Gamified Virtual Reality for Program Code Structure Comprehension

Roy Oberhauser [1] and Carsten Lecon [1]

[1] Department of Computer Science, Aalen University, Aalen, Germany

Abstract - **When programmers view program code text, the abstract and invisible nature of the underlying program code structures remains inherently challenging for them to visualize. Widespread availability of virtual reality (VR) hardware and software now make VR visualization of program code structures accessible. In such potentially visually satiating environments, the application of gamification has the potential to provide an additional focus and motivational factor towards comprehending program structures. Towards this end, this paper describes our Gamified Virtual Reality FlyThruCode (GVR-FTC) approach which gamifies our immersive metaphorical visualization of any given software code structure. Our initial results show that VR-based gamification (specifically code dependencies and modularization) can be more fun and motivational and support structural program comprehension better than using a PC-based text editor for a similarly gamified situation.**

Index Terms - **Virtual reality; Gamification; Software visualization; Program comprehension; Software engineering; Computer education.**

I. Introduction

The rapid digitalization of society is inexorably linked with an increased demand for and dependence on software. Accordingly, the amount of program source code produced and maintained worldwide by software developers is increasing dramatically. At least 2bn lines of code (LOC) can be accessed by 25k developers at Google[1], and well over a trillion lines of code (LOC) are estimated to exist worldwide with at least 33bn added annually (Booch, 2005). In one study[2], 11m professional software developers are estimated to be producing or maintaining program code, excluding hobby and other IT-skilled workers.

Given such a volume of code, these developers continue to struggle with comprehending unfamiliar complex code structures and dependencies utilizing common display forms of program source code or the two-dimensional Unified Modeling Language (UML). Reasons for this include cognitive limitations as well as the inherent invisibility of software, which remains an essential difficulty for software construction, as the reality of software is not embedded in space (Brooks, 1995). A vision of walking through a 3D visualization of software architecture has been described (Feijs and De Jong, 1998), and in prior work we developed our 3D (non-VR) fly-through code (FTC) approach for navigating software structures (Oberhauser, Silfang, and Lecon, 2016). Yet the potential of VR and game engines has not been fully realized in software engineering (SE) tools, and their practicality with off-the-shelf VR hardware has been insufficiently explored. To address this, in prior work (Oberhauser and Lecon, 2017) we described our VR-based FlyThruCode (VR-FTC) approach for visualizing, navigating, and conveying program code information interactively in a VR environment using a game engine to support exploratory, analytical, and descriptive cognitive processes (Butler et al., 1993).

Although VR-FTC provided a VR-based software visualization capability, it did not directly support learning. According to game designer C. Crawford (Crawford, 1984), learning is a fundamental motivation for all game-playing. Computer games involving the learning of some knowledge area are known as educational computer games (Wolf, 2012). Serious games (Michael and Chen, 2005) (Abt, 1970), be they digital games (DG) or not, have an explicit educational focus and can be used to educate, train, and inform. (Connolly et al., 2012) identified 7392 papers that included 129 with empirical evidence and found that the most frequently reported outcome and impact of playing games were affective and motivational as well as knowledge acquisition or content understanding. Furthermore, DG-based learning research has largely shown that it is now accepted that DG can be an effective learning tool (Van Eck, 2006). With respect to gamification in software

E-mail: `roy.oberhauser@hs-aalen.de, carsten.lecon@hs-aalen.de`

[1] http://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/

[2] http://www.infoq.com/news/2014/01/IDC-software-developers

engineering (SE), (Connolly et al., 2007) found that users liked game-based learning and found it motivating and enjoyable. However, within SE and computer science (CS) education, the potential of serious DG in combination with VR for motivating and enhancing program code comprehension remains insufficiently explored. Furthermore, in our previous study on gamification in SE (Oberhauser, 2016), most SE DGs were simulation-based and focused primarily on management aspects, with almost no SE DG dealing with software design or code structural aspects.

Extending our VR-FTC paper (Oberhauser and Lecon, 2017), this paper contributes a digital gamification approach to VR-based program code structure visualization called Gamified Virtual Reality FlyThruCode (GVR-FTC). Our prototype demonstrates the viability of the approach while the empirical study investigates its potential.

The paper is organized as follows: the following section discusses related work; Section 3 then describes our solution approach. In Section 4, details about the realization are provided. Our evaluation is described in Section 5, which is followed by a conclusion.

II. Related Work

Work related to software visualization includes Teyseyre and Campo (Teyseyre and Campo, 2009), who provide an overview and survey of 3D software visualization tools across the various SE areas. Software Galaxies[3] gives a web-based visualization of dependencies among popular package managers and supports flying, with each star representing a package clustered by dependencies. CodeCity (Wettel and Lanza, 2007) is a 3D software visualization approach based on a city metaphor and implemented in SmallTalk on the Moose reengineering framework. Buildings represent classes, districts represent packages, and visible properties depict selected metrics, with Wettel et al. (Wettel et al., 2011) showing a significant improvement in task correctness and task completion time. X3D-UML (McIntosh, 2009) provides 3D support with UML in planes such that classes are grouped in planes based on the package or hierarchical state machine diagrams. As to VR approaches, Imsovision (Maletic, Leigh, and Marcus, 2001) visualizes object-oriented software using electromagnetic sensors attached to shutter glasses and a wand for interaction. ExplorViz (Fittkau, Krause, and Hasselbring, 2015) is a browser-based web application that uses Javascript-based WebVR to support VR exploration of 3D software cities using Oculus Rift together with Microsoft Kinect for gesture recognition. In contrast, GVR-FTC, visualizes software structures by leveraging common game engine VR capabilities and a single VR system and controller set (not requiring trained gestures) for an immersive VR software visualization environment. It is also unique in providing multiple dynamically-switchable and customizable metaphors that support tagging, searching, and filtering of visual objects. An oracle in the form of a virtual tablet and keyboard are unique for providing an additional intuitive interaction capability within the VR landscape for accessing diverse external non-VR SE tool data.

With regard to digital gamification within SE, Pedreira et al. [25] carried out a systematic mapping of studies by extending the ISO/IEC 12207 to classify the SE process areas that were gamified for 29 primary studies from 2011-2014 and concluded that: the application of gamification in SE is in an initial stage, research in this area is quite preliminary, there is little sound evidence of its impact, and that there is scarce empirical evidence. They found the focus to lie mostly on software design, and to a lesser extent software requirements, project management, and other support areas. With regard to the use of gaming within the SE education field, Connolly et al. [17] performed a literature search of games-based learning in SE and "found a significant dearth of empirical research to support this ap3proach." We were unable to find work related to the application of VR-based DG in the area of program code comprehension.

Our approach GVR-FTC supports the gamification of software program structures in VR for exploration, analysis, description, training and education. In contrast to other games, our GVR-FTC can gamify any provided real project program code, rather than creating some simulated or fictional environment, and thus motivate the learning of an actual program for a software organization.

III.Solution Approach

GVR-FTC utilizes a 3D application domain view visualization (Oberhauser, Silfang, and Lecon, 2016) of program code structure (i.e. the software architecture). On these often invisible abstract structures, it provides a non-textual perspective, arranging customizable symbols in 3D space and enabling fly-through navigation. For example, certain information typically not readily accessible is visualized, such as the relative size of classes (not typically visible until multiple files are opened or a UML class diagram is created), the relative size of packages to one another, and the dependencies between classes and packages.

A. Architecture

Figure 1 shows the GVR-FTC game-engine-based architecture. Assets are used by the game engine (Unity in our implementation) and consist of Animations, Fonts, Imported Assets (like a ComboBox), Materials (like colors and reflective textures), Media (like textures), 3D Models, Prefabs, Shaders (for shading of text in 3D), VR SDKs, and Scripts. Scripts consist of Basic Scripts like user interface (UI) helpers, Logic Scripts that import, parse, and load project data structures, and Controllers that react to user interaction. Logic Scripts read Configuration data about Stored Projects and the Plugin System (input in XML about how to parse source code and invocation com-

[3] http://github.com/anvaka/pm/

mands). Logic Scripts can call Tools consisting of General and Language-specific Tools. General Tools currently consist of BaseX, Graphviz, PlantUML, and Graph Layout - our own variant of the KK layout algorithm for positioning objects. Java-specific tools are srcML, Campwood SourceMonitor, Java Transformer (invokes Groovy scripts), and Dependency Finder. Additional tools and applications to be easily integrated as Plugins.

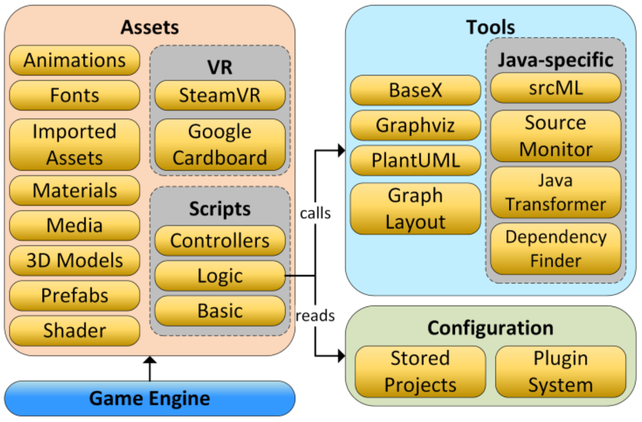


Figure 1: Software architecture.

B. Principles

Solution principles include: multiple dynamically switchable and tailorable 3D visual metaphors (space, terrestrial, custom); delineated grouping metaphors of representative objects for code packages or components (solar systems, glass bubble or tree-lined cities); directed connection metaphors for object dependencies (pipes, light rays); flythrough navigation (i.e. motion) via camera movement by controllers in an anchored scene; an oracle to readily access data (virtual tablet); tagging support to custom label objects in the landscape; and immersive background sounds.

C. Code Structure Extraction Process

The structure extraction process consists of: (1) modeling generic program code structures, metrics, and artifacts as well as visual objects, (2) mapping the model to a visual object metaphor, (3) extracting a given project's structure and metrics, (4) visualizing a given model instance within a metaphor, and (5) supporting navigation through the model instance (via camera movement based on user interaction).

D. Gamification

The intention of our serious digital gamification approach is to motivate users to familiarize themselves with the actual (not fictional) textual source code or code structures of any given project within a (VR or non-VR) visualization environment. For this, various DGs were realized (BLong and DepEnd) and others are foreseen, for instance, to gamify project knowledge (program code or structural) comprehension of a given project by reconstructing missing structures or by searching and exploring it to find certain program or structural elements. A game aspect of ranking is also included by showing the top scorers at the end of a game, and the reward system thus permits one type of knowledge or understanding comparison with cohorts.

IV. Realization

We now describe the VR environment and metaphors before detailing the implemented DGs.

A. VR Environment

The HTC Vive, a room scale VR set, is used to track the movement of a head-mounted display and two wireless handheld controllers (see Figure 2) using two 'Lighthouse' base stations (not shown). Note that the pictured wall is mirroring what the subject is viewing. The touchpad on the left hand-held controller controls altitude (up, down) and the right one direction (left, right, forward, backward) to realize flythrough navigation by moving the camera position. A laser pointer for selection becomes visible when the controller enters the view field, (shown in the universe metaphor in Figure 2) and a selected object (class) changes to a whitish color and is pointed to by a rotating inverted pyramid to track smaller objects during navigation.

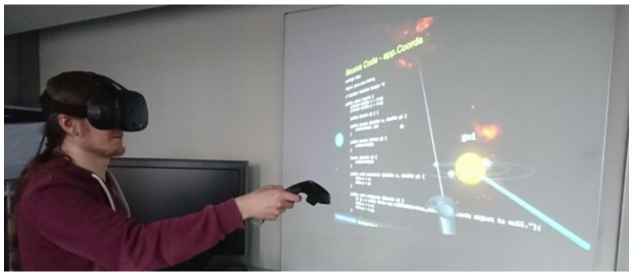


Figure 2: Subject using Vive HTC headset and controller (visible in a scene making a planet selection in solar system).

B. VR Metaphors

A welcome room in the form of a space vehicle cockpit is shown initially for metaphor and game or other destination selection (Figure 3). To exemplify support for multiple 3D visual metaphors, a universe (Figure 4) and a terrestrial metaphor (Figure 5) were implemented, as they are relatively universal metaphors - more detailed justification is described in (Oberhauser, Silfang, and Lecon, 2016). In the terrestrial metaphor, labeled glass bubble cities represent packages (Figure 5); buildings represent classes (Figure 6, labeled in blue at the top) - the number of stories represent some metric (in this case the number of class methods); and colored pipes show directed dependencies. In the universe, packages are represented by a solar system (Figure 2), classes via a labeled planet (Figure 4), and dependencies between classes or packages are indicated via directed colored light beams.

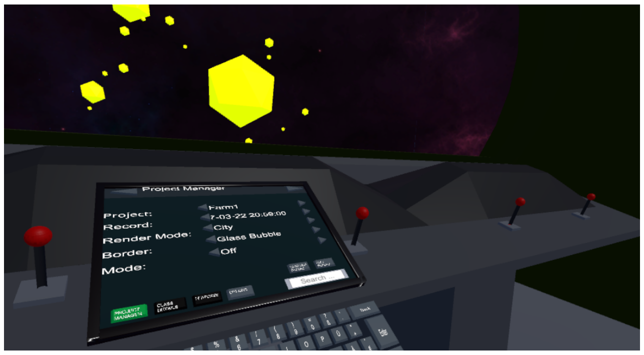

Project Manager
Project: Farm1
Record: 7-03-22 20:59:00
Render Mode: City
Border: Glass Bubble
Mode: Off

Figure 3: Welcome room cockpit for metaphor selection.

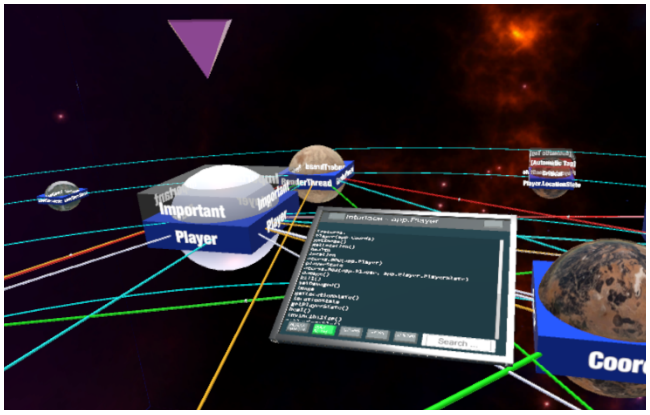


Figure 4: Universe metaphor showing the selected planet (class) via an inverted pyramid as an arrow. Tags on planets are evident above their label, and the oracle shows the interface for the selection.

While our non-VR 3D FTC variant utilizes a semi-transparent Heads-Up Display (HUD) paradigm to show various informational screens, in VR mode we determined that a HUD was not practical, since these screens contain large amounts of text that must be fairly opaque to be readable - hiding the background landscape. Moreover, any head movement shifts the then hidden landscape and can thus cause disorientation, while the focal point must be further inset than on a monitor. Thus, we chose to use a virtual tablet as an oracle (Figure 7) for providing source code (Figure 7), code metrics (Figure 5), interfaces (Figure 4), dynamically generated UML diagrams, tagging, filtering, and project data for a selected object. Two buttons at the top support switching to the previous or next screens, a scrollbar is on the right, and various features buttons are placed at the bottom. To enable users to more easily recall objects, persistent tagging permits labels matching automatic patterns or any manually inserted label to be placed on an object (e.g., the Important Tag on the Player class in Figure 4).

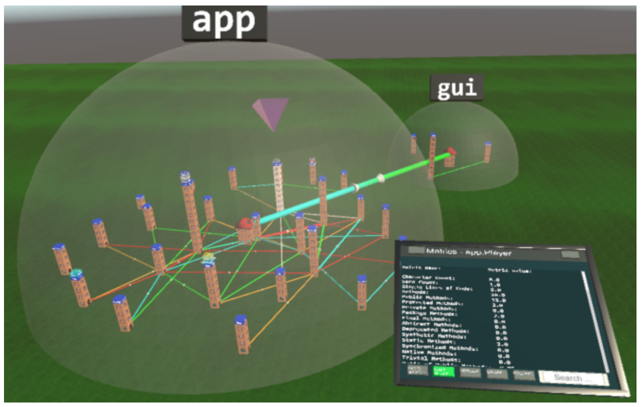


Figure 5: Terrestrial metaphor with bubbled cities (packages) and the oracle showing metrics for a selected object (notice upside-down pyramid pointer).

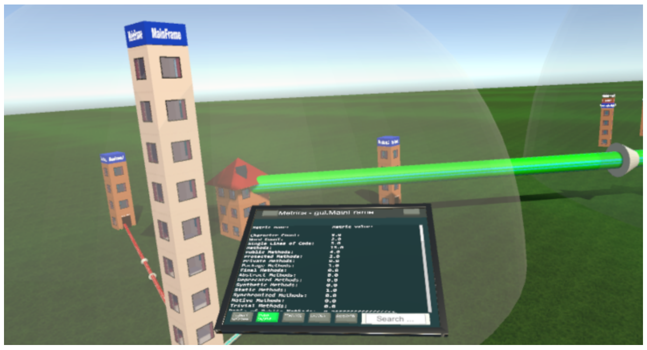
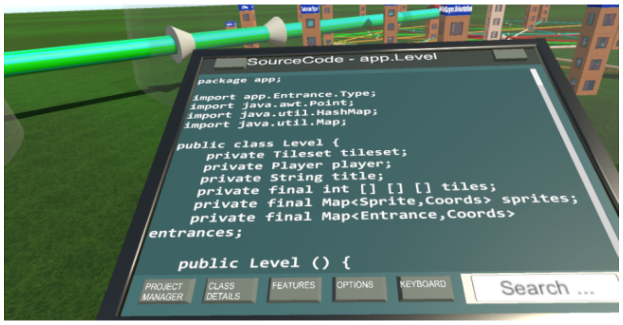


Figure 6: Terrestrial metaphor with building (class) selected and oracle (tablet) showing metrics for the selected object (pink building).

As to the internal realization, XML is used to hold relevant source code, metrics, and metadata. For extracting existing code structure information into our model, srcML (Maletic, J. et al. 2002) is used to convert source code into XML and BaseX used for XML storage. Campwood SourceMonitor and DependencyFinder extract code metrics and dependency data, and plugins with Groovy scripts and a configuration file are used to integrate the various SE tools. The game logic for both DGs described in this section was realized within the Scripts module of Figure 1.



```
package app;

import app.Entrance.Type;
import java.awt.Point;
import java.util.HashMap;
import java.util.Map;

public class Level {
    private Tileset tileset;
    private Player player;
    private String title;
    private final int [] [] [] tiles;
    private final Map<Sprite,Coords> sprites;
    private final Map<Entrance,Coords>
entrances;

    public Level () {
```

Figure 7: The oracle (virtual tablet) interface showing source code for a selected object. A bidirectional (dependency) pipe is seen in the background.

C. Dependency Structure Gamification (DepEnd)

While in a visually appealing metaphor, the game DepEnd seeks to motivate players to familiarize themselves with source code dependency structures. Referencing key game elements from the Game Ontology Project (GOP)[4] (Zagal and Bruckman, 2008), the main DG elements are:

- *Goal*: The goal is to earn points by correctly determining the directed dependencies between program code classes by analyzing the source code via our oracle (virtual tablet) and then indicating the determined dependencies via drag-and-drop of a connector. Further, time is tracked and compared in the rankings to motivate the user to work quickly.
- *Interfaces*: As to hardware, a VR headset is used for the visual display and VR controllers are used to interact in VR. Source code is accessed via an oracle (virtual tablet). A virtual laser pointer on the controller is used to connect classes with directional dependencies.
- *Rules*: Points and the time necessary function as the reward system and are used for comparison with other high scorers. One point is accrued for each correctly set dependency, no points are given for each missing dependency, and -1 points are given for each incorrect dependency (between the wrong objects or in the wrong direction) - in order to penalize guessing.
- *Entity Manipulation*: All visualized dependencies between classes are hidden when the game is started. Class objects, be they buildings in the terrestrial or planets in the universe metaphor, are then selected individually and, after code analysis within the oracle, a dependency is dragged-and-dropped (drawn) onto another class object.

[4] http://www.gameontology.com

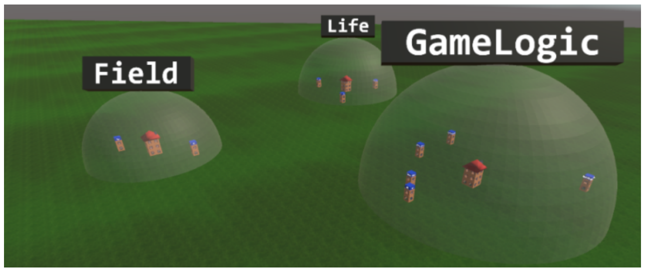


Figure 8: Initial DepEnd game setup in the City metaphor.

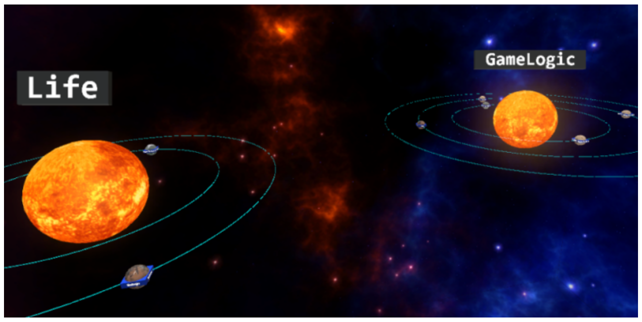


Figure 9: Initial DepEnd game setup (Universe metaphor).

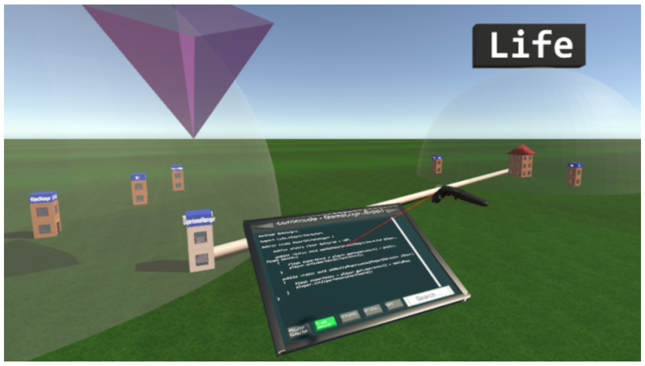
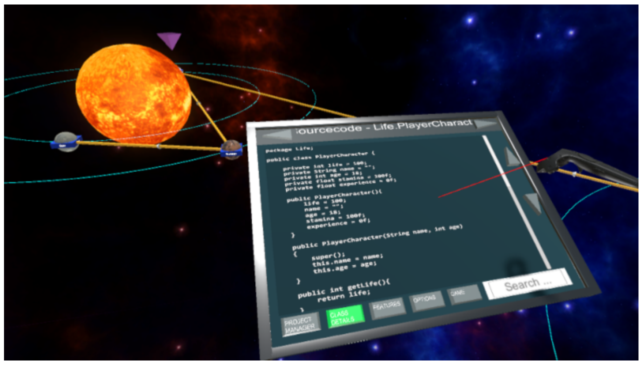


Figure 10: Directed dependency created in City metaphor.

Initially all dependencies between classes and packages are hidden (Figure 8 and Figure 9). On the oracle, access to the source code Java import statements (Figure 7) and the Interface screen (Figure 4), which shows inbound and outbound dependencies, is restricted to prevent cheating. Time tracking starts as soon as soon as the project is loaded. The oracle is then used to read and analyze the source code to determine class dependencies without knowing the actual import statements. The missing directed connections are then inserted by the user by selecting a class with the controller and releasing the connector via drag and drop on another class (Figure 10, Figure 11, Figure 12, and Figure 13). For bidirectional dependencies, the procedure is repeated starting with the opposite class. When finished the user selects "Check" on the "Game" screen, stopping the clock. The result analysis is shown in Figure 14 and Figure 15, where connectors are colored green when correct, yellow when wrong (in direction or between the wrong objects), and red when missing. The player can then navigate to see where mistakes were made and check the source code. The highest scores are shown in a table on the oracle for comparison Figure 16 with the ability to add one's name if one has a top score.



```
package Life;
public class PlayerCharacter {
    private int life = 100;
    private String name = "";
    private int age = 18;
    private float stamina = 100f;
    private float experience = 0f;
    public PlayerCharacter(){
        life = 100;
        name = "";
        age = 18;
        stamina = 100f;
        experience = 0f;
    }
    public PlayerCharacter(String name, int age)
    {
        super();
        this.name = name;
        this.age = age;
    }
    public int getLife(){
        return life;
    }
```

Figure 11: Determining dependencies in the Universe metaphor.

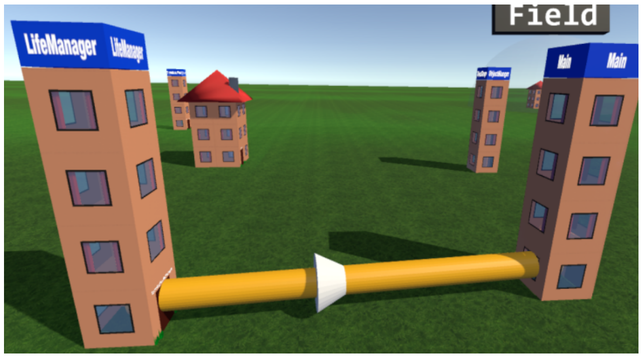


Figure 12: A directed dependency in the City metaphor.

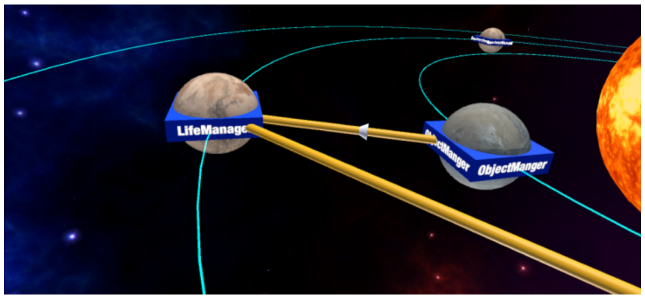


Figure 13: A directed dependency in the Universe metaphor.

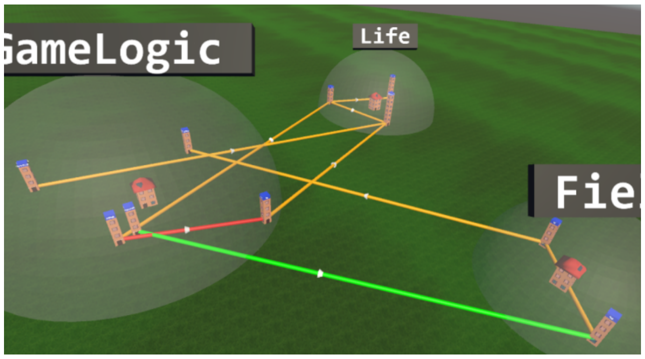


Figure 14: DepEnd game analysis (City metaphor).

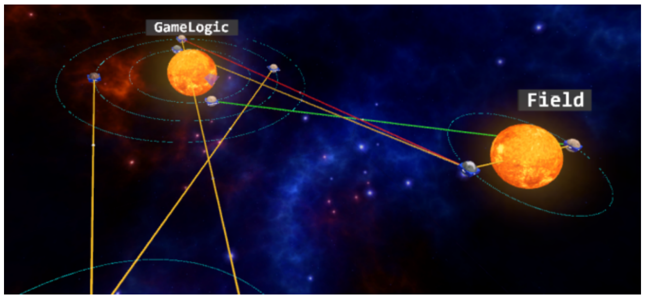
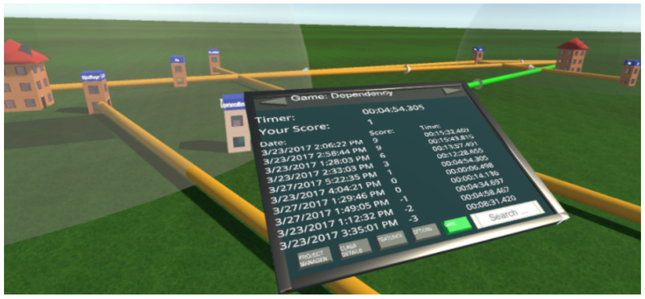


Figure 15: DepEnd game analysis (Universe metaphor).



```
Timer:        00:04:54.305
Your Score:   1
Date:                    Score:   Time:
3/23/2017 2:06:22 PM     9        00:15:32.403
3/23/2017 2:58:44 PM     9        00:15:43.815
3/23/2017 1:28:03 PM     6        00:13:57.491
3/23/2017 2:33:03 PM     3        00:12:28.655
3/27/2017 5:22:35 PM     1        00:04:54.305
3/23/2017 4:04:21 PM     0        00:00:06.498
3/27/2017 1:29:46 PM     0        00:00:14.136
3/27/2017 1:49:05 PM     -1       00:04:34.697
3/27/2017 1:12:32 PM     -2       00:04:58.467
3/23/2017 3:35:01 PM     -3       00:08:31.420
```

Figure 16: Game ranking screen.

D. Code Modularization Gamification (BLong)

The focus of the game BLong is for users to familiarize themselves with the modularization of a code project and be able to recall its structural modularization. Referencing key game elements from the GOP, the main DG elements are:

- *Goal*: The goal is to earn points while being timed by determining the module (Java package) for each element (Java class) while being timed and then to indicate this by drag-and-drop of the object onto the correct module.
- *Interfaces*: A VR headset is used for the visual display, VR controllers are used to interact in VR to access code via an oracle (virtual tablet) and to place classes in packages (in accordance with the metaphor).
- *Rules*: Points and the total time duration function as the reward system and are used for comparison with other high scorers. One point is accrued for each correct and -1 points for each incorrect placement, in order to penalize guessing.
- *Entity Manipulation*: Class objects, be they buildings in the terrestrial or planets in the universe metaphor, are selected individually and dragged-and-dropped onto some package object (e.g., bubble or sun) for grouping placement.

First, based on Java packages as a grouping mechanism, the configuration of classes within packages is presented (Figure 17 and Figure 18). The user then has unlimited time to visually impregnate the modularization configuration (or its implicit underlying principles or patterns). Once the user starts the game, the classes are placed randomly in a line external to empty packages (Figure 19 and Figure 20) and the user is timed as to how quickly they can correctly reproduce the modularization by moving classes via drag-and-drop to the appropriate package (Figure 21 and Figure 22) while minimizing the need to access the source code to determine its package. In contrast to DepEnd, no interaction with the oracle for code analysis is required, since the correct allocation is shown visually before starting the game, unless one forgets where an object belongs and needs to look. When finished, the user selects "Check" on the "Game" screen, stopping the clock. The result analysis is shown in Figure 23 and Figure 24, where correctly placed objects are colored green and otherwise red to provide visual feedback. The player can then navigate to see where mistakes were made and also check the source code. The highest scores are shown in a table on the oracle for comparison (Figure 25) with the ability to add one's name if one has a top score.

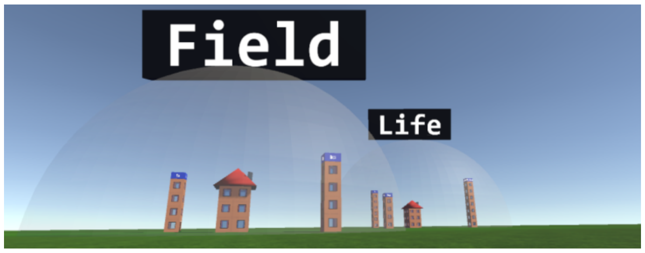


Figure 17: Example initial BLong module depiction in the City metaphor.

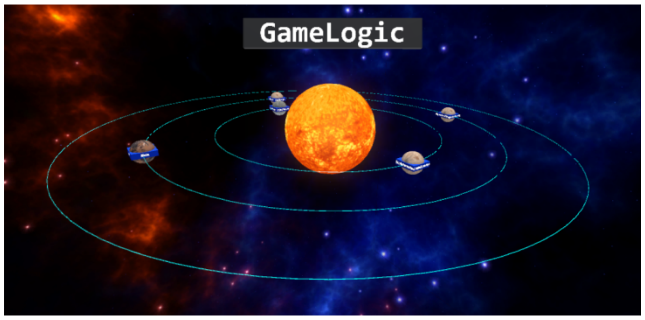


Figure 18: Example initial BLong module depiction in the Universe metaphor.

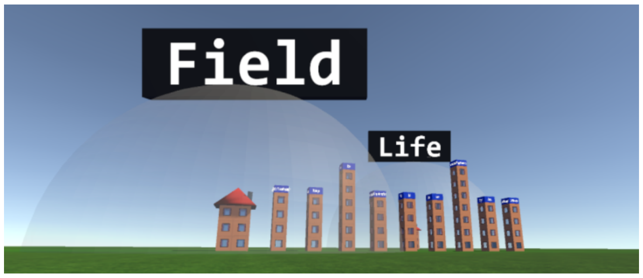


Figure 19: Initial BLong game start (City metaphor).

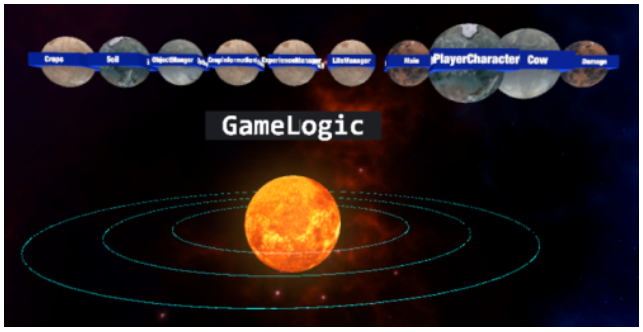


Figure 20: Example initial BLong game start in the Universe metaphor.

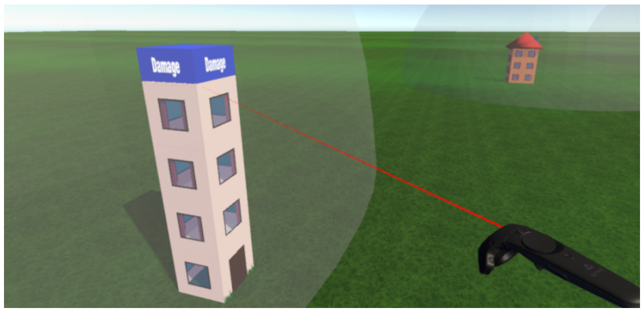


Figure 21: BLong drag-and-drop in the City metaphor.

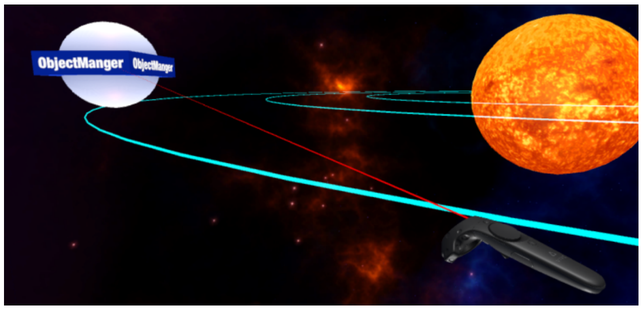


Figure 22: BLong drag-and-drop in the Universe metaphor.

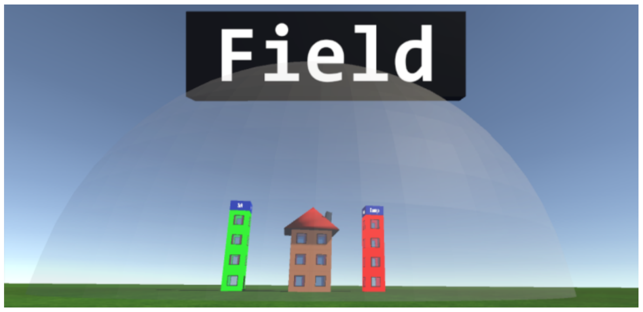


Figure 23: BLong game analysis in the City metaphor.

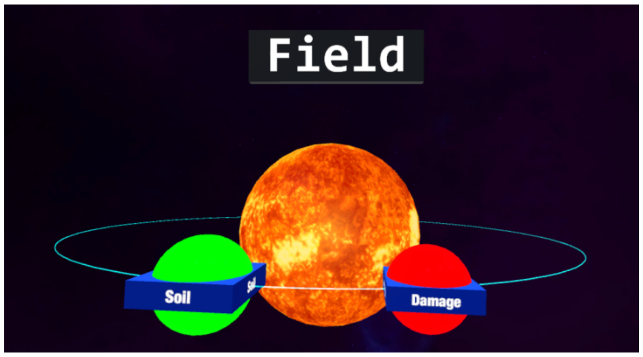
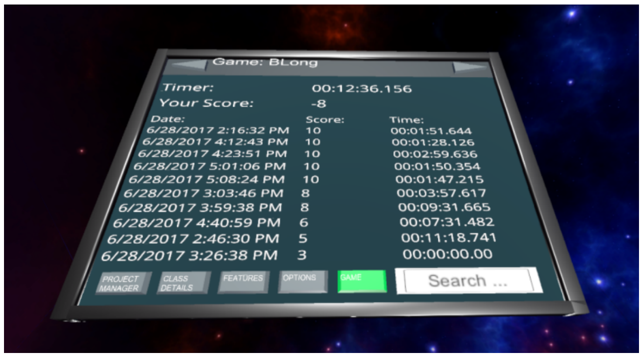


Figure 24: BLong game analysis in the Universe metaphor.



Timer: 00:12:36.156
Your Score: -8

Date:	Score:	Time:
6/28/2017 2:16:32 PM	10	00:01:51.644
6/28/2017 4:12:43 PM	10	00:01:28.126
6/28/2017 4:23:51 PM	10	00:02:59.636
6/28/2017 5:01:06 PM	10	00:01:50.354
6/28/2017 5:08:24 PM	10	00:01:47.215
6/28/2017 3:03:46 PM	8	00:03:57.617
6/28/2017 3:59:38 PM	8	00:09:31.665
6/28/2017 4:40:59 PM	6	00:07:31.482
6/28/2017 2:46:30 PM	5	00:11:18.741
6/28/2017 3:26:38 PM	3	00:00:00.00

Figure 25: BLong points and time ranking.

V. Evaluation

The experiential and motivational benefits of VR-FTC for VR novices were underscored in our previous empirical study (Oberhauser, Silfang, and Lecon, 2016). Furthermore, in our prior work (Oberhauser and Lecon, 2017), we noted no significant difference in usability efficiency of VR-FTC vs. desktop mouse usage for SE analysis tasks. Thus the focus of this evaluation was on the games themselves, for which we utilized a convenience sample of six Master of Computer Science students. Due to various scheduling restrictions and resource limitations, we were unable to expand the number of subjects for this study.

While BLong and DepEnd can use any real project of any number of packages, classes, and dependencies, for the empirical evaluation we initially limited the experiment to placement of a maximum of 10 objects or dependencies, just exceeding the cognitive working memory limit (Miller, 1956). Otherwise, for BLong users might attempt to discover allocation patterns for classes to certain packages rather than being able to remember an arbitrary configuration. Also, this limit reduced the experiment duration. While the object grouping was primarily a domain association, to ensure that users inspected and paid attention to the entire configuration, a few objects were randomly assigned to a contrary package (an exception to expectations). When questions requiring the editor (Notepad++) occurred, for BLong a table was provided and a mark needed to be made to indicate in which of the 3 packages each of the 10 classes were; for DepEnd a drawing of the 3 packages and their classes was provided and they only needed to draw the dependencies.

To analyze the effects of GVR-FTC gamification with regard to code modularization using the BLong game, Table 1 shows the averaged results in relation to typical editor usage (using Notepad++ on a PC) for BLong. BLong had a lower error rate (5.0% vs. 8.3%) and was faster to complete (157 vs. 209 seconds) versus an editor. The game was more fun (4.7 vs. 4.0) and the subjects felt they understood the structure better (3.8 vs. 3.3) versus an editor (Notepad++), whereby the mouse-based editor interface (which they are used to) was found to be more intuitive (4.3) than the VR controller interface of BLong (3.8).

For GVR-FTC gamification of dependency structures using the DepEnd game, Table 1 shows that DepEnd had a much higher error rate than editor (Notepad++) usage (40% vs. 6.7%). This may be due to the requirement in DepEnd to access the oracle and analyze the code to determine dependencies, and then visually draw these using the VR controllers without forgetting the relation start, end, and direction. Using the editor, the user need only draw a line on the answer sheet and move on. This also explains the higher duration (504 vs. 401 seconds) for DepEnd vs. editor usage, since more cumbersome interaction to place both ends of the dependency were required. This also affected its intuitiveness (3.8 vs. 4.6 for the editor). However, DepEnd had a higher fun factor (4.0 vs. 3.0) and was felt to provide a better structural understanding (4.2 vs. 2.6), which underscore the positive potential of VR visualization and gamification for program comprehension.

When comparing the DepEnd to BLong games, BLong had fewer erroneous placements (5% vs. 40% for DepEnd) and was more fun than DepEnd (4.7 vs. 4.0). Users found both DepEnd and BLong to be equivalently helpful in understanding the project structure (4.0 vs. 3.83 for BLong). DepEnd required users to read and analyze code to determine dependencies, whereas BLong required only visual object analysis. BLong was just as intuitive as DepEnd to use.

Table 1: Averaged results for VR and editor-based games.

	BLong	Editor vs. BLong	DepEnd	Editor vs. DepEnd
Error rate	5.0%	8.3%	40%	6.7%
Duration (seconds)	157	209	504	401
Fun factor[1]	4.7	4.0	4.0	3.0
Intuitiveness[1]	3.8	4.3	3.8	4.6
Structural understanding[1]	3.8	3.3	4.2	2.6

[1]Scale of 1 to 5 (best)

As to possible threats to validity, the enjoyment factor might be independent of the gamification factor and be based, for instance, primarily on the attractiveness of the visualization environment or perhaps the newness of such a VR application. However, based on the open comments that asked about motivation and which reported game-like motivation, in our opinion this is unlikely. While the results of our study are not statistically significant due to the limited sample and resources, we interpret these preliminary results to show that the gamification approach holds promise and future work will pursue this further with a more comprehensive empirical study and a more tightly integrated gamification interface.

VI. Conclusion

The GVR-FTC VR-based gamification approach provides an immersive VR flythrough game experience of any given program code structure (currently only Java is realized) using multiple metaphors for visualizing, navigating, and conveying program code information interactively. Two games (DepEnd and BLong) were realized to demonstrate gamification's potential for improving the comprehension of structural dependencies and code modularization. In contrast to other approaches, it can gamify any given program code project (currently only Java projects are supported). The evaluation showed that the VR-based games were more fun and provided better program structure comprehension support than equivalently gamified text editor interaction. While our evaluation sample was not statistically significant, our results underscore the potential for gamification to help focus a VR user's attention in a visually distracting environment. Future work will investigate gamification's potential with regard to program code comprehension with a more comprehensive empirical study and will look into improving the game interfaces to reduce error rates and improve efficiency.

Acknowledgments

The authors thank Sinan Emecan, Alexandre Matic, Camil Pogolski, and Patrick Sprenger for their assistance with the design, implementation, and evaluation.

References

Abt, C. 1970. Serious Games. The Viking Press.

Booch, G. 2005. The complexity of programming models. Keynote talk at AOSD 2005, Chicago, IL, Mar. 14-18, 2005.

Brooks, F. P. Jr.. 1995. The Mythical Man-Month. Boston, MA: Addison-Wesley Longman Publ. Co., Inc.

Butler, D. M. et al. 1993. Visualization reference models. In Proc. Visualization '93 Conf. IEEE CS Press, 337–342.

Connolly, T.M., Boyle, E.A. MacArthur, E., Hainey, T., and Boyle, J.M. 2012. A systematic literature review of empirical evidence on computer games and serious games, Computer Education, 59, pp. 661–686.

Connolly, T., Stansfield, M. and Hainey, T. 2007. An application of games-based learning within software engineering. British Journal of Educational Technology, 38(3), pp. 416-428.

Crawford, C. 1984. The art of computer game design. McGraw-Hill/Osborne Media.

Feijs, L. and De Jong, R. 1998. 3D visualization of software architectures. Comm. of the ACM, 41, 12 (1998), 73-78.

Fittkau, F., Krause, A., and Hasselbring, W. 2015. Exploring software cities in virtual reality. In Proc. IEEE 3rd Working Conf. Software Visualization (VISSOFT), IEEE, 130-134.

Maletic, J. et al. 2002. Source code files as structured documents. In Proc. 10th Int. Workshop on Program Comprehension, IEEE, pp. 289-292.

Maletic, J. I., Leigh, J. and Marcus, A. 2001. Visualizing software in an immersive virtual reality environment. In Proc. 23rd Intl. Conf. on Softw. Eng. (ICSE 2001). IEEE.

McIntosh, P. M. 2009. X3D-UML: user-centred design, implementation and evaluation of 3D UML using X3D. Ph.D. dissertation, RMIT University.

Michael, D.R. and Chen, S.L. 2005. Serious games: Games that educate, train, and inform. Muska & Lipman/Premier-Trade, 2005.

Miller, G. A. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review. 63 (2): 81–97.

Oberhauser, R. 2016. An Ontological Perspective on the Digital Gamification of Software Engineering Concepts. Journal on Advances in Software, IARIA, ISSN: 1942-2628, vol 9, no 3 & 4, pp. 207-221.

Oberhauser, R., and Lecon, C. 2017. Virtual Reality Flythrough of Program Code Structures. In Proc. of the 19th ACM Virtual Reality International Conference (VRIC 2017). ACM.

Oberhauser, R., Silfang, C., and Lecon, C. 2016. Code structure visualization using 3D-flythrough. In Proc. 11th Int'l Conf. on Comp. Sc. & Educ. (ICCSE), IEEE, pp. 365-370.

Teyseyre, A. R. and Campo, M. R. 2009. An overview of 3D software visualization. Visualization and Computer Graphics, IEEE Transactions on, 15, 1 (2009), 87-105.

Van Eck, R. 2006. Digital game-based learning: It's not just the digital natives who are restless, EDUCAUSE review, 41(2), pp. 16-30.

Wettel, R. and Lanza, M. 2007. Program comprehension through software habitability. In Proc. 15th IEEE Int'l Conf. on Program Comprehension, IEEE CS, pp. 231–240.

Wettel, R. et al. 2011. Software systems as cities: A controlled experiment. In Proc. of the 33rd Int'l Conf. on Software Engineering, ACM, pp. 551-560.

Wolf, M. 2002. The medium of the video game. University of Texas Press.