Lab II - PolyMorpher Prototype Product Specification

Colten S. Everitt

Team Silver

Old Dominion University

CS411W

Professor Thomas Kennedy

26 February 2018

Version 1

Author Note

Colten S. Everitt, Department of Computer Science, Old Dominion University.

This research was done under the supervision and guidance of Thomas Kennedy.

Correspondence concerning this article should be addressed to Colten Everitt, Department of Computer Science, Old Dominion University, Norfolk, VA 23529.

Contact: cever015@odu.edu

Table of Contents

List of Figures

List of Tables

Lab 2 - PolyMorpher Product Specification

1 Introduction

PolyMorpher is a programming game and outside resource solution for undergraduate (predominately first-year) Computer Science (CS) students at Old Dominion University (ODU). Object-Oriented Programming (OOP) and problem solving skills will be taught throughout the game to increase the probability of a CS student passing introductory CS classes, with the end goal of that student graduating with at least a bachelor's degree in CS. PolyMorpher is a downloadable executable file that is run with a Management Simulator to a Tangible User Interface (TUI). The programming game is currently being developed using the Unity Software Development Kit (SDK) with the C# and JavaScript programming languages.

1.1 Purpose

PolyMorpher is a programming game that runs off the player's machine and allows the player to play an engaging single player story, while learning the key concepts of CS. The gameplay offers a user-friendly TUI and incorporates all of the necessary aspects of modern computer-based games.

The initially targeted end users for PolyMorpher are students who are currently enrolled in the CS degree program at ODU. Since the game will be made by a team of students currently in the CS degree program at ODU, the game will be partially owned by ODU and therefore sold by ODU. After students at ODU begin to utilize PolyMorpher, ODU might make the decision to sell the game to other universities, colleges, or educational institutions. PolyMorpher is also made for anyone who is generally interested in programming, outside of the CS department or at ODU.

Similar to the end users, the initially targeted customer for PolyMorpher is ODU. Since the game was made at ODU by students in the CS department, ODU will be the first targeted customer. PolyMorpher could also be sold to other universities, colleges, or educational institutions that currently offer a CS degree program. Middle or high school teachers might be interested in buying a license to PolyMorpher for their classes. PolyMorpher is also targeted towards anyone seeking to gain more knowledge in computer programming, OOP concepts, and problem solving skills.

1.2 Scope

Programming is intimidating for the uninitiated. As a result, first time ODU programming students drop out or switch majors. Existing tools fail to teach OOP concepts and problem solving skills. Extensive evidence that a significant section of first time ODU CS students are struggling with programming is shown in the Appendix (Section 3.3 Reasons for Creating PolyMorpher for Students). The current CS curriculum at ODU is failing to teach OOP concepts and problem solving skills effectively enough for students to truly understand them. These students might try to look for outside resources aside from what ODU provides, end up not finding any, and then fail classes, or worst case switch majors or drop out of college altogether.

Today, most programming games use OOP but do not actually teach OOP to the players. This is a major problem; CS and computer programming can not be fully learned without OOP concepts and problem solving skills. Current programming games do not only require low experience levels, while both using and teaching OOP, and only focusing on one programming language. PolyMorpher will provide all these necessary features to create a programming game that will give students a solid foundation in CS.

PolyMorpher will address OOP concepts and problem solving through the use of a Management Simulator and a Tangible User Interface (TUI). A Management Simulator, along with a TUI through a game application, has been shown to be the best way to teach OOP concepts and problem solving. In addition, games enhance interest among new learners. The basic nature of game interaction inherently gives players a more natural way to learn content. Games also change the learning style from traditional to more dynamic. One way this is accomplished is by not requiring an instructor to be present at all times; learning can occur on more fluid, rather than set, schedules.

1.3 Definitions, Acronyms, and Abbreviations

API: Application Program Interface - A tool for assisting developers in creating applications

Computer: a programmable electronic device designed to accept data, perform prescribed mathematical and logical operations at high speed, and display the results of these operations

Computer Programming: a process that leads from an original formulation of a computing problem to executable computer programs

Computer Science (CS): the science that deals with the theory and methods of processing information in digital computers, the design of computer hardware and software, and the applications of computers

Design: an outline, sketch, or plan, as of the form and structure of a work of art, an edifice, or a machine to be executed or constructed

Git: version control system for tracking changes in computer files and coordinating work on those files among multiple people

GitLab: web-based git repository manager the includes wiki and issue tracking

Gradle: an open-source build automation system that was designed for multi-project builds

GUI: Graphical User Interface - A graphical display for users of electronics to interact with the content displayed

JavaScript: a programming language commonly used in web development where the the code is processed by the client's browser

Management Simulator: a way to simulate the management of a game in an organized fashion

MySQL: an open source multi-user database management system

Non-Technical Game: user-friendly gameplay able to be utilized by non-technical users

Non-Technical User: user who lacks formal education or knowledge in computer science, computer programming, object-oriented programming, or problem solving skills

Object-Oriented Programming (OOP): A schematic paradigm for computer programming in which the linear concepts of procedures and tasks are replaced by the concepts of objects and messages

ODU: Abbreviation for Old Dominion University

Platform: an integrated set of packaged and custom applications tied together with middleware

PolyMorpher: a programming game that focuses strictly on teaching OOP and problem solving skills

Problem Solving: the process of finding solutions to difficult or complex issues

Programming Game: a video game which incorporates elements of computer programming into the game, which enables the player to direct otherwise autonomous units within the game to follow commands in a domain-specific programming language

Regression Testing: a type of application testing that determines if modifications to the application have altered the application negatively

Software Development Kit (SDK): a set of software development tools that allows the creation of applications for a certain software package

Student Involvement: the amount of physical energy students exert and the amount of psychological energy they put into their college experience

Student Progression Dilemma: the problem of CS majors at ODU not advancing through the CS course schedule in order to graduate with a CS degree

TUI: Abbreviation for Tangible User Interface

Ubuntu: open-source Linux operating system

Unity: a popular game development platform

User-Friendly: easy to comprehend by non-technical users

Virtual Machines: emulations of computer systems that provide functionalities of physical computers

Web Application: a client-server computer program in which the client (including the user interface and client-side logic) runs in a web browser

Wiki: a website on which users collaboratively modify content and structure directly from the web browser

1.4 References

12 Free Games to Learn Programming. (2016, April 25). In Mybridge. Retrieved from https://medium.mybridge.co/12-free-resources-learn-to-code-while-playing-games-f7333043de11

Batten, C. (Narrator). (2017). CS410 Dungeon Escape Demo (Short Version) [Online video]. Online: YouTube. Retrieved from https://www.youtube.com/watch?v=ynhdd1IKgps

Batten, C. (Narrator). (2017). CS410 Project Dungeon Demo [Online video]. Online: YouTube. Retrieved from https://www.youtube.com/watch?v=ynhdd1IKgps

Batten, C. (2017, November 21). CS410 Tech Demo 2 (Download Source Code). In PolyMorpher. Retrieved from http://www.cs.odu.edu/~410silver/references.html

Batten, C. (2017, November 29). VersionControlFlow. In draw.io. Retrieved December 21, 2017, from https://www.draw.io/?state=%7B%22ids%22:%5B%221IQj6SYJqC6YLAK_qMRVIQkHiUmr9laBu%22%5D,%22action%22:%22open%22,%22userId%22:%22108692003133590583047%22%7D#G1IQj6SYJqC6YLAK_qMRVIQkHiUmr9laBu

Batten, C. (2017, October 26). CS410 Dungeon Escape Demo (Download Source Code). In PolyMorpher. Retrieved from http://www.cs.odu.edu/~410silver/references.html

Batten, C. (2017, October 26). CS410 Dungeon Escape Demo (Play Now). In PolyMorpher. Retrieved from http://www.cs.odu.edu/~410silver/references.html

Edraw. (2017, May 12). Standard Flowchart Symbols and Their Usage. In Edraw Visualization Solutions. Retrieved from https://www.edrawsoft.com/flowchart-symbols.php

Everitt, C. (2017, September 6). Current Process Flow. In draw.io. Retrieved December 21, 2017, from https://www.draw.io/?state=%7B%22ids%22:%5B%220B-5KdQEdqLUPdnBFUnp2V05uMEE%22%5D,%22action%22:%22open%22,%22userId%22:%221086920031335905830 47%22%7D#G0B-5KdQEdqLUPdnBFUnp2V05uMEE

Everitt, C., & Dang, D. (2017, September 24). currentProcessFlow. In draw.io. Retrieved December 21, 2017, from https://www.draw.io/?state=%7B%22ids%22:%5B%220B3Bc95zBWXg9TFZ6X0FMU1NTdEk%22%5D,%22action%22:%22open%22,%22userId%22:%221086920031335905830 47%22%7D#G0B3Bc95zBWXg9TFZ6X0FMU1NTdEk

Everitt, C., Santos, K. & DeArce, N. (2017, November 27). Work Breakdown Structure (WBS). In draw.io. Retrieved December 21, 2017, from https://www.draw.io/?state=%7B%22ids%22:%5B%220B-5KdQEdqLUPWnNoSHhIUGg2OTQ%22%

5D,%22action%22:%22open%22,%22userId%22:%221086920031335905830 47%22%7D#G0B-5KdQEdqLUPWnNoSHhIUGg2OTQ

Everitt, C., Santos, K. & DeArce, N. (2017, October 13). ProcessFlowDiagram_silver. In draw.io. Retrieved December 21, 2017, from https://www.draw.io/?state=%7B%22ids%22:%5B%220B_xBnZ1ge4PlZTVjV3h6Y2pGSWc%22%5D,%22action%22:%22open%22,%22userId%22:%221086920031335905830 47%22%7D#G0B_xBnZ1ge4PlZTVjV3h6Y2pGSWc

Few, S. (2008, February 5). Practical Rules for Using Color in Charts. In Perceptual Edge. Retrieved from http://www.perceptualedge.com/articles/visual_business_intelligence/Rules_for_using_color.pdf

Kennedy, T. (2017, September 6). kennedyData. In Google Drive. Retrieved from https://drive.google.com/drive/u/1/folders/0B_xCQd8Vk2BnSU1hNnJwSXB1NEE

O'Neill, M. (2017, March 6). Computer Science Before College. In Computer Science Online. Retrieved from https://www.computerscienceonline.org/cs-programs-before-college/

Riley, P. (2017, September 14). Using Games to Introduce Programming to Students [PowerPoint slides]. Retrieved from http://www.cs.odu.edu/~410silver/references.html

Santos, K., Riley, P. & Dang, D.(2017. December 7) Risk matrix and description tables in Design Presentation. Retrieved from https://docs.google.com/presentation/d/

1oY9lkSAHvg2OIRkljYJNZWCqVTbiw45STKglsJUQjJI/edit#slide=id.g283e74317a_0_177

Stokes, J. (Narrator). (2017). CS410 Programming Game Pitch [Online video]. Online: YouTube. Retrieved from https://www.youtube.com/watch?v=QBvgzFgZaOQ&feature=youtu.be

Stokes, J. (2017, October 9). CS410 Programming Game Pitch (Download Source Code). In PolyMorpher. Retrieved from http://www.cs.odu.edu/~410silver/references.html

Team Silver. (2017, December 13). Prototype PowerPoint Presentation. In *PolyMorpher*. Retrieved from https://docs.google.com/presentation/d/e/2PACX-1vSidnjCKAuVEtKshHkyO7A-OfW3qWIKRkxcp0em412WwL1ig6SFmnqrMUyHr8-FMvzvaRjmcKYiCytq/pub?start=false&loop=false&delayms=3000&slide=id.g25ab9a9d23_0_1542

Team Silver. (2017, November 21). Design PowerPoint Presentation. In *PolyMorpher*. Retrieved from https://docs.google.com/presentation/d/e/2PACX-1vSllslBDmSvRfMI9nbrp0RmRaPRsHNz7YWDfKNiF5sg15cp7ycQ774MuMgm4G4qhR6hohTiUQrrjRdo/pub?start=false&loop=false&delayms=3000&slide=id.g25ab9a9d23_0_1542

Team Silver. (2017, October 25). Feasibility PowerPoint Presentation. In *PolyMorpher*. Retrieved from https://docs.google.com/presentation/d/e/2PACX-1vReG6Sodx-

gVFro1ByYMOYHSyiSRiU5HW-Su-PyMVGO8F4CQ7pY49tB_pJecVAprukso GaP_00ozhmR/pub?start=false&loop=false&delayms=3000&slide=id.g25ab9a9d 23_0_1542

Team Silver. (2018, February 26). Lab 1. In *docs.google.com*.

"The Benefits of Video Games." abcnews (2011, December 26). Retrieved October 19, 2017, from http://abcnews.go.com/blogs/technology/2011/12/the-benefits-of-video-games/Good-Morning-America

Unity Technologies. (2017, August 10). Company Facts. In Unity. Retrieved from https://unity3d.com/public-relations

Unity. (2016, July 6). Unity - Scripting API. In Unity. Retrieved December 21, 2017, from https://docs.unity3d.com/530/Documentation/ScriptReference/index.html

Unity. (2017, October 11). Asset Store. In Unity. Retrieved December 21, 2017, from https://www.assetstore.unity3d.com/en/

1.5 Overview

In this product description paper, a general description of the prototype of PolyMorpher will be given. It will highlight PolyMorpher's architecture, functionality, and interfaces. It will also go over the capabilities, features, and components of PolyMorpher. The information provided in the remaining sections of this document includes a detailed description of the hardware, software, and external interface architecture of the PolyMorpher prototype. The product specification requirements provided in Lab II Section 3.1 can be found in a separate document.

2 General Description

The primary goal of the PolyMorpher prototype is to provide a working demonstration of the PolyMorpher product. This is accomplished by implementing only the necessary components and features of the full product. As a prototype, the product will still be playable like other programming games, while also teaching OOP concepts and problem solving skills. OOP concepts that will be taught include: abstraction, encapsulation, polymorphism, and inheritance.

2.1 Prototype Architecture Description

PolyMorpher prototype architecture will be identical to that found in the completed product. The prototype for PolyMorpher will be separated into three main components: the PolyMorpher application, the Unity file structure, and the PolyMorpher website.

- The PolyMorpher application: A stand-alone downloadable executable file that allows the user to play the game locally on a personal computer. It will be downloadable directly from the PolyMorpher website.
- The Unity file structure: The entirety of the Unity file structure is contained in the PolyMorpher application. The “StreamingAssets” directory in the Unity file structure will be directly accessed and modified by the user for use in “morphing” game objects.
- The PolyMorpher website: The website will contain the PolyMorpher application for users to download. A play guide will also be available here for users to view game rules and tips.

Figure 1 shows the website and components of PolyMorpher, and how those components interact with each other.

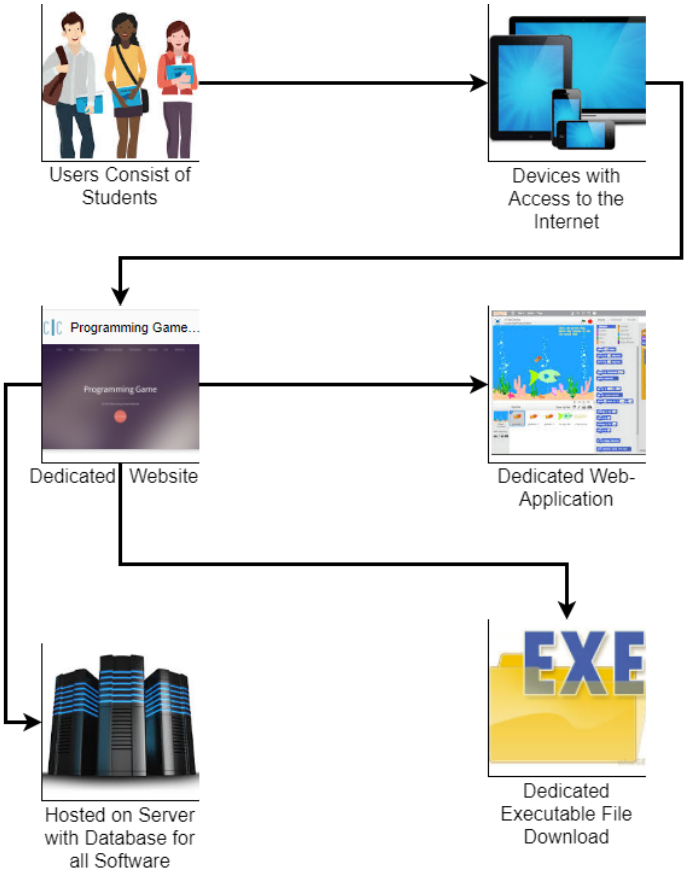

Figure 1: Major Functional Components Diagram - Prototype Version (Team Silver, 2017)

2.2 Prototype Functional Description

The PolyMorpher prototype will be compatible with virtually all operating systems: Windows, Linux, and MacOS. Both the completed product and the prototype will be able to be downloaded off the PolyMorpher website. A summary of the PolyMorpher prototype deliverable features can be seen in Tables 1 and 2. The primary differences between PolyMorpher's completed product and its prototype are displayed in Table 2.

KEY
Fully Functional
Partially Functional
Eliminated

Table 1: Key for Features of the Complete Product versus the Prototype (Team Silver, 2017)

Elements	Description	Real World Product	Prototype
Teaches Polymorphism	Provision of a single interface to entities of different types		
Teaches Abstraction	Technique for arranging complexity of systems		
Teaches Encapsulation	Building of data with the methods that operate on that data		
Teaches Inheritance	When an object or class is based on another object or class, using the same implementation		
Single Language Taught	A single programming language will be focused on C#.		
Single Player	Focused on an experience targeted to interact with only one player		
Downloadable .EXE File	Desktop application version of the game		
Game Assets	Primary components that are used as building block to construct the more complex features and levels of the game		
Developed Story	Narrative used to drive progression or direct player throughout a more guided/linear experience		
Portable Compiler	Code compiler used to run player-made code on the fly in game		
Tutorial Section	Precursor series of levels meant to help the player adjust to the in-game toolset given to them and also prep them with knowledge of the language(s) they will be working with		
Player-Made Content	Variant of Sandbox Level, potentially allows the player to share custom levels with one another		
Sandbox Level	Open level where the player has access to all tools at once and can build their own level sequences and puzzles		
Multiple Languages	Alternative programming languages for the player to use and learn in-game		
Multiple Player	An experience geared toward multiple players interacting with a game environment together		
Web Application	Web based version of the game running in-browser		

Table 2: Features of the Complete Product versus the Prototype (Team Silver, 2017)

Table 2 shows that the prototype will not be implementing multiple programming languages as alternative programming languages for the player to use and learn in-game. It will not be implementing the multiplayer gameplay feature of experience being geared toward multiple players interacting with the game's environment simultaneously. The prototype also will not be implementing the ability to access and play the game through a web-application;

instead, a downloadable executable file will be available for game access. The features that will only partially be implemented are the player-made content, along with the sandbox level. These features allow the player to share custom levels with other players. A sandbox level is defined as an open level where the player has access to all tools at once and can build their own level sequences and puzzles. All else will be fully implemented.

2.3 External Interfaces

There are five types of external interfaces for PolyMorpher: Hardware Interface, Software Interface, User Interface, API Book Interface, and Compiler Interface. The player will also require a compatible device with the necessary hardware and software specifications to operate the game.

2.3.1 Hardware Interface

PolyMorpher is designed to operate on the user's local machine. For the specific hardware the player has to have to run the game, a fourth generation i3 Intel Processor is the minimum requirement. With PolyMorpher being a two-dimensional game style, it will utilize minimal resources.

2.3.2 Software Interface

All of the software behind PolyMorpher is hosted on the CS server at ODU. The specific software in order to run the game is Windows 7, 8, 10, or Linux or MacOS operating system. The software required to run the game is cross-platformed so that way basically anyone in the world can play it.

2.3.3 User Interface

- Computer screen: This is used to display the game after the user downloads the executable file off PolyMorpher's website. The game will be displayed in the Unity gameplay interface; it will not stop running until it is terminated. Figure 2 shows a technical demonstration of a rapid prototype interface a user could see displayed on the screen.

Figure 2: Rapid Prototype Technical Demo of PolyMorpher, By: Joel Stokes (Team Silver, 2017)

- Keyboard: This is used for moving the main character, moving in on objects to "morph," and searching the API Book interface for help with C# coding conventions. The keyboard is also used for typing in the C# code in the text editor box to "morph" each object.

- Mouse: This is used for selecting each object to “morph”, choosing the API Book interface to open in a text box, clicking multiple buttons to submit and cancel code, and traversing the general TUI of the game.

2.3.4 API Book Interface

The API Book Interface is opened when the API Book button is clicked in the game. The API Book Algorithm describes the functionality of PolyMorpher’s API Book button in the game. The API Book Algorithm acts as the primary method of information distribution from game designer to player. It interacts directly with the Compiler Algorithm by determining the knowledge base the player has to exploit in the Compiler Algorithm. It also directly influences the outcome of the gameplay/challenges by offering the player a multitude of tools to interact with their environment. The dataflow algorithm for the API Book Algorithm is displayed in Figure 3.

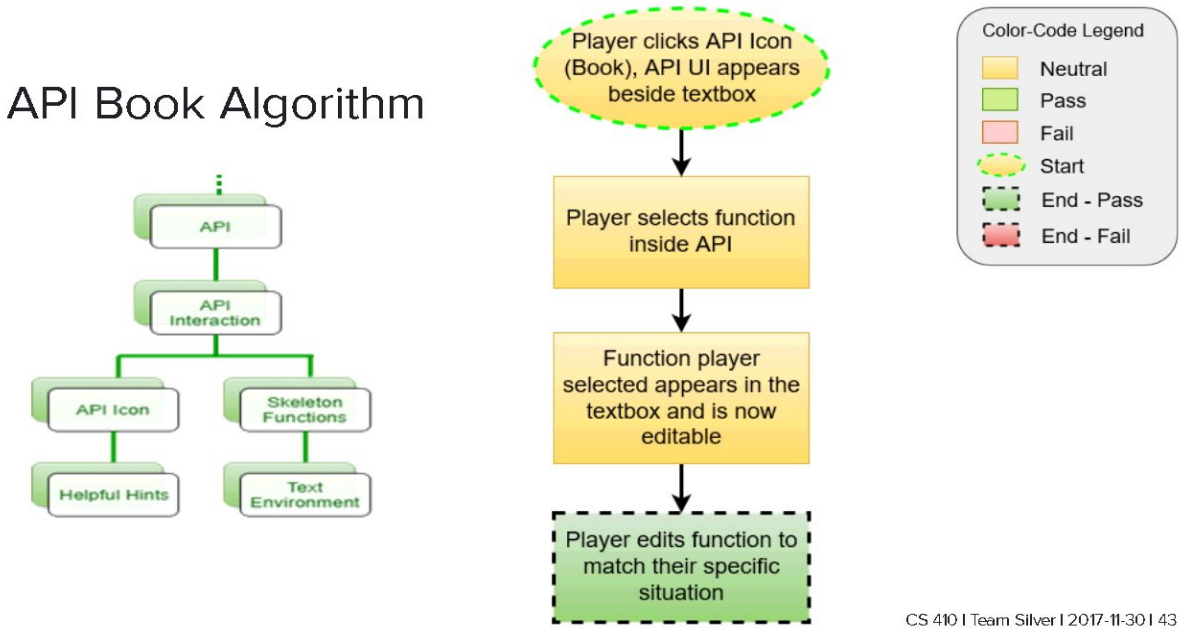

Figure 3: Dataflow Algorithm - API Book Algorithm (Team Silver, 2017)

2.3.5 Compiler Interface

The Compiler Interface is opened whenever any editable object is selected with the mouse by the user to "morph." The Compiler Interface is essentially a text box that includes incomplete C# code that is to be edited by the user to create syntactically correct code. The code can be made correct to fit each unique situation, and to make each editable object have the desired functionality to complete each level in the game. The Compiler Interface with compile, reset, and cancel buttons is shown in Figure 4.

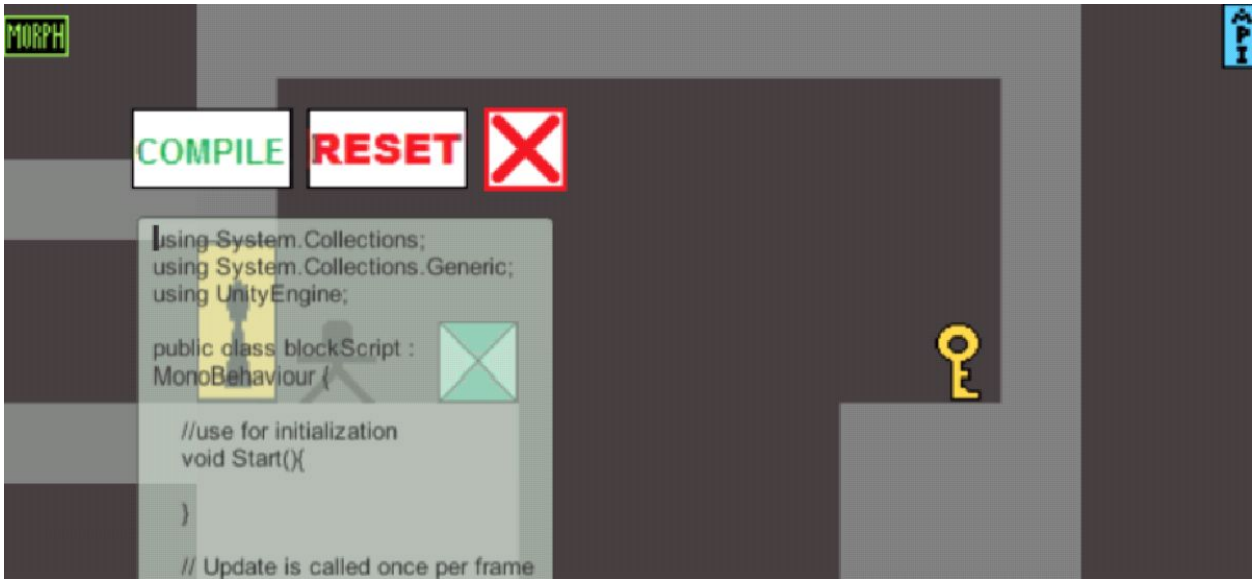

Figure 4: CS410 Technical Demonstration #2, By: Casey Batten (Team Silver, 2017)

When the compile button is clicked in the Compiler Interface, the Compiler Algorithm processes the code that was input and returns it back as either passing or failing to compile. In addition, the Compiler Interface allows the player to write custom scripts in order to morph certain objects to progress in PolyMorpher. The Compiler Algorithm controls and continuously affects the back-end system of the game itself. It directly determines the behavior of objects in the game's environments at a fundamental level. It is also responsible for the amount of control

and open design power the player is granted during gameplay. The Compiler Algorithm is, in fact, co-dependant on the API Book Algorithm. This is based on the tools the player is likely to find there to use within the Compiler Algorithm's in-game dependencies. The dataflow algorithm for the Compiler Algorithm is displayed in Figure 5.

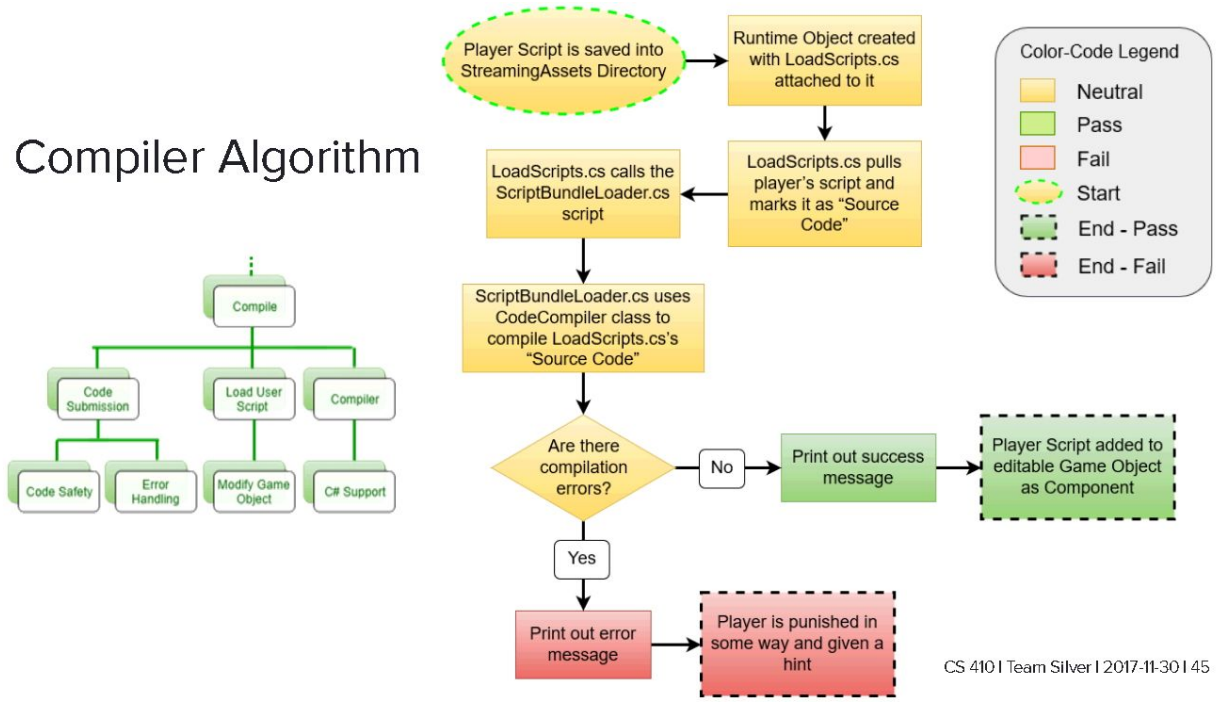

Figure 5: Dataflow Algorithm - Compiler Algorithm (Team Silver, 2017)

3 Specific Requirements

The functional requirements of the PolyMorpher prototype, found in section 3.1, are located in a separate document. Lab II Section 3.1 contains all requirements necessary to complete the prototype. Each requirement contains specifications to ensure that nothing is to be missed during implementation.

Appendix

PolyMorpher will have all of the features and capabilities necessary to be a successful and competitive programming game. If any issues arise, Team Silver will take all necessary actions to make sure that this prototype is built on time and properly. Each team member has been assigned a role and will work in tandem to ensure that the prototype is completed as specified; all of the important requirements will be made first priority. Any extra features or plugins will be implemented after the completion of the main requirements to make sure the programming game is fully functional upon its launch.

3.3 Reasons for Creating PolyMorpher for Students

Recent statistics show that students in the CS degree program at ODU are in decreasing class sizes as they progress from the lower level to the upper level CS classes. Some of this decrease in class size may be due to the fact that some of the lower level CS classes are required for other majors as well as CS, wherein the higher level CS classes are only required for CS degrees. However, there is still a significant decrease in class size when the focus is only on CS students. PolyMorpher is going to help balance and lessen this decrease in class size as CS students progress through their CS degree.

Reiterating the Problem Statement, students are in desperate need of a supplemental resource that will teach them the programming skills needed to pass CS classes at ODU. Once PolyMorpher is introduced to students, there will be a drastic reduction in situations where they do not understand the basic fundamentals of CS, fail CS classes, drop out, or switch majors.

In addition, there is a ‘Student Progression Dilemma’ at ODU. Students are not following the CS course series in the expected order. The collected data provides purpose behind the need

to reassess methods of assisting students' progress in how they learn the topics at hand. The provided data also shows exactly when during the learning process that this assistance would be most effective. Changes in class volume could indicate that students are leaving the CS major for less programming intensive fields.

Furthermore, there is a significant decline in enrollment of students who progress from the classes CS150 to CS250, and then continue on to the classes CS330, CS361, and CS350 at ODU. This decline in enrollment may be directly related to differing major requirements and course overlap. However, the decrease in student body is still significant enough to warrant a deeper look into later classes in the major path. In addition, decreasing class sizes show a steady decline in CS course enrollments as course level difficulty advances. These decreases may be indicative of students either dropping out of the CS program or changing majors. The Student Progression Dilemma statistics from the ODU Factbook are displayed in Figure 7 as a pie graph.

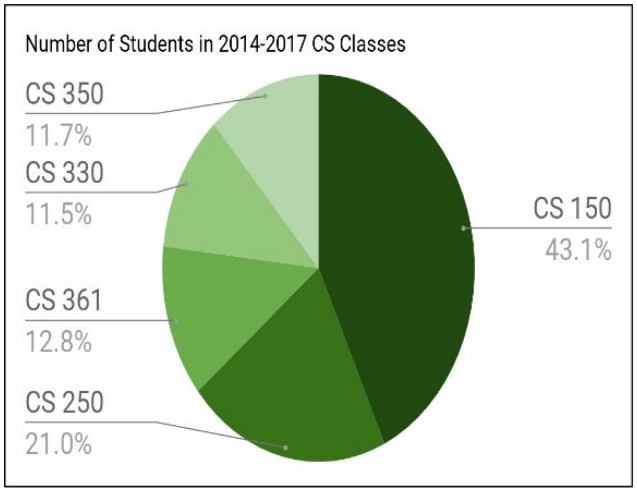

Figure 7: Student Progression Dilemma - Pie Chart (Team Silver, 2017)

Figure 7 further shows the breakdown of the total number of students from 2014 to 2017 in CS classes at ODU. According to the ODU Factbook, from 2012 to 2016 the number of

undergraduate CS majors increased from 284 to 429. This shows the high demand in the CS degree path at ODU. From 2014 to 2015 there were roughly 672 students enrolled in CS150, compared to only about 327 students enrolled in CS250 from 2015 to 2016. In the years of 2016 to 2017, there were roughly only 199 students enrolled in CS361, 180 students enrolled in CS330, and 182 students enrolled in CS350. Figure 7 displays these statistics in the form of percentages to emphasize the impact of decreasing course sizes.

The ‘white noise’ of students progressing from the classes CS150 to CS250 at ODU has to be accounted for. There is a less dramatic decrease in course sizes than initially thought from the CS courses that are requirements for other majors. For example, CS150 is the first course that CS majors have to take and is considered a ‘service course’ by ODU. This means that it is also required to be taken by Physics, Math, Engineering, and Mod-Simulation majors. CS250 is required to be taken by CS, Mod-Simulation, and Computer and Electrical Engineering majors. CS330 is required to be taken by CS and Mod-Simulation majors. CS361 is required to be taken by CS and Computer and Electrical Engineering majors. Lastly, CS350 is required to be taken by CS and Computer Engineering majors. Table 3 further displays the Student Progression Dilemma taking into account the ‘white noise’ of overlapping different majors.

[This space intentionally left blank]

	CS 150	CS 250	CS 361	CS 330	CS 350
2013-2014	804	327	161	111	93
2014-2015	672	367	208	203	148
2015-2016	937	327	217	195	183
2016-2017	920	337	199	180	182

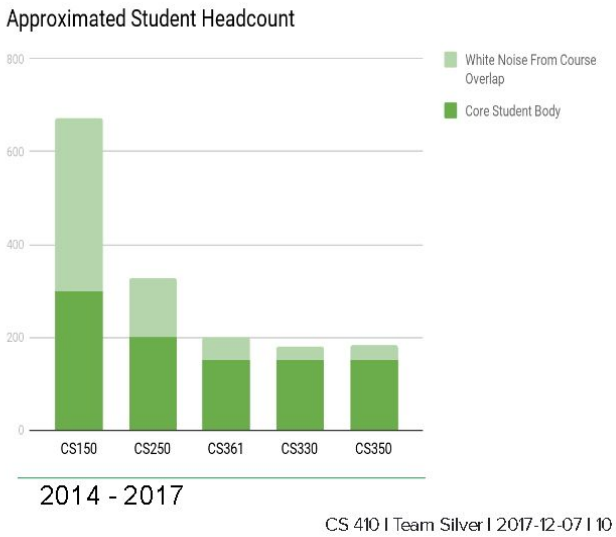

Table 3: Student Progression Dilemma - Table & Graph (Team Silver, 2017)

As shown from Table 3, the approximated student headcount of the core student body for CS majors is significantly less than when also considering the 'white noise' from course overlap. Again, the values with the 'white noise' from course overlap are shown in Figure 7. The core student body from 2014 to 2017 consists of approximately 300 students in CS150, 200 students in CS250, 175 students in CS361, 180 students in CS330, and 178 students in CS350. As evidenced above, only CS150 and CS250 are drastically affected by the 'white noise' from course overlap.