**Time Traveler Math**

**Educational Game for Elementary School Students**

**Hendrik Choy, Richard Hernandez, Nathan Nguyen**

**Final Draft v1.0**

**April 24th, 2015**

**CEN 4935 Senior Software Engineering Project**

**Instructor: Dr. Janusz Zalewski**

**Software Engineering Program**

**Florida Gulf Coast University**

**Ft. Myers, FL 33965**

**Glossary**

Android: An operating system used in computers, phones and tablets.

Camera: Provides the device a view of the game world

Protagonist: Also named the “Time Traveler”. The player-controlled main character of this game.

Sprites: An image integrated into a larger scene.

**1. Introduction**

The objective of this project is to develop an educational software math game, called Time Traveler Math or TTMath for short, on the Android platform for young students starting at Grade 3. The game can potentially scale up towards high school students with critical thinking problems through puzzles. The game engine that is used for the game is called Libgdx [2]. Eclipse is used as the environmental development for this project with the Android platform version 4.0.3 or higher as the target device for the release. Tiled Map Editor is the software used for level map creation.

The software has two modes, Maze Mode and Mental Math mode. In the game mode Maze Mode, the player explores a maze, which represents a specific time era in the background of the game, with puzzles from start to finish. The map will have obstacles that can impede a player’s progress, and items that can help a player. The player must explore the maze and search for keys which unlock a certain passage in the maze. The final key in the dungeon unlocks the final door to the end of the maze. When the player reaches the final door, a math question appears and the player must solve it in order to finish the level. The second game mode, Mental Math, is a mode that lets players solve math problems in a time limited situation. There are several “Paths” a player can choose in both game modes. These “Paths” represents math difficulty, and the higher the progression in the levels, the harder the math problems become. Both modes will have a score function where players can compare their scores to themselves or other players. The score function will exist as an incentive for players to improve their score and compete with other players.

**2. Software Requirements Specification**

**2.1 General Project Description**

The project currently uses an Android emulator and Android phone versions 4.0.3 or higher as the environmental system used to test TTMath. Figure 2.1.1 depicts a user using the software on an Android device where data can be sent to the web server. The user would use any Android devices to start the game, and a class called LevelManager would load the map level. Beating the level will show the user his/her score which is loaded to the local database stored on the phone and can optionally be loaded to the web server to compare different high scores. Figure 2.1.2 is a context diagram showing a simple context of how the system works in general. Figure 2.1.3 is a generic view of the screen that is usually displayed when the user is in Maze Mode.

The app will have two game modes that teach the same math concepts. The math concepts taught will start at 3rd grade difficulty and scale up from there. Both game modes will start with teaching addition and subtraction. The difficulty of the addition and subtraction will slowly rise as the user progresses through the game. Once the user has mastered this concept they will move on to multiplication and division. Each concept will be taught separately to prevent the game from being too difficult. As the difficulty rises multiple math concepts will be introduced at the same time to increase the user’s math skills.

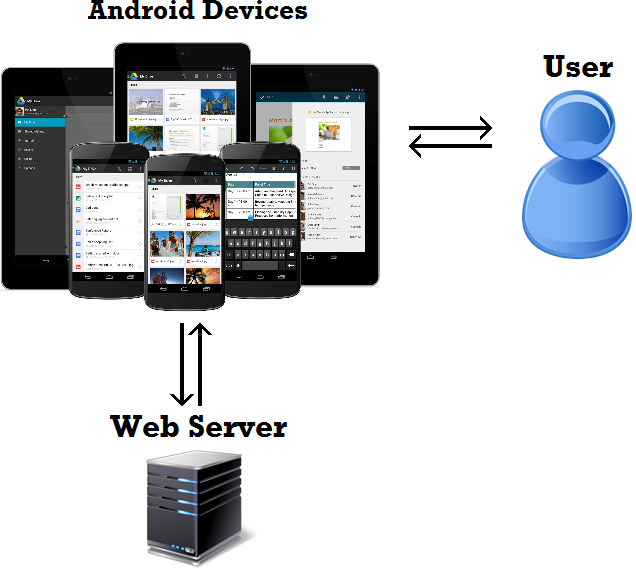


Figure 2.1.1 Physical Diagram

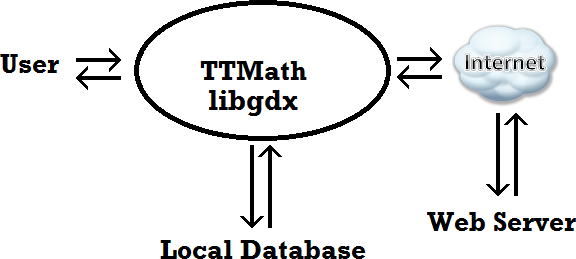


Figure 2.1.2 Context Diagram

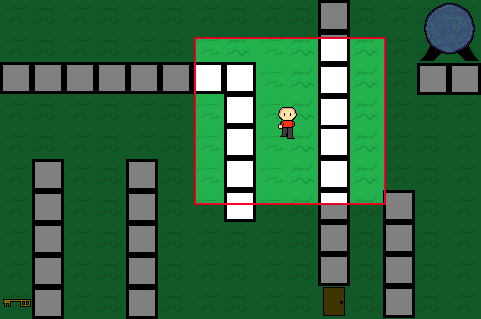


Figure 2.1.3 Generic view of the Screen

**2.2 Functional requirements**

The games will be operated through the screen of an Android device. The generic view of the screen is shown in Figure 2.1.3. The viewable area of the game is denoted by the red box.

The user operates the game by moving a protagonist on the screen. The screen view of the current game area is called a camera.

**2.2.1 Protagonist movement**

2.2.1.1 The Protagonist’s movement shall be controlled by touch motions from the user

2.2.1.2 The Protagonist shall move in only one of four directions based on touch input: up, down, left, or right.

2.2.1.3 The Protagonist shall have the ability to detect collisions from objects in the game.

2.2.1.4 The Protagonist shall have the ability to detect item objects in the game and pick the up.

**2.2.2 Camera Screen**

2.2.2.1 The camera screen shall be centered with a top-down orthogonal view on the protagonist..

2.2.2.2 The camera screen shall direct the player on where to look when either the maze game, math question screen, or main menu is up.

**2.2.3 Educational**

2.2.3.1 A math question shall be presented to the player when opening a door.

2.2.3.2 If the player gets the math question wrong, the key used shall disappear and another key shall be generated in the map, unless it is the special key.

2.2.3.3 Mental Math mode shall be able to produce questions infinitely as long as the player does not choose any wrong answers three times.

**2.2.4 Game Interface**

2.2.4.1 The game interface shall use touch motion to detect user input.

**2.3 Design Constraints**

The software shall run on an Android device (phone or tablet) with a minimum version of 4.0 or higher.

**2.4 Other Concerns**

The game shall have an 8 bit look.

The current plan is to accomplish the following required goals by the end of April:

1. Map Levels - 70%
2. Entity of the player, doors, and keys - 100%
3. Player movement - 100%
4. Problem Screen - 100%
5. GUI interface - 90%

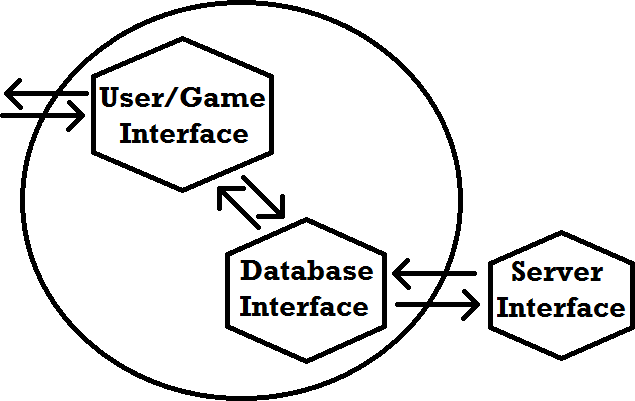
**3. Design Description**

**3.1 Design Details**

This project is comprised of one piece of software that runs on an Android device. The software controls what education game mode is displayed to the user. When a user picks a game mode the game renders the selected mode on the Android device. The user can pause or end the game mode if they wish at which point it will bring them back to the main menu screen. The user can choose to end the game which will close the application and bring them back to android menu.

**3.2 Software Architecture**

The software architecture of the game system is shown in figure 3.2.1. The user starts at the MainGame, which loads the ScreenManager. The screen manager always loads the MenuScreen first, and then usually the GameScreen is loaded. The game screen then loads the level manager which loads the assets into the game screen. The camera is reused by the screen manager for any screens used. The software architecture diagram in figure 3.2.1 is subject to changes in the future.



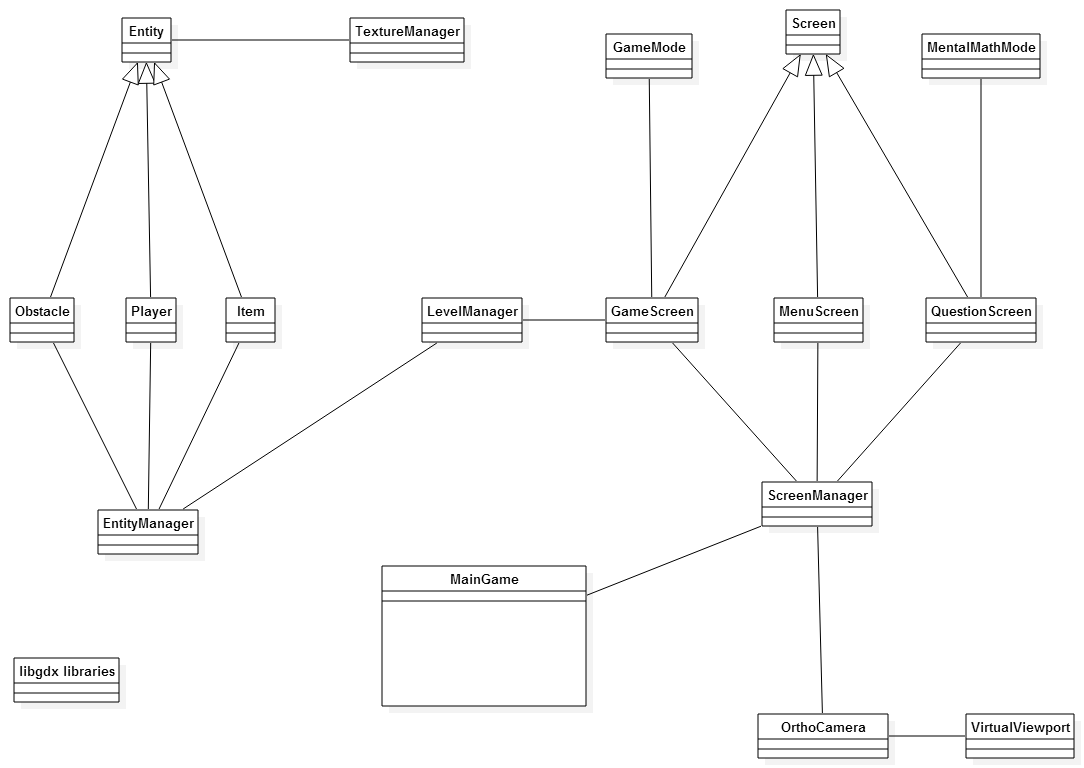


Figure 3.2.1 Class Architecture of the system

**3.3 Dynamic Perspective of the Game System**

The libgdx framework includes these lifecycle events: create, resize, render, pause, resume, and dispose. This application lifecycle is not specific to Android, although it applies as well. These methods are all event-driven and there is no explicit main game loop unlike most traditional game libraries. Any time the program changes screens (eg: when going from Menu Screen to the Game Screen), the new screen is created and the old screen is disposed. Resize ensures the width and height of the screen is correct. Render draws all the game objects onto the screen. Pause and resume allow the user to push the home button on the Android device and still keep their application in the background.

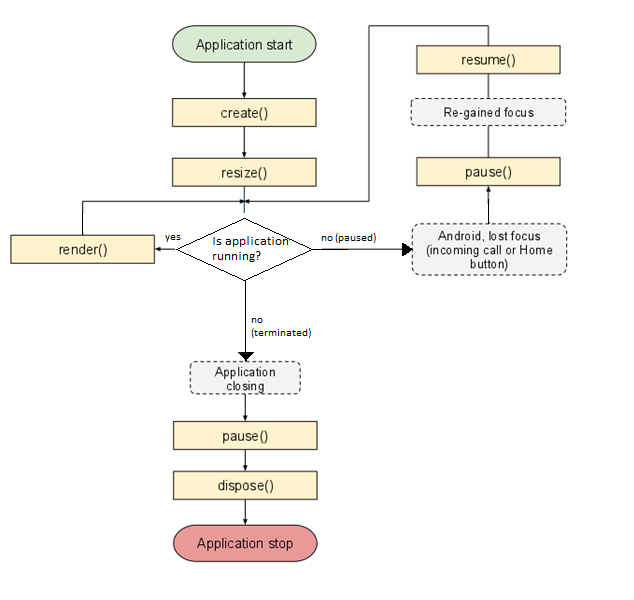


Figure 3.3.1 Application flowchart of TTMath [1]

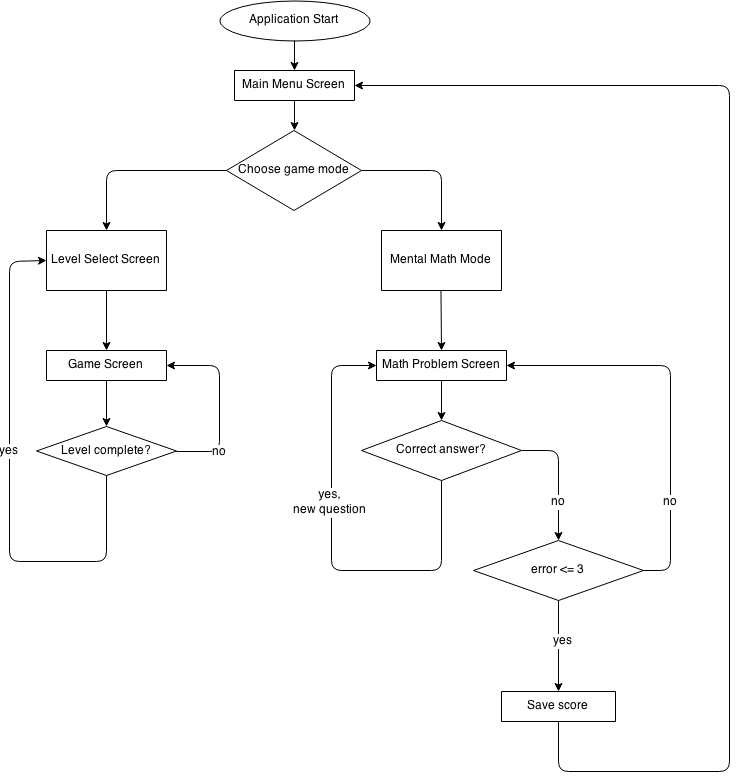


Figure 3.3.2 Dynamic perspective for TTMath

**4. Implementation**

**4.1.1 Arithmetic Operations**

The TTMath Education Android app is based on touch implementation and touch events for smart phones that contain the Android OS. The user can choose two different modes to play in and six different paths that represents a combination of the basic arithmetics used in this game (addition, subtraction, multiplication, and division). The arithmetic used in each question is based on the path taken and is generated by the code below.

private int operationGenerator(int path){

Random rand = new Random();

int difficulty = rand.nextInt(1 + path);

int[] operations = getPathsArray(path);

do{

if(operations[difficulty] != -1){

break;

}

difficulty = rand.nextInt(1 + path);

}while(operations[difficulty] == -1);

return operations[difficulty];

}

Once the arithmetic for the question has been chosen a question is created using these operations and random numbers. The question generated is multiple choice where there is only one correct answer. The code below is used to generate the answers used in the multiple choice question. There are a total of four answers the user can choose which are to be generated randomly except for the correct answer which will replace one of the randomly generated answers.

private int[] randomizeAnswer(int answer){

Random rand = new Random();

int answers[] = new int[5];

int max = answer+6;

int min = answer-6;

int realA = rand.nextInt(4+1);

for(int i=0;i<4;i++){

answers[i] = rand.nextInt(max-min+1)+min;

}

answers[realA] = answer;

answers[4] = realA;

return answers;

}

**4.1.2 Protagonist Movement and Collision**

Movement is handled by an interface from Libgdx called “InputProcessor”. There are actually three events that are handled by the engine: touchDown is when the system detects when the screen is touched, touchDragged is when the system detects the finger or mouse dragging itself across the screen, and touchUp is when the system detects the finger or mouse being released from the screen. The code below is an example of processing movement in the touchDragged event.

@Override

public boolean touchDragged(int screenX, int screenY, int pointer) {

isMoving = true;

//moving down

if(Gdx.input.getDeltaY()>10 && !collidesBottom()){

setTextureRegion(animations[0][0]);

movingDown = true;

setDirection(0,-170);

}

return false;

}

Collision is handled differently. Detecting collision and rerunning detection around the area of player highly depends on the GPU and CPU power of the system. Below is an example of a collision detection that consistently runs as the player moves.

public boolean collidesBottom() {

if(this.pos.y<5){

return true;

}

//checks each step beyond the player

for(float step = 0; step < playerWidth; step += collisionLayer.getTileWidth() / 2){

if(isCellBlocked(((this.getPosition().x + step)+5) / tileWidth, ((this.getPosition().y)-2) / tileHeight))

return true;

}

return false; }**5. Testing**

**5.1 Description**

A test driven development like approach was used to test the mechanics the game uses: movement, player, math question GUI, and keys and doors.

**5.2 Test Cases and Results for Maze Mode**

|  |
| --- |
| 5.2.1 |
| Description: Testing Protagonist movement with touch motions. Specifically, moving the Protagonist to the left. |
| Input: Pressing a finger on the screen and dragging the finger to the left |
| Expected: Player moves according to the direction of the drag |
| Result: Player moved according to the direction of the drag |

|  |
| --- |
| 5.2.2 |
| Description: Testing player overall movement with touch events. Specifically, moving the Player to any direction with no error or unexpected results. Only four directions, up, down, left, and right. |
| Input: Pressing a finger on the screen and dragging the finger in multiple directions |
| Expected: Player moves according to the direction of the drag |
| Result: Player moved according to the direction of the drag |

|  |
| --- |
| 5.2.3 |
| Description: Testing player collision detection with objects that have collision. If there is an object that has collision, the player should not be able to walk pass the object. |
| Input: Move the player towards an object with collision and try to pass the object. |
| Expected: Collision detection stops player from passing the object. |
| Result: The Collision detection does not work all the time, and the player can phase through the wall. |

|  |
| --- |
| 5.2.4 |
| Description: The player should be able to pick up an item. Specifically, a key. |
| Input: Move the player towards the key. |
| Expected: The player picks up the key. |
| Result: The player picked up the key. |

|  |
| --- |
| 5.2.5 |
| Description: The camera screen should be centered on the Protagonist when the user starts a game. |
| Input: Start a level by clicking on the game button. |
| Expected: Camera screen should centered on the Protagonist. |
| Result: Camera screen is centered on the Protagonist. |

|  |
| --- |
| 5.2.6 |
| Description: According to the current screen that is being shown to the user, the camera screen shall display the correct portion that needs to be displayed to the user |
| Input: Go to mental math mode. |
| Expected: Only the question should be rendered and displayed to the user. |
| Result: The question is be rendered and displayed to the user. |

|  |
| --- |
| 5.2.7 |
| Description: The player should lose a key when the player opens a door. |
| Input: Go to a door and answer the question correctly. |
| Expected: The door should be opened and the key gone. |
| Result: The door is opened and the key is gone. |

|  |
| --- |
| 5.2.8 |
| Description: The player should still lose a key if the question was answered incorrectly. |
| Input: Go to a door and choose the wrong answer to the question |
| Expected: The key should be gone and the door still closed |
| Result: The key is gone and the door is still closed. |

|  |
| --- |
| 5.2.9 |
| Description: The game interface is based on touch input events. For this test, only the game screen should be shown when the user chooses the maze mode. |
| Input: Touch the maze mode button, |
| Expected: The camera screen should displays the maze and the protagonist. |
| Result: The camera screen displays the maze and the protagonist. |

**5.3 Test Cases and Results for Mental Math**

|  |
| --- |
| 5.3.1 |
| Description: Math operation is dependent on the path chosen by the user. Choosing path 1 should only yield addition and subtraction math arithmetic questions. |
| Input: Choose path 1 as the setting for mental math. |
| Expected: Only addition and subtraction math questions should appear. |
| Result: Only addition and subtraction math questions appear. |

|  |
| --- |
| 5.3.2 |
| Description: Answering questions correctly in succession should yield harder questions for the users to answer. |
| Input: Answer multiple math questions correctly. |
| Expected: The next couple of math question should be more difficult than the previous questions. |
| Result: The next couple of math question are more difficult than the previous questions. |

|  |
| --- |
| 5.3.3 |
| Description: Pressing the top right corner of the screen should take the user back to the main menu |
| Input: touch the top right corner of the screen when the math question is opened. |
| Expected: User should be back to the main menu. |
| Result: User is back to the main menu. |

|  |
| --- |
| 5.3.4 |
| Description: Getting a total of three wrong answers should end mental math and return the user back to the main menu |
| Input: Answer the question wrong three times. |
| Expected: The user should be taken back to the main menu. |
| Result: The user is taken back to the main menu. |

**6. Conclusion**

Troubles we had working on this project are similar in aspect to any other project that is new in knowledge. Most of the time spent in the project is researching the tools that are used to create the product. We spent most of our time working with the Libgdx engine in order for it to function as we need it to. The most recurring problem was rendering objects on the screen, and adjusting the camera correctly for the user to see what is actually happening. For example, we had trouble showing the math questions to the users on different devices. Either half the question was shown or the question text was too small. The problem was that we had to find a way to present a 2D world in a 3D plane, and some of the classes required some form of method calling that reduce the difficulty of rendering objects on a 3-D field. Other than this problem, we would definitely have different approaches to this project if we were able to restart it. One of the different things we can do is focus on the aspect of how to render objects on the screen, and control the frames per second. We focused mostly on what was object was going to be placed on the screen rather than how we should do it with any other object. The software lifecycle of our product helped us understand that we must put priority and work in different aspects in the project in order to be efficient.

Most if not all project goals that have been set have been accomplished. A fully functional game is currently present and functional. The educational aspect is currently there and working. A fun experience while playing the game is subjective so it is hard to measure how much of this goal has been accomplished. This project, if used correctly, has the ability to extend the learning opportunities to anywhere you have a smartphone. This provides endless possibilities for the end user and future developers. Future developers may extend this project by adding in other math concepts and providing more game modes.

**7. References**

[1] Zechner, Mario, libgdx Developer’s Guide, Badlogic Games, July 14, 2014. URL: <https://github.com/libgdx/libgdx/wiki/The-life-cycle>

[2] Zechner, Mario,libgdx, Badlogic Games, March 12, 2014. URL: <https://github.com/libgdx/libgdx>

**Appendix**

Test Report

**1. TestCase:**  5.2.1

Description: Testing Protagonist movement with touch motions. Specifically, moving the Protagonist to the left.

Input: Pressing a finger on the screen and dragging the finger to the left

Expected Result: Player moved according to the direction of the drag

Actual Results Obtained: behavior is as expected.

Test Pass.

**2. TestCase:**  5.2.2

Description: Testing player overall movement with touch events. Specifically, moving the

Player to any direction with no error or unexpected results. Only four directions, up, down, left, and right.

Input: Pressing a finger on the screen and dragging the finger in multiple directions

Expected Result: Protagonist moves according to the direction of the drag

Actual Result Obtained: Moved according to the direction of the drag.

Behavior is as expected.

Test Passes.

**3. TestCase:**  5.2.3

Description: Testing player collision detection with objects that have collision. If there is an

object that has collision, the player should not be able to walk pass the object.

Input: Move the player towards an object with collision and try to pass the object.

Expected Result: Collision detection stops player from passing the object.

Actual Result Obtained: Behavior is as intended, collision detection works.

Test Passes.

**4. TestCase:**  5.2.4

Description: The player should be able to pick up an item. Specifically, a key.

Input: Move the player towards the key.

Expected: The player picks up the key.

Actual Result Obtained: Protagonist picks up key. Behavior is as expected.

Test Passes.

**5. TestCase:**  5.2.5

Description: The camera screen should be centered on the Protagonist when the user starts a

game.

Input: Start a level by clicking on the game button.

Expected Result: Camera screen should centered on the Protagonist.

Actual Result Obtained: Behavior as intended.

Test Passes.

**6. TestCase:** 5.2.6

Description: According to the current screen that is being shown to the user, the camera screen shall display the correct portion that needs to be displayed to the user

Input: Go to mental math mode.

Expected: Only the question should be rendered and displayed to the user.

Actual Result Obtained: Single question displayed. Behavior as intended.

Test Passes.

**7. TestCase:**  5.2.7

Description: The player should lose a key when the player opens a door.

Input: Go to a door and answer the question correctly.

Expected: The door should then be opened and the key removed from protagonist.

Expected Result: The door is opened and the key is gone.

Actual Result Obtained: Behavior technically is as intended.

There are two different keys to obtain in the game: the regular key and

the master key. The bug is regular keys can open the master door, which should not be possible.

Test passes, with reservation.

**8. TestCase:**  5.2.8

Description: The player should still lose a key if the question was answered incorrectly.

Input: Go to a door and provide the wrong answer to the question

Expected Result: The key should be gone and the door still closed.

Actual Result Obtained: The key is gone and the door is still closed.

Result is as intended.

Test Passes.

**9. TestCase:** 5.2.9

Description: The game interface is based on touch input events. For this test, only the game

screen should be shown when the user chooses the maze mode.

Input: Touch the maze mode button.

Expected: The camera screen should display the maze and the protagonist.

Actual Result Obtained: Behavior is as intended. Recommendation: add space in between Map

Select and map buttons

Test Passes.

**10. TestCase:**  5.3.1

Description: Math operation is dependent on the path chosen by the user. Choosing path 1

should only yield addition and subtraction math arithmetic questions.

Input: Choose path 1 as the setting for mental math.

Expected Result: Only addition and subtraction math questions appear.

Actual Result Obtained: Not present in the application yet.

Test Fails.

**11. TestCase:**  5.3.2

Description: Answering questions correctly in succession should yield harder questions for the

users to answer.

Input: Answer multiple math questions correctly.

Expected Result: The next couple of math question should be more difficult than the previous

questions.

Actual Result Obtained: Behavior is as expected.

Test Passes.

**12. TestCase:** 5.3.3

Description: Pressing the top right corner of the screen should take the user back to the main

menu

Input: touch the top right corner of the screen when the math question is opened.

Expected Result: User should be back to the main menu.

Actual Result Obtained: Functions as expected, but the selection area is ridiculously small.

Area of where to press should increased in size.

Test Passes\* with reservations.

**13. TestCase:**  5.3.4

Description: Getting a total of three wrong answers should end mental math and return the user back to the main menu.

Input: Answer the question wrong three times.

Expected Result: The user should be taken back to the main menu.

Actual Result Obtained: The user is taken back to the main menu.

Behavior is as intended. Suggestion to authors: If the same answer is selected more than once,

it should not count against you. The same answer should not be presented more than once.

Test Passes.