Applying Visual GUI Testing to Computer Game Testing

Maria Camenzuli
School of Design and Informatics
Abertay University
DUNDEE, DD1 1HG, UK

ABSTRACT

Context
Although test automation is a de-facto standard in general software engineering, the games industry still relies on manual testing as the primary means of quality assurance. One reason for this is the fact that existing testing tools are not robust enough to gracefully cope with software as dynamic and interactive as a computer game.

Aim
This paper presents a plan to assess whether visual GUI testing, an emergent GUI testing technique that uses computer vision to locate components on screen, can be applied to computer game testing. The aim is to create a suite of automated tests for a game that will not break when GUI elements move around on screen or are distorted by artistic animations.

Method
A visual GUI testing tool will be developed and then used to create a suite of regression tests for an existing, full-fledged, 2D game that was recently released commercially.

Anticipated Results
The study is anticipated to show that visual GUI testing is a viable technique for creating robust automated tests for computer games at a high-level.

Keywords

Computer Games, Computer Game Testing, Visual GUI Testing, Computer Vision

1. INTRODUCTION

Although software quality is as important in the development of computer games as it is in any other software development endeavour, the practices of quality assurance and testing used in game development vary significantly from those used in the development of software in other domains. Studies by Murphy-Hill et al. (2014) and, more recently, by Santos et al. (2018) indicate that there is significantly less test automation used in the development of games when compared to the development of other software. These studies identified a number of key reasons that seem to be behind the human-centred approach to testing that is prevalent in games development.

To begin with, when dealing with games, testing needs to account for measures that are much too abstract and subjective for an automatic process to quantify, such as how fun the game is and how good the controls feel to the player. On the more quantifiable side of things, since games are highly interactive applications that often give the player the freedom to achieve the same thing in a multitude of different ways, the input and state space that can be tested in a game level could be mathematically enormous. Another challenge is posed by the non-determinism injected into the game by the randomness often used in games to keep the player from getting used to certain patterns, and more recently perhaps, from emergent AI behaviour. Yet another problem is that automated testing in games is viewed as very fragile, with tests easily breaking when game designers inevitably make tweaks to the game parameters.

Interestingly, Murphy-Hill et al. (2014) also noted that apart from the undoubtedly hard technical problems that face game developers who look to automate testing, these developers may also be held back by the business reality that game play testers are quite cheap and easy to hire relative to software testing engineers. However, one could argue that given that the lifetime and maintenance phase of games are increasing in length due to the popularity of online delivery services such as Steam, the investment in building repeatable, reliable, automated tests, particularly regression tests, might end up being quite cost effective in the long run. There will be an even stronger case for automation if new research into the core issues that have hindered automation in the games industry so far can spark the creation of new tools that effectively address these problems.

In an attempt to help address the problem of fragility in automated game testing, this paper presents a proposal to develop a visual GUI testing tool that can be used to automatically locate and interact with GUI elements in a running game intelligently. The tool will use computer vision to locate GUI elements in a rendered game frame, without the need for access to the game's source code or any of its internal data. By relying solely on computer vision to interact with the game's rendered GUI, the tool will mimic user behaviour while being robust to possible changes in the layout of the GUI. Through this study, the question of whether visual GUI testing can be successfully applied to computer game testing will be answered.

2. BACKGROUND

Despite the fact that automated testing of software at the GUI level has been the subject of several studies, support for this type of testing is limited by weaknesses in existing GUI testing techniques that might be holding back companies from moving away from manual GUI testing. This is noted by Alégroth & Feldt (2017), who also point out that these existing tools and techniques have been classified in three generations. These generations of tools are differentiated by the approach they take to interact with and make assertions on an application's GUI.

The first generation approach is to provide the automated process with the exact screen space position of GUI components. A test engineer writing an automated test script using this method would specify the exact screen space position of a button, for example, and then have the script click on that location. This approach lacks any form of robustness and results in very easily breakable tests. If the GUI designer were to move components around, or if the application is run on a different resolution than the one used while the tests where being written, the tests are bound to break. To get around the need for test engineers having to specify screen space coordinates in their scripts, a commonly used approach is Record and Replay (R&R) (Borjesson & Feldt 2012). Using this process, mouse and keyboard input are first recorded automatically while a human user is manually interacting with the GUI. Then the test script simply replays the human user's actions automatically, forming the automated test. However, when R&R is simply recording screen space positions, the tests created using this technique are still very fragile.

The second generation approach is to locate and interact with GUI components through lower-level access to the GUI library being used, or to any other API-based hooks that may be available. This is commonly done when testing web-pages, for example, because web browsers expose the structure of web-pages through the Document Object Model (DOM), a programming interface that gives automated processes the ability to view and change the structure, style, and content of a web-page. The R&R method can also be applied here, with code references to components being stored for replay instead of screen coordinates (Borjesson & Feldt 2012). While this method of GUI testing improves on the first generation by being significantly more robust to high-level changes in the GUI, it is still sensitive to structural changes in the code of the system under test. Another problem with this approach is that it is only applicable when the software being tested provides an API that can be used for locating GUI components of interest to the tester.

The third and final generation approach is to use computer vision on the GUI in order to locate components of interest. This technique is referred to as Visual GUI Testing (VGT), and it is a relatively recent and emergent technique. Using an image of what a GUI component, such as a button, is expected to look like, a test engineer can specify that the test script should locate the section of the screen that looks like the specified component. Since this technique uses computer vision to locate GUI components on frames or images of the rendered GUI, it is not vulnerable to changes in GUI layout or structural code changes, but is instead sensitive to big changes in the appearance of visual components. Academic research has produced a handful of tools that apply this technique, including Sikuli by Chang et al. (2010), and JAutomate by Alegroth et al. (2013).

Research by Alégroth & Feldt (2017), Borjesson & Feldt (2012), and others, have provided support for the applicability of VGT in practice for general software testing. However, there does not seem to be any research into the applicability of VGT in the testing of computer games, and this is something this proposal seeks to address. It seems reasonable to argue that VGT could be effective in the computer game development industry, because it is a technique that performs testing at a very high level, something practitioners have expressed a preference for when surveyed by Murphy-Hill et al. (2014). Apart from this, it is very robust to layout changes, a very desirable feature for testing software as dynamic as a game. Naturally, layout changes are not the only cause of transformation in a GUI during the runtime of a game, since game GUIs may contain other dynamic elements such as animated sprites. It would be useful to know how the robustness of VGT holds up in the face of such changes.

With regards to justification for investing in automated testing in game development in general, Buhl & Gareeboo (2012) present an industry case study. The authors describe how investing time in test automation, after getting management and team buy-in, resulted in a significant boost in quality over time in the development process of a commercially released game.

3. METHODOLOGY

The goal of this study is to assess whether visual GUI testing can be applied to automated game testing in order to create tests that are resilient to changes in the GUI. For the scope of this project, only 2D games will be considered. The hypothesis under test is that an automated test for a 2D game that uses VGT to locate and interact with GUI elements will not break when GUI elements move around on screen or vary slightly in appearance from one frame to another due to the presence of artistic animation, assuming that the animation does not distort the appearance of a component in a way that would render it unreasonable for computer vision algorithms to identify it as the same component.

The proposed plan is to develop a VGT tool and evaluate whether the technique can be applied to computer game testing by using it to write a suite of regression tests for a game. An alternative approach would be to use an existing VGT framework, such as Sikuli (Chang et al. 2010) or JAutomate (Alegroth et al. 2013). However, these tools were built with general software GUIs in mind, not computer games. Because of this, one could argue that the computer vision algorithms that they use are unlikely to factor in strong support for things such as sprite animation. This is hard to verify, because not all of these tools are open-source. A tool specifically designed for use in computer game testing is preferable in the context of this study because it will be built to support the dynamic and animated GUIs of computer games.

The game that will be tested is Wargroove (Chucklefish 2019), a recently released, 2D, turn-based tactics game that is entirely GUI driven. As shown in the screenshot in figure 1, the player is presented with a battlefield laid out on a grid based map. When it is his or her turn, the player can click on units to select them, then click on map tiles to make the unit move there, attack an enemy, or issue any other command available to the unit in the given context. The game has been released commercially and is not open-source, so its source code and internal data are inaccessible. Although many games can benefit from VGT, given their graphical nature, Wargroove is an ideal game to use as the test subject for this project for a handful of reasons. To begin with, the game does not need to be modified to setup

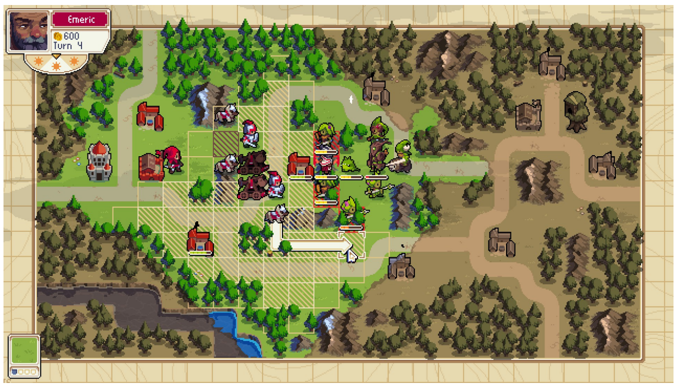

Figure 1: A screenshot showing the gameplay of Wargroove (Chucklefish 2019) running on Windows. The game world is a 2D tile-based map, and units on the battlefield are given commands via mouse clicks on GUI elements.

custom test scenarios specifically for this project, because the game itself has an inbuilt map editor that can be used to create custom maps and game scenarios. Apart from this, the game is GUI driven and is played using animated units that will move around the map during the course of the game, making it conveniently aligned with what the project hypothesis sets out to test. Another noteworthy point is that most actions in Wargroove, such as combat, have deterministic outcomes, which allow tests to have a specification of the exact expected outcome for a given action. Finally, the fact that it is a full-fledged game that was recently released commercially means that this project has the opportunity to validate that VGT is applicable in real-world industry game development projects.

In order to develop the proposed testing tool, this study will need to begin by conducting an investigation into applicable computer vision methods. OpenCV (Bradski 2000) will be used to support the implementation of the tool. To be able to test the fundamentals of Wargroove, the tool should be able to do the following.

- Locate, interact with, and make assertions on animated units visible on any section of the battlefield.
- Locate, interact with, and make assertions on components of the game HUD and in-game menus.
- Identify and interact with highlighted map tiles. Highlighted tiles indicate that a selected unit can be given a command on those map tiles.

Given these capabilities, automated tests can be written to verify the following aspects of the game, among others.

- All available friendly units are able to be selected and given a command during the player's turn.
- When the player hovers the mouse over a unit or tile, descriptive information is displayed on the HUD.
- The HUD always displays an icon of the player's commander unit and how much gold the player has.
- When a unit is given a move command, it moves to the specified location.

As an evaluation, manual debugging techniques will be used to verify that the VGT tool is supplying the automated tests with the correct coordinates for each GUI component used by the tests.

The main risk in undertaking this project will be getting a viable VGT tool implemented in the given time frame. Should the tool not be finished in time, Sikuli (Chang et al. 2010) can be used as a fallback framework. Although this is not ideal, given that it is not designed to work specifically with games, it is an open-source project, so changes could potentially be made to it if necessary.

4. CONCLUSION

Computer games are highly interactive and dynamic software applications. Apart from allowing players to achieve the same thing in a multitude of different ways, game designers will modify game parameters frequently throughout the lifecycle of a game. Because of this, test fragility is one of the key issues holding the games industry back from using more test automation in computer game development. This research proposal has laid out a plan to investigate whether visual GUI testing, a technique that uses computer vision to locate GUI components on screen, can be used to write automated tests for games that are robust enough to handle changes in the layout of a GUI, and the presence of other animations.

The anticipated result is that VGT will be shown to be a viable technique for creating robust automated tests for games. Given that the games industry is lagging behind the rest of the software engineering world when it comes to adoption of automation, this study could potentially help the games industry make tangible progress towards finally being able to reap the well known benefits of automation.

5. PROJECT SCHEDULE

The proposed project schedule is illustrated in figure 2.

References

Alégroth, E. & Feldt, R. (2017), 'On the long-term use of visual gui testing in industrial practice: a case study', *Empirical Software Engineering* **22**(6), 2937–2971.

Alegroth, E., Nass, M. & Olsson, H. H. (2013), Jautomate: A tool for system-and acceptance-test automation, *in* '2013 IEEE Sixth International Conference on Software Testing, Verification and Validation', IEEE, pp. 439–446.

Borjesson, E. & Feldt, R. (2012), Automated system testing using visual gui testing tools: A comparative study in industry, *in* '2012 IEEE Fifth International Conference on Software Testing, Verification and Validation', IEEE, pp. 350–359.

Bradski, G. (2000), 'The OpenCV Library', *Dr. Dobb's Journal of Software Tools* .

Buhl, C. & Gareeboo, F. (2012), Automated testing: a key factor for success in video game development. case study and lessons learned, *in* 'Proceedings of the 30th annual Pacific NW Software Quality Conferences', pp. 545–559.

Chang, T.-H., Yeh, T. & Miller, R. C. (2010), Gui testing using computer vision, *in* 'Proceedings of the SIGCHI

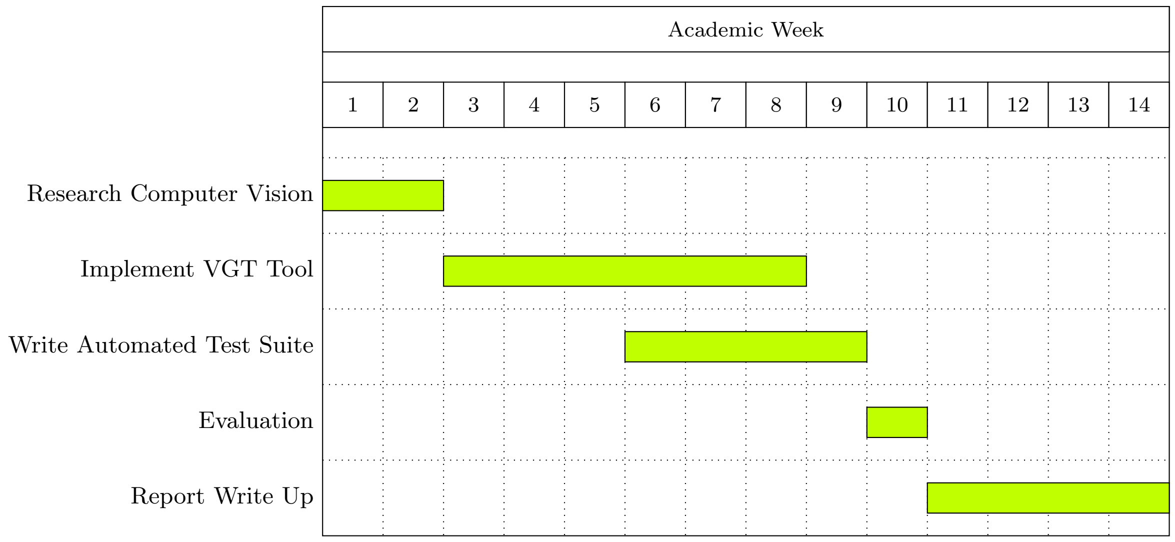

Figure 2: Gantt chart illustrating the project schedule.

Conference on Human Factors in Computing Systems', ACM, pp. 1535–1544.

Chucklefish (2019), 'Wargroove', [Video game]. Chucklefish.

Murphy-Hill, E., Zimmermann, T. & Nagappan, N. (2014), Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?, *in* 'Proceedings of the 36th International Conference on Software Engineering', ACM, pp. 1–11.

Santos, R., Magalhaes, C., Capretz, L. F., Neto, J. C., da Silva, F. Q. B. & Saher, A. (2018), 'Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners', *arXiv preprint arXiv:1812.05164* .