The Icarus Protocol

Project Requirements and Specifications

Lincoln Middle School

Collin Nelson;

Anna Ueti;

9.27.22

TABLE OF CONTENTS

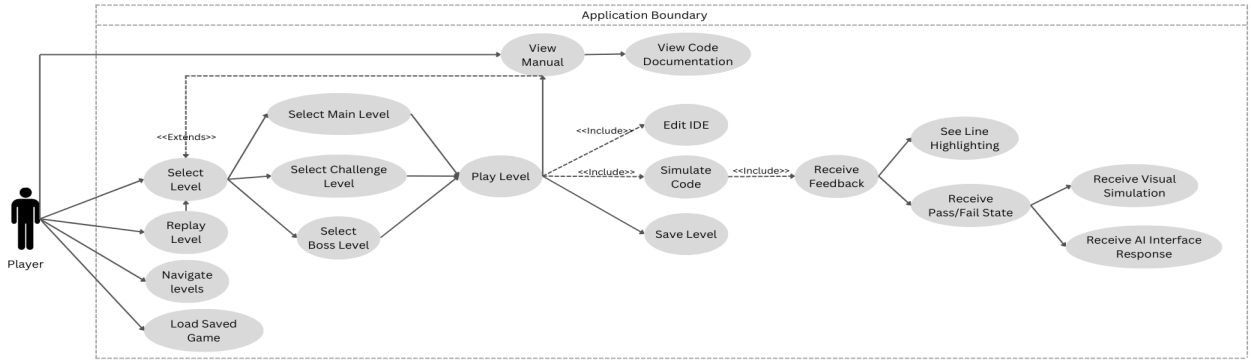
I. Introduction

In 2021, the games industry reported estimated revenues of about 198.40 billion dollars, solidly positioning themselves as one of the world's most immensely popular and valuable forms of entertainment. This growth is only projected to continue rising as well, with market research firms anticipating revenues as high as 339.95 billion by 2027 (*Gaming market size, share: 2022 - 27: Industry growth* 2021). The power of games to capture the minds and attentions of players across the world cannot be discounted, and given this power, there is also incredible potential for games to capture the attention of players and direct it towards learning and personal advancement as well as towards entertainment.

There is a powerful overlap between games and education, an overlap that innovators have only recently begun to truly explore. The feedback loops and reward models designed and honed to keep people engaged in modes of entertainment can be useful for intriguing students in classwork with a reputation for being uninteresting. Our team aims to explore the potential of games as a tool for education by producing a fully featured game designed to teach basic programming skills to students at a middle school or early high school level with no prior programming experience. While games of a similar nature exist, they often fall into one of two traps which our team sees as pitfalls. Some, while employing the surface level appearance of a game, fail to truly embody the game design principles that make games powerful for learning; Others use a proprietary scripting language that has lessened impact in teaching actionable programming skills. In the construction of this project (working title: "Icarus Protocol") we aim to solve this problem by producing a game that is a genuinely fun and interesting experience, while also serving as an effective tool for teaching real Python programming.

II. System Requirements Specification

II.1. Use Cases



*See the glossary for more in depth descriptions of diagram stakeholder

Select Level

Pre-condition	- On level select screen
Post-condition	- Level information is displayed

	- Play button is available
Basic Path	1. Locate level icon in level list 2. The user selects a main mission by clicking 3. The level name, and progress is displayed
Alternative Path	- In the 2nd step, the user selects a challenge level. Instead of level progress by phase the complete and incomplete objectives are displayed.
Related Requirements	- Level-Select - Main Levels - Challenge Levels

Replay Level

Pre-condition	- On level select screen
Post-condition	- Level information is displayed - Play button is available
Basic Path	1. Locate a completed level icon in level list 2. Click on level to show data about level
Alternative Path	
Related Requirements	- Level-Select

Navigate Levels

Pre-condition	- On level select screen - Level information for a level is being displayed
Post-condition	- New level information is displayed - Play button is available
Basic Path	1. Locate a different level icon in level list 2. Click on level to show data about level
Alternative Path	1. The user interacts with hotkeys (such as arrow keys) or with menu buttons to navigate to the next level or previous level.
Related Requirements	- Level-Select

Load Saved Game

Pre-condition	- On main menu screen
Post-condition	- Previously completed levels are reflected in level list
Basic Path	1. Locate the Continue button 2. Click on the Continue button
Alternative Path	
Related Requirements	- Saving and Loading - Cloud Save and Load

View Manual

Pre-condition	- On level select screen
Post-condition	- Manual of code examples is displayed
Basic Path	1. Locate manual icon 2. Click manual icon
Alternative Path	1. At the precondition step, Player is on the Level screen
Related Requirements	- Accessible In-Game Python Documentation

Play Level

Pre-condition	- Selected level and main level type
Post-condition	- Receive level feedback
Basic Path	1. The player enters the level, the screen displays a section for the python IDE, the ship simulation, and the instruction/tip panel. 2. The player click on the panel for the python IDE, typing to modify the python code 3. The player clicks on Simulate button 4. The Python background tests check and determine whether the player's code is successful or not. 5. The code is successful, the ship simulation plays an animation for the task being successful 6. The level data is saved
Alternative Path	- At the 5th step, the code is not successful, the ship simulation plays an animation for the task failing, and player returns to the 2nd step.
Related	- Functional Python IDE

Requirements	- Operational IronPython Integration - IDE Syntax Highlighting

II.2. Functional Requirements

II.2.1. User Interface and Python IDE

Functional Python IDE: The application needs to contain a functional Python IDE for writing and simulating code in-game. This IDE needs to allow the user to write code either freely, or by filling in blanks in a fixed code-block. The resulting code then must be packaged as a python string to be simulated by the engine.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 0: Essential and required functionality.

IDE Syntax Highlighting: The Python IDE should implement some level of basic syntax highlighting. This highlighting should be able to recognize and distinguish between strings, integers, and applicable keywords, highlighting them with different text colors to make programming easier. In addition, background colors or lines in the IDE should be used to indicate the tab index of a block of code, helping to distinguish what is contained within certain code blocks.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 1: Desirable Functionality

Level-Select: The application needs to feature a UI that allows the players to select a level, view level details, and choose to play the level if they wish. This UI needs to accommodate the potential addition of levels outside of a linear mission order, should such levels be added to the game. This UI must also limit players to selecting levels they have unlocked by completing certain prerequisite conditions.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 0: Essential and required functionality.

Accessible In-Game Python Documentation: The game must include a documentation window in its UI, accessible from the menu screens or from within missions. This documentation must include explanations of the concepts introduced in the games, including example code and detailed instructions and tips. These entries should be itemized by concept, and should unlock as the concept is introduced in the game, hiding information that is not yet relevant to the user.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 0: Essential and required functionality

II.2.2. IronPython Integration

Operational IronPython Integration: The game must contain a successful integration of IronPython, a popular Python to .NET integration, with our C# code. This integration must allow us to run user-generated code, which must be sandboxed to prevent unintended access to the game code or the operating system. Errors must be handled gracefully, with no interruption to the game itself.

Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 0: Essential and required functionality.

II.2.3. Core Game Systems

Main Levels: The game must feature a collection of at minimum 5 main “levels”. Each level will consist of multiple phases of escalating complexity, introducing, developing, and synthesizing a distinct concept. Specifically the game must at a minimum introduce the concepts of statements, conditionals, loops, and lists.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 0: Essential and required functionality.

Boss Levels: The game will end with at least one “boss level”. The goal of this level is to synthesize a variety of already-introduced concepts into a single mission. This is distinguished from main missions by the fact that it includes no new concepts, but instead uses multiple phases of increasingly difficulty to challenge existing knowledge.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 1: Desirable functionality.

Challenge Levels: The game will include a variety of “challenge missions”. These missions are short, but difficult missions consisting of a single phase. This phase contains a unique challenge based on knowledge acquired in previous missions, but doesn’t introduce any new concepts. These challenges can be used to develop skills and provide additional completion objectives for motivated users.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 2: Extra features or stretch goals.

Saving and Loading: As the player completes objectives, the game should automatically store their accomplishments, including facets of level completion, level unlocks, and the last simulated code for each level. This saved data should be automatically loaded if the player selects to “continue” from the main menu. This data will be stored locally in an accessible location of the user’s computer.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 0: Essential and required functionality.

Cloud Save and Load: The game will store the saved data somewhere attached to a cloud storage service (i.e. Google Cloud tied to the user’s Google account) along with local save. If the player is utilizing this cloud storage then the game will sync the local save data with the cloud save data upon startup.
Source: Internal requirements elicitation among members of the team.
Priority: Priority Level 2: Extra features or stretch goals.

II.3. Non-Functional Requirements

Self Containment:

The system shall be fully self contained. In this context what that means is that completing the game and learning the material shouldn’t require any outside Googling, or instructor help. All the resources and help should be provided in-game.

Fun:

The game shall be fun to play. It should be an experience that students enjoy going through, that teaches while engaging the player in a satisfying and enjoyable way. This can be measured through playtest surveys and player metrics.

Effective Teaching:

The system shall be an effective teaching tool. An average student playing through all of the main missions and engaging with an adequate amount of side-content should exit the experience having learned actionable skills in alignment with grade-appropriate standards. (*Ospi,* 2018)

Actionable Skills:

The system shall teach skills which extend outside of the realm of the game. An average student who completes the game and demonstrates understanding of in-game concepts, should also feel more confident in their understanding of Python, and their ability to use real programming principles.

Efficient Performance:

The system shall perform efficiently enough to be playable on machines available to students at LMS. “Playable” means that the game should generally maintain a framerate of >30 fps without clear stuttering or slowdowns during normal gameplay.

Intuitive Use:

The game should be intuitive to use for most students, and should not require the student to read detailed installation instructions, or operating procedure in order to understand how to play the game and interact with its systems.

Visual Interest:

The system shall be visually interesting and engaging to look at. This means that most actions should result in some kind of visual and/or auditory feedback to the user, and that success and failure should be marked with distinct visual effects.

III. System Evolution

One of the major factors of consideration within this design is Software Evolution. Part of the nature of this project is user play testing with the students within Lincoln Middle School (LMS). Once the alpha build of the game is available, our team will have a small group of testers at LMS play it and give feedback regarding their understanding and any problems that arise. Additionally, our team will debrief with our contact liaison, Mr. Davis, who is the student’s technology teacher, about the teaching environment while the students were testing and any other observations and feedback he has. Our team will do this testing process several times. With each iteration, we will refine, refactor, or modify the game so that the experience is improved. With this process in mind, we will create code that is modular and well documented so that it is easy to modify and extend, as needed. Furthermore, our team is aware that this is a preliminary testing design and is subject to change in the future.

Additionally, our team understands that the lesson requirements should be modular, as the passage of time may effect the expected standards from our client or the CSTA 6-8th grade learning standards for computer science (*Ospi,* 2018). As such, our team will design the game with modularity in mind, making it easy to update and improve curriculum as needed.

IV. Glossary

Player: Player is term refering to our primary application user.

Feedback Loops: A term in game design which refers to using the players success or failure to adapt and adjust the output of other systems. This could mean adjusting up the difficulty of challenges if the player appears to be succeeding too easily, or providing hints only when the player shows signs of struggling.

Reward Models: A game's reward model is the structure and array of visual, auditory, and mechanical rewards that are given to the player to incentivize success, as well as the specific situations in which rewards are given to incentivize the desired behavior.

Scripting Language: A programming language, usually an interpreted language, which can be integrated in with a compiled executable to allow for the adjustment or extension of existing features or content without recompilation of the primary executable or exposure of its source code.

Level: A self-contained stage of the game, consisting of one or more phases, and teaching a complete and distinct concept. Analogous to a lesson for the purposes of education.

IDE: Integrated Development Environment: A software or part of a software which can be used to develop other applications. In our case the IDE is the component of our game that allows you to develop functional Python code.

V. References

Computer Science Teachers Association (CSTA), "Computer Science 6-8 Standards," *Ospi*, 2018. [Online]. Available: https://www.k12.wa.us/student-success/resources-subject-area/computer-science/computer-science-k-12-learning-standards. [Accessed: 20-Sep-2022].

"IronPython," *IronPython.net /*. [Online]. Available: https://ironpython.net/. [Accessed: 27-Sep-2022].

"Gaming market size, share: 2022 - 27: Industry growth," *Gaming Market Size, Share | 2022 - 27 | Industry Growth*, 2021. [Online]. Available: https://www.mordorintelligence.com/industry-reports/global-gaming-market. [Accessed: 20-Sep-2022].