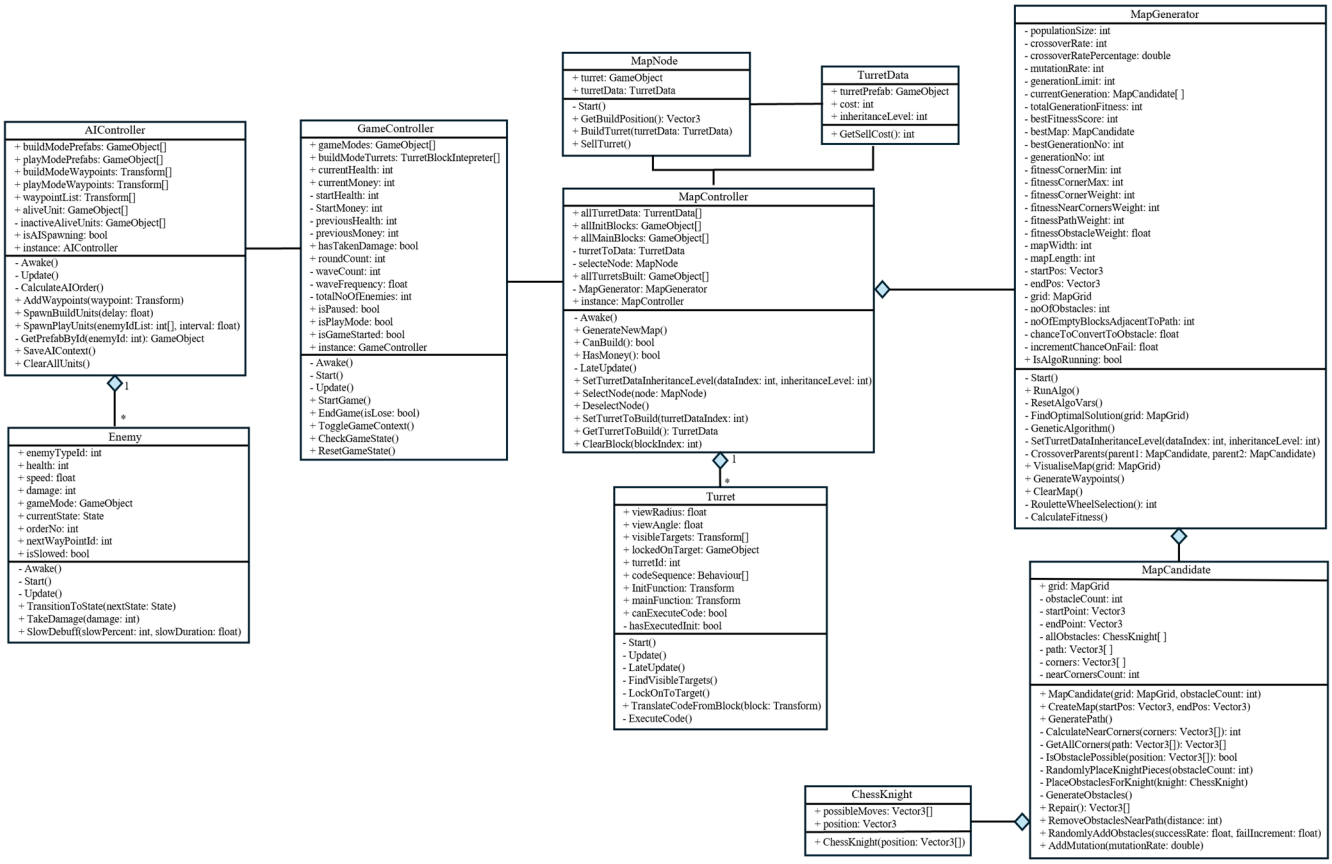

Fig. 5. UML class diagram of core components.

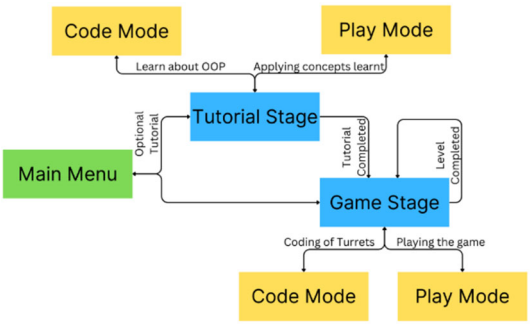

Fig. 6. Game flow and game scenes sequence.

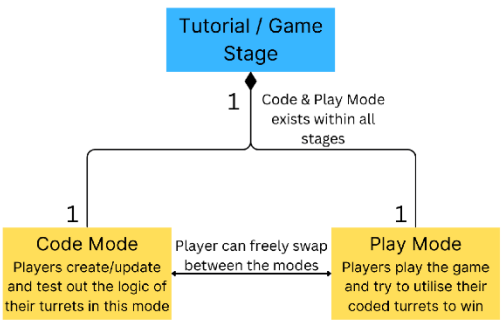

Fig. 7. UML class diagram of the game design.

The code mode features a block-based programming interface. This is the mode in which players can code their turrets. A sample screenshot is shown in Fig. 8. It allows players to visualise the output of their code and encourages experimentation with the code blocks and OOP concepts.

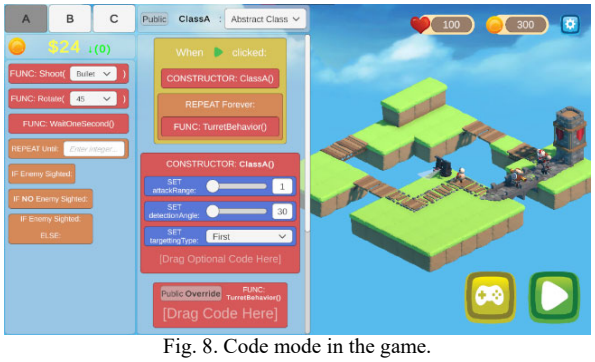

Fig. 8. Code mode in the game.

In play mode, players place the turrets that they have coded in code mode and play the game accordingly. The play mode's objectives are to complete various quests and prevent enemies from entering the castles. A sample screenshot is illustrated in Fig. 9.

Fig. 9. Play mode in the game.

The ability to freely switch between the code mode and play mode encourages experimentation, which allows players to visualize and receive real-time feedback, addressing the challenges of learned helplessness. Moreover, it promotes interactions between players and the game, thus increasing engagement and minimizing idle behavior.

1) Tutorial stage

The goal of the tutorial stage is to familiarize players with the game controls and the four basic principles of OOP: Abstraction, Inheritance, Polymorphism, and Encapsulation. The process of teaching players is split into seven learning activities in sequence: Introduction, Abstraction, Inheritance, Encapsulation, Polymorphism, Game Explanation, and a final assessment of understanding, shown in Fig. 10.

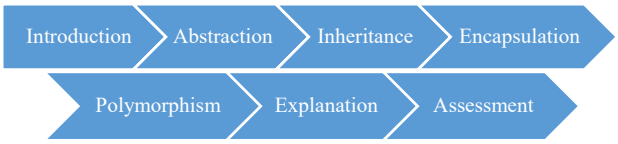

Fig. 10. Learning activities in the tutorial stage.

The tutorial stage is designed to present theories to players through an interactive slideshow, followed by a tutorial video and an accompanying exercise. The learning activities are designed in code mode to facilitate the familiarisation of the concept of block programming and OOP. A sample screenshot of the tutorial video is shown in Fig. 11.

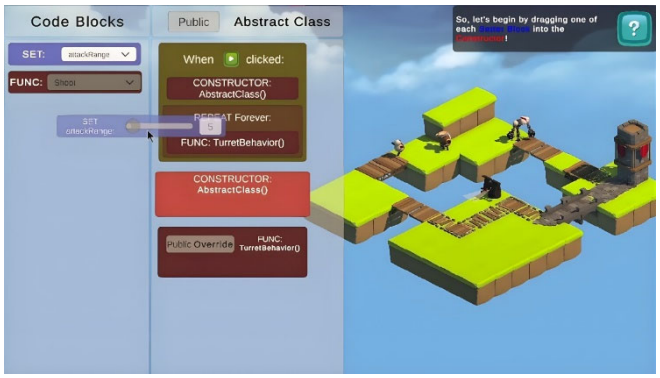

Fig. 11. Sample tutorial video to teach players the five activities.

In the play mode of the tutorial stage, players are able to apply their newfound knowledge in a risk-free and stress-free environment. Players are implicitly introduced to quests during the final phase, eliminating the need for detailed explanations. Shown in Fig. 12, an example quest is outlined in red, to eliminate at least 10 simulated enemies. In the event where players wish to replay the tutorial, they can pause the game and restart the tutorial stage.

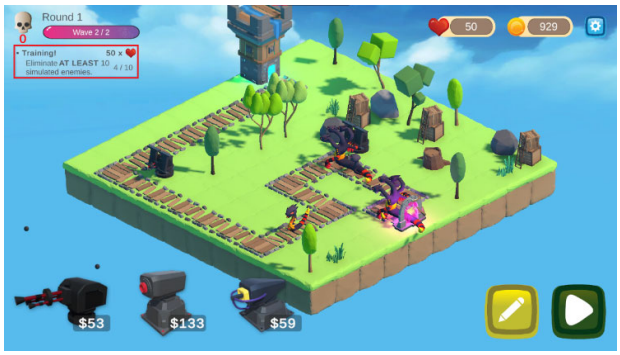

Fig. 12. Tutorial quest in play mode.

After completing all learning activities and quests in the tutorial stage, players are greeted with a congratulatory message banner. They are granted the choice of restarting the tutorial stage, or proceeding to the game stage, or returning to the main menu, shown in Fig. 13.

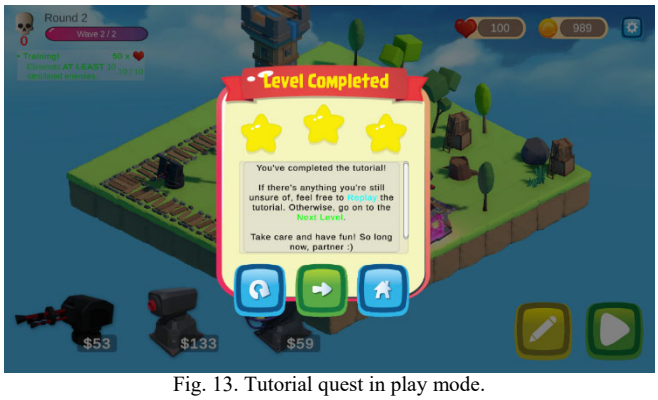

Fig. 13. Tutorial quest in play mode.

Fig. 14. A randomly generated game map.

2) Game stage

The game stage allows for the application of OOP concepts and theories. It offers endless gameplay as the game scales infinitely in difficulty through its random map generation. Upon each game round's completion, a completely new map, set of quests, and difficulty will be generated. This is made possible by leveraging the GA and A-star algorithms in the map generation. The algorithms are designed to be highly customizable as they feature various parameters that can be adjusted, which can be observed in the Map Generator class within the UML class diagram shown in Fig. 5. Additionally, Fig. 14 showcases a generated example of a map that differs from previous game scenes. Lastly, several creational and behavioural design patterns are utilised to modularise the game components, which ensures the scalability of the game.

IV. Results and Discussions

To evaluate the efficacy of the developed OOP game, a case study is performed on students from tertiary levels. The case study includes 15 voluntary participants between the ages of 18 and 25, including ten participants from non-Computing Science (CS) background, and five participants with CS background and programming knowledge. The five participants with CS background are assigned into the CS group. While the ten non-CS participants are equally assigned into the control group and experiment group randomly, who had no prior experience with software programming knowledge and educational games, as shown in Table 1.

Table 1. Participants and grouping

Participant No.	Education Level	Age Group	Programming Experience	Group
1	University	22–25	None	Control
2	University	22–25	Minimal	Experiment
3	University	22–25	Minimal	Control
4	University	22–25	None	Experiment
5	Polytechnic	18–21	Minimal	Experiment
6	University	22–25	None	Control
7	University	22–25	None	Experiment
8	University	22–25	None	Experiment
9	University	22–25	None	Control
10	University	22–25	None	Control
11	University	24–26	Experienced	CS
12	University	24–26	Experienced	CS
13	University	24–26	Experienced	CS
14	University	24–26	Experienced	CS
15	University	24–26	Experienced	CS

When selecting participants for the control group and experiment group, it is important to ensure that they had minimal software programming experience or exposure. It ensures that the assessment results are not influenced by prior knowledge or experience of software programming. The term programming experience is defined as previous exposure to Introductory Programming courses but is not pursuing or has not pursued the education in any variations of CS or ICT.

The control group and experiment group who had minimal programming experiment are invited to participate in the learning, quiz assessment, and survey. While the CS group only participates in the survey, as they already had prior experience of OOP and programming before this experiment. The procedure of the case study is carried out as follows.

1) Participants in the control and experiment groups are provided copies of OOP lecture slides. A traditional lecture is given on the contents of the slides, accompanied by a brief questions and answers session.
2) Participants are divided into their respective groups and ushered into separate rooms. A self-study session with permitted discussion is held. Both control and experiment groups conduct self-directed learning on the lecture slides, but the developed OOP game is introduced to the experiment group.
3) A zero-discussion assessment is attempted by the 10 participants. The assessment consisted of 12 multiple choice questions with four options shuffled randomly. Each question carries the same mark weightage.
4) After the assessment, participants of the control group were allowed to play with the developed OOP game.
5) A survey questionnaire was taken by all 15 participants from three groups afterward.

The motivations behind the actions in Step 1 and 2 of the procedure are twofold: one is to establish the necessity of prior knowledge on the subject according to the literature review; the other is a way to validate the legitimacy of the developed educational game as a supplementary product to the existing curriculum to learn OOP.

The control and experiment groups are examined by a quiz, to test the basics of OOP, the four pillars of OOP, and applications of these pillars. The boxplot of results is shown in Fig. 15. The mean score of all participants is about 7.3. The participants in the experiment group performed better than the control group. The experiment group has a mean score difference of about 20% as compared to the control group. The mean score of the experimental group is about 11% higher than the overall mean score of all participants. The standard deviation of the experiment group at 1.82 is lower than the control group's 2.23. It indicates the experiment group's participants are more consistent in their understanding of OOP.

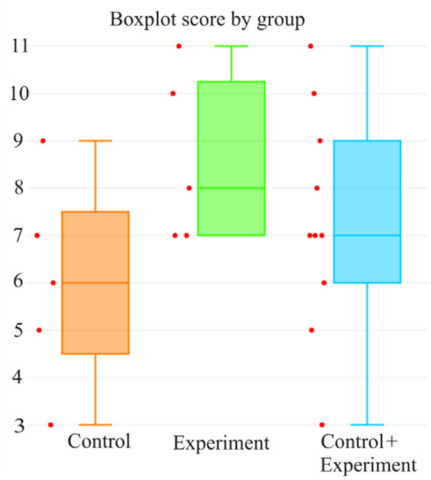

Fig. 15. Boxplot of assessment results for control and experiment groups.

Table 2. Questionnaire questions

No	Questions	Responses Type
Q1	How would you rate the OOP gameplay?	1 to 5 on a Likert Scale
Q2	How agreeable are you with the statement "The game was easy to navigate"?	1 to 5 on a Likert Scale
Q3	How agreeable are you with the statement "The game's controls were easy to pick up"?	1 to 5 on a Likert Scale
Q4	How agreeable are you with the statement "The gameplay was easy to understand"?	1 to 5 on a Likert Scale
Q5	How agreeable are you with the statement "The game improved my understanding of OOP."?	1 to 5 on a Likert Scale
Q6	How agreeable are you with the statement "The game made me interested in learning OOP."?	1 to 5 on a Likert Scale
Q7	Which learning methods do you prefer?	Traditional Classroom Lectures / Educational Game / Both / None
Q8	Do you think secondary to university level students playing the game could learn OOP concepts?	Yes / No / Others
Q9	Is there any feedback you would like to give regarding the game? (E.g., Ways to improve, features you would like to see, features you liked/disliked, etc.)	Short-text based response

All 15 participants from three groups (i.e., control group, experiment group, and CS group) are invited to perform the

survey questionnaire consisting of nine questions shown in Table 2. Six out of nine questions are rating questions with a five-point Likert scale: (1) Strongly Disagree, (2) Disagree, (3) Neutral, (4) Agree, (5) Strongly Agree. The remaining questions consist of multiple-choice questions and a short text-based question. The questionnaire is designed to solicit feedback of students, and classified as follows:

- **Q1**: Personal ratings of the game.
- **Q2–Q4**: The overall ease of use and understandability of the game.
- **Q5–Q6**: The perceived efficacy of the game with regard to its educational objectives.
- **Q7–Q8**: Personal preferences.
- **Q9**: Requirement gathering for future work on the methodology and game development.

The feedback of all 15 individuals is collected and summarized accordingly. The visualized response of the ratings of the game is shown in Fig. 16. Most participants gave the OOP game a perfect score, with an overall score of 4.73.

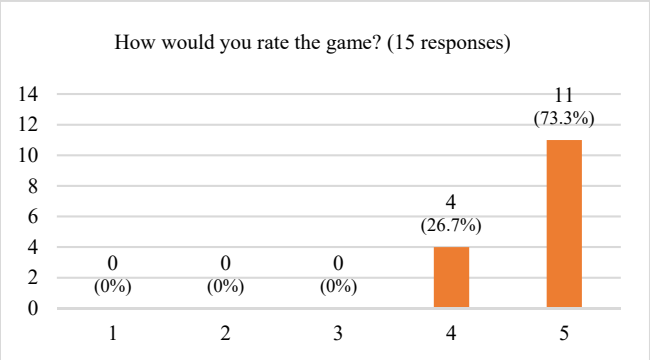

Fig. 16. Game rating for Q1.

The aggregated scores of Q2–Q4 are shown in Fig. 17, where the participants are asked about the overall ease of use and understandability of the OOP game. Most participants find the game design straightforward, while a minority feel neutral about it. It suggests that there is room for improvement. However, the positive results indicate that the game design is on the right track.

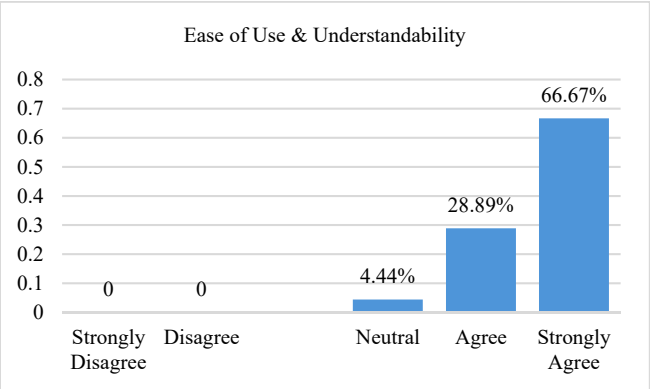

Fig. 17. Aggregated score on overall ease of use and understandability.

The aggregated scores of Q5 and Q6 are shown in Fig. 18, where the participants are asked about their perceived efficacy of the game regarding its educational objectives. It is observed that most participants feel that the game improves their interest in OOP programming. Fig. 18 shows a similar distribution of results to that of Fig. 17, but there is an increase in neutral responses. The increase in neutral responses is negligible as the participants in the CS group already have prior experience with OOP programming.

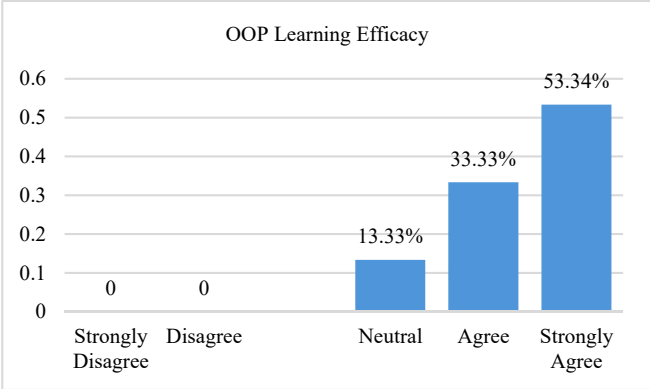

Fig. 18. Aggregated score on efficacy of the game regarding education.

It is observed in Fig. 19 that most participants prefer the educational game as their learning method, while a minority prefers both traditional classroom lectures and the OOP game. This indicates that the educational game successfully fulfills its role as a supplementary educational tool.

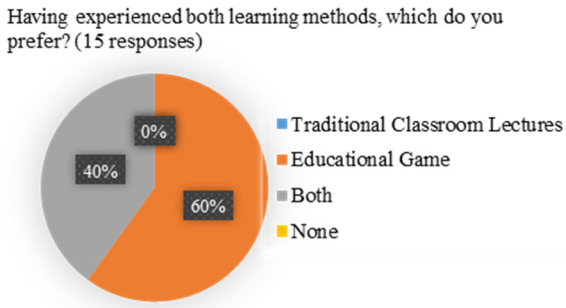

Fig. 19. Learning method preference in Q7.

It can be seen from Fig. 20, all participants agree that the game could be applied to education, with the majority supporting the idea of introducing it to secondary to university level students. It indicates a consensus on the potential benefits of incorporating the game into the curriculum. The varying opinions on the appropriate starting level further highlight the versatility and adaptability of the OOP game for different educational contexts.

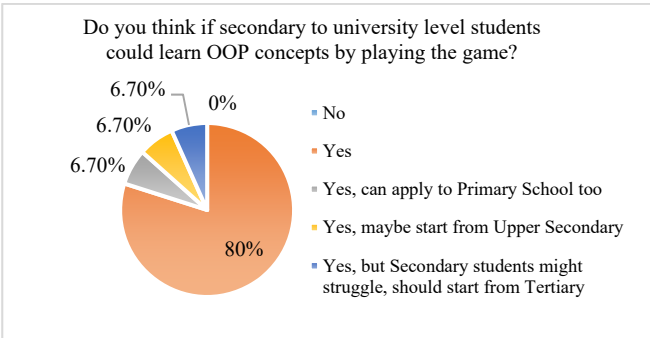

Fig. 20. Preference on applications to education level in Q8.

By analyzing the results obtained from the assessment and questionnaire, the case study provides evidence that our methodology is effective, practical, and feasible for OOP

learning. However, there are several limitations associated with the methodology in this work. Firstly, this work uses convenience sampling to select a small scale of 15 participants. This method of sampling has limitations, as it may not provide a representative sample of the population and may introduce bias into the study. The small sample size limits the generalizability of the findings. In future work, we may consider using a larger and more diverse sample to increase the external validity of the findings. We may use other sampling methods, e.g., random sampling or stratified sampling, for a more representative sample of the population.

Secondly, there is limited manpower in this research who furthermore worked on a part-time basis. This contrasts with usual game developments which consist of a team of game designers, game artists, and game programmers. The lack of manpower limited the scope and scale of the methodology.

Thirdly, it lacks a baseline for game development in this work, as the methodology is relatively novel. This work is complex in the objectives, which may have affected the ability to fully explore the methodology's potential.

V. Conclusion

In this study, gamification techniques are applied to the learning of OOP through the introduction of a novel block-based programming tower defence game. The game omits unnecessary syntax learning and exposes students to the fundamental concepts of OOP, including Abstraction, Inheritance, Encapsulation, and Polymorphism. It is important to note that the developed educational game is not intended to replace educators but rather to supplement their teachings of the existing curricula.

The case study shows that students who use the developed OOP game perform more consistently. Feedback gathered from the participants show that they respond positively and appreciate the inclusion of gamified learning. The incorporating of gamification into the learning can enhance the retention and absorption of information by providing students with an experimental environment to test, visualize, and validate their understanding of the OOP concepts.

Based on the feedback collected from all participants, improvements and recommendations for future work include:

- **Leaderboards**: Currently it lacks a leaderboard to rank players based on their scores. The inclusion of a leaderboard would add meaning to these game elements and benchmark among students.
- **Co-op Multiplayer**: It can be improved by adding a cooperative multiplayer feature to foster teamwork among students.
- **Customizability**: The platform allows educators to create custom maps and quests, to enhance the effectiveness as an educational tool.
- **Variety**: The current methodology has limited code blocks, turrets, and enemy variations. To improve the learning experience, a wider variety can be introduced.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, X.J.L.C., Y.C., and Q.C.; methodology, X.J.L.C., Y.C., and Q.C.; software, X.J.L.C.; validation, X.J.L.C.; formal analysis, X.J.L.C.; supervision, Q.C.; writing—original draft preparation, X.J.L.C.; writing—review and editing, X.J.L.C., C.K.S., Y.Z., M.W., and Q.C.; all authors had approved the final version.

References

[1] E. Macrides, O. Miliou, and C. Angeli, "Programming in early childhood education: A systematic review," *International Journal of Child-Computer Interaction*, vol. 32, p. 100396, Jun. 2022.

[2] T. Gkrimpizi, P. Vassilios, and M. Ioannis, "Classification of barriers to digital transformation in higher education institutions: Systematic literature review," *Education Sciences*, vol. 13, no. 7, 2023.

[3] P. L. Tin, "Coding as part of the mainstream curriculum," Ministry of Education Singapore, 04-Sep-2020.

[4] N. Cowan, "Working memory underpins cognitive development, learning, and education," *Educational Psychology Review*, vol. 26, no. 2, pp. 197–223, 2014. https://doi.org/10.1007/s10648-013-9246-y.

[5] Y. Soepriyanto and D. Kuswandi, "Gamification activities for learning visual Object-Oriented Programming," in *Proc. 7th International Conference on Education and Technology*, 2021, pp. 209–213.

[6] N. Gürefe and O. Bakalım, "Mathematics anxiety, perceived mathematics self-efficacy and learned helplessness in mathematics in faculty of education students," *International Online Journal of Educational Sciences*, vol. 10, no. 3, pp. 154–166, 2018.

[7] D. M. Berry, "The essential similarity and differences between mathematical modeling and programming," *Science of Computer Programming*, vol. 78, no. 9, pp. 1208–1211, 2013.

[8] S. E. Forsström and O. T. Kaufmann, "A literature review exploring the use of programming in Mathematics Education," *International Journal of Learning, Teaching and Educational Research*, vol. 17, no. 12, pp. 18–32, 2018.

[9] M. Laurent, R. Crisci *et al*., "Impact of programming on primary mathematics learning," *Learning and Instruction*, vol. 82, 2022.

[10] L. S. Vailshery. Most used languages among software developers globally 2022. *Statista*. [Online]. Available: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

[11] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification: Using game design elements in non-gaming contexts," *Annual Conference Extended Abstracts on Human Factors in Computing Systems*, 2011, vol. 66, pp. 2425–2428.

[12] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?—A literature review of empirical studies on gamification," *Annual Hawaii International Conference on System Sciences*, 2014.

[13] Y. Cai, W. van Joolingen, and K. Veermans, "Virtual and augmented reality, simulation and serious games for education," Springer Singapore, 2021. https://doi.org/10.1007/978-981-16-1361-6

[14] C. Ebert, A. Vizcaino, and R. Grande, "Unlock the business value of gamification," *IEEE Software*, vol. 39, no. 06, pp. 15–22, 2022.

[15] Q. Cao, B. T. Png, Y. Cai, Y. Cen, and D. Xu, "Interactive virtual reality game for online learning of science subject in primary schools," *IEEE International Conference on Engineering, Technology & Education*, China, 2021, pp. 383–389.

[16] S. Azmi, N. A. Iahad, and N. Ahmad, "Attracting students' engagement in programming courses with gamification," *IEEE Conference on e-Learning, e-Management and e-Services*, Malaysia, 2016, pp. 112–115.

[17] W. Toledo, S. J. Louis, and S. Sengupta, "NetDefense: A tower defense cybersecurity game for middle and high school students," *IEEE Frontiers in Education Conference*, Sweden, 2022, pp. 1–6.

[18] J. Figueiredo and F. J. García-Peñalvo, "Increasing student motivation in computer programming with gamification," *IEEE Global Engineering Education Conference*, Portugal, 2020, pp. 997–1000.

[19] W. H. Lim, Y. Cai, D. Yao and Q. Cao, "Visualize and learn sorting algorithms in data structure subject in a game-based learning," *IEEE International Symposium on Mixed and Augmented Reality Adjunct*, Singapore, 2022, pp. 384–388.

[20] A. N. Saleem, N. M. Noori, and F. Ozdamli, "Gamification applications in e-learning: A literature review," *Technology, Knowledge and Learning*, vol. 27, no. 1, pp. 139–159, 2021.

[21] A. P. Cavalcanti, A. Barbosa, R. Carvalho *et al*., "Automatic feedback in online learning environments: A systematic literature review," *Computers and Education: Artificial Intelligence*, vol. 2, 2021.

[22] J. Seanosky, I. Guillot, D. Boulanger *et al*., "Real-time visual feedback: A study in Coding Analytics," in *Proc. IEEE 17th International Conference on Advanced Learning Technologies*, 2017.

[23] T. Hainey, G. Baxter, J. Black *et al*., "Serious games as innovative formative assessment tools for programming in Higher Education," in

Proc. 16th European Conference on Games Based Learning, vol. 16, 2022.

[24] M. Aydin, H. Karal, and V. Nabiyev, "Examination of adaptation components in serious games: A systematic review study," *Education and Information Technologies*, 2022.

[25] P. Toukiloglou and S. Xinogalos, "A systematic literature review on adaptive supports in serious games for programming," *Information*, vol. 14, 2023. https://doi.org/10.3390/info14050277

[26] R. Kraleva, V. Kralev, and D. Kostadinova, "A methodology for the analysis of block-based programming languages appropriate for children," *Journal of Computing Science and Engineering*, vol. 13, no. 1, pp. 1–10, 2019. https://doi.org/10.5626/jcse.2019.13.1.1

[27] A. Vahldick, P. R. Farah *et al*., "A blocks-based serious game to support introductory computer programming in undergraduate education," *Computers in Human Behavior Reports*, vol. 2, 2020.

[28] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 1–15, 2010.

[29] O. Allen, X. Downs, E. Varoy, A. Luxton-Reilly and N. Giacaman, "Block-Based Object-Oriented Programming," *IEEE Transactions on Learning Technologies*, vol. 15, no. 4, pp. 439-453, 2022.

[30] A. E. Rais, S. Sulaiman, and S. M. Syed-Mohamad, "Game-based approach and its feasibility to support the learning of object-oriented concepts and programming," *Malaysian Conference in Software Engineering*, Malaysia, 2011, pp. 307–312.

Copyright © 2024 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).