(Written in 2012, very old, contribute if you feel like it)

# Introduction

## Abstract

The FBG has evolved out of the requirement of an intellectual mind and has been made with the intention of allowing users to have a fun time trying to set a high score whilst the programmer has a difficult time enforcing constraints in the background. This product does have similarities to other game playing programs, but is a self-contained product of its own.

The game modules are all self-written functions each performing one different task and no prior API’s or physics libraries are used. The user will not see the background classes or history unless they decide to develop their own game and implement it into the system. The module creator who designed the system will be the only one with knowledge of the user class details and characteristics.

## Purpose

The main purpose of this game is to provide users with a visual experience and to de-culminate stress by avoiding a mass array of colorful balls appearing from different parts of the screen.

This is the windows adaptation of the game written from scratch using raw physics and no prior physics libraries of engines. Hence, it serves as an immense learning experience in game coding and debugging. Also the source code will be published by me online for any other developers looking to understand the working of Falling Balls.

This sharing of different ideas encourages community and promotes creativity in others to make new games.

## Literature Survey

### Current System

* Allows a user to view previous High Scores.
* Allows user to select level, once unlocked.
* Allow users to register with a nickname.
* Keep track of a user’s history with their scores.
* Save High Scores
* End game
* Android OS

### Proposed System

* Ported to Windows OS
* Visualization of ball movement
* Multiple levels periodically appear.
* Multi-colored balls
* Save and View High Scores
* New maps
* DLC’s

## Definitions, Acronyms & Abbreviations

FBG: Falling Balls Game

IEEE: Institute of Electrical and Electronics Engineers

SRS: Software Requirements Specification

SDS: Software Design Specification

DLC: Downloadable Content

GUI: Graphical User Interface

N/A: Not Applicable

# Planning & Scheduling

## SDLC model

The incremental model is chosen for the development of this project as certain tasks will be completed within a specified timeframe and add additional changes such as more levels or features if required.

The incremental model justifies all the requirements of this project and is best suited over the other models mainly because many changes will be rolled out in increments as time progresses.

As this is a game, the incremental model best justifies the cause as there will always be a demand for more levels or newer maps or features and all these can be rolled out one after the other as DLC’s, i.e., in increments.

# Software Requirements Specification

## Functional Requirements

### Stakeholders

Anyone interested in my code writing ability and physics engine development are free to invest in my project. Also, the code for the FBG will be made free and open to the entire coding community for anyone who wishes to implement their own ideas and develop the game further in any direction of their own free will.

There are no licenses or patents on the FBG Windows game and I do not intend on doing so, hence no prior permission or credit is required for anyone who wishes to use this game for further development.

### Modules

The game is divided into 3 modules from a programmer’s point of view:

1. **Interface**:

This segment of the code basically deals with the opening screen, closing screen, LEVEL UP! dialogues and saving and displaying the scores. Every splash screen, figure and image was created on GIMP and is a part of the interface that the user can see and use.

2. **Ball Simulator**:

Several functions bundled up into one could be used as a Ball simulating module. This exhibits the modularity and polymorphic ability of my code. There are functions to perform every specific task. The ball movement is simulated using the physics formula such as time of flight = u\*sin (theta) \* g where g is gravity and assumed to be 10.0, sx = u\*cos (theta)\*t, etc.  
The ball is constantly redrawn by calculating its next position coordinate using the above formula and the left vertical of the screen is taken as the +Y and –Y axes, i.e, the ball begins motion on the –X axis to make it appear as though it is falling from the top of the Y axis.

3. **Player Simulator**:

One simple function takes care of this motion by redrawing the player.png file as the user uses the right and left arrow keys to make the figure move. Boundary conditions are checked for a 640x480 screen.

### User stories/requirements

**REQ-1:** Every game must have a splash screen or some sort of intro before playing the actual game, so that the user knows what to press. (I.e. a graphic showing what keys.)

**REQ-2:** Developer can choose to define a set of keys to be used for each game, a standard (like Nintendo, has only an A & B button)

**REQ-3:** Game will not use any Physics engines or development kits such as UDX. Everything will be done from scratch.

**REQ-4:** Game must be developed only in C++.

**REQ-5:** A standard screen resolution for the game. Non-adjustable.

**REQ-6:** A HELP document built into the game, so that the user can pause the game, and view help, if they don’t know what’s happening/or how to play.

**REQ-7:** A possible list of soundtracks for different scenarios in the game.

**REQ-8:** There must be a way to turn music [on/off]

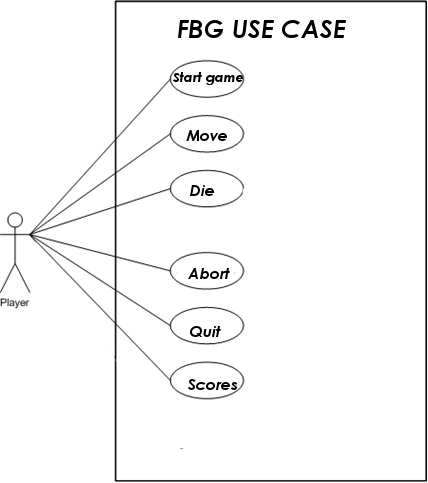
**REQ-9**: All graphics to be used and imported from parent folder.

**REQ-10**: Additional levels to be added by me or by third party users as and when they are created

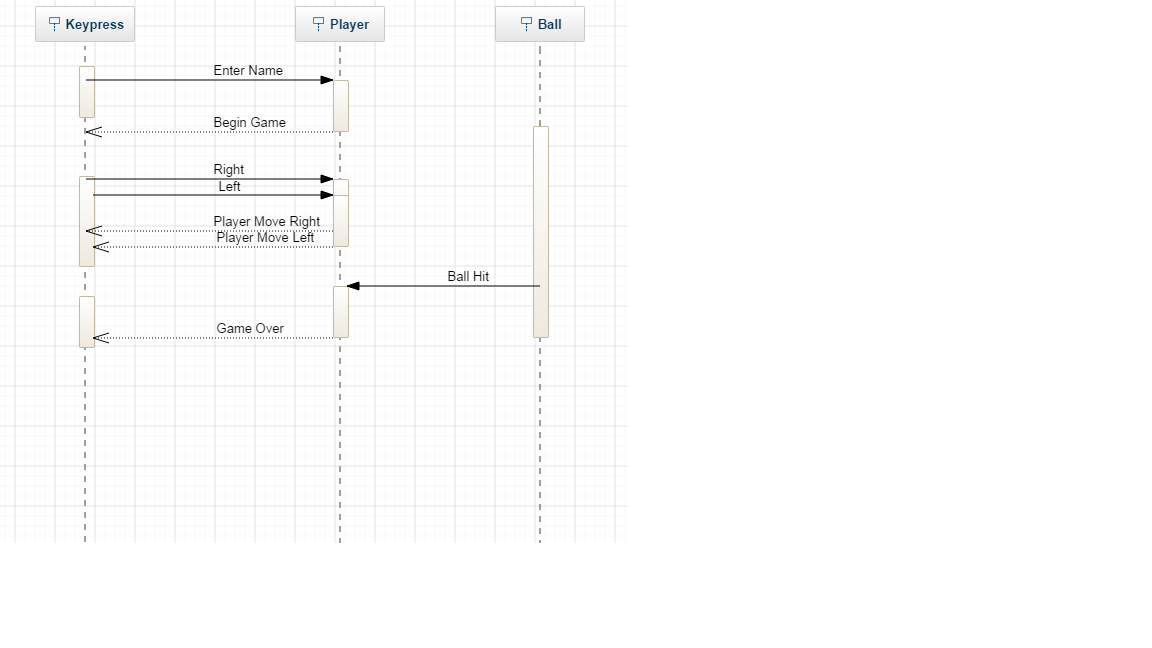
**REQ-11**: Existing levels are purely based on number of balls and player movement speed.

### Requirement Analysis

#### Use Case Diagram



#### Sequence Diagram



#### Use Case Description

* **Start Game** – The user presses ENTER or space bar on the intro screen to dismiss the introduction and start playing the game by entering their name.
* **Move** – The player moves using the arrow keys allowing for bi-directional movement on screen. To move, the following keys are used.
  + Left: Left arrow
  + Right: Right arrow
* **Die** – The player has the option of suicide by standing still on the screen without moving. A ball will eventually hit him
* **Scores** - To view high scores, the player must end the game.
* **Quit Game** – To stop playing, the player can quit by pressing the ESC key.
* **Abort Game** – If however, the game hangs up (never good and must be avoided), the player can abort the game by selecting CTRL-ALT-DEL keys simultaneously. In addition, in the event of certain errors, the player can end the game by following the on-screen instructions. Again, game errors must be avoided.

## System Requirements

Minimum Hardware Requirements:

* 512MB RAM or higher
* Intel Dual Core or higher
* 512MB GPU recommended
* 14” VGA Monitor

Minimum Software Requirements:

* Windows XP ( 32-bit)
* Dev C++ 4.9.9.2
* Allegro 4.2.1

## Non-Functional Requirements

## Performance Requirements

The FBG can withstand large amounts of usage at one time and will not crash or freeze at a moment of high activity and implementation. In addition, the system can accept large amounts of information at once.

Additionally, the number of modules accepted is high enough that memory levels on the do not reach their max where no more modules would be accepted. However, a possible solution may be to place a limit on the number of modules the system can allow, but after a certain time period of the module existing on the system, it is immediately removed/archived so that another new module can be updated. This may be decided after the system has begun use and the development is re-evolved to future system requirements.

Other performance requirements will be added as the product is in use and the issues surrounding performance will be addressed and new obstacles discovered.

## Safety Requirements

In order to ensure the safety of the FBG, I ensure that accidents will not occur, and that if by chance one does, that the consequences of it are minimal. I attempt to prevent hazards by designing for the worst-case scenarios and preventing their occurrence, no matter how farfetched the assumption. In addition, in the case that not all hazards are predicted, the system includes protection features that will minimize fallout from various incidents. For instance, virus protection is utilized to confirm that all modules are scanned for viruses, Trojans, or other harmful attachments, which may infiltrate the game playing system.

These safety requirements will have been implemented in the final stages of design, prior to the testing stage so as to prevent any severe crashes or accidents during the testing stages.

## Security Requirements

The database contains all of the user names, scores and other information that must be protected from hackers who would try to infiltrate the system and steal any personal or user information and try to login under a stolen name. In addition, the modules that users load onto the site and the site’s original games are all protected from outside hackers who may want to negatively alter the code that’s present for current games on the site without logging in or registering. Moreover, the modules that are loaded into the system are scanned for viruses, Trojans, or other attachments that can weaken the security of the system.

A developer familiar with firewalls may implement one to ensure that data privacy is guaranteed for all parties involved and the FBG is invulnerable to middleman attacks.

The security requirements of the FBG system will be implemented in the actual design stages of the system during the integration of components and front end designing. This will guarantee that the security components are integrated correctly into the system and that there will be less vulnerability in the testing stages of the production process.

## Software Quality Attributes

I will be confirming that all the IEEE standards and Rules of the World Wide Web Consortium will all be taken into consideration and followed to the best of the my ability in all aspects of project design, planning, and implementation.

I have made sure that there is no infringement on copyright laws and licensing. This is a purely fictional and original game made out of creativity, individually.

Besides licensing, I have ensured that all of the requirements of the system (such as the help guide) uphold all IEEE standards and will prevent any lawsuits or injustices on the part of the user.

## Assumptions & Constraints

There will be a brief set of instructions for each game, a user manual for the games in general, an email address to contact with any problems and questions, and a user manual for the code. With all of this information available to the user, there should be no difficulty in using the game playing system or troubleshooting the player if any problems arise.

The game modules themselves may be dependent upon shared software sources from the Internet, but it is dependent upon the decision of each user as to whether they will develop their own games entirely on their own.

The game will be tested and run on a 32-bit Operating System on Dev C++ 4.9.1 and Allegro 4.2.1 ONLY.

# Software Design Specification

## Architecture Design

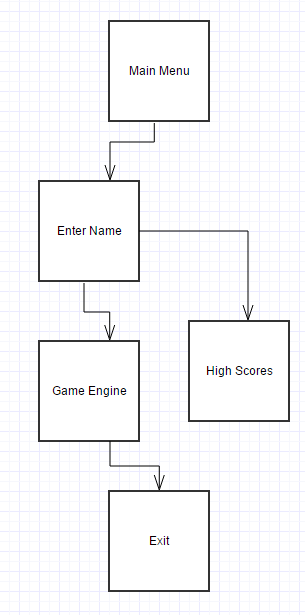


Fig: Architecture Diagram

## System Models

### Context Model

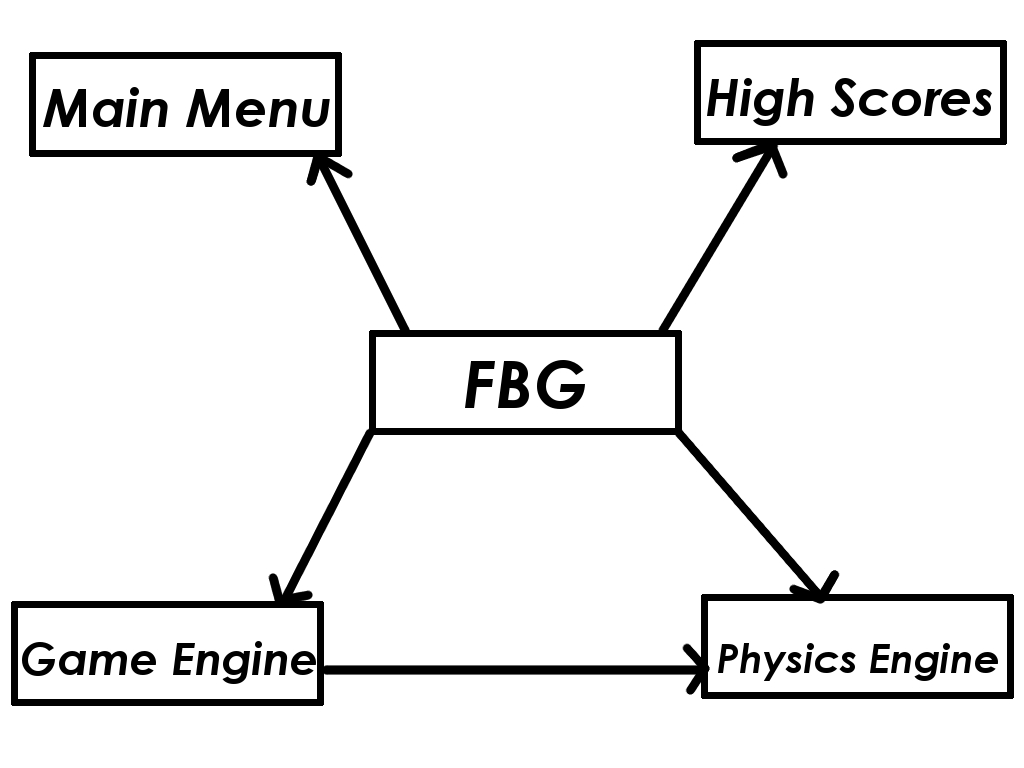
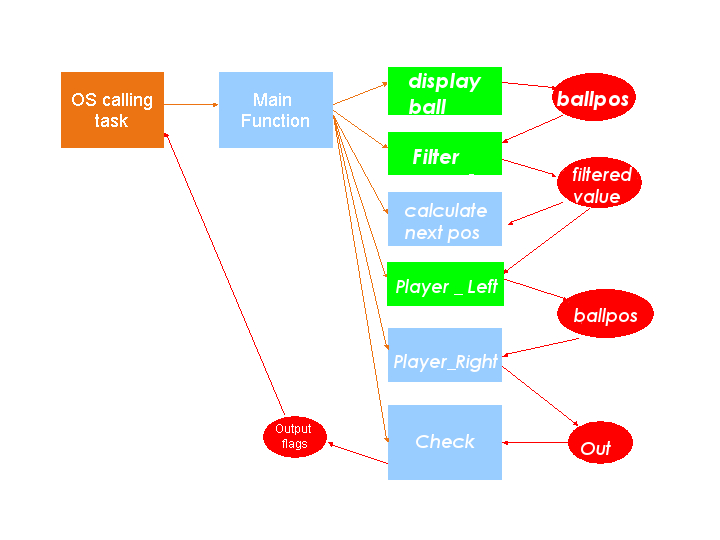


Fig: Context Model Diagram

### Behavioral/Process Model

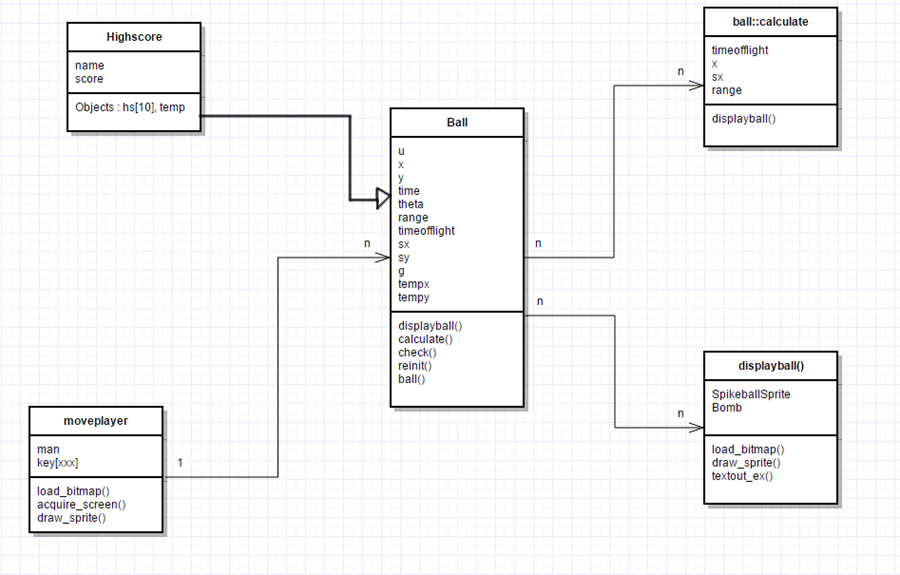
Fig: Data Flow Diagram



**Process Model Chosen**: Incremental Model as described earlier.

### Object Model

#### Class Diagram



## User Interface Design



Fig: Splash Screen

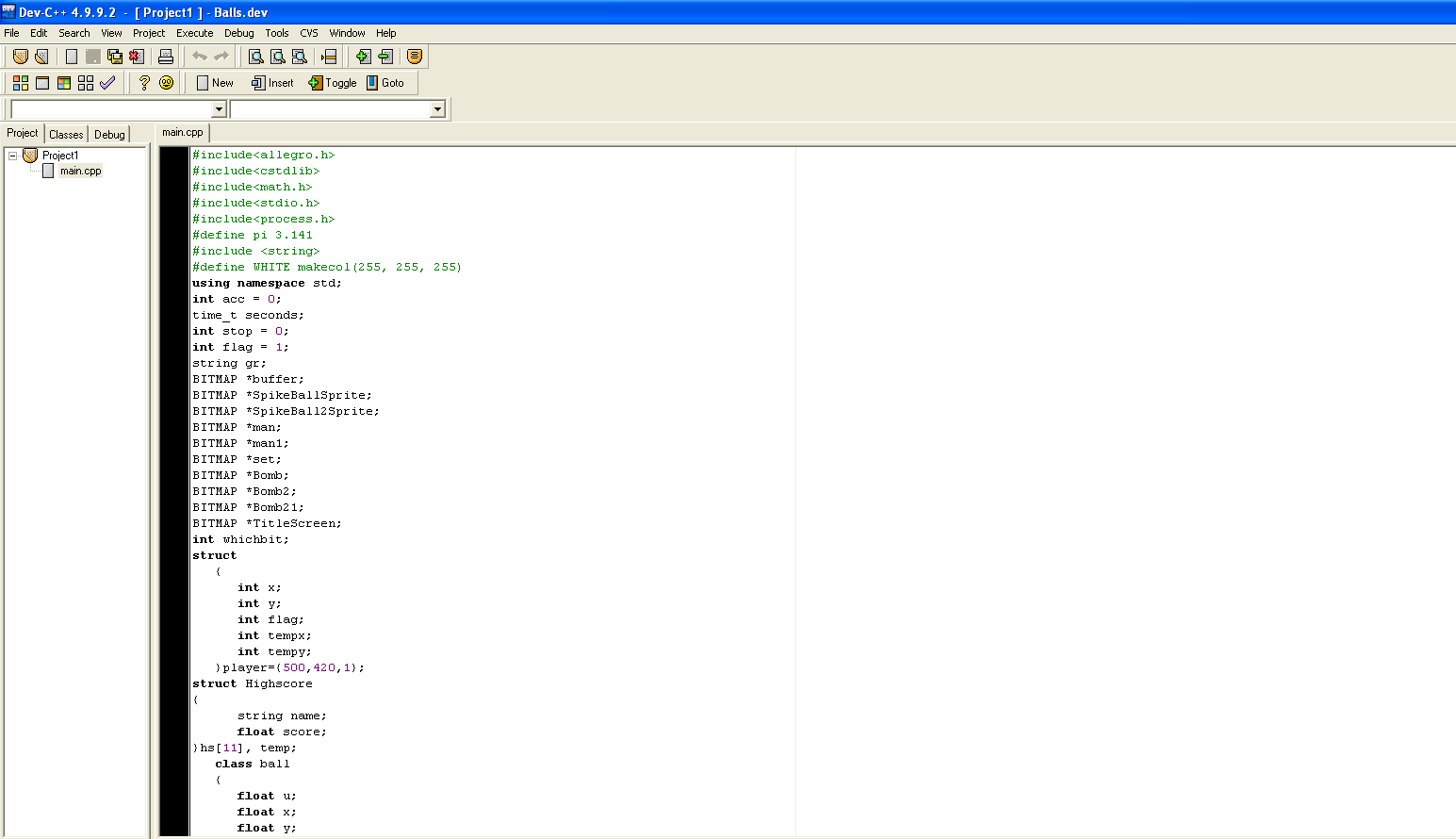
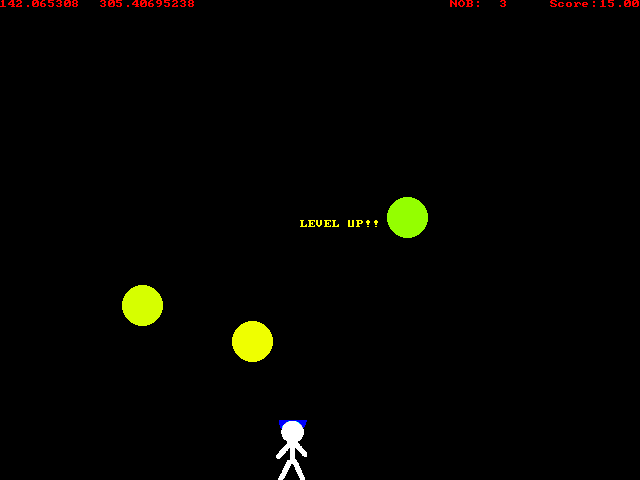
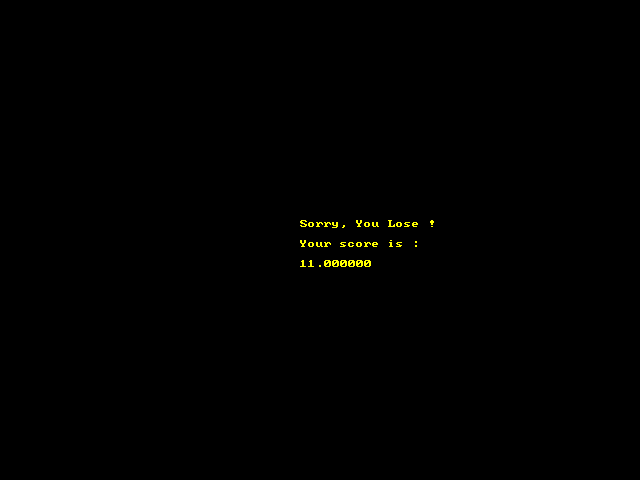


Fig: Allegro Library Usage





## Alternate Designs

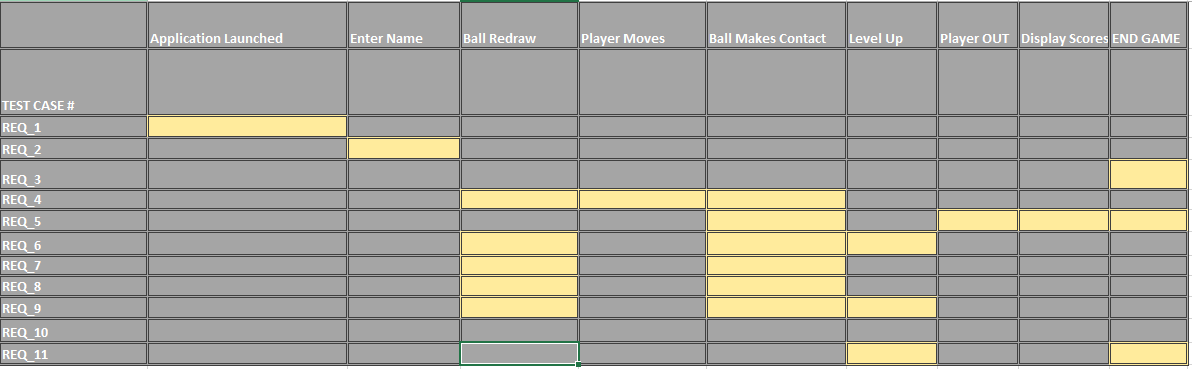
There were no other available models that could be chosen as this game was written intuitively and purely from scratch using genuine splash screens and physics logic.  
  
However, a Physics Engine may have been a possible option but the physics of this game was created purely using standard formulae just to test the creators ability.

# Testing

## Test Cases



## Traceability Matrix



## Test Execution Report

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |
|  |  |  |  |  |  |  |  |
| **Iteration/ Cycle** | 4 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| *EXECUTED* | PASSED | 10 |  |  |  |  |  |
| FAILED | 1 |  |  |  |  |  |
| *(Total) TESTS EXECUTED (PASSED + FAILED)* | | **11** |  |  |  |  |
| PENDING | | | 0 |  |  |  |  |
| IN PROGRESS | | | 0 |  |  |  |  |
| BLOCKED | | | 0 |  |  |  |  |
| *(Sub-Total) TEST PLANNED* | | | **11** |  |  |  |  |
| *(PENDING + IN PROGRESS + BLOCKED + TEST EXECUTED)* | | |  |  |  |  |
| DEFERRED | | | 0 |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Feature/Components** | **# Tests Planned** | **% Tests Executed** | **% Tests Passed** | **# Deferred/ blocked** | **Status** | **Remarks** |  |
| *New Levels and Maps* | *~5* | *40* | *40* | *2* |  | *0* |  |

## Testing Tools

No specific tools were used for testing.

The game was tested through public opinion and hands-on usage to identify and fix bugs reported by other users.

# Configuration Management

V1.0 – Contained Basic Intro Screen and High Score file handling Mechanism

V1.1 – Integrated Player Movement

V1.2 – Integrated Ball Movement

V1.3 – Integrated color changing balls and basic level up

V1.4 – New maps and levels. Still in Testing!

# Bibliography

<https://play.google.com/store/apps/details?id=com.reverie.game.fallingball&hl=en>

<http://devpaks.org/details.php?devpak=1>

<http://dev-c.soft32.com/>

<http://www.cppgameprogramming.com/cgi/nav.cgi?page=allegbasics>

http://formulas.tutorvista.com/physics/projectile-motion-formula.html