Test Specification

Introduction

Goals and Objectives

GameForge is a graphical tool used to aid in the design and creation of video games. A user with little or no experience with Microsoft DirectX and/or Visual C++ programming can construct his or her own basic 2D-arcade games. GameForge also assists experienced programmers by generating the Microsoft DirectX code and Microsoft Windows9x overhead necessary for basic game construction, allowing them to concentrate on more detailed game design issues and implementation. The idea is to limit the amount of actual code written by the user, while providing an interface that is easy to use yet complex enough to remain functional.

The testing process for GameForge has a number of goals. The software will be thoroughly tested for coding bugs and logic errors. In addition to testing for bugs, GameForge will be tested to ensure that it is of the utmost quality. It will be expected exhibit the following qualities: well-executed software, high production values, easy to use interface with a common Windows 'feel', and a full featured engine with reasonable performance on wide variety of machines.

Statement of Scope

GameForge will be tested on a number of levels, beginning with unit testing (using white box testing methods), integration testing (using black box testing methods), validation testing, and ending with high order system testing with a public beta.

A number of design principles will be validated during the testing process. The interface should be easy to use. Data should read and write flawlessly from the database. The creation wizards and level editor must be robust, and data must export flawlessly to the game engine.

The game engine's unaltered code should produce bug free results. Data files created by the world builder must be read in flawlessly. The engine should also maintain a reasonable frame rate on various machine speeds.

The help files must be robust and easy to navigate. Tutorials must be complete and easy to understand. The manual must sufficiently compliment the help files. The software should install easily and flawlessly, and the installation of DirectX 7.0a should be part of the install. Uninstallation must completely uninstall only the files that are part of GameForge.

Major Constraints

GameForge has a drop-dead delivery date of 4/17/00.

The minimum system requirements to run the software must adhere to the following specifications:

- 200 MHz CPU,
- 32 MB RAM
- 4 MB Video Card

Testing will only occur on systems with Windows 98, DirectX 7.0a (with the DirectX SDK), and Microsoft VC++. Systems with other software configurations will not be tested.

Test Plan

Software (SCIs) to be Tested

Interface Components to be Tested

- Level editor
- New World Wizard
- New Sprite Wizard
- Database Read/Write
- File exporter

Engine Components to be Tested

- Object Handler / Engine Core
- Data Loaders
- Draw Handler
- Sound Handler
- Input Handler
- Text Handler
- Logic Handler

Help Components to be Tested

- Help Files Search
- Tutorials

Installation Components to be Tested

- Software Install Process
- Software Uninstall Process

Testing Strategy

Unit Testing

All unit testing will be done in White Box fashion. Testing will be conducted using Basis Path testing methods, because of its simplicity and high effectiveness. Loop testing will also be conducted to compliment the Basis Path testing.

Individual engine components (Draw Handler, Sound Handler, etc.) will be tested separately. Due to the GameForge engine's modular design, there is no need for test beds. All components can be tested through the Object Handler. The following engine components will be unit tested:

- Object Handler / Engine Core
- Draw Handler
- Sound Handler

- Input Handler
- Text Handler
- Logic Handler
- Data Loaders

The interface will be unit tested in five parts:

- Level editor
- New World Wizard
- New Sprite Wizard
- Database Read/Write
- File exporter

Integration Testing

Because GameForge has a highly modular design, top-down and bottom-up integration will occur simultaneously. However, the system will be integrated incrementally, to control the amount of bugs that need to be fixed at any given time. The engine will be integrated in the following order: draw handling, input handling, sound handling, logic handling. Tests will be conducted in a black box fashion.

Validation Testing

Combined engine components will be tested as a whole. To maintain maximum control over the testing criteria, all data files will be made specifically for testing purposes. The level builder will be tested to ensure proper communication between the interface and the database. Testing will be done in a black box fashion.

GameForge will be closely examined to ensure conformity with the System Requirements Specification. Noted criteria being examined are: ease of use and Principle of Least Astonishment regarding the interface, as well as a reasonable performance level, understandable code, and sufficient AI routines for the engine.

High-Order Testing

The High-Order testing will be performed on the complete, integrated system. “In-house” beta testing will take place at this time and PA Software staff members will attempt to construct fully functional games using GameForge. The software will be stress tested and performance tested.

Parties outside PA Software will be asked to help with the final testing phase: public beta testing. Each beta tester will be given a copy of the software, and the preliminary help files (these will not be completed until immediately prior to GameForge's final build.) Beta testers will be expected to submit bug reports and any opinions they may have concerning the software (especially the interface layout.)

Testing Resources and Staffing

Resources

No special resources are required beyond those already needed for development.

Staffing

Test Team Leader – Jonathan Schmoll
Unit Testing Coordinator – Ken Nelson
Integration Testing Coordinator – Matthew Forster
System Testing Coordinator – Bill Lord
Beta Testing Coordinator – Jonathan Schmoll

Test Work Products

Frame Rate Counter

In order to monitor the number of frames the engine is capable of producing per second, an additional piece of software was developed. The frame rate counter is a valuable tool in determining system performance.

Test Record Keeping

Microsoft Excel will be used to evaluate immediate test results. After the results have been evaluated, they will be submitted to a Microsoft Access database for storage.

A test log is kept to monitor the tests that have been applied. An error, or 'bug' log is kept to monitor any problems that have arisen during testing. Also, a beta tester report form exists to aid beta testers in organizing their communication with PA Software. Examples of these forms can be found under the "Test Record Keeping and Test Log" heading later in the document.

Test Metrics

Function Point: this metric will be used when calculating statistics pertaining to the complete testing process.

Bang Metric: this metric will be used to provide an indication of the number of test cases required.

Cyclomatic Complexity: this metric will be used to target modules as candidates for extensive unit testing. Modules with high cyclomatic complexity are likely to be more error-prone.

Breadth of Testing: this metric provides an indication of testing completeness.

Depth of Testing: this metric provides a measure of the percentage of independent basis paths covered by the testing versus the total number of basis paths in the program.

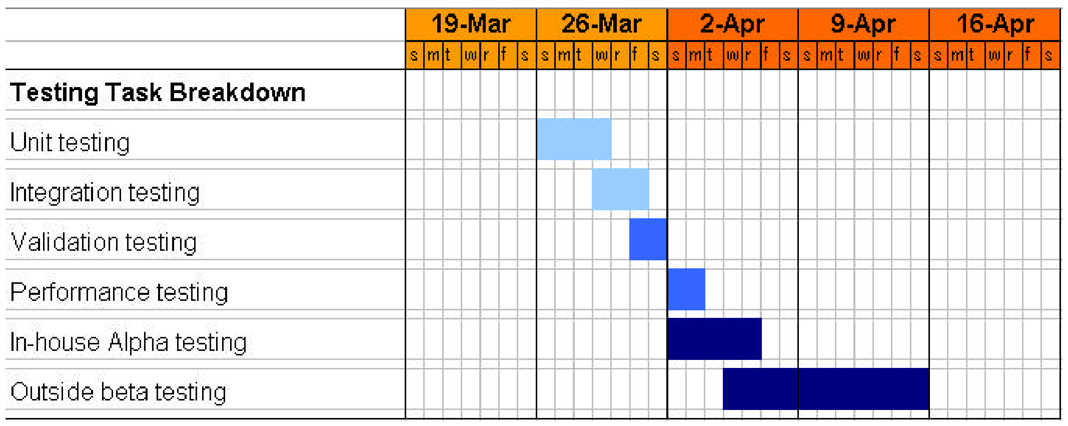
Fault Profiles: this metric is used to prioritize and categorize uncovered errors.

Frames per second: this metric is used to gauge the performance of the game engine.

Testing Tools and Environment

Microsoft Visual Basic and Visual C++ are used as testing tools as well as the testing environment. Test data files will be constructed for unit and integration testing. A Frame Rate Counter is also used in determining program performance. There are no other special tools or hardware.

Test Schedule



Test Procedure

Software (SCIs) to be Tested

Interface Components to be Tested

- Level editor
- New World Wizard
- New Sprite Wizard
- Database Read/Write
- File exporter

Engine Components to be Tested

- Object Handler / Engine Core
- Data Loaders
- Draw Handler
- Sound Handler
- Input Handler
- Text Handler
- Logic Handler

Help Components to be Tested

- Help Files Search
- Tutorials

Installation Components to be Tested

- Software Install Process
- Software Uninstall Process

Testing Procedure

Unit Test Cases (Interface)

Testing Procedure for Component: ***Interface Level Editor***

The level editor will be tested in a white box fashion. Pre-constructed sprites will be loaded into the level and all editor functions will be tested.

Stubs and/or Drivers for Component: *Interface Level Editor*

- None.

Test Cases for Component: *Interface Level Editor*

- Pre-constructed sprites will be loaded into the tree-view control and then placed into the level editor.
- Views will be rotated between sprites, backgrounds, and collision areas using the ‘view layer’ command.

- The property view of loaded sprites will be checked using the right-click menu.
- The mini-map will be used to quickly access various sections of a level.

Purpose of Tests for Component: *Interface Level Editor*

The purpose of these tests is to ensure correct operation of all controls in the interface, as well as verifying proper placement of backgrounds, sprites, and collision areas.

Expected Results for Component: *Interface Level Editor*

The interface is expected to perform within design specifications.

Testing Procedure for Component: ***New World Wizard***

The New World wizard will be tested in a white box fashion. The wizard will be stepped through and all functions will be tested.

Stubs and/or Drivers for Component: *New World Wizard*

- None.

Test Cases for Component: *New World Wizard*

- The wizard will be stepped through and all controls will be tested.

Purpose of Tests for Component: *New World Wizard*

The purpose of these tests is to ensure correct operation of all controls in the wizard.

Expected Results for Component: *New World Wizard*

The interface is expected to perform within design specifications.

Testing Procedure for Component: ***New Sprite Wizard***

The New Sprite wizard will be tested in a white box fashion. The wizard will be stepped through and all functions will be tested.

Stubs and/or Drivers for Component: *New Sprite Wizard*

- None.

Test Cases for Component: *New Sprite Wizard*

- Various data configurations will be entered into the attribute pages of the wizards to test for impossible combinations that are still allowed.
- Sprite animations will be created from test bitmaps.

Purpose of Tests for Component: *New Sprite Wizard*

The purpose of these tests is to ensure correct operation of all controls in the wizard, and the proper creation of sprite objects.

Expected Results for Component: *New Sprite Wizard*

The interface is expected to perform within design specifications.

Testing Procedure for Component: ***Database Read / Write***

The database will be tested in a white box fashion. A sample database will be loaded into the database.

Stubs and/or Drivers for Component: *Database Read / Write*

- None.

Test Cases for Component: *Database Read / Write*

- An attempt will be made to load the data into the tree view.
- Sprites placed in the level editor will be verified with the corresponding data in the database.

Purpose of Tests for Component: *Database Read / Write*

The purpose of these tests is to ensure correct read and write operations made by the database.

Expected Results for Component: *Database Read / Write*

The database is expected to perform within design specifications.

Testing Procedure for Component: ***File Exporter***

The file exporter will be tested in a white box fashion. A completed database will be exported to data files. Data format will be examined.

Stubs and/or Drivers for Component: *File Exporter*

- None.

Test Cases for Component: *File Exporter*

- A completed database will be exported to data files.
- A working engine prototype will be tested with these files.

Purpose of Tests for Component: *File Exporter*

The purpose of these tests is to ensure correct data output by the database.

Expected Results for Component: *File Exporter*

The database is expected to perform within design specifications.

Unit Test Cases (Engine)

Testing Procedure for Component: ***Object Handler / Engine Core***

The Object Handler will be tested in a white box fashion. It will be tested with various forms of data to determine data flow through the software architecture (data must be passed into the proper components.)

Stubs and/or Drivers for Component: *Object Handler / Engine Core*

- The required Windows code is used as a test bed.

Test Cases for Component: *Object Handler / Engine Core*

- Test data will be pushed through the Object Handler.

Purpose of Tests for Component: *Object Handler / Engine Core*

The purpose of these tests is to ensure correct data flow through the system.

Expected Results for Component: *Object Handler / Engine Core*

The component is expected to perform within design specifications.

Testing Procedure for Component: ***Data Loader***

The Data Loader will be tested in a white box fashion. It will be read in various test data files and pass them to the Object Handlerto determine data flow through the software architecture.

Stubs and/or Drivers for Component: *Data Loader*

- The Object Handler is used as a test bed.

Test Cases for Component: *Data Loader*

- Test data will be pushed through the Object Handler.

Purpose of Tests for Component: *Data Loader*

The purpose of these tests is to ensure correct data flow through the system.

Expected Results for Component: *Data Loader*

The component is expected to perform within design specifications.

Testing Procedure for Component: ***Draw Handler***

The Draw Handler will be tested in a white box fashion. It will receive data from the Object Handler, and will be examined while executing the received commands. Surfaces must be created and destroyed properly.

Stubs and/or Drivers for Component: *Draw Handler*

- The Object Handler is used as a test bed.
- The Data Loader is used to parse information from the data files into the Object Handler.

Test Cases for Component: *Draw Handler*

- Surfaces will be created and blitted to the screen using the Draw Handler.
- Surfaces will be moved and scaled.

Purpose of Tests for Component: *Draw Handler*

The purpose of these tests is to ensure proper drawing of sprites.

Expected Results for Component: *Draw Handler*

The component is expected to perform within design specifications.

Testing Procedure for Component: ***Sound Handler***

The Sound Handler will be tested in a white box fashion. It will receive data from the Object Handler, and will be examined while executing the received commands. Buffers must be created and destroyed properly.

Stubs and/or Drivers for Component: *Sound Handler*

- The Object Handler is used as a test bed.
- The Data Loader is used to parse information from the data files into the Object Handler.

Test Cases for Component: *Sound Handler*

- Sound Buffers will be created and played using the Sound Handler.
- Sounds will be stopped during play and looped for continuous play.

Purpose of Tests for Component: *Sound Handler*

The purpose of these tests is to ensure proper playing of sounds.

<u>Expected Results for Component: *Sound Handler*</u>

The component is expected to perform within design specifications.

<u>Testing Procedure for Component: ***Input Handler***</u>

The Input Handler will be tested in a white box fashion. It will receive data from the Object Handler, and will be examined while executing the received commands. Input devices must function properly

<u>Stubs and/or Drivers for Component: *Input Handler*</u>

- The Object Handler is used as a test bed.
- The Data Loader is used to parse information from the data files into the Object Handler.

<u>Test Cases for Component: *Input Handler*</u>

- Sprites will be controlled using the keyboard, mouse, and a joystick.

<u>Purpose of Tests for Component: *Input Handler*</u>

The purpose of these tests is to ensure proper player control.

<u>Expected Results for Component: *Input Handler*</u>

The component is expected to perform within design specifications.

<u>Testing Procedure for Component: ***Text Handler***</u>

The Text Handler will be tested in a white box fashion. It will receive data from the Object Handler, and will be examined while executing the received commands. Text must be displayed properly.

<u>Stubs and/or Drivers for Component: *Text Handler*</u>

- The Object Handler is used as a test bed.
- The Data Loader is used to parse information from the data files into the Object Handler.

<u>Test Cases for Component: *Text Handler*</u>

- Various text will be displayed to the screen.

<u>Purpose of Tests for Component: *Text Handler*</u>

The purpose of these tests is to ensure proper displaying of text.

Expected Results for Component: *Text Handler*

The component is expected to perform within design specifications.

Testing Procedure for Component: ***Logic Handler***

The Logic Handler will be tested in a white box fashion. It will receive data from the Object Handler, and will be examined while executing the received commands. All game logic must function properly.

Stubs and/or Drivers for Component: *Logic Handler*

- The Object Handler is used as a test bed.
- The Data Loader is used to parse information from the data files into the Object Handler.

Test Cases for Component: *Logic Handler*

- Enemies will be attributed with various AI routines.
- Various combinations of world attributes will be set (gravity, friction, etc.)

Purpose of Tests for Component: *Logic Handler*

The purpose of these tests is to ensure proper functionality of game logic.

Expected Results for Component: *Logic Handler*

The component is expected to perform within design specifications.

Unit Test Cases (Help / Tutorials)

Testing Procedure for Component: ***Help File Search***

The Help File Search will be tested in a white box fashion. The help file search engine will be tested for completeness and functionality.

Stubs and/or Drivers for Component: *Help File Search*

- None.

Test Cases for Component: *Help File Search*

- The help data will be search for various pieces of information.

Purpose of Tests for Component: *Help File Search*

The purpose of these tests is to ensure robust and functional searching of help information.

Expected Results for Component: *Help File Search*

The component is expected to perform within design specifications.

Testing Procedure for Component: ***Tutorials***

The Tutorials will be tested in a white box fashion. The tutorials will be tested for ease and completeness.

Stubs and/or Drivers for Component: *Tutorials*

- None.

Test Cases for Component: *Tutorials*

- The available tutorials will be followed step by step.

Purpose of Tests for Component: *Tutorials*

The purpose of these tests is to verify that the tutorials are complete, correct, and easy to use.

Expected Results for Component: *Tutorials*

The component is expected to perform within design specifications.

Unit Test Cases (Software Installation)

Testing Procedure for Component: ***Installation and Uninstall***

The software installation process will be tested in a black box fashion. The installation of all files and registry entries will be tested. All files must be removed upon uninstallation of the software.

Stubs and/or Drivers for Component: *Installation and Uninstall*

- None.

Test Cases for Component: *Installation and Uninstall*

- The software (incomplete) will be installed using Wise InstallMaster 8.0.
- The software (incomplete) will be uninstalled using Wise InstallMaster 8.0.

Purpose of Tests for Component: *Installation and Uninstall*

The purpose of these tests is ensure proper installation and removal of all necessary files.

Expected Results for Component: *Installation and Uninstall*

The installation package is expected to perform within design specifications.

Integration Testing

Testing Procedure for Integration

The system will be integrated incrementally, to control the amount of bugs that need to be fixed at any given time. The engine will be integrated in the following order: draw handling, input handling, sound handling, logic handling.

The system will be tested for errors in a black box fashion, after each component is integrated

Stubs and Drivers Required

- The Object Handler is used as a test bed.
- The Data Loader is used to parse information from the data files into the Object Handler.

Test Cases and Their Purpose

- Each component will be attached to the Object Handler in the order specified above.

Expected Results

The system is expected to integrate without major flaws.

Validation Testing

Testing Procedure for Validation

The features and functionality in the final system will be cross-referenced with the Software Requirements Specification document to verify that the software demonstrates conformity with the requirements.

Test Cases and Their Purpose

- Features corresponding to the design requirements will be evaluated.

Expected Results

The software will perform within the specifications of the Software Requirements document.

Pass/Fail Criterion for All Validation Tests

Features corresponding to the design document requirements do not need to coincide verbatim with the requirements. Instead, it is important that they retain the spirit of the requirements.

High-Order Testing (System Testing)

Recovery Testing

No Recovery testing will occur. While system failures are undesirable, termination of the program in the event of a crash is acceptable.

Security Testing

No Security testing will occur. There are no security issues with GameForge.

Stress Testing

The world builder and the engine will be loaded with abnormally high sprite counts (with attributes and sounds) to determine how much GameForge can handle.

Performance Testing

The engine will be loaded with an increasing number of sprites while the frame rate is monitored using the Frame Rate Counter.

Alpha/Beta Testing

Alpha testing will occur in-house. Members of the test team will attempt to construct a complete, working game using GameForge. Games will be constructed with and without the aid of the tutorials.

Beta testing will be semi-public. Select individuals outside of PA Software will be asked to participate in beta testing GameForge. Individuals will be selected based on their lack of programming knowledge, and their experience with image processing. Testers will be expected to submit bug reports, as well as their opinions concerning performance and interface layout.

Pass/Fail Criterion for All Validation Tests

Features corresponding to the design document requirements do not need to coincide verbatim with the requirements. Instead, it is important that they retain the spirit of the requirements.

Testing Resources and Staffing

Testing Resources

No special resources are required for testing beyond those already needed for development.

Test Staff

Test Team Leader – Jonathan Schmoll
Unit Testing Coordinator – Ken Nelson
Integration Testing Coordinator – Matthew Forster
System Testing Coordinator – Bill Lord
Beta Testing Coordinator – Jonathan Schmoll

Test Work Products

Frame Rate Counter

In order to monitor the number of frames the engine is capable of producing per second, an additional piece of software was developed. The frame rate counter is a valuable tool in determining system performance.

Test Record Keeping and Test Log

Test Record Keeping

Microsoft Excel will be used to evaluate immediate test results. After the results have been evaluated, they will be submitted to a Microsoft Access database for storage.

A test log is kept to monitor the tests that have been applied. An error, or 'bug' log is kept to monitor any problems that have arisen during testing. Also, a beta tester report form exists to aid beta testers in organizing their communication with PA Software.

The report format for beta testers is below:

Beta Tester Report Form Name ________________________________	
Defect Report	**Enhancement Report**
Was the defect found in the interface or the engine?	Will the proposed enhancement occur in the interface or the engine?
Specifically, where in the interface or engine did the defect occur?	Specifically, what module of the interface or engine would you like to see the enhancement made?
Is the defect reproducible? If so how?	Is the enhancement a cosmetic or a functional enhancement? Give us the details of your enhancement:
Does the defect crash the system?	
Additional Comments:	

Test Log

The following is an example of the test log format:

TEST LOG (Unit Testing)					
Date	**Test Type**	**System**	**Module**	**Coordinator**	**Results**
2-11	Unit Testing	World Builder	Mini-map	Matt	Mini-map working to specification.
2-17	Unit Testing	World Builder	Player Death	Ken	Player death working but player bullets kill player.
2-19	Unit Testing	World Builder	TreeView Control	Matt	TreeView control loads database information correctly, but needs further error checking.
3-2	Unit Testing	World Builder	TreeView Control/Level Editor	Matt	Sprites dragged and dropped to the level editor are not visible, though accurately stored in database.
3-2	Unit Testing	World Builder	Layer Selection Box	Ken/Matt	Correct layer is identified, but not all sprites from each layer are displayed properly.
3-2	Unit Testing	World Builder	New World Wizard	Matt	Information is correctly placed in database, controls such as text boxes, combo boxes needs better error checking against invalid user input.
3-2	Unit Testing	World Builder	New Sprite Wizard	Ken/Matt	Sprite’s attributes are correctly input into the database, but need to be checked for invalid combinations.
3-5	Unit Testing	World Builder	New Sprite Wizard	Bill	Bitmaps are loaded properly, but animation frames cannot be deleted once added.
3-5	Unit Testing	World Builder	Sprite Animation Form	Jon	Individual frames jump during animation cycle in animation box.
3-7	Unit Testing	World Builder	File Exporter	Jon	Data in database is converted into data files correctly.
3-14	Unit Testing	World Builder	New World Wizard	Bill	Transparent color is working properly.
3-14	Unit Testing	World Builder	New Level Wizard	Matt	Need error checking for invalid input.
3-18	Unit Testing	World Builder	New Level Wizard	Matt	Data is entered properly into database.
3-30	Unit Testing	World Builder	New Sprite Wizard	Matt	Sounds are attached to sprites properly.
2-19	Unit Testing	Engine	Object Handler	Ken	Data pushes through object handler properly. Code needs to be optimized for efficiency.
2-19	Unit Testing	Engine	Data Loader	Ken	Engine reads data files from World Builder properly. Need END attribute for each sprite.
2-19	Unit Testing	Engine	Draw Handler	Ken	Draw surfaces are created properly. Unacceptable memory usage. Surfaces must be released thoroughly.
2-21	Unit Testing	Engine	Draw Handler	Ken	Draw surfaces swap animations properly.
2-23	Unit Testing	Engine	Sound Handler	Jon	Sounds of different formats are not playing properly. Increase functionality

					to include multiple formats simultaneously.
2-25	Unit Testing	Engine	Input Handler	Ken	Joystick support not working properly with current engine architecture. Fix this.
3-1	Unit Testing	Engine	Input Handler	Ken	Keyboard working properly. Escape key kills surfaces properly.
3-3	Unit Testing	Engine	Logic Handler	Ken	Collision detection not working properly for large sprites.
3-3	Unit Testing	Engine	Logic Handler	Ken	Gravity is not working correctly, but cannot be used with jumping. Jumping algorithm needs improving.
3-3	Unit Testing	Engine	Logic Handler	Ken	Bouncing causes player to lose control.
3-3	Unit Testing	Engine	Logic Handler	Ken	Shooting animations are incorrect.
3-4	Unit Testing	Engine	Logic Handler	Ken	Large moving sprites pass through smaller sprites.
3-4	Unit Testing	Engine	Logic Handler	Ken	Any sprite death kills player, regardless of interaction with player.
3-5	Unit Testing	Engine	Logic Handler	Ken	Player Health is not updating correctly.
3-6	Unit Testing	Engine	Text Handler	Ken	Score is not displayed properly.
3-6	Unit Testing	Engine	Logic Handler	Ken	Bullets do not effect sprites at all. Bullets stop in mid air.
3-7	Unit Testing	Engine	Logic Handler	Ken	Jumping shoots player off top of screen.
3-7	Unit Testing	Engine	Logic Handler	Ken	Player can float when the jump button is held down.
3-7	Unit Testing	Engine	Draw Handler	Ken	Color blit not displaying correct RGB value.
3-10	Unit Testing	Engine	Sound Handler	Ken	Get status not returning correct value.
3-23	Unit Testing	Engine	Data Loader	Ken	Constant Crashing fixed.