



Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

Introducing Education on Test Driven Development into the Pytch Learning Environment

Matthew Lynch

Supervisor: Professor Glenn Strong

April, 2022

**A dissertation submitted in partial fulfilment of the requirements for the
Final Year Project of Masters Year of Electronic & Computer Engineering**

"I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at <http://www.tcd.ie/calendar>.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at <http://tcd-ie.libguides.com/plagiarism/ready-steady-write>."

ABSTRACT

The education of young and novice programmers is an ever-evolving field due to the prominence of new programming languages in the information industry. Current introductory programming environments are dominated by web-based tools such as MIT's Scratch service (Scratch Foundation, 2021). The goal of the Scratch platform is to offer engaging learning content targeted at younger users and children through block based programming in order to introduce the basics of programming logic and hopefully encourage the next generation of programmers. However, there is a gap in this educational field from the step from Scratch-like environments into the industry standard tools that are taught and used in Computer Science courses at third level education today. These introductory platforms are also only used to form a base knowledge of programming logic in the minds of new programmers where there is potential for introducing other industry standard practices such as Test Driven Development.

Pytch, which forms its name from a combination of 'Python' and 'Scratch', is a web-based programming environment centred around the education of the Python language through editing and compiling python code in browser, but also offers Scratch-like block component descriptions of its functions. This allows the platform to be a steppingstone platform for novice programmers with experience with Scratch to better adjust themselves to actual code editing in Python, rather than using the Scratch block based framework. The currently under development Pytch platform already has a series of pre-built projects that can be run as fully completed projects in the Pytch environment or they can be experienced through Pytch's tutorial environment which offers a step by step guide on the chronological building of the project.

This project analysed the current tutorial structure of Pytch by studying its design from both a frontend website and also backend application perspective through what documentation material was available and also from investigating the file and system structure manually. The goal of this project is to provide a feasible evolution of the Pytch tutorial system to include tutorial chapters that include practices of Test Driven Development. This is achieved by gaining an understanding of Pytch's tutorial design and structure so that new chapters and new features could be integrated into an existing Pytch tutorial.

This project implemented and evaluated the use of Python assertion tests within the Pytch environment to ensure that they communicate correctly with the Pytch web terminal. A visual proof of concept was then created by modifying the html of a currently existing tutorial chapter to evaluate the best practice for delivering the information on Test Driven Development. Finally, a functional proof of concept was created that offered an existing Pytch tutorial with chapters on Test Driven Development integrated at certain parts of the tutorial. The tests were successfully implemented along with their corresponding tutorial chapters and an outline of what was learned about forming the optimal tests for inclusion in educational work intended for novice programmers was also included.

The results of this project show that the inclusion of Test Driven Development is possible within the existing Pytch educational environment and also shows on a wider scope that it is feasible for developers of educational projects to include an introduction to Test Driven Development in their work by forming tests to aid in not only the education of the language they are working with, but also allows them to design these tests to be used for checks on the student's code that are closely related to the individual project rather than simply using built in syntax error checking that would come with a standard compiler.

ACKNOWLEDGEMENTS

I would like to thank Professor Glenn Strong for his guidance over the course of this project. His support, advice and expert knowledge of the Pytch environment played a vital part in the achievements of this project especially in the aid of setting up a virtual environment that could run Pytch from a Linux environment on my local machine which was an important factor in this project's development stages due to these strange times. The meetings throughout the year where you warmly shared your wisdom helped keep the focus on the work at hand and made the process enjoyable so thank you for that.

I would also like to thank Dr Ben North for allowing me access to Pytch as an environment to study it and base my project work around it. The platform has a lot of potential and the work you have done so far is not only useful but was also easy to understand when I went to look under the hood at how Pytch operates.

Both Glenn and Ben were open to lending support and guidance to this work if needed and I am very grateful for that.

I would like to say thanks to my friends that have supported me throughout this Masters year with your words of encouragement throughout the process especially when I was at my busiest.

Finally, I would like to thank my family, especially my parents who have been extremely supportive of my studies this year. While at times staying focused and optimistic was a struggle especially when contracting Covid in the middle of the year, your belief in me and support is something I will always treasure.

CONTENTS

1	Introduction	1
1.1	What is Pytch?	1
1.2	Project Objectives	2
1.3	Ethics & Gender Dimension	3
1.4	Work Plan.....	4
1.5	Potential Beneficial Outcomes of this Project	4
1.6	Project Scope	5
1.7	Project Results	5
1.8	Dissertation Roadmap.....	6
2	Background	7
2.1	Literature Review	8
2.1.1	Web-based Python Programming Environments	8
2.1.2	Educational Environments with Test Driven Development	10
2.1.3	State of the Art of Existing Testing Suites	11
2.2	Pytch Frontend and Tutorial Layout	12
2.3	Current Architecture of Pytch Backend	16
2.3.1	Pytch-Build	16
2.3.2	Pytch-Tutorials	17
2.3.3	Pytch-VM.....	17
2.3.4	Pytch-Webapp.....	17
2.3.5	Pytch-Website	17
2.4	Pytch Tutorial Creation and Compiling	18
2.4.1	Tutorial File Structure	18
2.4.2	Github Tutorial Commits.....	19
2.5	Tests in Pytch	20
2.6	Chase Tutorial	21
3	Design & Implementation of TDD into Pytch Tutorials.....	22
3.1	Final Test Design	22
3.2	Running Pytch Locally	23
3.2.1	Window Subsystem for Linux.....	23
3.2.2	Oracle Virtual Machine	24
3.3	Testing of Python Tests in Pytch Environment	25

3.3.1	Python Assert Testing in Pytch.....	25
3.4	Visual Proof of Concept Through Tutorial HTML	26
3.5	Functional Proof of Concept Through altering Tutorial	27
3.5.1	Design Decisions	27
3.5.2	Implementation Process	28
3.6	Chase Tutorial's new Integrated Tests.....	29
3.6.1	Background Image Validation	29
3.6.2	User Input Sanity Checks.....	30
3.6.3	Verifying the Operation of the Random Function	31
4	Evaluation	32
4.1	Evaluation of WSL versus Oracle VM	32
4.2	Implementation Evaluation	33
4.2.1	Validation of Assertion Tests	33
4.2.2	Visual Proof of Concept of Test Chapter	33
4.2.3	Functional Pytch TDD Chapters.....	34
4.3	Test Driven Development in Environments for Novice Programmers	35
4.3.1	Simplicity First	35
4.3.2	Relevant and Engaging Test Content	35
4.3.3	Setting Up a Strong Foundation for Understanding	36
4.3.4	Comprehensive Error Messages	36
4.3.5	Ensure the Tests Improve the Project's Validation.....	36
4.3.6	Summary	37
5	Conclusion.....	38
5.1	Future Work	39
	Bibliography	40

LIST OF FIGURES

Figure 2.1 Pytch Homepage	12
Figure 2.2 Pytch Programming Environment with each section segmented in red	13
Figure 2.3 Pytch Functions with Scratch Like Blocks representing their functionality	14
Figure 2.4 Pytch Tutorial Environment	15
Figure 2.5 tmux Window running Pytch-build scripts.....	16
Figure 2.6 Example of Pytch Tutorial File Structure.....	18
Figure 2.7 Tutorial Chapter defined in a Pytch tutorial.md file	19
Figure 2.8 Screenshot of Chase tutorial output canvas	21
Figure 3.1 Image of Failing Pytch Compile on WSL.....	23
Figure 3.2 Oracle VM running Pytch App on Windows Machine.....	24
Figure 3.3 Example of the Inclusion of a Python assert test in Pytch	25
Figure 3.4 Screenshot of the Visual POC for TDD Tutorial Chapter in Pytch	26
Figure 3.5 Snippet of List of Tutorial Branches	28
Figure 3.6 Background Image Validation Test	29
Figure 3.7 User Input Movement Test	30
Figure 3.8 Verification of Random Functionality Test	31

LIST OF TABLES

Table 1 - Brief Outline of Project Work Plan	4
--	---

NOMENCLATURE

TDD	Test Driven Development
POC	Proof of Concept
CS	Computer Science
VM	Virtual Machine
OOP	Object Oriented Programming
WSL	Windows Subsystem for Linux
IDE	Integrated Development Environment
GUI	Graphical User Interface
HTML	HyperText Markup Language

1 INTRODUCTION

Computer Science education is an ever-evolving area of study and one that is vital in today's world as there is such a high demand for experience with programming languages. Many courses outside of Computer Science departments, especially business-related courses have come to include courses on beginner programming in recent years. The label of a 'novice programmer' is one that is almost to be considered a life skill in the modern world and the age at which Computer Science education is being introduced has become younger in recent years due to more child friendly education environments sprouting up across the globe.

With the increasing requirement for workers with experience in technology and computer science, there comes an increasing demand for educational material as well. Test Driven Development is a standard practice in the modern development architecture and the earlier a programmer is introduced to the concept of diligently testing their code the more comfortable and successful they will be when they go to work in the field of Computer Science.

The goal of this project was to evaluate the current position of a web-based programming environment that is targeted at novice programmers and determine whether the integration of education on Test Driven Development (TDD) is a plausible feat.

1.1 WHAT IS PYTCH?

Pytch is a web-based educational environment for teaching novice programmers the basics of the Python language. Its name and its core aesthetic takes inspiration from Scratch, MIT's child-friendly block based educational platform for introducing novice programmers to code logic through engaging projects. The name Pytch is a combination of both Python and Scratch, and likewise offers an engaging environment that allows users to compile simple projects in order to learn some of the basics of the Python language.

The compilation of Python code in the browser is made possible by the JavaScript package Skulpt (Skulpt, 2019), this package can be used to create custom python functions and offers the functionality to create a web environment for compiling Python code.

Each project available on Pytch offers two options, one simply allows the user to compile the fully completed project and observe and engage with the output project. The other more interactive option allows the user to follow a step-by-step tutorial of the same project where each component and class used in the project is explained and can be copied straight into the coding environment.

The success of Scratch has shown the effectiveness of compiling coding projects in an online web-based environments as it removes all potential package versioning or path related issues that a novice programmer would have to deal with if working on the same projects through an IDE on their own machine (Lappalainen, et al., 2010). Keeping the environment as user friendly as possible allows all the focus of the student to go directly towards understanding the basics of the Python language which is key to efficient teaching and learning especially for beginners.

Pytch is set up to allow teachers to either use currently available projects or even form their own projects to be completed by their students. The design of the system for creating and compiling Pytch tutorials will be explained further in Chapter 2.

1.2 PROJECT OBJECTIVES

The objective of this project was to determine useful interactive testing features that are not currently available in the Pytch Tutorial system and to implement a working version of those new features. After studying the state of the art and the current position of Pytch with its tutorial based education structure, it was determined that the inclusion of Test Driven Development was the best path to take for this project.

In order to complete this project, some of the following research objectives needed to be reached;

- Become familiar with the Pytch environment.
- Gain experience with using JavaScript.
- Learn about Scratch and Skulpt (Scratch is what inspired Pytch and Skulpt is an 'in-browser implementation of Python' that was used to build Pytch).
- Study the state of the art for other Python web based programming environments and educational environments for programming and TDD in education.

The development goals for this project are as follows.

- Determine the optimal way to include TDD into the Pytch Tutorials.
- Create a visual proof of concept for the inclusion of TDD chapters in an existing tutorial.
- Create tests for existing example tutorial to verify the possibility of TDD education in Pytch.
- Create a functional iteration of a current tutorial.
 - Includes multiple chapters on test driven development.
 - Ensure each contains tests on various parts of the respective tutorial.

Then there are the more general project goals of ensuring the project material will be ready to present and submit by the given deadlines.

1.3 ETHICS & GENDER DIMENSION

The topic of Ethical and Gender related challenges is important as while they can be challenging to pinpoint, they are always present in projects such as this, especially due to the educational nature of Pytch. The gender concern is more conventional as there will always be a level of gender bias when a project is created by a group where all members are of one gender. The example tutorials on Pytch are sufficiently gender neutral with regards to gender stereotypes however due to the nature of Pytch as a tool, it could be beneficial to have some sort of reminder to new teachers/professors who are about to create their own tutorial to be mindful of the theme of their work. There has been countless studies done over the years that exhibit the clear bias in projects across Computer Science (International Society for Technology in Education, Eugene, 1990) and it is vital that a tool such as Pytch that is targeted at novice and beginner programmers is not deterring anyone from engaging with the learning because of the material. While this is not directly related to the work that will be done for this project it is an important point of thought for any work in programming education so it's necessary to discuss.

There are a few ethical issues involved in this project that are worth noting, one issue that was a repeating occurrence in the literature review was the consequence of the overuse of automatic grading on student assignments. If a project is to be entirely assessed automatically it will heavily constrict the creativity of the students and almost remove any potential for originality or individuality from submissions from a body of students. While this would maximise the efficiency of the assignment process, it also forces current and future work to fit into a box created by the testing that can't be expanded upon. The overuse of automatic grading may lead to the absence of any abstract or unique ideas and approaches to coding projects which is an ethical issue. Unique solutions should be encouraged from the very beginning of programming education and this is where Pytch sits as a tool on the grander timeline of a programmers academic path. Pytch's use of tutorials to communicate the learning material also limits the user's creativity however the coding environment does allow for code to be modified so as long as the student's Python syntax is correct the project won't fail. Essentially there is potential for more abstract and creative work to be done by the student in the Pytch environment but there isn't currently any encouragement to do so. Creating tests that encourage user interactivity was an important takeaway from this analysis that was kept in mind during the test design.

Another more general ethical concern is to ensure that the developed product is stable and easy to use, especially since the target audience for this environment is novice programmers. It's vital that the tutorials and grading/feedback system are clear and are not confusing otherwise the environment won't prove useful in an education setting. It's important for both parties (users and developers) that Pytch serves its purpose in an efficient manner. The users will not want their time wasted by poorly coded elements and the developers will want feedback to offer important changes that can benefit the platform rather than the noise of bug reporting in the feedback of users.

1.4 WORK PLAN

A brief outline of the core actions taken during this project can be seen in Table 1 below. The project was planned to have the majority of the research work done in semester one in parallel with the setup work around running a local instance of Pytch on a personal Windows laptop. The general learning objectives such as studying JavaScript and researching the state of the art were carried out throughout the whole year. The main implementation work of the project was then carried out in semester two after a sufficient amount of learning was done surrounding Pytch and its different components.

	Task	Time
1	Learn Javascript	Throughout Project
2	Research TDD and Python Testing	Throughout Project
3	Get a local development version of Pytch to run	First Semester
4	Study Pytch Documentation	First Semester
5	Study Pytch Tutorial compiling setup	First Semester
6	Practice Github project rebasing	Second Semester
7	Create a Visual POC of Test Chapter in Pytch	Second Semester
8	Create a Functional Integration of Test Chapters into Pytch Tutorial	Second Semester

Table 1 - Brief Outline of Project Work Plan

1.5 POTENTIAL BENEFICIAL OUTCOMES OF THIS PROJECT

The potential benefits of this project include the prospect for an evolved tutorial system on the Pytch platform that leans into education on Test Driven Development and allows for a more engaging experience for students as they make their way through the tutorials. The inclusion of Test Driven Development at this early stage of novice programming will also benefit the students in the long run when they go to work in the software development industry in the future. A successful implementation of these tests could also open a new door for the education process on Pytch where the projects could become more assignment-like rather than tutorial based as future developers could potentially use this testing structure to evaluate a student's performance. The experience of designing optimal tests for an education environment outlined in the evaluation of the project work could also be used to influence future work that may be completed in the same field.

1.6 PROJECT SCOPE

The development and implementation work of this project is focused solely on the development of the tutorial system within Pytch specifically. The developments and evaluations made during this project are tailored to the ongoing development of the Pytch environment. The issues and obstacles encountered during this project are likely to be niche to Pytch's design. A big focus of the work completed was to determine what was currently available to be implemented within the Pytch environment to create the Proof of Concept tutorial chapters. The idea would be that similar tests could be built for all current and future Pytch tutorials. The Test Driven Development chapters built were designed specifically for the chosen Tutorial and it is highly likely that the best tests to use in other tutorials would differ from the ones created for this project.

However, there are greater concepts and ideas that are learned about the procedure of including Test Driven Development that could be viewed from a wider angle than just Pytch as a platform and more so as general learning as to what works when it comes to the designing and implementation of Test Driven Development in educational material targeted towards younger students and novice programmers.

1.7 PROJECT RESULTS

This project was successful in demonstrating the feasibility of using Python tests in the Pytch environment. A functioning proof of concept of Test Driven Development integration into a Pytch tutorial was created and also proven to work successfully. After designing the tests to be added to an existing Pytch tutorial, the core characteristics of what makes a useful test for Test Driven Development in a learning environment were evaluated and outlined to be used as a potential future guideline for developers looking at integrating education on testing into their own work.

1.8 DISSERTATION ROADMAP

Chapter 2 of the dissertation describes the background work of the project. The background work includes a literature review of the state of the art for both web-based python programming platforms and the use of Test Driven Development in Computer Science education. The current design of Pytch will be investigated with a particular focus on the Pytch Tutorial design both from a frontend and backend perspective. Chapter 3 will then give an overview of the development stages for the inclusion of Test Driven Development chapters inside an existing Pytch tutorial. Each step is discussed and any obstacles that were identified and dealt with during the implementation process are addressed. Chapter 4 is an evaluation of the projects work, primarily addressing the success of the integration of Test Driven Development chapters within the chosen Pytch tutorial. An evaluation of what was learned through the process of designing suitable tests is also laid out in this section. Chapter 5 is then a summary of the work done during this project, what was achieved and concluding analysis. There is also a future work section that highlights where this work could be taken.

2 BACKGROUND

Pytch is a web-based programming environment that promotes Python education for novice programmers. In this section, a literature review will be done on the state of the art of web-based python programming, plus a review of work that has been done to date on using Test Driven Development and automatic testing in educational environments.

Following that state of the art the section, an insight into the current frontend layout for a Pytch user will be given, then an overview of Pytch's system architecture will be highlighted and finally the backend structure of creating and compiling tutorials will be examined.

The background section will end with an investigation into tests currently being used in the Pytch backend and then a brief summary of the content of the 'Chase' tutorial that is modified to contain this project's implementation of Test Driven Development tutorial content that is described in Section 3.

2.1 LITERATURE REVIEW

Due to the context of this project, the research spanned different areas of state of the art approaches of CS Education. It was essential for this project to study the state of the art of other web-based Python environments and compare them to the current state of Pytch. Research was also done into any material that could be found on the use of Test Driven Development in CS education.

2.1.1 Web-based Python Programming Environments

The first area that was investigated as part of the literature review was the state of the art of currently existing programming environments used for the education of programming languages. While the primary point of interest in this state of the art was environments that included integrated testing or automatic grading, it was an interesting field to study as it allowed for a comparison of Pytch to other similar tools and even some competitors.

One such example called APOGEE offers an automatic project grading and instant feedback system within an environment (Fu, et al., 2008) that was designed by the authors. This paper highlighted the necessity and potential complications of creating a platform including the automatic grading of student projects. The main issue that was highlighted in this particular paper is the potential difficulty of creating a uniform grading system that will cater for all student submissions. The challenge that the APOGEE platform faced was dealing with GUI tests. Creating tests for a project in which the students would set their own names for certain HTML components proved impossible. The APOGEE system used Ruby testing scripts with set names for HTML controls so any difference in these control names would cause the testing script to fail. Their solution to this was to simply provide a *'GUI project "shell" (without detailed logic implementation)'*.

GUI elements take up a significant part of the current Pytch tutorials and most educational tools as they make the education process more engaging, this means that the issues highlighted here surrounding GUI testing will bleed into Pytch also. One key motivator for the automatic grading of computing projects highlighted in this paper was efficiency. The fast evaluation that automatic grading and feedback offers allows for the education process to speed up, automatic grading and testing removes the tedious process of an instructor or teacher being required to manually check all aspects of a students work. The APOGEE platform is targeted at college courses which means that any time that can be saved is of high value, especially for large college classes. Pytch is mainly targeted at beginner Python programmers who are most likely to have experience with Scratch-like environments (secondary school or early Computer Science courses) which is a large demographic, so this same principle of time efficiency does apply for Pytch.

CodeSkulptor was another similar tool that offers a web-based environment for Python projects through Skulpt (Skulpt, 2019). This tool is more so angled towards programmers with a higher proficiency of Python programming as it is less of an educational tool and more of a web platform for showcasing student projects. One paper that was aimed at creating and integrating an execution visualizer for CodeSkulptor (Tang, 2015) offered a short description of how unit tests are carried out through Skulpt and for the visualiser mode created as part for their project. This paper again highlighted the challenges of testing GUI related code, it explains how GUI changes and manipulations are difficult to quantify and record for comparison purposes. It also mentions how different browsers can cause elements to behave differently, the conclusion was that manual testing is still a requirement for any GUI related assessment.

Further research into the state of the art of educational coding environments uncovered other projects and papers that offered a similar service as Pytch does. Pythy (Edwards, et al., 2014) is a platform for web-based Python programming that is also targeted at novice programmers. Finding this similar platform was a massive help in comparing Pytch to the state of the art. The most vital thing this paper offered was that Pythy does include a grading tool and feedback. A small survey was also completed at the end of this paper on two small classes of novice students, the most common response for Pythy's 'most used feature' and 'saved the most time' was auto project testing. This highlights the positive effects of a working feedback system within the environment and was a huge affirmation of the motivation behind this project. Another important outcome of the survey on Pythy was that the 'most frustrating' component of the environment was unclear errors, unfortunately the paper doesn't give much more information about this feedback or context for this, but it is an important point that error messages for these environments must be clear and well-articulated as the targeted demographic is novice programmers who won't deal well with advanced technical errors. This is something that played a large factor in the design decisions made for the tests implemented in this project.

2.1.2 Educational Environments with Test Driven Development

Another area that was investigated as part of the background of this project was literature that involved comparing established teaching methods with test driven development for programming education (Goodwin & Drange, 2016). This paper discussed an environment called 'T-FLIP' which was used to educate novice programming students on the Java language. The paper discusses a study in which two similar courses from nearby institutions in Norway where one class uses more test-driven development while the other uses '*a more standard approach*' to teaching programming that requires manual evaluation from the teachers. The conclusion of the study suggested that there are both benefits and disadvantages to using test driven development instead of current standard practices, the biggest advantage being exposure to test driven development at an early stage in programming education as it is an industry standard. The disadvantage to test driven development and automatic testing is that the rigid nature of these tests does not allow for much creativity from the students. The tests are strict and looking for definitive results from required project features, meaning any passionate students who have a desire to go above and beyond with their project will actually find their marks hurt by this system. This is an unfortunate unavoidable consequence for programming projects that are solely assessed using automatic testing. The other big concern that was uncovered from this study was the realisation that there is a high possibility for cheating and plagiarism with an automatic testing system over an experienced programmer evaluating the projects. This is a concern if the entire process is controlled automatically, however as mentioned in the paper, there are many plagiarism detection tools available today that can run checks for similarities between student's work.

The comparison between the two forms of education determined that the automatic testing and test driven development system offered potential for students to work and receive useful feedback immediately and also independently, they didn't have to wait for their professors to evaluate the work, while there is a loss in flexibility for the outcome of a student's programme and also a risk of unmonitored plagiarism, the general increase in efficiency and the opportunity for infinite re-evaluation of student work left the authors with the conclusion that TDD and automatic testing is a highly beneficial part of programming education.

Another paper based around the inclusion of Test Driven Development in an '*adaptive learning environment*' (Chien, et al., 2007) concluded similar findings surrounding the potential rigid and '*inflexible*' nature of Test driven development when forming more complex tasks. The findings were that the use of TDD in grading is that it encouraged students to initially look at output correctness rather than good coding performance which is not desired when trying to teach the students the full process of project development.

2.1.3 State of the Art of Existing Testing Suites

The state of art of currently used tools for automatic grading was also researched. There were many papers comparing different testing tools and suites, some of which do a general comparison of many different testing tools with regards to certain parameters (Caiza & Del Alamo, 2013) while others do a more in depth analysis of one or two tools i.e. (Heckman & King, 2018).

One paper provided an overview of many of the most popular testing tools at the time for different languages (Caiza & Del Alamo, 2013). The analysis is split into what are labelled as ‘mature’ tools and ‘recently developed’ tools, most of the recently developed tools examined in this 2013 paper were Java centred testing tools which won’t be of much use to Pytch, however some of the mature tools such as Web-CAT (Virginia Tech, 2021) which was made by the same group that made Pythy (Edwards, et al., 2014). Web-CAT is referenced quite often among literature that centres around grading and testing for web-based programming and is regularly highlighted as one of the most versatile tools due to how customizable and extensible it is. It offers a *‘plug-in-style architecture’* to allow professors to create their own tests and grading systems for their web-based assignments and exams.

2.1.4 Literature Review Conclusions

Overall, the takeaways from the analysis of current python web environments were that code testing is a vital component of an efficient programming education system, however, caution must be taken when dealing with GUI tests as they can be unreliable across browsers and are very sensitive to variable name changes. The Pythy (Edwards, et al., 2014) survey also showed how appreciated the automatic feedback is by student users however great care must be taken when forming feedback messages so not to confuse any novice programmers.

The opinion of automatic grading and testing in programming education is positive but there are also many warnings that are made apparent across the literature. This style of testing and grading does greatly improve the efficiency of the learning process and also provides an early introduction to industry standard processes. The issues that are met with using automatic tests are mainly centred around the somewhat constrictive programming nature that automatic grading creates if used in excess. The takeaway is that the motivation for this project is valid and Pytch would benefit from testing and feedback features, but it will be important to create an environment that can allow for engaging tests that fit well with the teachings of any given Pytch tutorial. These are conclusions that had a significant impact on the designing of the tests for this project.

2.2 PYTCH FRONTEND AND TUTORIAL LAYOUT

This section is to exhibit the Pytch platform so that a better understanding of the current state of Pytch and its tutorial structure can be reached. The Pytch platform offers many ready made python projects that involve a high level of interaction for students who are new to Python programming to engage with their learning process. These projects are offered through the Pytch website and are served in a web-based environment. The front page of Pytch can be seen in Figure 2.1.

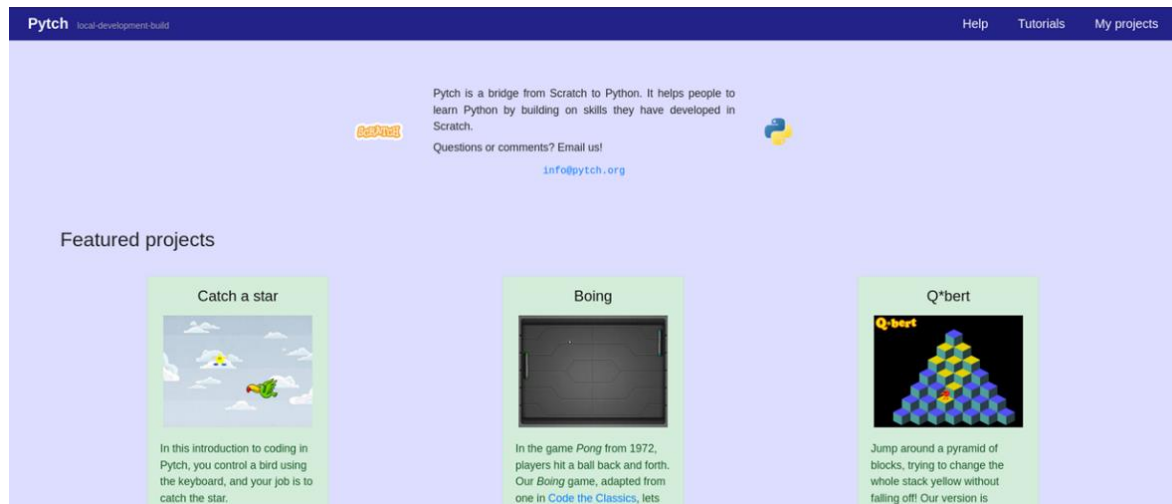


Figure 2.1 - Pytch Homepage

This dashboard allows user to freely navigate through a variety of different python tutorials and also offers links to the Pytch documentation (North & Strong, 2022) and access to previously worked on projects.

The environment shown in Figure 2.2 is the fully completed 'Pong' game project with each segment of the Pytch project environment separated with a red border. The Pytch environment offers an editable coding environment, a canvas for demonstrating the output of the current coding project and a section at the bottom of the page to display the terminal messages plus any static material used in the project.

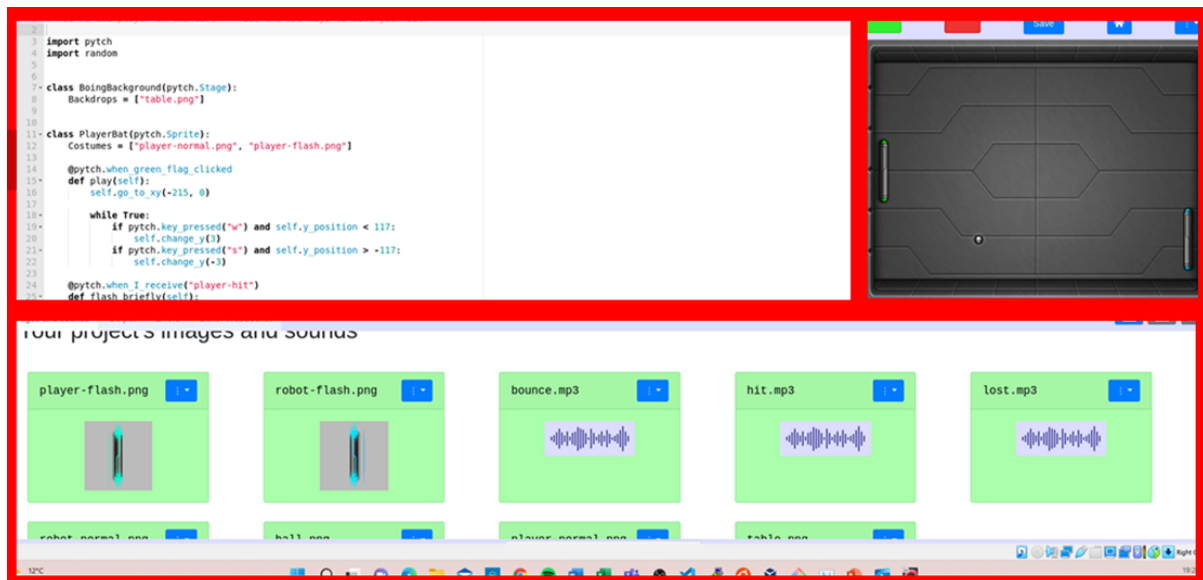


Figure 2.2 Pytch Programming Environment with each section segmented in red

This design allows for users to make changes and see those edits exported in real time, the canvas will show any working changes and if there is an error in their code the bottom panel will change to the error terminal and the code error will be highlighted to the student.

The current range of Pytch projects and custom components that have been built are focused on event based programming and the use of Sprites (North & Strong, 2022). This style of programming was a point that had to be kept in mind when deciding what structure should be given to any new chapters that are to be included in this environment.

There is also a tab that can be viewed that shows a Scratch-like block representation of different Pytch functions. This offers users with a prior experience with Scratch the opportunity to understand the step between reading Scratch blocks to reading Python decorators which can be seen in the 'Events' section of this Pytch sidebar tab in Figure 2.3.

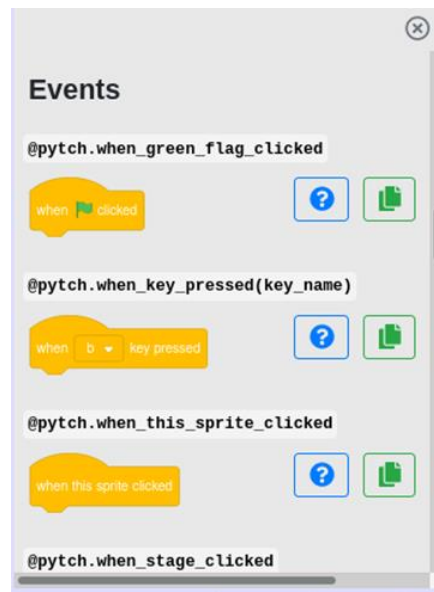


Figure 2.3 Pytch Functions with Scratch Like Blocks representing their functionality

The tutorials are laid out the same way a normal fully finished project would be except the main tab at the bottom contains traversable chapters that show the project completed chronologically with each step comprehensively explained with the intention of allowing novice programmers to follow along comfortably (Figure 2.4).



Figure 2.4 Pytch Tutorial Environment

This tutorial environment is where the implementation of this dissertation resides, however this is only the frontend layer to what is an intricate system that needs to be understood in order to start making any real change to the platform.

2.3 CURRENT ARCHITECTURE OF PYTCH BACKEND

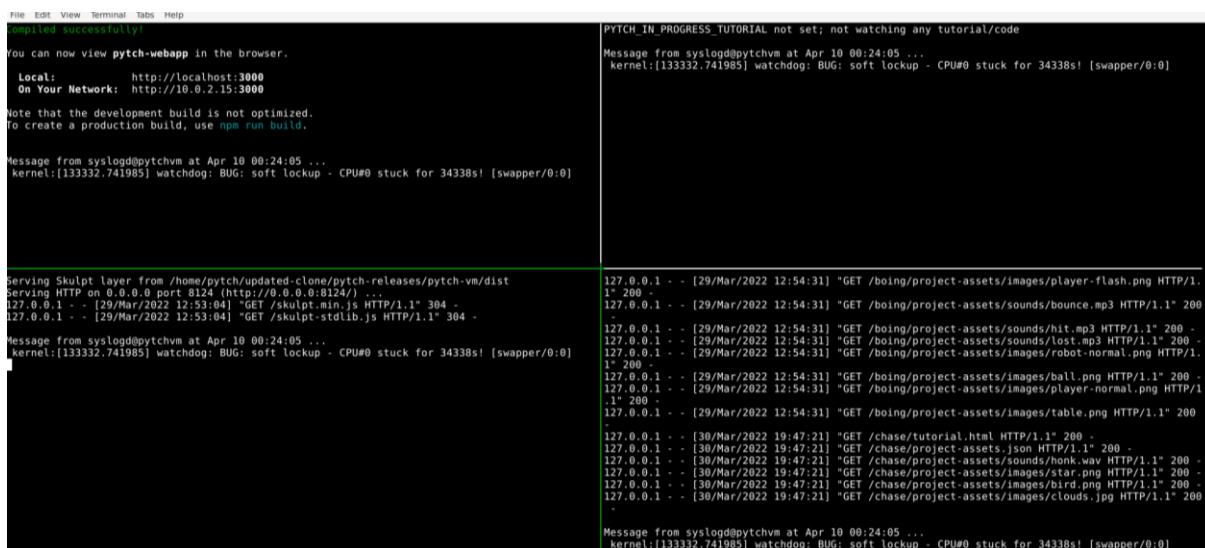
Pytch is developed on Github and all modules and submodules of Pytch exist within the Pytch Github repository (North & Strong, 2022). There are five core submodules that are all kept under the umbrella of the pytch-releases 'superproject' that is used to run the whole Pytch app together. The five submodules are as follows:

- **Pytch-releases**
 - **Pytch-build**
 - **Pytch-tutorials**
 - **Pytch-vm**
 - **Pytch-webapp**
 - **Pytch-website**

These submodules all offer a different functionality in the composition of the Pytch environment and each will be individually highlighted (North & Strong, 2022). It is to be noted that the following process is related to running the developer version of Pytch on a local machine.

2.3.1 Pytch-Build

As the name suggests the pytch-build submodule is what initiates the building and running of the developer version of Pytch on a local server. The scripts that are run within pytch-build will initiate a terminal multiplexer session using tmux (Figure 2.5) which allows for multiple windows to show information coming from the different corners of the project. All the current tutorials that are to be shown in the environment will be called and hosted on a particular localhost port. The same is done for the custom Skulpt modules that are part of the Pytch build. The web-app will also be initialised from within these scripts and once all the different segments are up and running the Pytch website will be available from the base URL.



```
File Edit View Terminal Tabs Help
Compiled successfully!
You can now view pytch-webapp in the browser.
Local: http://localhost:3000
On Your Network: http://10.0.2.15:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
Message from syslogd@pytchvm at Apr 10 00:24:05 ...
kernel:[133332.741985] watchdog: BUG: soft lockup - CPU#0 stuck for 34338s! [swapper/0:0]
PYTCH_IN_PROGRESS_TUTORIAL not set; not watching any tutorial/code
Message from syslogd@pytchvm at Apr 10 00:24:05 ...
kernel:[133332.741985] watchdog: BUG: soft lockup - CPU#0 stuck for 34338s! [swapper/0:0]
Serving Skulpt layer from /home/pytch/updated-clone/pytch-releases/pytch-vm/dist
Serving HTTP on 0.0.0.0 port 8124 (http://0.0.0.0:8124/) ...
127.0.0.1 - - [29/Mar/2022 12:53:04] "GET /skulpt-main.js HTTP/1.1" 304 -
127.0.0.1 - - [29/Mar/2022 12:53:04] "GET /skulpt-stdlib.js HTTP/1.1" 304 -
Message from syslogd@pytchvm at Apr 10 00:24:05 ...
kernel:[133332.741985] watchdog: BUG: soft lockup - CPU#0 stuck for 34338s! [swapper/0:0]
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/images/player-flash.png HTTP/1.1" 200 -
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/sounds/bounce.mp3 HTTP/1.1" 200 -
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/sounds/hit.mp3 HTTP/1.1" 200 -
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/sounds/lost.mp3 HTTP/1.1" 200 -
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/images/robot-normal.png HTTP/1.1" 200 -
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/images/ball.png HTTP/1.1" 200 -
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/images/player-normal.png HTTP/1.1" 200 -
127.0.0.1 - - [29/Mar/2022 12:54:31] "GET /boing/project-assets/images/table.png HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2022 19:47:21] "GET /chase/tutorial.html HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2022 19:47:21] "GET /chase/project-assets.json HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2022 19:47:21] "GET /chase/project-assets/sounds/honk.wav HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2022 19:47:21] "GET /chase/project-assets/images/star.png HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2022 19:47:21] "GET /chase/project-assets/images/bird.png HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2022 19:47:21] "GET /chase/project-assets/images/clouds.jpg HTTP/1.1" 200 -
Message from syslogd@pytchvm at Apr 10 00:24:05 ...
kernel:[133332.741985] watchdog: BUG: soft lockup - CPU#0 stuck for 34338s! [swapper/0:0]
```

Figure 2.5 tmux Window running Pytch-build scripts

2.3.2 Pytch-Tutorials

The pytch-tutorials submodule is where a lot of the most frequent developments are made and also where this project's development resides the most (North & Strong, 2022). This is the sub-repository that holds all the iterations of the different tutorials that are to be made available. There is an 'index.yaml' file at the top of this directory that defines which iteration of each tutorial is to be called and used. Each tutorial and each iteration are stored as a separate Github branch of the repository. The process of how these tutorials are constructed and fetched is explained further in Section 2.4. The pytch-build will call for a zipped file of the current tutorial versions to be used and will then unzip them and serve them on a particular port as described in 2.3.1.

2.3.3 Pytch-VM

The pytch-vm submodule holds all the Pytch Python components that are built with Skulpt (Skulpt, 2019) as part of the Pytch library. All the different components created with Skulpt to create the Scratch-like components for Pytch such as the Stage and Sprite actors are housed here alongside all the unit tests that can be run (North & Strong, 2022). There are tests for the JavaScript side of Pytch but also python tests that are used to test the functionality of the Pytch custom components. This style of testing and its relevance to this project will be discussed further in the following Section 2.5.

2.3.4 Pytch-Webapp

The pytch-webapp submodule contains the React Application that defines the structure of the website and declares the structure of the pages and how they are connected. It also contains the styling of the different sections and components of the website. The react app served here also has a suite of Jest tests (Meta - Facebook, 2022) that verify its proficiency.

2.3.5 Pytch-Website

This submodule simply contains all the static content for the website that is non-IDE related, primarily centred around the Pytch documentation pages (North & Strong, 2022).

2.4 PYTCH TUTORIAL CREATION AND COMPILING

The core of the education process in Pytch is the tutorial chapter format in which it delivers the Python projects to students. This section will show the tutorial structure and the different pieces of a Pytch tutorial. How the tutorials are kept and served will also be highlighted.

2.4.1 Tutorial File Structure

Each tutorial follows an identical structure with the same file labelling system (Figure 2.6), there will be a `code.py`, `summary.md` and `tutorial.md` file for every tutorial. There will also be asset folders, one for any images or sounds files to be used in the project and another file for assets related to the summary card for the tutorial, for example a screenshot of the project output to be displayed beside the summary.

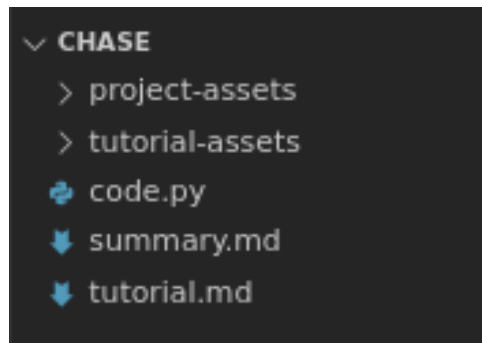


Figure 2.6 Example of Pytch Tutorial File Structure

The `summary.md` file is a small markdown file that will be used to fill in the content of the summary card for a given tutorial when a user navigates to the 'Tutorials' page on the Pytch application that lists all the available tutorials.

The `tutorial.md` file is what defines the different chapters of the tutorial and what content should be displayed within them, i.e. The headers, text content and code snippets.

The `code.py` file stores the final python project with all its elements. What allows for this file to be linked and segmented into different tutorial chapters is through Github commits.

Any change to the `code.py` file that is desired as a code snippet in a tutorial chapter will be given a commit message with a specific layout that is looked for by the tutorial compiler. Each code snippet for its respective tutorial chapter is defined in the `tutorial.md` file.

2.4.2 Github Tutorial Commits

A unique feature of a Pytch tutorial step is the ability for a student to observe the current code snippet within the tutorial chapter description and also have the ability to copy the code just as it is formatted into the Pytch code editor. This is all made possible by the use of Github and its code commit process.

All Pytch tutorials and all their iterations are held in inside the Pytch-tutorials Github repository as separate branches. In order to build a tutorial in Pytch, the specific branch name must be given to the tutorial compiler by declaring it in the aforementioned 'index.yaml' file (2.3.2). This branch will then be found in the repository and each commit will be iterated over until the tutorial is fully built and ready for rendering on the webpage.

However, there are many different changes that are made to these branches through development so the compiler was built to identify different chapter steps by giving these chapters specifically laid out commit messages that are structured as follows;

```
git add code.py
git commit -m "{#import-pytch}"
```

This structure of closed brackets with a hashtag symbol is how any step in the tutorial is identified by the compiler. It is important for the tutorial.md file to lay out the chapters in the exact same order as the order in which the commits appear in the repository.

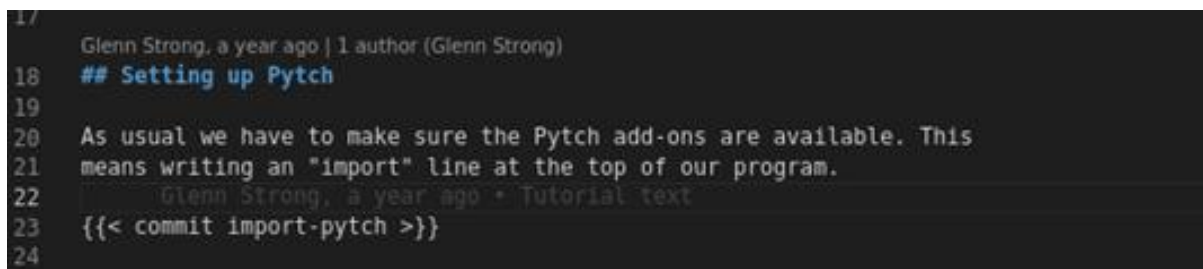


Figure 2.7 Tutorial Chapter defined in a Pytch tutorial.md file

The code snippet image shown in Figure 2.7 is an example of a chapter defined in the tutorial.md file. The design here exhibits the use of a title, then a brief description of what the code snippet will be used to perform and then finally the code snippet itself which would be loaded in using the observed format. The key takeaway from the study of this process is the possibility to insert new chapters into a currently existing Pytch tutorial if enough care is taken while undertaking a git rebase.

2.5 TESTS IN PYTCH

While the Pytch documentation does offer useful information about the structure of Pytch and how best to set it up locally, manual investigation was required to understand aspects of the environment such as where and how Pytch uses tests to validate its service. Uncovering this was a high priority for this project due to the objectives revolving around adding Test Driven Development to the tutorial experience.

The aim here was to identify where testing and unit testing is used in the application and to evaluate whether any inspiration could be taken from these tests for the development and implementation process of this project.

What was found was a healthy amount of test suites containing unit tests that are written in both JavaScript and Python. These tests are used to verify the functionality of Pytch components build with Skulpt for the Pytch platform.

There are JavaScript tests that validate that the logic behind the components defined using the Skulpt package work as they are expected to that are being validated using Jest (Meta - Facebook, 2022). For instance, Pytch offers a Python function called 'self.touching()' that verifies whether a sprite of the current class is currently colliding with another sprite that is defined as an input to this function. There is a test file that creates two objects in an environment and then ensures that this function outputs the correct Boolean value that is expected in different circumstances where the two observed objects are colliding or not.

On top of this JavaScript test file there is also a suite of Python unit test files that create smaller environments to test the same components but from the Pytch environment rather than in the background.

While these tests were beneficial for inspiration purposes, the actual system behind their use doesn't match up well with the intent of using these tests in real time as part of a Pytch tutorial. They are not part of the real time verification of user code in the Pytch environment but rather a validation of the functionality of Pytch components that would be run after making core changes to Pytch components outside of the production application. Therefore, they don't offer the level of response or feedback that this project's objective of integrating tests into the real time testing of student code inside the Pytch web environment.

2.6 CHASE TUTORIAL

The final section of the background is a brief overview of the 'Chase' or 'Catch the Star' tutorial that was selected as the tutorial to be used for the development stage of this project. This tutorial is an objected oriented programming (OOP) centred project that encourages the student to create a small interactive game in which the user has control to move a bird sprite around a 2D environment. Another star sprite is programmed to move randomly around the environment using python's 'random' function. When a collision between the bird and star is detected at any point the bird will output a 'Squawk' sound and the star will disappear and reappear somewhere else in the 2D environment.

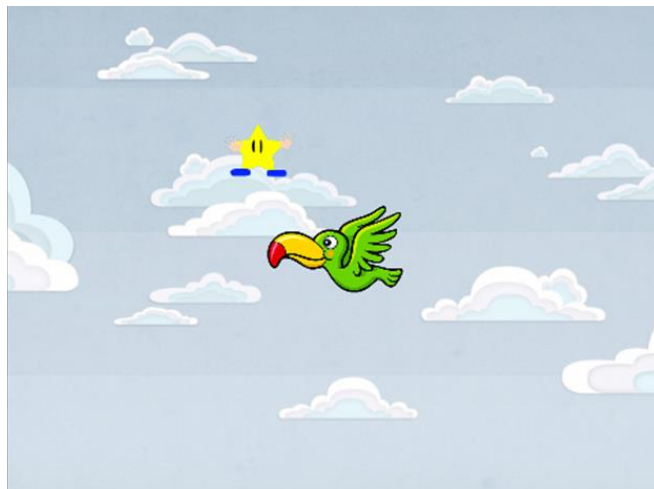


Figure 2.8 Screenshot of Chase tutorial output canvas

This tutorial is used to highlight the use of multiple Sprite classes in a 2D environment and to show the logic of collision detection in Python. Other useful features taught in this tutorial are the navigation and understanding of a 2D coordinate system in order to move objects around the screen using both user input and automatic methods.

The identified components of this tutorial's core teachings are the important foundation to build from in the design of the tests to be integrated into the project. The tests created for the Test Driven Development chapters are tailored for this tutorial specifically so that they fit in seamlessly with the intended teaching outcomes while also educating the users on the practice of Test Driven Development.

3 DESIGN & IMPLEMENTATION OF TDD INTO PYTCH TUTORIALS

The development work for this project consisted of some key milestones along the path to completing a final functioning proof of concept of Test Driven Development tutorial chapters in a Pytch environment. Each iteration of the implementation will be discussed in this section. The different iterations are as follows:

- Successfully run Pytch on a local machine for development.
- Testing the possibility of Python testing methods in a Pytch environment.
- Create a visual proof of concept of a chapter containing Python tests and a description of the step.
- Integrate Test Chapters into a Pytch Tutorial to create a functional proof of concept for TDD in Pytch.

3.1 FINAL TEST DESIGN

Through careful examination of the Chase tutorial, and with regards to the final conclusions drawn from the project's literature review in Section 2.1.4, the design of a set of test chapters to be added was decided upon. The general motive behind the design of these tests was to create engaging and useful tests that could be easily explained and understood in the tutorial, but also to ensure that no additional Pytch functions or components were introduced that were not already present in the current tutorial. The idea behind this being that the tests should only aid in the users learning of Python and Test Driven Development without adding any extra layers of confusion that already exist with the learning of a programming language.

The first test was designed to give a very soft introduction into the area of assert testing by simply testing that the background image for the project was the correct image file. While this test doesn't offer a huge amount of engagement for the user, it's an important first encounter with the principle of the inclusion of tests in the process of coding. The intention with this chapter is also to include an interactive optional task through a suggestion at the end of the chapter to change the background image in order to allow the student to witness the assert test failing and producing an error in the Pytch terminal.

The second test design involves more interactivity with Pytch components being used in the project and some 2D coordinate theory. This test involved verifying the position of the Bird sprite before and after the press of one of the arrow keys, then asserting the new position is correct relative to which key was pressed. This test felt very in line with the aspects of python being taught with this particular tutorial and fit the flow of the tutorial chapters nicely.

The third and final test added to this tutorial was a test that would verify that the Star sprite does move when it is supposed to. The Star sprite in this project is programmed to move randomly around the 2D environment and this test uses Pytch components to verify the stars position, waits a few seconds, then tests the Star's position again. The test would assert the stars position has changed since the previous check which therefore ensures that the star is successfully gliding around the environment as it should be at this stage of the tutorial.

3.2 RUNNING PYTCH LOCALLY

Pytch is an application that is still in development. At this stage of its build, Pytch has only ever been built to run on a Linux machine. This created an added challenge to the beginning of this project's cycle as there was some setup work that needed to be completed before the implementation work could get fully underway.

Due to the majority of this work being completed remotely due to the pandemic, it was essential that an editable version of Pytch could be accessed and modified to complete the objectives of this project.

Both WSL (Windows Subsystem for Linux) and a Virtual Machine using Oracle's VM application (Oracle, 2022) were both investigated as solutions and will both be discussed in this section.

3.2.1 Window Subsystem for Linux

When initially looking at running Pytch on a local windows machine the WSL system was the first thing that was looked to as it was readily available on the windows machine and didn't require any extra installations. While the Linux compatibility layer was successful at running, the Pytch application did not run successfully with errors stemming from the python virtual environment. In the end after several attempts and supervisor support the decision was made to redirect towards using a virtual machine instead.

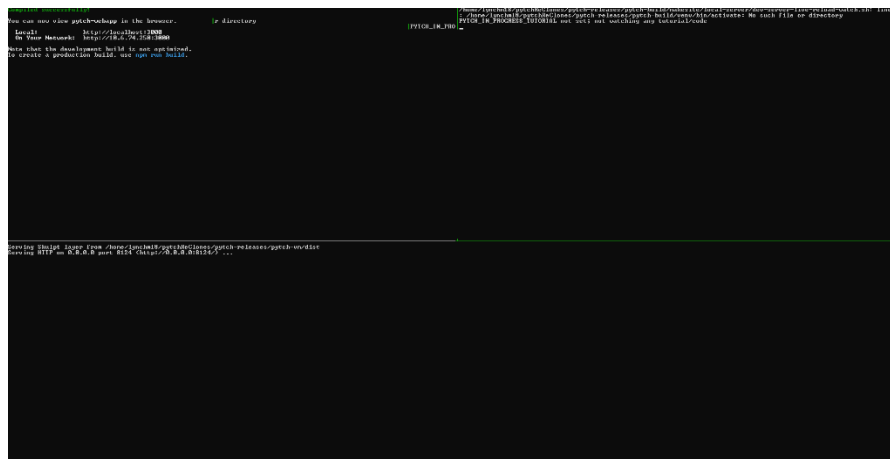


Figure 3.1 Image of Failing Pytch Compile on WSL

3.2.2 Oracle Virtual Machine

The Oracle Virtual Machine was the environment in which the development work for this project took place. It offered an environment where the Pytch application could build cleanly and could offered easy interaction with the web environment. Initially work was completed on the latest version of Pytch that had been cloned directly from the main Pytch Github (North & Strong, 2022).

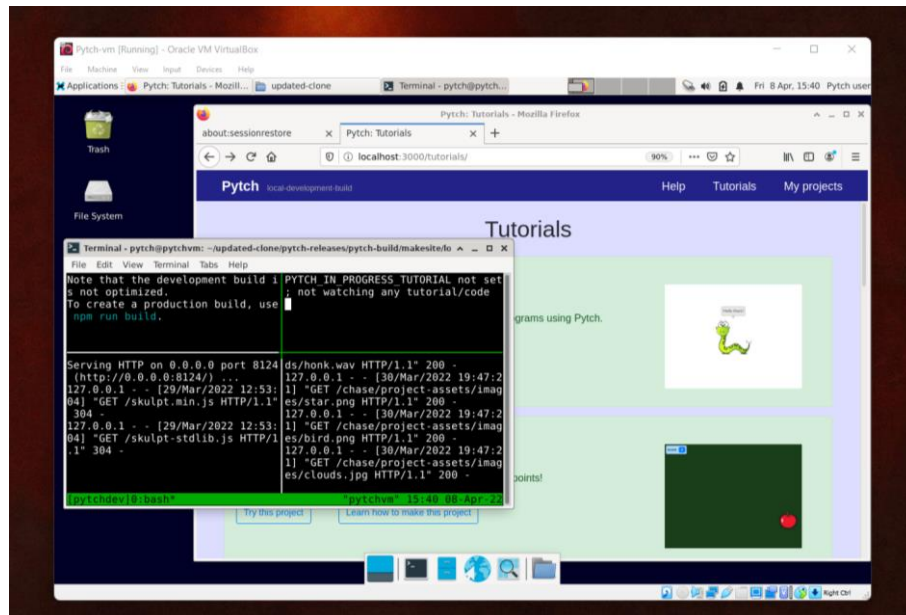


Figure 3.2 Oracle VM running Pytch App on Windows Machine

As the project progressed into the latter stages, the creation of functional changes prompted a need for rebasing. This called for a new repository to be cloned but this time from a fork of the main Pytch Github into a personal Github repository where the development work was free to experiment and modify larger components of the Pytch application that couldn't have been pushed to the main Github at this stage.

A brief evaluation of these two environments and how the Virtual Machine performed will be covered in Chapter 4.

3.3 TESTING OF PYTHON TESTS IN PYTCH ENVIRONMENT

The first step of development was to verify the possibility of using Python testing methods inside the Pytch environment. This was done by opening a currently existing Pytch tutorial and compiling an early tutorial step with an assert test as part of the class.

3.3.1 Python Assert Testing in Pytch

A Python assert test is a simple test that allows you to check for a specific part of the code to output an expected result. If the test is successful, the code will continue to run. If the expected value is not compatible with the actual value, a custom error message will be output to the terminal. This test of feasibility was successful and showed that the Python assertion test worked in the Pytch environment and interacted well with the Pytch terminal.

```
7 class Sky(pytch.Stage):
8     Backdrops = ["clouds.jpg"]
9     @pytch.when_green_flag_clicked
10 def checkRender(self):
11     assert self.Backdrops == ["clouds.jpg"], "Background Image not defined Properly"
12     print("Background Rendered successfully!")
13
```

Figure 3.3 Example of the Inclusion of a Python assert test in Pytch

The verification of the assert test working with the Pytch environment indicated that this was the way forward for showing novice programmers how to include simple unit tests into their work as a first step in Test Driven Development.

3.4 VISUAL PROOF OF CONCEPT THROUGH TUTORIAL HTML

In the initial stages of studying Pytch and its tutorial compiling system, a visual proof of concept was created by manipulating the HTML file that is built to serve an existing tutorial in order to gain an understanding of how the chapters on Test Driven Development would look. When a tutorial is built, a static html file is compiled using the design defined in the respective tutorial.md file. The webpage's HTML file is editable and due to Pytch being built with React, the file could be saved and when the webpage was reloaded the changes were visible. This was done prior to the development into manually editing the structure of a tutorial on the Github repository as a visual proof of concept to aid in the later final design of these chapters.



Figure 3.4 Screenshot of the Visual POC for TDD Tutorial Chapter in Pytch

This proof of concept gave insight into how these extra chapters should be integrated into the already active tutorial. The two options for inclusion of this new material were to either add in full new chapters between current chapters that would follow on from the addition of a section with a chapter on what tests to run on this section, or to integrate the new snippet of test code into the bottom of an existing chapter. It was decided the creation of additional chapters that slotted into the current tutorial structure was the way to implement them.

3.5 FUNCTIONAL PROOF OF CONCEPT THROUGH ALTERING TUTORIAL

3.5.1 Design Decisions

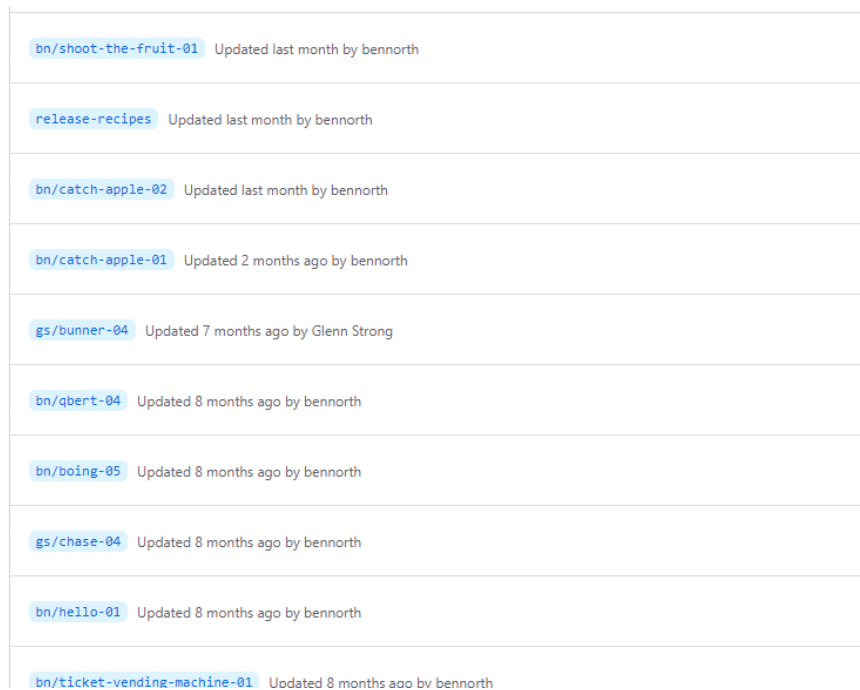
The final stage of development was to implement new test centred chapters into the already existing 'Chase' Tutorial described in Section 2.6. This was done by designing the tests for different sections of the code.py file for this tutorial. It was decided for a few reasons that the tests to be included in this proof of concept were to be unique from each other and also to cover the project components that were specific to the current tutorial. The reason for the implementation of unique tests being that if the testing process was too repetitive it would likely make the tutorial process mundane which is the least desirable quality for educational material. It also forced more attention to be applied to the second motivation behind the tests designed, the aim of unique tests meant that more of a span of Pytch components are covered in these tests.

It was also decided that there shouldn't be an oversaturation of tests in the tutorial chapters so not to take too much attention away from the learning of the basics of Python. The step from a Scratch environment into a more code heavy environment like Pytch is already a challenge that novice programmers are dealing with, so the motivation here is to keep things simple and not over teach in order to make the material more engaging and memorable for the students. The original code for this project is also less than 100 lines of code so adding too much extra testing does significantly take away from the original material.

3.5.2 Implementation Process

Once the tests were designed and a new final version of the code.py for the chase project was created. A new clone of the most updated Chase tutorial branch was created, the process of carefully rebasing the tutorials commits then began. Completing an accurate rebase was imperative to this project's implementation as it was the process that allowed for the seamless integration of new chapters into the middle of an existing project. As explained in Section 2.4, the Pytch tutorial authoring system requires that the chapters be committed in chronological order from start to finish.

The important aspects of this rebase process that were learned were that in order to rebase the new code into the tutorial properly, the individual steps that the test were added to would have to be pulled down and added to rather than just having the tests available in the final code project.



bn/shoot-the-fruit-01	Updated last month by bennorth
release-recipes	Updated last month by bennorth
bn/catch-apple-02	Updated last month by bennorth
bn/catch-apple-01	Updated 2 months ago by bennorth
gs/bunner-04	Updated 7 months ago by Glenn Strong
bn/qbert-04	Updated 8 months ago by bennorth
bn/boing-05	Updated 8 months ago by bennorth
gs/chase-04	Updated 8 months ago by bennorth
bn/hello-01	Updated 8 months ago by bennorth
bn/ticket-vending-machine-01	Updated 8 months ago by bennorth

Figure 3.5 Snippet of List of Tutorial Branches

This realisation lead to the thorough examination of the tutorial commits at the neighbouring steps to these new test chapters. The new chapters would then be committed by copying the previous chapter step into the code.py file, then adding the new code and creating the commit. These commits would then be inserted into their intended slots using an interactive git rebase from the Oracle VirtualBox terminal. Once this was complete, this new tutorial branch would replace the current 'Chase' branch in the index.yaml file on the 'release-recipes' branch seen above in Figure 3.5 Snippet of List of Tutorial Branches, and when Pytch is rebuilt the tutorial would be successfully updated with the intended changes.

3.6 CHASE TUTORIAL'S NEW INTEGRATED TESTS

This section gives an outline of the of the tests that were created and included in the 'Chase' project with an accompanying screenshot of each test displayed in the Pytch environment.

3.6.1 Background Image Validation

The first test integrated into the 'Chase' tutorial was the addition of an assertion test to verify that the background image being rendered is the 'clouds.jpg'. The test will print a message to the terminal if it passes.



Figure 3.6 Background Image Validation Test

This test was intentionally kept very simple as it is the students first introduction to tests in the project so it doesn't include any of the more complex Pytch functions to avoid confusing the students. This test offers a useful teasing introduction into the process of Test Driven Development and offers an opportunity to see the assert test in action in a simple manner. A note is left at the end of this chapter prompting the student to change the image being imported into the 'Backdrops' object in order to see the error flag in the terminal.

This use of the project's background image as an initial test also works well as it is very obvious to the student when the background image is changed in the code and rendered in real time. This test offers a comprehensive introduction that the rest of the tests build from.

3.6.2 User Input Sanity Checks

The second test included in the tutorial is an assertion test that is integrated into the 'move' functions that are created as part of the tutorial in order to give the user control over the movement of the Bird in the project scene. These tests verify that the bird moves the correct amount in the right direction after every key press from the user. This test is useful as it fits in naturally with the already existing project function while also increasing the level of understanding that the student can gain about the use of 2D coordinate geometry and key press input listeners in Python.

Change the code like this: ?

```
20 20
21 21 ....@pytch.when_key_pressed("ArrowRight")
22 22 ....def move_right(self):
+ 23 .....x_curr = self.x_position
23 24 .....self.change_x(self.speed)
+ 25 .....x_new = self.x_position
+ 26 .....x_diff = x_curr - x_new
+ 27 .....assert x_diff == -self.speed, "should have moved the correct amount Right"
24 28
25 29 ....@pytch.when_key_pressed("ArrowLeft")
26 30 ....def move_left(self):
```

Figure 3.7 User Input Movement Test

This test is centred around more complex Python and Pytch related content, so it was imperative that the test was kept as simple and well aligned with the project code as possible. There is an identical test inserted into each of the four move functions.

Initially there was a print called after every successful movement in their respective direction but after testing the code a few times it was decided these prints cluttered the terminal too much so they were removed, these tests then became silent tests unless they failed in which case the assertion test message would be passed to the terminal as the error description.

3.6.3 Verifying the Operation of the Random Function

The final test that was integrated into the tutorial was an assertion test on the Star sprite that verifies that the star is moving around the environment. The star class makes use of the python 'random' to select the next coordinate in the environment for the Star to move to.

This assertion test makes use of the Pytch 'when_green_flag_clicked' and 'wait_seconds' functions which are both previously used in this project. The 'when_green_flag_clicked' is a python decorator that will call the following function as soon as the project is initiated. The 'wait_seconds' is a custom function that simply waits the input amount of seconds before progressing through the current function.

Change the code like this: ?

```
77 77 .....while True:
78 78 .....if self.touching(Bird):
79 79 .....self.hide()
+ 80
+ 81 .....@pytch.when_green_flag_clicked
+ 82 .....def testMovement(self):
+ 83 .....    pytch.wait_seconds(10)
+ 84 .....    x_start = self.x_position
+ 85 .....    y_start = self.y_position
+ 86 .....    pytch.wait_seconds(2)
+ 87 .....    assert self.x_position != x_start, "star should have moved X"
+ 88 .....    assert self.y_position != y_start, "star should have moved Y"
+ 89 .....    print("Star Successfully Moved!")
```

Figure 3.8 Verification of Random Functionality Test

If this test fails a message will display to the terminal as an error alerting the user that the star sprite has not moved when it was supposed to. If it passes a short message will be printed to the terminal indicating that the star has moved successfully.

This test reinforces the students understanding of these custom Pytch functions while also offering a useful test on a new concept to the student which is the random function. The inclusion of the random function was unique to this particular Pytch project so it was deemed important to form a test that would include it in some way.

4 EVALUATION

This chapter examines the results of this projects implementation and also looks at what was learned with regards to the implementation of Test Driven Development in educational material that is targeted at younger audiences.

4.1 EVALUATION OF WSL VERSUS ORACLE VM

While I believe the Windows Subsystem for Linux is a useful tool that was the most accessible solution for working in a Linux environment during this project, the niche nature of Pytch's development and the lack of history of its use with the tool meant that any unknown errors that were encountered took a long time to address and troubleshoot. Without time constraints it would have been of benefit to get the Pytch environment running successfully on WSL, but this wasn't the case in this project's timeline so other solutions had to be looked at.

The Oracle VM offered an environment that had previously been used to run the Pytch environment so there was confidence and experience from the Pytch developers that could help troubleshoot any errors or blockers that were met. This tool was what was used for this projects development and offered an adequate solution. The only downfall was the higher level of GPU required to run this service on the laptop and it meant that running the virtual machine hindered the ability to run much else on the machine at the same time. Sharing screens on Teams or Google Meets became an impossible task while running the VM as the laptop would come to a grinding halt.

Overall, the Oracle VM was sufficient in enabling the development work for this project, but it could be a useful future piece of work to get a successful running of Pytch on the WSL.

4.2 IMPLEMENTATION EVALUATION

The implementation procedure undertaken during this project mainly consisted of proof of concept work to study the feasibility of integrating certain aspects of Test Driven Development into the Pytch environment. The final result of this implementation was a working example of an existing Pytch tutorial with new chapters that encourage the use of Python assert tests in order to promote the thought process of Test Driven Development at an early stage of programming education.

4.2.1 Validation of Assertion Tests

The initial work of validating that Python assert tests functioned in the Pytch environment and work with the output terminal was imperative to the rest of the implementation process and did prove to be successful. The assertion tests were chosen as they can be added into the code very easily and are easily readable. They are contained to one line of code and only involve a conditional input and a string variable to be used as the error output. This test is perfect for use as an introduction into Test Driven Development as they offer sufficient functionality with minimal chance for confusion.

4.2.2 Visual Proof of Concept of Test Chapter

The development of a visual proof of concept was invaluable to the design process that followed surrounding the creation of the test driven tutorial chapters. The only issue with this work was that it was completed by editing the static HTML file that was created by the tutorial compiler, meaning any time the compiler was restarted this work would be lost. While this was not ideal and evidently couldn't be used in any final model, it did serve the purpose of offering an initial insight into what the final project would look like.

The editing of the HTML file was also a tedious process and definitely not the way to make tutorial changes efficiently. However, at the time of this development the work wasn't being completed off a forked Github repository of the Pytch project, so the steps required to create new functioning chapters through making new tutorial branches and rebasing wasn't an option at the time. The process of digging through this HTML file offered insight into the structure of the Pytch tutorial design that were not available in the documentation or even in the tutorial compiler file. Therefore, this implementation became a helpful process for investigation into Pytch.

4.2.3 Functional Pytch TDD Chapters

The final implementation step was creating functional chapters based on the inclusion of Test Driven Development into the Pytch tutorial. This was the biggest piece of development work of this project as it showed that the inclusion of chapters on Test Driven Development were not only a possibility but a valid option.

The key takeaway from the implementation process of the chapters was that a very in depth understanding of this code was required before any rebasing of the Github repository could be done successfully. Due to the fact that the design of these tutorials leans heavily on how each individual commit is defined and ordered, it became apparent that great care would need to be taken when inserting new chapters into the middle of the tutorial structure. This was touched on in Section (3.5.2) and did give some trouble initially with small visual errors coming through on the webpage side due to slight mismatching of code in the newly added chapters. This was fixed by completing a full new rebase where the commit errors were corrected and was finished successfully offering the resulting evolved Pytch tutorial.

The product of this implementation offers an evolved Pytch tutorial that contains chapters that are used to encourage the use of tests in project code for young developers. The process that was undertaken in order to achieve this was specifically for Pytch's environment and tailored to how Pytch is designed. The tests created for the 'Chase' tutorial could be used in some form in other Pytch tutorials however they were tailored to flow with the chase tutorial specifically. All this considered, the tests and the corresponding chapters created offered a compact and effective solution to the addition of education on Test Driven Development into the Pytch environment.

4.3 TEST DRIVEN DEVELOPMENT IN ENVIRONMENTS FOR NOVICE PROGRAMMERS

Through the process of developing and implementing Test Driven Development chapters into a Pytch tutorial, a few learnings were taken that are worth noting. These mainly involve what was discovered through the studying of external material plus examining what worked best when designing the test chapters.

4.3.1 Simplicity First

The one core principle that already exists within the Pytch education system and other environments such as Scratch is that keeping the work as simple and easy to understand as possible should always be the main priority. This idea had massive impacts on the design choices throughout this project as the aim is to implement new tutorial chapters that flow naturally with the already existing tutorial steps. This is why the use of the assertion test was decided upon as it offers a very straight forward approach of completing expectation tests on any variable in the project. Assertion tests are very straight forward to explain and understand and their inclusion offered a method of including education on the basics of Test Driven Development without creating blockers for the core education of Pytch which is teaching Python programming. The assertion tests are also very compact which again leans into the idea of keeping the code and tutorial learning as simple but efficient as possible. Adding tests that require a large quantity of lines would distract from the rest of the project which is not desired.

4.3.2 Relevant and Engaging Test Content

Another core principle of the design process for the new tutorial chapters was the tests being created must be both useful to aid the tutorial material but also unique from each other as not to become repetitive. Sustaining attention and focus from younger demographics is a challenge so any potential catalysts to losing focus such as the tests becoming mundane and repetitive is the antithesis of what is trying to be accomplished in this project and with Pytch as a whole. The tests should be designed to be interactive and should directly engage with the material of the tutorial project they are a part of.

The process taken in this project's implementation was to identify what the core learning of the tutorial was and what made it unique from the other tutorials and then focus on implementing most of the tests around those concepts. The key teaching of the Chase tutorial is the remote movement of Sprites in an environment and detecting the interaction between multiple Sprites in an environment. It also introduces python math's random function so testing on this was also looked at. From this analysis of the Chase tutorial, the tests that were designed tested that the Bird sprite moved correctly according to the key pressed by the user, and that the random function was working for the star's movement. Pytch's custom functions for obtaining the 2D coordinates of Pytch sprites were used along side these tutorials in order to obtain their current position for the tests.

4.3.3 Setting Up a Strong Foundation for Understanding

The only slight exception to the principle of having the test contain very tutorial relevant content was the first test's design. The test verifying the background image doesn't offer much further education on Pytch's components or on Python programming but is used as a direct teaching on the use of an assertion test in the environment and offers an understanding on how it works. This is purposefully the first test chapter that the student will encounter as it offers a very open and obvious method of teaching Test Driven Development with no possible distraction from potentially unfamiliar Pytch functions or new Python concepts. This test is one that can be done at the start of any of the Pytch tutorials as it is GUI testing the backdrop image of the canvas. Every current Pytch tutorial includes visual outputs for the obvious reason that it creates a far more engaging learning experience hence making this a universally applicable test.

4.3.4 Comprehensive Error Messages

Another point of design that was very important to the final deliverable of these test chapters was that every test's error message should be well articulated and very relevant to the test that is being run. From the study completed by the creators of the Pythy platform (Edwards, et al., 2014) it was made apparent that the biggest point of potential confusion for students was vague or overly complex/technical error messages. Great care should be taken in the forming of these error messages as it could be the difference between a student spending three minutes or three hours troubleshooting.

4.3.5 Ensure the Tests Improve the Project's Validation

There is an added security that is given to the project once these tests tutorial tailored tests are a part of the code. This project's work is beneficial for tutorial creators and educators as it offers more specific error checking than purely syntax errors that the standard Python compiler would pick up on. The inclusion of custom tests allows for the expected functionality of project components to be tested instead of just the correctness of the student's code. This is more so highlighting the benefits of including Test Driven Development and code testing in general but is an important point to note, especially when moving into future work and the potential for creating grading systems and assignment environments.

While the points were made about the importance of simplicity in these tests, there should be a valid purpose for each test included in aiding the testing of the code's specific use. This is especially useful for the custom Pytch functions that are used in the tutorials.

4.3.6 Summary

To summarise, the inclusion of test driven development in these tutorials allows for specific verification of the projects functionality. It is important therefore that the tests created cover as many aspects of the tutorial as possible while also being straight forward and in line with the teachings of the tutorial. It is also imperative that the error messages created for these tests are comprehensive and in line with the test being run. The tests should also be compact and somewhat discrete as to not take away from the rest of the teachings of the project. Once these conditions are hit the tests should be sufficiently simple but also effective at delivering the necessary lesson on Test Driven Development that is intended. The tests don't need to be ground breaking or overly complicated, the motivation here is the easy introduction of novice students to the idea of Test Driven Development and try to make them comfortable with the process so that they may start to use it in other work.

5 CONCLUSION

This chapter consists of a summary of the work done during the project and dissertation process. The potential future work for this project is also highlighted.

This project designed and implemented new chapters for Pytch’s tutorial environment that include education on Test Driven Development relative to the existing tutorial material. The Pytch environment was studied from both perspectives of frontend GUI and backend architecture in order to understand and develop a succinct design for the integration of the new tutorial chapters. An existing Pytch tutorial was chosen and analysed in order to determine what tests would best fit its design.

During the timeline of this project many new skills were also learned outside of the understanding of the Pytch environment and the state of the art of Test Driven Development in CS education. JavaScript and React were both studied throughout this project and the use of a Virtual Machine to access Pytch on a Linux machine was uncharted territory before this project began. A better understanding of the Linux OS was gained through the use of this Virtual Box. Working on a somewhat proprietary developing project rather than widely used open source environments was something new that while it posed a challenge, offered plentiful new experience that would never have been acquired otherwise.

The results from Chapter 3.6 highlight the successful development process that was completed during this project with the integration of new Test Driven Development chapters that are tailored to the intended learning outcomes of the ‘Chase’ tutorial. The product of this implementation can be looked upon as a proof of concept of the feasibility of design and inclusion of engaging Test Driven Development processes in the Pytch environment, and similar tests could be designed and implemented into all other tutorial projects if desired in the future.

During the process of designing and implementing these new tutorials on testing it was also demonstrated what discoveries were made relating to how to create the optimal tests for inclusion in educational content for novice programmers on a wider scope than just the Pytch environment. The different aspects highlighted in Section (4.3) show all the extra findings about designing these tests that were learned. The conclusions on this work were that the tests created should fall in line with the intended teachings of the current project without adding too many new features that could potentially distract from the original intended teachings of the specific tutorial and also that they should span as many of the prominent features of project as possible. The error messages should also be ensured to be very comprehensive and easily understandable for novice programmers. The tests should also be designed to be as discrete as possible to also not distract from the main body of tutorial code.

Overall, this project was successful in including Test Driven Development Education into the Pytch learning environment and also additionally provided a view into what works and what doesn’t with regards to Test Driven Development integration into learning environments and projects targeted at novice and young programmers on a wider scope than just the Pytch environment.

5.1 FUTURE WORK

As part of the Future Work section a differentiation will be outlined between what sort of future work would be completed based on what sort of time frame the project was extended to. Due to the nature of this project and its results in demonstrating the feasibility of something that doesn't currently exist in the Pytch environment, there are many potential directions in which this work could expand.

The first idea in further development that is the most natural follow up to the work completed here would be to include similar test driven development chapters across the rest of the Pytch tutorials. This could be done by taking the evaluation work completed in this project to determine a useful set of tests to add to each tutorial and then implement these new chapters using the git rebasing process described in Section (3.5.2).

Aside from the integration of this project's work across Pytch, there are other areas of Pytch and other ways in which these tests could be utilised. The test chapters could be designed as an obstacle that the student must overcome before they can continue along the rest of the tutorial as somewhat of a sanity check. There is potential for redesigning the tutorial compiler so that the tutorial could not be continued until the test has been passed successfully by the user. Every test a user passes could potentially be stored as a representation of their learning progress.

Expanding further on the idea of using the outcome of the tests as a student progression record. The development of a programming assignment environment in Pytch as well as a tutorial environment could be something of great benefit for Pytch. Looking at the current state of the art of other web-based Python environments like Pythy (Edwards, et al., 2014) there is an existence of automatic grading systems in similar projects to Pytch. The addition of an automatic grading system would allow Pytch to become more whole in its purpose as an educational platform. It would allow for teachers and professors to use the platform to create assignment projects that could then be graded and become part of the curriculum mark for the students, which would be a massive step for this application.

BIBLIOGRAPHY

International Society for Technology in Education, Eugene, 1990. *In Search of Gender Free Paradigms for Computer Science Education*. Nashville, s.n., pp. 10-15.

Caiza, J. & Del Alamo, J., 2013. *Programming Assignments Automatic Grading: Review of Tools and Implementations*. Valencia, s.n., pp. 5691-5700.

Chien, L.-R., Beuhrer, D. J. & Yang, C.-Y., 2007. *An adaptive environment in DICE system with TDD model*. Villach Austria, National Chung Cheng University.

Edwards, S., Tilden, D. & Allevato, A., 2014. *Pythy: Improving the introductory Python programming experience*. Atlanta, s.n., pp. 641-646.

Fu, X. et al., 2008. *APOGEE – Automated Project Grading and Instant Feedback System for Web Based Computing*. Portland: s.n.

Goodwin, M. & Drange, T., 2016. *Teaching Programming to Large Student Groups through Test Driven Development - Comparing Established Methods with Teaching based on Test Driven Development*. Rome, s.n., pp. 281-288.

Heckman, S. & King, J., 2018. *Developing Software Engineering Skills using Real Tools for Automated Grading*. Baltimore, Association for Computing Machinery New York, pp. 794-799.

Lappalainen, V., Itkonen, J. I. V. & Kollanus, S., 2010. *ComTest: A Tool to Impart TDD and Unit Testing to Introductory Level Programming*. Jyväskylä, University of Jyväskylä, Finland.

Meta - Facebook, 2022. *JestJS.io*. [Online]
Available at: <https://jestjs.io/>
[Accessed 01 04 2022].

North, B. & Strong, G., 2022. *Pytch Developer Documentation*. [Online]
Available at: <https://www.pytch.org/doc/developer.html>
[Accessed 01 04 2022].

North, B. & Strong, G., 2022. *Pytch Documentation*. [Online]
Available at: <https://www.pytch.org/doc/index.html>
[Accessed 01 04 2022].

North, B. & Strong, G., 2022. *Pytch Github Repository*. [Online]
Available at: <https://github.com/pytchlang>
[Accessed 01 01 2022].

North, B. & Strong, G., 2022. *Pytch Tutorial Documentation*. [Online]
Available at: <https://www.pytch.org/doc/build-tools/tutorialcompiler/index.html>
[Accessed 01 04 2022].

North, B. & Strong, G., 2022. *Pytch VM Documentation*. [Online]
Available at: <https://www.pytch.org/doc/vm/developer/index.html>
[Accessed 01 04 2022].

Oracle, 2022. *Oracle VM - VirtualBox*. [Online]
Available at: <https://www.oracle.com/uk/virtualization/virtualbox/>
[Accessed 01 04 2022].

Scratch Foundation, 2021. *Scratch*. [Online]

Available at: <https://scratch.mit.edu/>

Skulpt, 2019. *Skulpt Documentation*. [Online]

Available at: <https://skulpt.org/docs/index.html>

[Accessed 01 04 2022].

Tang, L., 2015. *A Browser-based Program Execution Visualizer for Learning Interactive Programming in Python*, Houston: Rice University.

Virginia Tech, 2021. *Web-CAT Projects*. [Online]

Available at: <https://web-cat.github.io/projects/>