Interrobang

Interactive Puzzle Game

Joey Martinez, Merci Magallanes, and John Hardy

# **‽**

# 

# Table of Contents

Title page  
 Table of Contents

1.0 Preliminary Project Proposal

1.1 Description

1.2 Justification

2.0 Proposal Presentation Slides  
  
 4.0 Software Development Plan  
 4.1 Plan Introduction  
 4.1.1 Project Deliverables  
 4.2 Project Resources  
 4.2.1 Hardware Resources  
 4.2.2 Software Resources  
 4.3 Project Organization  
 4.4 Project Schedule  
 4.4.1 PERT / GANTT Chart  
 4.4.2 Task / Resource Table  
 4.4.3 Class Schedule  
  
 5.0 Requirements Specification  
 5.1 Introduction  
 5.2. CSCI Component Breakdown  
 5.3 Functional Requirements  
 5.4 Performance Requirements  
 5.5 Project Environment Requirements  
 5.5.1 Development Environment Requirements  
 5.5.2 Execution Environment Requirements  
  
 6.0 Software Design Description  
 6.1. Introduction  
 6.1.1 System Objectives  
 6.1.2 Hardware, Software, and Human Interfaces  
 6.2 Architectural Design  
 6.2.1 Major Software Components  
 6.2.2 Major Software Interactions  
 6.2.3 Architectural Design Diagrams  
 6.3 CSC and CSU Descriptions  
 6.3.1 Class Descriptions  
 6.3.2 Detailed Interface Descriptions  
 6.3.3 Detailed Data Structure Descriptions  
 6.3.4 Detailed Design Diagrams  
  
 10.0 Testing  
 10.1 Unit Test Plan  
 10.1.1 Unit Test Descriptions  
 10.2 Integration Test Plan  
 10.2.1 Integration Test Descriptions  
  
 11.0 Lessons Learned

15.0 Project Source Code

# 1.0 Project Proposal

## 1.1 Description

*Interrobang* is a looping 3D puzzle solver. Looping means that we will be using the same set of 2 or 3 levels over and over again for each puzzle, with small tweaks made to the level on each iteration. We want our game to be three dimensional because it will give us more opportunities to make the gamer interesting and interactive. The most important part of the game is that each iteration will contain a puzzle where the player must find some secret item or trick to continue to the next level.

Interrobang will be similar to other 3D puzzle games such as: Portal, Anti-chamber, Museum of Simulation Technology, and Mirror’s Edge. Obviously, we do not have enough time to mimic the level of detail of these games, but we plan on using them as a foundation for our ideas. Interrobang will run on most personal computers and will run using the Unity game engine. The target audience for our game will be anyone over 8 years of age who enjoys puzzle games.

## 1.2 Justification

1. This project will allow us to demonstrate what we have learned in game design classes, as well as our ability to learn new programming languages. We will also be working on a decent user interface that will utilize principles we learned in interaction design.
2. The project will involve using C# to program mechanics within the game which will expand our knowledge of programming languages, as well as following through with design ideas and focusing on usability, similar to interaction design. This project will also build on our knowledge of game design, as well as furthering our understanding of Unity.
3. The project can be scaled up or down during the project, as we get a feel of our capabilities. We will start the project by bringing in the basic mechanics of the game, and once these concepts are realized, we can focus on content of the game, and depending how much time we have, either make the game shorter or longer.
4. The scalability of the project allows it to fit within the time constraints of the semester.
5. The tools necessary for this project are Unity, and possibly some 3D modeling software, which both teammates can gain access to.
6. Both teammates have expressed interest in game design, and want to use this experience to further their understanding of game design.

# 2.0 Proposal Presentation Slides

Project proposal presentation slides can be found in the “documents” folder of the Interrobang repository.

# 4.0 Software Development Plan

## 4.1 Plan Introduction

Interrobang is a First Person 3D Puzzle Video Game built around a looping map design. We believe this project will be an appropriate project for 401, as it will force us to see through design decisions, and realize these ideas through code. It was also allow us to learn new software, as well as the C++ programming language. The project is also scalable, as the basic idea for the game is easily doable within a single, but can grow far beyond this, depending on how much our team is able to get done. Likewise, this will allow us to learn skills necessary for working in a group, and how to properly divide work so that it will get done in the most efficient manner. Development will involve learning basics of Unity and Maya, brainstorming ideas for the game, learning best practices for implementing these ideas, modeling basic objects for the game in Maya, and developing the mechanics of the game through various C++ scripts. The culmination of this process will result in a fully functioning game with most if not all of the brainstormed ideas working as intended.

|  |  |
| --- | --- |
| Sub-task Completion Dates | |
| Basic movement is working in Unity: Basic character movement is working | Week 5 |
| Basic Level is finished in Unity: The general format of what the main level within the game is finished | Week 5 |
| Brainstorming for assets: Assets the project needs to get developed in Maya are discussed so that they can eventually make their way into the game. | Week 6 |
| Level Testing: Level is tested against the player movement. Changes are made until both movement and level complement each other | Week 6 |
| Asset Creation: Assets have been created and added to the game. More assets are given to the asset creator to continue creating. | Week 7 |
| Visual Design: Visuals of the game are discussed, art director is tasked with converting the basic level into a level that incorporates this visual design concept. | Week 7 |
| MVP Completed: Minimal Viable Product finished, which includes character moving flawlessly in the main level, with the character able to loop through the level five times, and complete five different puzzles each time, to demonstrate the core tenants necessary for the game. |  |

### 4.1.1 Project Deliverables

|  |  |
| --- | --- |
| Project Deliverables | |
| Project Proposal Presentation / Document  ● Present powerpoint and document outlining Interrobang to class | Week 02 |
| Requirements Specification  Initial Development Schedule (Part of SDP draft)  ● List out what we will need for the project and create an initial schedule to get things done by  Oral Status Reports  ● Oral description of how Interrobang is moving along. | Week 05 |
| Written Status Reports in the project SDF  ● Actual written document outlining progress on Interrobang | Week 06 |
| Software Development Plan Document  ● The entire finished plan for how Interrobang will be developed | Week 07 |
| Written Status Reports in the project SDF  ● Actual written document outlining progress on Interrobang | Week 08 |
| Written Status Reports in the project SDF  ● Actual written document outlining progress on Interrobang | Week 10 |
| Project Preliminary Design Review Presentation  ● Show off the minimum viable product of what we have so far.  Software Design Description Document (Architecture Section)  ● Go into gorey detail about how Interrobang works in a description document. | Week 11 |
| Project Critical Design Review Presentation  ● Show off what changes we have implemented from previous week  Software Design Description Document  ● Finishing touches on description document  Written Status Reports in the project SDF  ● Actual written document outlining progress on Interrobang | Week 12 |
| Presentation/Demonstration  ● Formal presentation of Interrobang  Test and Integration Plan  ● Formal plan on how testing will be implemented into game  Written Status Reports in the project SDF  ● Actual written document outlining progress on Interrobang | Week 14 |
| User's Manual Final Updates  ● Finished user’s manual with details on how to play Interrobang  Oral Status Reports  ● Oral description of how Interrobang is moving along. | Week 15 |
| FINAL Product Delivery (Final Report and Code)  ● Final report on how development of Interrobang turned out along with all the code used for the game.  ● December 19th  User's Manual Final Updates  ● Finished user’s manual with details on how to play Interrobang | Week 16 |
| FINAL Project Presentation  ● Final presentation of Interrobang | Week 16 |

## 4.2 Project Resources

### 4.2.1 Hardware Resources

|  |  |  |
| --- | --- | --- |
| Resource | Development | Execution |
| Windows 10  Intel Core i7, 2.4 GHz  8 GB ram | X | X |
| Mac OS  Intel Core i5, 2.7 GHz  8 GB | X | X |
| Windows 10  Intel Core i3, 2.3 GHz  8 GB ram | X | X |

### 4.2.2 Software Resources

The two main resources that will be used in this project are Unity and Maya. Unity will handle the bulk of the core game design processes, as well as house the finished video game application. Maya will be used to design and model assets for the game to be imported into Unity. Music composition software Finale and music mixing software Fl Studios will be used as well in order to develop sound clips and music for the game. Visual Studios will be used to write and compile code written in C++.

|  |  |  |
| --- | --- | --- |
| Resource | Development | Execution |
| Windows 10 OS/Mac OS | X | X |
| Unity | X |  |
| Maya | X |  |
| Visual Studio | X |  |
| Finale | X |  |
| Source Tree | X |  |
| FL Studios | X |  |

## 4.3 Project Organization

● Merci Magallanes: Art Director, Asset Creation/3D modeler, UI Artist, Animator

○ Merci is in charge of creating the overall art direction of the project to make sure the visual style of the game is consistent and complements the rest of the game. Along with this, she is tasked with using Maya to make any assets that are not easily available within Unity. Simple animations may also be necessary, and thus Merci will be the one to handle the situations in which animation is appropriate. Lastly, she will be given the duty of designing a user interface that is both effective and complements the games visuals.

● Joey Martinez: Project Manager, Game Designer, Level Designer, Additional Programming, Music Producer/Audio Mixing and Mastering Engineer

○ Joey will handle manage the overall project in terms of organizing each member in the project and their roles within the group. He will make sure that each person knows what needs to get done during each week. Likewise, the overall concept of the game will be designed by Joey and it will be Joey’s job to make sure everyone on the group knows of this direction. Lastly, Joey will handle making sure music and audio clips make it into the game, and that there is an overall acceptable quality to all of the audio files.

● John Hardy: Lead Programmer, UI Developer, Technical Director, Quality Assurance Tester, Version Control Manager

○ John is tasked with developing most of the code used to handle the designs expected to be in the game. He will make sure that all the design decisions developed by Joey will be interpreted into the game correctly through code, so that it best reflects the original intent of the design. Likewise, he is tasked with making sure that all code is tested and working correctly. He will also be tasked with making sure the UI will adhere to the design given by Merci. Lastly, he is in charge of making sure version control happens correctly, and in a clear and efficient manner.

● Christian Alva: Music Composer

○ Christian will handle creating a score that both engages the player and complements the tone and feel of the game.

The Project manager will handle making sure everyone knows the work that needs to get done each week, and to remind each member of the planned schedule. Communication of this will happen at biweekly meetings in the Keck Lab. Communication outside of meetings will happen through Slack, and in urgent cases, text message. All members, except for the music composer, will be pushing code to Github. Each member will be expected to communicate effectively through commit messages, during meetings, and through slack messages the changes they have made so that everyone knows the current state of the project. Overall, version control will be overviewed by the Version Control Manager to make sure conflicts are handled, everything is up to date, and that all commits contain working, efficient, and clean code.

## 4.4 Project Schedule

### 4.4.1 GANTT Chart

The GANTT chart is located in the “documents” folder of the Interrobang repository.

### 4.4.2 Task / Resource Table

|  |  |  |
| --- | --- | --- |
| Task | Persons Assigned | Equipment |
| Basic Character Movement | John Hardy | Computer, Unity, Visual Studio |
| Basic Level Design | Joey Martinez | Computer, Unity |
| Level Testing | John Hardy, Joey Martinez | 2 Computers, Unity |
| List of Necessary Assets | Merci Magallanes, Joey Martinez | 2 Computers |
| Basic Assets Made | Merci Magallanes | Computer, Maya |
| Preliminary Functionality (Death, win teleport) | John Hardy, Joey Martinez | 2 Computers, Unity, Visual Studio |
| Death Screen | John Hardy, Merci Magallanes | 2 Computers, Unity, Visual Studio |
| Win Screen | Merci Magallanes, John Hardy | 2 Computers, Unity, Visual Studio |
| 5 puzzles designed and implemented | Joey Martinez, John Hardy | 2 Computers, Unity |
| Minimal Art Design Included | Merci Magallanes | Computer, Unity, Maya |
| Level Advancement | John Hardy | Computer, Unity, Visual Studio |
| Advanced Movement | John Hardy | Computer, Unity, Visual Studio |
| Basic Animations | Merci Magallanes, John Hardy | 2 Computers, Unity, Maya, Visual Studio |
| Basic Player Interaction (button presses, spikes, etc.) | John Hardy | Computer, Unity, Visual Studio |
| Last Necessary Assets | Merci Magallanes | Computer |
| Miscellaneous Animations | Merci Magallanes | Computer, Unity, Maya |
| 15 puzzles Made and Implemented | Joey Martinez | Computer, Unity |
| Miscellaneous Tweaks (objects aligned correctly, aesthetic appearances, etc.) | Joey Martinez, John Hardy | 2 Computers, Unity |
| Basic Testing | Joey Martinez, John Hardy | 2 Computers, Unity |
| Advanced Player Interaction (Speed up pads, bouncy spikes, Teleportation Pads, etc.) | John Hardy | Computer, Unity |
| Forest Ending Added | Joey Martinez | Computer, Unity |
| Music Mixed | Joey Martinez | Computer, FL Studios, Finale |
| Music Added | Joey Martinez | Computer, Unity |
| Sound FX added | Joey Martinez | Computer, Unity |
| Start Screen | John Hardy, Merci Magallanes | 2 Computers, Unity |
| Setting Screen | John Hardy | Computer, Unity |
| More artwork and Assets added to help tone of game | Merci Magallanes | Computer, Unity, Maya |
| Testing/Bug fixing | John Hardy | Computer, Unity, Visual Studio |
| More Puzzles Added | Joey Martinez | Computer, Unity |
| Voice Over for Poems | Joey Martinez | Computer, Unity, FL Studios, Finale |
| Credits | John Hardy | Computer, Unity, Visual Studio |

# 5.0 Software Requirements Specification

## 5.1 Introduction

Interrobang will be a fun, interactive puzzle game designed to improve spatial awareness and critical thinking. In order to create such a game, several resources are required in order to design, model, develop, play test, and build this game. Most of these requirements will be satisfied using a game development environment called Unity paired with a code-writing environment known as Visual Studio. All other requirements will largely be centered around team communications and version control software.

## 5.2 CSCI Component Breakdown

Interrobang is composed of the following Computer System Configurations:

5.2.1 Game Execution CSC -- Many parts to the program-execution pipeline

5.2.1.1 Graphics Rendering CSU -- In game graphics rendering pipeline

5.2.1.1.1 DirectX 11 Rendering module -- Graphics will be handled on DirectX 11 platform

5.2.1.2 Input Handling CSU -- Input handling process

5.2.1.2.1 Windows Mouse and Key Input Drivers -- Reports mouse and key input

5.2.1.2.2 Player Controller Script -- Handles mouse and key input

5.2.1.3 Audio Handling CSU -- Audio manipulation process

5.2.1.3.1 Windows audio decoder libraries -- Decodes audio files

5.2.1.3.2 Game Manager Script -- Handles the timing of audio decoding and playback

5.2.1.4 Asset Manipulation CSU -- Handles storage and delivery of Assets

5.2.1.4.1 Windows File System -- Stores Assets on Client’s machine

5.2.1.4.2 Unity Game Backend -- Retrieves Assets from File System

5.2.1.5 Game State Management CSU -- Keeps track of current game state

5.2.1.5.1 Game Manager Script -- Stores game state and executes game events

## 5.3 Functional Requirements

Interrobang should function as any other game would be expected to function. The game should initialize when the client starts the game. The game will assume its initial state and stay in that state until the correct inputs are given by the user, then it will carry on to the next state based on the given conditions, and will continue to do so repeatedly. Given a specific state, the game will render a specific set of images to the screen including: 3D objects and 2D Graphical User Interfaces. The game will also output audio based on a specific state as well. When the user wishes to terminate the game, a specific set of inputs must be given and the game will terminate properly. The game will also have safeguards in place for unintended failures such as power-loss, improper termination, and code errors. When the game is restarted, it should load the last known state based on the state saved by the game previously.

5.3.1 The graphics rendering pipeline shall represent the current game state to the user.

5.3.2 The input handling pipeline shall interpret inputs given by the user for the game.

5.3.3 The audio management system shall represent the current game state to the user.

5.3.4 The asset manipulation system shall manage assets for the game.

5.3.5 The game management software shall:

5.3.5.1 control the state of the game.

5.3.5.2 trigger asset utilization.

5.3.5.3 trigger audio playback.

5.3.6 The Unity Game Backend should manage unintended failures.

5.3.7 The game management software should load the game state from the previously saved state of the game.

5.3.8 The game shall allow the player to use the camera to view optical illusions in order to solve puzzles.

5.3.9 The game shall allow the player to utilize the “gravity” of the level by being able to walk/run/jump on the ceiling or the ground at certain times

5.3.10 The opening screen shall display the “How To” button which describes the controls and instructions of the game.

5.3.11 The opening screen shall display the “Play” button which allows the player to run and play the game.

5.3.12 The opening screen shall initially show the player staring down a dimly lit hallway.

5.3.13 The game shall allow the player to walk around the level.

5.3.14 The game shall allow the player to run around the level.

5.3.15 The game shall allow the player to jump around the level.

5.3.16 The game shall allow the player to interact with various objects in order to progress through the level.

5.3.17 The game shall allow the player to interact with various objects in order to progress through the level

5.3.17.1 The game shall allow the player to press a button to open a door.

5.3.18 The game shall allow the player to traverse through the looping levels.

5.3.19 The game shall allow the player to lose the level by running into spikes.

5.3.20 The game shall allow the player to spawn back at the start of the level after losing a level.

5.3.21 The game shall not allow the player to clip through the outer walls of the level.

5.3.21.1 The game shall not allow the player to clip through the exit of the level as an easy way of winning.

5.3.22 The game shall make sure that the player shall be able to collide/interact with the walls of the level.

5.3.22.1 The game shall not allow the player to clip through the platforms of the level.

5.3.23 The game shall make sure that the player’s point of view follows the mouse movement/location

5.3.24 The game shall not allow the player to “slide” around the level

5.3.24.1 The game shall allow the player to walk/run/jump around each level properly

5.3.25 The game shall allow the player to move around the world using the directional keys (WASD).

5.3.26 The game shall allow the player to run using the ‘Shift’ key.

5.3.27 The game shall start with showing the player an intro start screen

5.3.28 The game shall give the player the option to start the game via keyboard press (Enter Key).

5.3.29 The game shall render a 3D image to be rotated around to offer a background to the intro start screen while it waits for input from the player

5.3.30 The game shall fade the intro screen out and replace it with the image of the main game to be played by the player.

5.3.31 The game shall render lighting correctly so that the player can accurately see the screen and the level around them.

5.3.32 The game shall allow the player to slow there movement in the air by holding the back movement key (S key).

5.3.33 The game shall allow strafing in a slower manner than moving forward to back (Movement given from key presses A and D shall move the player more slowly than key presses W and S).

5.3.34 The game shall start the player on a standable ground so as to allow the player to move correctly instead of falling endlessly.

5.3.35 The game shall indicate a clear direction of movement to the player to indicate in which direction they should move initially.

5.3.36 The game shall allow the player to press a key to pause the game (Escape Key).

5.3.37 The game shall allow the player to navigate a menu from the pause screen.

5.3.38 The game shall allow the player to quit the game from the pause menu.

5.3.39 The game shall allow the player to access a settings menu from the pause menu.

5.3.40 The game shall allow the player to navigate a menu from the settings menu.

5.3.41 The game shall allow the player to change mouse sensitivity from the settings menu.

5.3.42 The game shall allow the player to change key bindings for movement of the player.

5.3.43 The game shall allow the player to change the volume of sound effects in the settings menu.

5.3.44 The game shall allow the player to change the volume of music

## 5.4 Performance Requirements

Interrobang should also perform like most other games would be expected to. Given a recommended set of hardware and software specifications, the game should perform at a level the the user finds enjoyable to play. Processes like rendering graphics, audio outputs, and key inputs should be handled quickly so that the game feels responsive. Note that all of these performance requirements will be met only if the game is run with the specified hardware and software needed to run it.

5.4.1 The graphics rendering pipeline shall always display the game at 30 frames per second or more.

5.4.2 The input handling pipeline shall interpret inputs given by the user within 1 second given.

5.4.3 The audio management system shall output specific sounds within 1 second of being triggered.

5.4.4 The asset manipulation system shall retrieve assets for the game within 10 seconds.

5.4.5 The game management software shall change game states given the right user inputs within 1 second.

5.4.6 The game management software shall utilize specific assets given the correct state within 1 second.

5.4.7 The game management software shall trigger audio playback given the correct state within 1 second.

5.4.8 The game management software should load the game state from the previously saved state of the game within 1 second.

## 5.5 Project Environment Requirements

### 5.5.1 Development Environment Requirements

Following are the hardware requirements for development of Interrobang:

|  |  |
| --- | --- |
| **Category** | **Requirement** |
| Processor | Intel Core i3, 2.30 GHz or Better |
| Hard Drive Space | 2 Gb |
| RAM | 4 Gb or better |
| Display | 800x600 or better |

The large 2Gb hard drive space requirement is necessary for the storage of all the assets used by the game. The game uses a dynamic graphics rendering system that allows display sizes less than 800x600 to be used as well as irregular screen aspect ratios. It is recommended that the user run the game at the native resolution of their computer or at 800x600 at a minimum to prevent screen distortions. Sound cards are built-in systems into modern motherboards so it is expected that the user will have a functioning sound card already.

Following are the software requirements for developing Interrobang:

|  |  |
| --- | --- |
| **Category** | **Requirement** |
| Operating System | Windows 10 |
| Graphics Rendering System | DirectX 11 or Newer |
| Input Handling | Keyboard and Mouse Drivers |
| Audio Playback and Decoding | Audio Drivers |
| 3D Asset Design | Maya Studio |
| Compiler | Microsoft Visual C# |

Computers using a previous version of DirectX will not be able to run Interrobang whatsoever. Newer versions of DirectX should be compatible with older games in most cases. Although the game has not been tested on older operating systems, it should still be compatible with Windows 7 and 8.

### 5.5.2 Execution Environment Requirements

The hardware requirements for executing Interrobang are equivalent to those for developing Interrobang. The same hardware clarifications apply as well.

Following are the software requirements for executing Interrobang:

|  |  |
| --- | --- |
| **Category** | **Requirement** |
| Operating System | Windows 10 |
| Graphics Rendering System | DirectX 11 or Newer |
| Input Handling | Keyboard and Mouse Drivers |
| Audio Playback and Decoding | Audio Drivers |

The properties specified above regarding older versions of Windows and DirectX still apply when executing Interrobang.

# 6.0 Software Design Description

## 6.1. Introduction

### 6.1.1 System Objectives

### 6.1.2 Hardware, Software, and Human Interfaces

## 6.2 Architectural Design

The overall architecture of the project will be split between the user interface, the player controller, and the game manager. Graphics and physics systems will be modeled and controlled by the Unity game engine. The user interface will be presented in a menu, and will handle starting the game, exiting the game, or changing settings within the game. The player would be confronted with these menus when starting up the game program, and when pausing the game while it is running. The user interface will not relate to when the player is actually playing the game, as the player is only maneuvering through the level, and should not be bothered by menus. The player controller script will handle character movement and game mechanics that the player can perform. The way the character moves throughout the level will be managed within the player controller and all updates to how the player moves will be facilitated here. If the player has any changes with its overall mechanics, this will also be handled here. The game manager script will handle all other interactions outside of the player. All states of the level, as well as scene transition, and level interaction with the player, will be handled by the game manager script. In short, the game manager handles the current overall game state, including how the player has interacted with the level, and whether the player has won or lost at any given moment.

### 6.2.1 Major Software Components

### 6.2.2 Major Software Interactions

### 6.2.3 Architectural Design Diagrams

## 6.3 CSC and CSU Descriptions

### 6.3.1 Class Descriptions

6.3.1.1 Detailed Class Description 1

. . . 6.3.1.n Detailed Class Description n

### 6.3.2 Detailed Interface Descriptions

### 6.3.3 Detailed Data Structure Descriptions

### 6.3.4 Detailed Design Diagrams

## 6.4 Database Design and Description

### 6.4.1 Database Design ER Diagram

### 6.4.2 Database Access

### 6.4.3 Database Security

# 10.0 Testing

Many of the features in Interrobang are complex and require a wide swath of testing to ensure their functionality. Because of this, we have implemented a series of individualized and generalized testing schemes to test all areas of our interactive puzzle game. These including everything from manual benchmarking of specific tasks to full-blown beta testing with inexperienced users.

## 10.1 Unit Test Plan

When designing individual tests for each unit of our software, it is imperative that we test all aspects of each module. Otherwise, exploits and bugs will quickly become apparent in our game and will diminish from its immersiveness.

### 10.1.1 Unit Test Descriptions

Our unit tests mostly consist of manual self-performed tests on the module in question. Most often, this is as simple as running the game and testing that there is no glitchiness or unexpected behavior with that given component. Such testing has been conducted on the following modules to verify the following results:

10.1.1.1 The W, A, S, and D keys control movement in the forward, left, back, and right directions respectively.

10.1.1.2 The mouse can be moved upwards, downwards, left, right, and in any combination of any two directions to created camera movement in that respective direction.

10.1.1.3 Holding the shift key while the player is moving increases the player’s movement speed.

10.1.1.4 Pressing the spacebar causes the player to “jump” by lifting them off the ground temporarily.

10.1.1.5 Placing the player over a button causes it to be pressed.

10.1.1.6 Holding the player over a pressed button keeps the button in the pressed position.

10.1.1.7 After a short period of time while not pressed by the player, the button will return to the “up” state. Button functionality continues as normal.

10.1.1.8 Doors can be opened, and will not open any further from their desired open position.

10.1.1.9 Doors can be closed, and will not close any further from their desired closed position.

10.1.1.10 When the player touches spikes or falls out of the map, they “die” and are moved to a spawn location.

10.1.1.11 The players spawn location changes as they move through the levels.

10.1.1.12 When the player lands on a moving platform, they stick to it and are moved along with the platform.

## 10.2 Integration Test Plan

When designing a fully functional game with minimized bugs, it is imperative that all components not only function as expected on an individual basis, but that they work as one cohesive unit during gameplay.

### 10.2.1 Integration Test Descriptions

Our integration tests mostly consist of what would be considered ad hoc testing. The idea was to run the game environment in as close to the true environment it would be run in once it is finished. This consisted of having many people play through various sections of the game to ensure no unexpected behavior or bugs within the game. This was was deliberately chosen to most closely reflect game state, as well as *adhere* to certain game design practices that involve game testing and debugging. The integration of various unit tests yields the following results:

10.2.1.1 When the button is pressed, the door opens, swinging out 90 degrees exactly, allowing player passage.

10.2.1.2 When the player dies, they are spawned at the beginning of the level they were currently playing through, but only after they have pressed the button from the previous level. Otherwise, the player respawns as the beginning of the game.

10.2.1.3 The player navigates through progressively harder levels by pressing a button to open a door leading to the next level.

10.2.1.4 When the player dies, the spawn location that corresponds to the level that the player died in is used to respawn the player.

# 15.0 Project Source Code

All Interrobang source code can be found in the Interrobang repository following this path:

“Interrobang/Assets/Scripts”

Note that there are several scripts in this directory that have been imported and modified from external sources. These include: FirstPersonController.cs, FOVKick.cs, LerpControlledBob.cs, and CurveControlledBob.cs. These scripts have been imported from the first person controller asset provided by Unity’s standard assets package system and were then modified to meet the needs of our project.