


Tile-Based Games for Object-Oriented Programming Learning: A Modular Base Code Approach

João-Paulo Barros^{1,2} 

¹*Polytechnic Institute of Beja, 7800- 295 Beja, Portugal*

²*Center of Technology and Systems (UNINOVA-CTS) and Associated Lab of Intelligent Systems (LASI),
2829-516 Caparica, Portugal*

Keywords: Programming Assignment, Active-Learning, Programming Project, CS1, OOP, Pedagogy, Education.

Abstract: Object-oriented programming (OOP) courses pose significant challenges for students mastering numerous interrelated concepts. Creating engaging assessment tasks aligned with course objectives is also crucial for instructors. This paper addresses students' challenges in learning object-oriented programming (OOP) and instructors designing effective assessments. A new approach to OOP assessments using scaffolded, tile-based game projects is presented. It uses a modular base code that students use and expands upon in increasingly complex games. This structure allows for applying and assessing core OOP concepts throughout the project. A specific implementation of this approach, using a particular tile-based game, is detailed. Student perceptions were positive regarding the tile-based game projects, suggesting that this approach is engaging and effective for learning OOP. The provided base code serves as a practical example for instructors. The work offers instructors a concrete and adaptable method for assessing OOP competencies in a way that promotes deep learning and is less susceptible to issues with AI assistance in student work. Scaffolded game projects, built upon a common base code, can improve student engagement and facilitate a more rigorous and confident evaluation of their OOP skills.

1 INTRODUCTION


A fundamental focus of programming courses is the completion of programming assignments. These should lead students to achieve or even surpass the intended learning outcomes. To that end, assignments must be engaging, including significant and aligned content and objectives.

Games, due to their obvious appeal to most students, are a topic often explored when teaching introductory programming (e.g., (Becker and Quille, 2019; Bayliss and Strout, 2006; Cliburn and Miller, 2008b; Sung, 2009; Martins et al., 2019)) and even as a means to start learning about programming (e.g., (Vahldick et al., 2014; Livovský and Porubán, 2014)). Additionally, games are a topic students already know and have practised to some degree.

This paper proposes using tile-based games (sometimes named grid-based games) in a scaffolding approach, from very simple to more demanding games, towards completing a half-to-one-semester-long object-oriented programming project. To that

end, we present a base code template that can easily be tailored to program a large family of tile-based and traditional physical board games. Hence, teachers can use it to generate games with a similar structure. Students can then use some games as learning examples, and others can be presented as assignments. As an example of a concrete application of the base code, we then present a specific programming project, already proposed elsewhere (Barros, 2024), as a particular case for using the presented base code. More specifically, this paper includes a presentation of the base code template and a specific instance of the Sokoban game and its application in an object-oriented programming course. It also presents the results of a student survey to collect students' perceptions and opinions and briefly compares student grades in the project and a final computer-based exam of the same course. It also provides information allowing the replication of the programming project and several lessons learned.

In the following section, we present the motivation. Section 3 presents related work, and Section 4 presents the structure and behaviour of the base

^a  <https://orcid.org/0000-0002-0097-9883>

code template that can be tailored to different game assignments. Section 5 presents the Sokoban game assignment and the base code given to students built from the code template. Section 6 presents the results of a student survey and a comparison between students' grades in the programming project and a final computer-based exam. Finally, Section 7 concludes.

2 MOTIVATION

Controlled environments, like oral or written individual examinations, provide limited learning opportunities for students. However, they are less prone to cheating and give a precise, even if not complete, measure of individual performance. Differently, it is well-known that people learn by doing and preferably in a less controlled environment (e.g., (Biggs et al., 2022)). Programming projects, typically conducted totally or mostly outside classes, have these characteristics. Their time and context are closer to the real world compared to in-class assignments or traditional exams, and this, along with the absence of a strict and short time limit, increases student motivation and opportunities for deep learning. However, to be effective, programming assignments also need to be aligned with course objectives, namely to provide justified adequate opportunities to apply the course contents while being able to engage students. Object-oriented programming courses are challenging with regard to those objectives, as numerous contents need to be justifiably present in the programming project to make it aligned, engaging, and having the right level of difficulty.

Several studies with different kinds of games conclude that students prefer game assignments (e.g., (Bayliss and Strout, 2006; Cliburn and Miller, 2008b; Sung, 2009; Martins et al., 2019)). In particular, interviews with students in (Cliburn and Miller, 2008b) revealed that "all students, whether they were regular game players or not, liked the idea of providing games as assignments." The authors also concluded that games should have as much structure as possible; well-known games are preferable and should have a graphical element. Other studies also concluded that CS1 students preferred more structured, less open-ended assignments (Cliburn and Miller, 2008a; Cliburn et al., 2010). Differently, another work presented board games as open-ended first-year projects. The students' feedback was positive, and the authors concluded that the project motivated them to understand basic computer science concepts and see beyond the notion that computer science is just programming (Bezakova et al., 2013). Additionally, an-

other study concluded that assignments with graphical user interfaces are perceived as more challenging and realistic (Ball et al., 2018).

Based on our experience with several course editions, tile-based and board games provide a suitable basis for engaging and aligned programming projects for introductory object-oriented and GUI programming courses. More specifically, they offer a perfect fit to apply some simple GUI construction and GUI-based interaction, an application of the model-view-controller pattern, abstraction, encapsulation, information hiding, modularity, inheritance, interface implementation, dynamic binding and polymorphism.

The following section presents a base code easily tailored to a large set of simple tile-based board games, for example, Tic-Tac-Toe, Connect Four, Fifteen, Minesweeper, Mastermind, Checkers, Chess, and others. The objective is to use the program as a user interface builder and rule checker, not to enable the computer to play the game. If one wants the computer to be able to play, then there is a vast difference between those games, and the problem becomes suitable to AI courses or the use of suitable game engines to support the game logic. Here, we intend the developed programs to incorporate the game logic to forbid illegal moves by the human player, to apply the game rules (e.g., scoring points), and to detect the end of the game.

3 RELATED WORK

The quest for aligned and engaging assignments is continuous, as significantly testified by numerous articles and sessions, most notably the nifty assignments project (Stanford, 2024), which has been collecting short, exciting assignments. As stated in the project, "The most important aspect of a Nifty Assignment is that students love it, and it teaches something many teachers want to cover."

Available libraries facilitate graphical game programming (e.g., (Roberts et al., 2008)), but no code base template for tile-based or board games exists to our knowledge. In (Chen and Cheng, 2007), a semester-long project for game development using the C++ language is presented. The objectives include learning object-oriented concepts and event-driven programming using application programming interfaces. However, the games to be developed have significant multimedia and interaction components, and a specifically tailored library is used. Drake and Sung also propose a specifically tailored library, presented as a simpler alternative to the Java Task Force library (Roberts et al., 2008), to develop a list of thirty-two

board, card, and dice games for teaching programming in CS1 and CS2 (Drake and Sung, 2011).

4 A BASE CODE TEMPLATE FOR BOARD GAMES

Visualization is widely recognized as an important learning tool in programming (e.g. (Lian et al., 2022)). In particular, the concepts of "class" and "object" and their relations are pretty tricky for students to grasp and have been the object of several pedagogical tools (Kölling, 2015), most notably BlueJ (Michael Kölling and Rosenberg, 2003). GUI programming provides a constructive way to create and visualize objects and their actions. Additionally, tile-based games are engaging for most students and can have a simple graphical interface that novice students can build. GUI programming is also an important topic in a computer science curriculum (Force, 2020). The base code template that we propose is a variation of the well-known *model-view-controller* (MVC)(Krasner and Pope, 1988) (see Fig. 1). However, regarding the code structure, the *controller* and the *view* parts are in the same package we named *gui*. This base code template provides a basis for creating code for different programming projects for tile-based games. When starting the program, one object in the *gui* package (an instance of a class implementing the *View* interface) creates the *GameModel* singleton (in the *model* package) passing a reference of itself to it. This way, the model stores a reference to the object of type *View*. Then, the code in the *gui* package creates the board user interface (the *view*) and defines the event handlers (the *controller*), becoming ready to handle user actions. It is important to note that the reference in the *model* is a reference to an object of type *View*, a (Java) interface so that the *model* package has no compilation-dependencies to the *View* type classes. Also, more than one class implementing the *View* interface can exist. In that case, the model will have one reference to each respective *View* type object. All *View* classes are forced to implement the *View* (Java) interface so that the model object can send its messages to the *View* type objects. As in the original MVC (e.g. Fig. 3 in (Krasner and Pope, 1988)), the message-sending and dependency updating follows the sequence *user-controller-model-view* and the *view* can query the model to update itself as the model provides a public interface (as a set of public methods) to that end.

For each user action in the gui (for example, a mouse click) the *controller* sends a message to the GameModel singleton. The latter updates the game

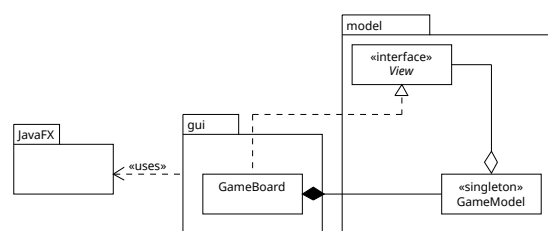


Figure 1: Class diagram for the basic board game structure.

state and then calls a method in the `View` interface to update what the user sees in the *view*. Each update the *model* needs to do in the *view* corresponds to a method in the `View` interface and is implemented in the `View` implementing class. As stated, the *view* can query the model to update itself.

Finally, it is interesting to note that the base code can easily be used for more challenging projects in more advanced courses, namely the implementation of solvers (e.g., (Li et al., 2014)) or the creation of new puzzle (maps) (e.g., (De Kegel and Haahr, 2020)).

5 SCAFFOLDING ACTIVITIES

All the students who used the proposed code base had already completed an introductory programming course in Java. The code base was used in the follow-up programming course on object-oriented programming. From the very beginning of the semester, the students were introduced to the template in Fig. 1 and used it to develop a program to allow two human players to play Tic-Tac-Toe ("noughts and crosses") using the computer as the board. The program enforced the rule of alternating the players (either drawing a "cross" or a "nought") and checking for draws or winning states. This slow step-by-step development took four weeks, with three hours of lab classes each week. As an additional example, students were also given a complete program for the game "Fifteen," which also used the base code template. Then, after three more weeks of applying test-driven programming (for the "model" code) and inheritance, always in programs using the given template, they started the autonomous development of the Sokoban final project, which we present in the following sections.

6 THE SOKOBAN ASSIGNMENT AND ITS APPLICATION

Starting from the presented base code template, we now report on applying a half-semester programming

project for implementing the Sokoban game as proposed in (Barros, 2024). Sokoban is a classical computer game created by Hiroyuki Imabayashi, in 1981, initially released the following year (Dor and Zwick, 1999; Wikipedia contributors, 2022). It has been praised as “the quintessential “block-pushing” game” (Wolf, 2008). Regarding the user interface and the interaction, it has a board (a “map”) made of “tiles” in a grid. The player character is subject to the game rules, namely how it can move. It is usually classified as a one-player puzzle game, as the player has to figure out how to push a set of boxes to specific positions in the grid (map). The player can only push one box to the adjacent position if that position is free. There can be “walls”, which are positions the player or the boxes cannot occupy, and positions marked as the final boxes’ positions. The game is played in a sequence of levels of increased difficulty, which correspond to different maps (puzzles). There are numerous maps on the internet, which is a helpful bonus as they cannot be random to guarantee a solution. As a significant example, the website <https://sokoban.info/> provides “more than 7500 levels”.

As intended for all assignments using the base code template presented in the previous section, the Sokoban base code was created in two steps: (1) starting from the template, a working solution for the Sokoban game was created, then (2) the Sokoban assignment base code was created by removing some code from the working solution. Both steps are critical to adjust the difficulty as more code or dependencies will increase the difficulty for students to understand the given code, and less code will provide insufficient scaffolding. Also, to better clarify the intended requirements, a video with the game playing using the working solution was presented to students. The same two steps should be followed to create the base code for other board game projects.

Figure 2 shows the diagram for the base code given to students for the Sokoban project. A set of game-specific classes and methods were added to the template while maintaining the described MVC behaviour. However, most classes were incomplete (the ones in light grey in Fig. 2) or missing (the ones in dark grey in Fig. 2). Two packages, each one for a different *view*, were given: `guixtext` and `guiimages`. However, only `guixtext`, the simpler one, with no images, was complete. Students had to implement the graphical one with the board images and a menu to get a GUI-based and text-based one-player game. Both shared the same `model` code. This validated the objective of having two views for two Sokoban games.

7 RESULTS

After the project concluded, students were invited to complete a survey. We also compared students’ pass and fail rates in the project with the final computer-based exam grades.

7.1 Student Survey

After the project submission, all thirty-four students who submitted their project were invited to fill out a short questionnaire. We got twenty-five answers: sixteen were first-time students, and nine were repeating students. In an online questionnaire, they were asked to grade several aspects. Eight questions used a semantic differential scale from 1 to 10, and six were open questions. The questions and the respective answers are presented next.

Next, we used stacked bar charts to present the survey results for the answers, using a semantic differential scale from one to ten. We separated the results between first-time and repeating students. For the open-ended questions, we presented all the answers for each student except for one repeating and one novice student complaining about how the grading was assigned.

As seen in Fig. 3a, students found the project slightly tricky, and repeating students found it slightly more difficult for all colleagues in general but not so much for themselves (see Fig. 3b).

As shown in Fig 4a, only one student did not enjoy the project subject: twenty out of twenty-five gave it eight or more out of ten. This is aligned with other studies that have already been cited regarding using games in programming. The enjoyment level was also positive for all except one, although slightly less than the subject preference (see Fig. 4b).

When asked to point out their likes and dislikes, the repeating student’s single dislike was the detail of the requirements. The novice students had largely positive opinions, with a few complaints about some requirements, time, and self-motivation details.

Question: Tell us briefly why you liked it or did not like it:

The opinions of the six repeating students who answered were the following:

- “I found the game quite interesting, as well as the whole structure.”
- “I don’t have anything negative to say, perhaps each requirement should be more explicit.”
- “I think it’s fun to remake a game because I believe it’s something that most students enjoy, and it may also have been the reason they joined the course in the first place.”

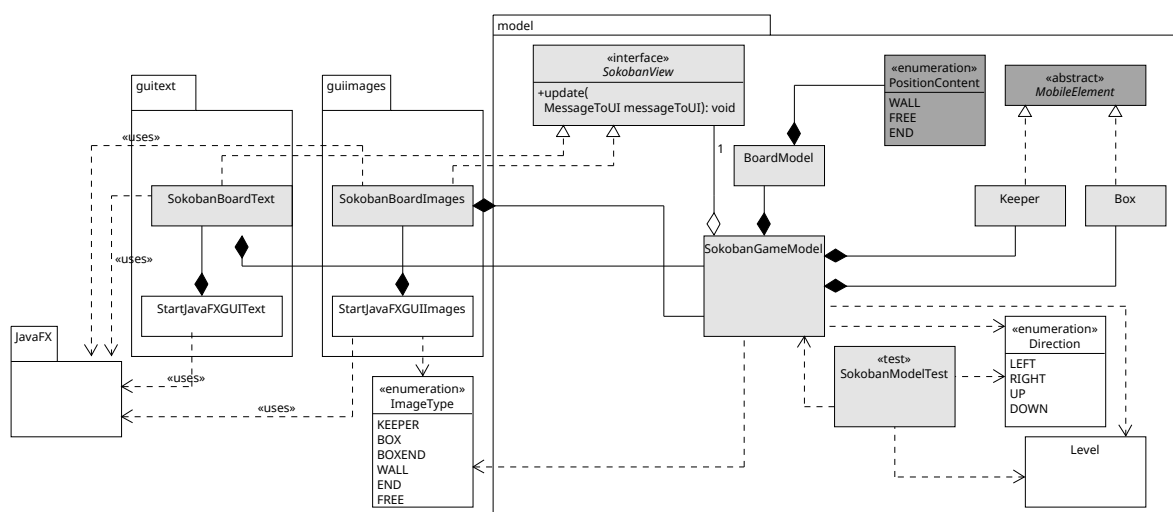
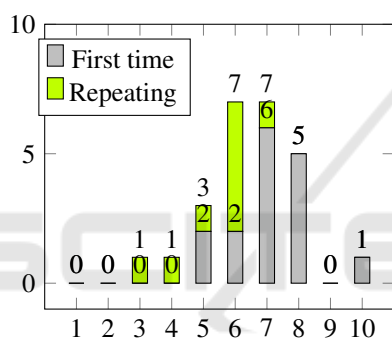
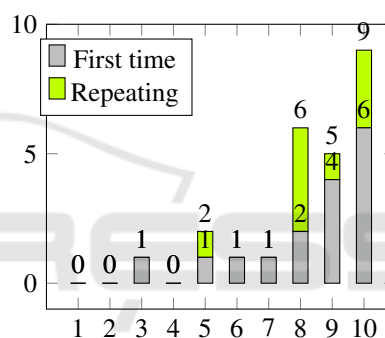


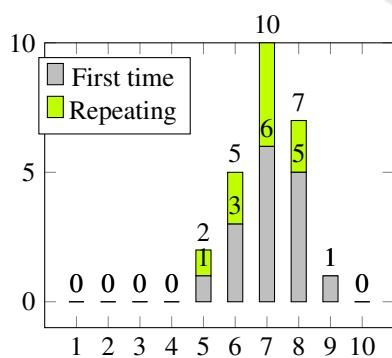
Figure 2: Class diagram for Sokoban base code.



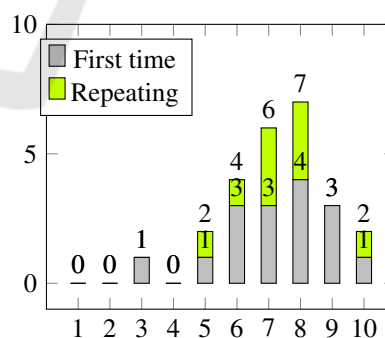
(a) Results for the assertion "How would you rate the level of difficulty of the project for yourself?" 1-very easy, 10-very difficult.



(a) Results for the assertion "How do you classify the subject of the work (the Sokoban game)?" 1-I did not like it at all, 10-I liked it very much.



(b) Results for the assertion "How would you rate the difficulty of the work for your classmates in general?" 1-very easy, 10-very difficult.



(b) Results for the assertion "How do you rate your enjoyment of doing the project?" 1-I did not like doing this project at all, 10-I very much enjoyed doing this project.

Figure 3: Perceived difficulty of the project.

- "It's an existing game that uses the logic of a game that we all know, and if you interpret the logic well, it becomes easier to do."

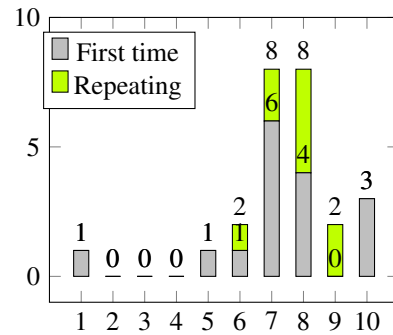
- "The fact that it's a game with the possibility of movement."

The opinions of the twelve novice students who answered were the following:

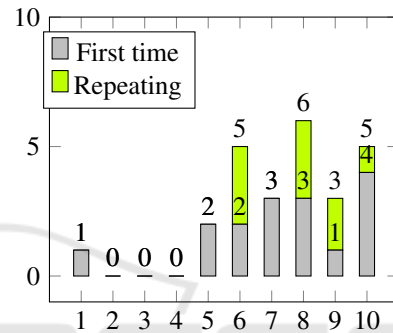
- "I really liked it because it was a game I didn't know and I became a fan of it";
- "I prefer to do the work from scratch, without a base code."
- "It was difficult to do it using GitHub as I had almost no previous experience with it."
- "I think the work was accessible. Personally, I enjoyed working on it because there were more complex parts that posed some challenges, and I enjoyed having to brainstorm like that. In other words, I think there was a balance; for those who wanted to get a minimum score of 10, I think people could do it without too much difficulty, but for those who wanted to go further, there were some challenges, so I think it makes total sense!"
- "In a way, it was a big puzzle with pieces to insert. It was hard, and I lost my hair, but the feeling of completing it was good. Maybe in the instructions you should include more details about what you can and can't do. Some of my colleagues and I were wondering if it was possible to change the basic code in a way that wasn't too extensive.";
- "Despite being a simple game, it contains a lot of content from what was taught in class, and it is possible to put into practice various parts of the code given throughout the guides."
- "The theme of the work as a whole was strange because I'm not used to it."
- "I have no motivation."
- "The work was really interesting, and I feel I learned a lot during its creation; the biggest difficulty was the intensity of the hours needed to do it."
- "The theme of the work makes it more interesting to be involved in, as you can see your progress as you test the game with new features."
- "The challenge of improving the game's code and features."
- "A little more time would have helped, as would having had some more practice before starting the work."

Students preferred to have a base code not only to get the job done (see Fig. 5a) but also to learn more (see Fig. 5b). However, 16 in 25 found it difficult to understand, which we see as unsurprising, as it was the largest piece of code they had ever encountered.

When asked about the base code, some students gave their opinion: **Question: "If you wish, please leave your opinion on the base code (e.g., defects and/or virtues)".** The opinions were the following:



(a) Results for the assertion "How do you rate the contribution of having a base code to GET the job done?" 1-It made it much harder to do the project, 10-It made it much simpler to do the project.



(b) Results for the assertion "How would you rate the usefulness of having a base code to LEARN more when doing the project" 1-I did not learn anything due to the base code, 10-I learned a lot more because of the base code.

Figure 5: Base code usefulness.

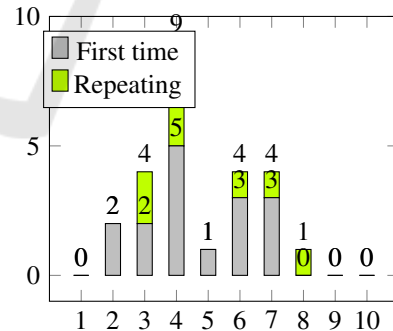


Figure 6: Results for the assertion "How difficult was it to UNDERSTAND the base code?" 1-Very difficult to understand, 10-Very easy to understand.

- "I thought that the base code was very well presented and explained, which helped solve the work."
- "As the base code had a lot of classes and methods, it took a lot of time to understand the code. I hope that in the next ones, there will be a summarized explanation of the base code."

The novices that answered also largely complained about an insufficient previous explanation of the base code in class.

- "I don't know if, for future work, I wouldn't give the students more freedom to create a project of their own from scratch instead of taking a base; it's different. Of course, the complexity would also have to be readjusted."
- "The hardest part was internalizing and understanding the base code, but despite that, I think it was well composed."
- "I think the base code should have been explained a bit over the course of the semester."
- "Initially, it was more difficult to understand the whole structure of the code due to its complexity and incompleteness."
- "It would have been very useful if the base code had been explained in class; we spent much time reading the base code, jumping from class to class until we understood where the functions were pulled from."

As optional replies, students were asked about what should be added (suggestions), stopped (negative), and continued (positive). The **suggestions** were the following:

- "I'd say handing in the assignment earlier, and monitoring the work more in class, not just at the end of the semester." (repeating student)
- "More time for presentations" (repeating student)
- "Have topics outside the world of games, for example, more commercial programs." (repeating student)
- "Having more time for presentations. I think a project like this, even a long one, should have more opportunity to be defended. Otherwise, we just focus on the negative points."
- "Instead of just one delivery, it would be a good idea to do it over the semester and add one or more features each time."
- "Explanation of the basic code"
- "Just work, not tests."
- "Explaining the base code in a dedicated class would greatly help. We asked the teachers questions in class, and that was fundamental. A class dedicated to the code would have advanced our work by several hours."
- "explaining the base code in class."

Regarding **negative points**, one repeating student asked for more detailed information about the grading

to be assigned to each requirement, and one novice student would have preferred no base code.

Regarding **positive points**, one repeating student stated, "I think that giving a base code helps a lot in solving a final project." Another student liked using the MVC pattern. Among the novice students, the statement and the existence of the base code were underlined by two students as they helped them understand how the project should work and how it should be built. One student considered they had plenty of time to complete the project.

7.2 Student Grading and Outcomes

After completing the programming project, students had to complete a computer-based test consisting of brief programming tasks based on a base code similar to the Tic-Tac-Toe game they had studied in class. The similar base code allowed the desired alignment between assessment and learning outcomes and between the project and the exam. In the exam, students were asked to add two functionalities and the respective test code to the given and known base code. All students who failed the programming project also failed the exam, and 68% of students who passed the project also passed the exam. In the individual oral examination, the 32% students who failed the exam managed to answer the questions about the submitted code. This demonstrates the need for the short exam to effectively filter out students who manage to explain the submitted group work without knowing how to do it. This is often due to excessive reliance on the work of the other group elements. Even so, if sufficient human resources are available, an alternative approach would be to conduct a longer oral examination, which would imply significantly more time.

8 CONCLUSIONS

The base code template provides flexible support for programming projects for tile-based games with a GUI programming component in object-oriented courses. The template was already tested for the games Tic-Tac-Toe and Fifteen as part of a scaffolding approach. The same template was also applied to the Sokoban game as a final graded project. The students' perceptions and feedback were very favourable. The base code template and the Sokoban project base code are available at <https://github.com/jpmprb/SokobanBase2023>. In future work, we will apply this template to additional games, enabling students to write code and engage with more interactive learning experiences. This

hands-on approach is expected to promote improved problem-solving skills and increased retention of key concepts.

ACKNOWLEDGEMENTS

This work was financed by Portuguese Agency FCT – Fundação para a Ciência e Tecnologia, in the framework of project CTS/00066

REFERENCES

- Ball, R., DuHadway, L., Hilton, S., and Rague, B. (2018). Gui-based vs. text-based assignments in cs1. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 1017–1022, New York, NY, USA. Association for Computing Machinery.
- Barros, J. P. (2024). Sokoban: An assignment for an object-oriented and gui programming course. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2024, page 1564–1565, New York, NY, USA. Association for Computing Machinery.
- Bayliss, J. D. and Strout, S. (2006). Games as a "flavor" of CS1. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, page 500–504, New York, NY, USA. Association for Computing Machinery.
- Becker, B. A. and Quille, K. (2019). 50 years of cs1 at sigcse: A review of the evolution of introductory programming education research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 338–344, New York, NY, USA. Association for Computing Machinery.
- Bezakova, I., Heliotis, J. E., and Strout, S. P. (2013). Board game strategies in introductory computer science. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 17–22, New York, NY, USA. Association for Computing Machinery.
- Biggs, J., Tang, C., and Kennedy, G. (2022). *Teaching for Quality Learning at University*. Open University Press, 5th edition.
- Chen, W.-K. and Cheng, Y. C. (2007). Teaching object-oriented programming laboratory with computer game programming. *IEEE Transactions on Education*, 50(3):197–203.
- Cliburn, D. C. and Miller, S. (2008a). Games, stories, or something more traditional: the types of assignments college students prefer. *SIGCSE Bull.*, 40(1):138–142.
- Cliburn, D. C. and Miller, S. M. (2008b). What makes a "good" game programming assignment? *J. Comput. Sci. Coll.*, 23(4):201–207.
- Cliburn, D. C., Miller, S. M., and Bowring, E. (2010). Student preferences between open-ended and structured game assignments in cs1. In *2010 IEEE Frontiers in Education Conference (FIE)*, pages F2H–1–F2H–5.
- De Kegel, B. and Haahr, M. (2020). Procedural puzzle generation: A survey. *IEEE Transactions on Games*, 12(1):21–40.
- Dor, D. and Zwick, U. (1999). Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228.
- Drake, P. and Sung, K. (2011). Teaching introductory programming with popular board games. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, page 619–624, New York, NY, USA. Association for Computing Machinery.
- Force, C. T. (2020). *Computing Curricula 2020: Paradigms for Global Computing Education*. Association for Computing Machinery, New York, NY, USA.
- Kölling, M. (2015). Lessons from the design of three educational programming environments. *International Journal People-Oriented Program.*, 4(1):5–32.
- Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49.
- Li, Z., O'Brien, L., Flint, S., and Sankaranarayanan, R. (2014). Object-oriented sokoban solver: A serious game project for ooad and ai education. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–4.
- Lian, V., Varoy, E., and Giacaman, N. (2022). Learning object-oriented programming concepts through visual analogies. *IEEE Transactions on Learning Technologies*, 15(1):78–92.
- Livovský, J. and Porubán, J. (2014). Learning object-oriented paradigm by playing computer games: concepts first approach. *Open Computer Science*, 4(3):171–182.
- Martins, V. F., Eliseo, M. A., Omar, N., Castro, M. L. A., and Corrêa, A. G. D. (2019). Using game development to teach programming. In *Handbook of Research on Immersive Digital Games in Educational Environments*, pages 450–485. IGI Global.
- Michael Kölling, Bruce Quig, A. P. and Rosenberg, J. (2003). The bluej system and its pedagogy. *Computer Science Education*, 13(4):249–268.
- Roberts, E., Bruce, K., Cutler, R., Cross, J., Grissom, S., Klee, K., Rodger, S., Trees, F., Utting, I., and Yellin, F. (2008). ACM Java Task Force. [Online; accessed 18-January-2024].
- Stanford (2024). Nifty assignments. [Online; accessed 17-January-2024].
- Sung, K. (2009). Computer games and traditional cs courses. *Commun. ACM*, 52(12):74–78.
- Vahldick, A., Mendes, A. J., and Marcelino, M. J. (2014). A review of games designed to improve introductory computer programming competencies. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–7.
- Wikipedia contributors (2022). Sokoban — Wikipedia, the free encyclopedia. [Online; accessed 13-October-2022].
- Wolf, M. J., editor (2008). *The video game explosion: a history from Pong to Playstation®and beyond*. Greenwood Press.