Software Requirements Specification (SRS)
Project X

Team: Group 5
Authors: Eric Carey, Shuhe We, Jacob Hempel, Alex Meier
Customer: Elementary Schools
Instructor: Dr. James Daly

1 Introduction

This Software Requirements Specification document consists of the following subsections:

1. Purpose
2. Scope
3. Definitions
4. Perspective
5. Functions
6. Characteristics
7. Requirements
8. Prototype
9. How to Use
10. User Scenarios

We start by going over why the game is being made, what the scope of the software is, and terms that need to be known to understand the software. We then delve into the context of the software, what requirements and functions are involved in developing and running it, and its characteristics. And finally at the end of the document one can find the prototyping, and instructions on how to run the prototype.

1.1 Purpose

This document is meant to explain the functions and detailed workings of the software for our Math Match edutainment game. Many of the sections cover how the software works and what the user will need to know in order to interact with it. Reading this document will give the reader the knowledge to understand exactly what the system does.

The intended audience for this document is anyone looking to utilize this game for education purposes. This will most likely not be the intended user of the game because the game is meant for elementary age children who will not have the education to understand the terminology and concepts discussed throughout. However, anyone invested in learning how this software works to help implement it in schools or other education formats, including teachers, administrators and IT personnel, will find all the information they need in this document.

1.2 Scope

The software product we are creating, and describing in throughout this document is Python based terminal application called Math Match. This software will be used to educate young children, specifically from the ages of x to x, basic math skills. They will be able to practice their addition and subtraction skills, and those more experienced can even move on to multiplication and division. We hope that by facilitation the focus on equivalent equations and numbers will strengthen their math skills as well as their decision making and timing. The objective of the game is to match each piece with either a number or equation on it in the hand with its equivalent number or equation bucket in

the game field. The ultimate goal, or the way to win rather, is to fill all of the buckets before they reach the bottom and fill the game field.

The software will facilitate drag and dropping of piece from the hand to the game field. It will also be able to evaluate equivalency and remove pieces and buckets that match up correctly. The buckets will be able to move down from the top to the bottom of the game field, only stopping when matching with a game piece. The software will time the user and generate a score based off of this time.

1.3Definitions, acronyms, and abbreviations

Bucket: A rectangular box holding an expression to be matched to and erased by the user.

Piece: The object with the expression the user is trying to match to an expression in a bucket.

Play field: The main area of the game that holds the buckets. This is the area where the user will drag the pieces to and match to a bucket.

Hand: This is the box that holds the pieces that are available for the user to drag to the play field.

Score Counter: A 4-digit number on the top left corner of the prototype screen displaying the user's score.

1.4Organization

The rest of this document will consist of different aspects of the actual software. We begin with a description of the system and it's different interfaces, followed by the functions being done. We then move on to the characteristics of the user and any constraints that might be present. Requirements and use cases follow after that. Finally the last few sections cover the prototype and how to use it.

1 Overall Description

This section will cover the functionality and characteristics of the software and the user.

2 Product Perspective

This product is a stand alone game, meant to be used as a supplementary addition to teaching and testing of elementary children's basic mathematical skills. This product can be found in schools or at home for fun educational purposes.

- Interface Constraints: Game will be restricted to on screen controls, selected by a mouse click made by the user
- System interface: reading the expression in the piece, outputting whether that is equivalent to the expression in the bucket.

- User Interface: mouse clicks and drags controlling the pieces
- Software interface: Pygame styled python code
- Hardware interface: any computer with python installed
- This game will take up very little memory
- Only a few operations for evaluating equivalency and determining what buckets pieces are in will be needed
- Site adaptation: user will customize the game by picking what difficulty game to load

3 Product Functions

The major functions of the product include difficulty selection, expression creation, expression comparison, and object removal.

While the game is playing, it will have to create new equations for the users to use in their matching. This may be done using a big list of valid equations with their solutions. This function will use the output of the difficulty selection area to determine which equations are valid. The game will also compare equations as the user attempts to match them. This will be used to determine if the two equation blocks should be deleted, of it the user's piece should be returned to the starting area. If the two equations match, then the game will remove both of them from the game world, and award points to the user.

4 User Characteristics

The intended user for our edutainment game are between the ages of 6 to 8 years old. These children will likely be in elementary school and just starting to learn basic mathematics. Users background should involve addition, subtraction, multiplication, division, as well as the concept of equality. Mixed operations will not be implemented in this version of the game so knowledge of the order of operations will not be necessary. There are several difficulties to our game, where the younger kids can practice simple addition and subtraction and the older kids can move on to multiplication and division. The desired audience should have a simple level of knowledge of how a computer works and be able to operate a mouse to drag and drop the pieces into place. Just a small amount of skill will be sufficient enough to operate and play this game.

Constraints

The application will be limited to the scope of the Python language and the Pygame library as these are the tools used to design the game. Development may also be constrained by the problems that come with collaboration. The developers will need to work together and divide tasks causing some to rely on others to get their parts done correctly. The only safety properties that need to be considered are developers compromising the system when possible additions to the complexity of expressions is

made for future levels. The user must not drag and drop pieces outside of the range of the game as this may cause unforeseen errors and crashes.

Assumptions and Dependencies

In order to run this application, a python3 installation is required as well as the pygame graphics library. Likewise, it is assumed that the user either has these installed on their system, or have the permissions required to do so. Additionally, it is assumed that the user will have some basic computer knowledge such as “dragging and dropping” items, and using the mouse to interact with buttons on screen. The user must have a computer running Windows, OSX, or any linux distribution. This application is not compatible with IOS, Android, or other mobile operating systems (including ChromeOS).

5 Apportioning of Requirements

This product will not have score or time based requirements to move on to the next level. The level to be loaded is simply a decision made by the user at the beginning of the game. There will also not be a change in complexity of expressions. The different levels will simply separate just addition and subtraction from all four major operations. Another aspect that will not be included in this version but may be added later is the addition of various sizes and shapes of the buckets. Everything will be symmetrical and move in a static pattern.

6 Specific Requirements

1.User should be asked to select either easy mode or hard mode contains different types of calculation before the game begins

1.1. When the user moves the mouse on the easy mode it should show suggested age 6-7

1.2. When the user moves the mouse on the hard mode it should show suggested age 7-8.

2.Program should generate 3 equations with different values after the user chooses the difficulty of the game.

2.1. Each of the equation should be displayed in a bucket on the top of the screen.

2.2. Easy mode should only have addition and/or subtraction in the expressions and no more than 3 numbers should be in the expression.

2.3. Hard mode should have 2 types of expressions. One is the equation have addition and/or subtraction. Another type is the expressions have multiplication and/or division. Also no more than 3 numbers should be in the expression.

3.When game begins, in each of the buckets on the top of the screen, one expression that has the same value as one of the functions in the piece should show and as time goes by there should be one more function showing.

3.1. The function showed in one of the buckets should not be the same as the function in another bucket

3.2.The function in a piece with a different size indicates a different score, smaller pieces means a higher score.

4.When user pulls the piece to the bucket that has the same value as the bucket both should be deleted and the game should add the appropriate score which shows on the top right corner

4.1.Initial score should be 0

4.2.The piece gets deleted only when the value of the function in the piece matches the user's choice in the top buckets

5.When the user pulls the wrong piece to the wrong bucket then the piece should go back to the original place in the hand.

5.1.When the user chooses the wrong function, time cost should increase and a new bucket should immediately added to the play field

6.When the buckets reach the bottom on the screen then the is game over

6.1.The score should show on the screen when the game over

6.2.Time cost should show on the screen when the game over

7 Modeling Requirements

Use Case Name:	
Actors:	
Description:	
Type:	
Includes:	
Extends:	
Cross-refs:	
Uses cases:	

Use Case Name:	Choose difficulty
Actors:	Player
Description:	Before the game starts, the player will be prompted to choose a difficulty setting. A pop up window displays buttons for easy, and hard.
Type:	Primary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	Initialize hard mode, Initialize easy mode

1.

Use Case Name:	Initialize hard mode
Actors:	Program
Description:	Show 3 random functions in 3 buckets on bottom of the screen. Functions should either contain addition and/or subtraction or multiplication and/or division.functions should have different values. Score will be initialized to 0 and shows on top right corner.
Type:	Secondary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	None

Use Case Name:	Initialize easy mode
Actors:	Program

Description:	Show 3 random functions in 3 buckets on bottom of the screen. Functions should either contain addition and/or subtraction .functions should have different values. Score will be initialized to 0 and shows on top right corner.
Type:	Secondary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	None

Use Case Name:	Game play
Actors:	Player and program
Description:	At the start of the game, the player is presented with one row of buckets in the playfield and a hand of pieces. The player may choose to drag a piece to one of the buckets and drop it. If the piece matches the bucket, both the piece and the bucket is deleted. If the piece does not match the bucket, a new row of buckets is inserted into the playfield. If the playfield becomes full (i.e. the first row reaches the bottom of the screen) a game over condition is met.

Type:	Primary and essential
Includes:	Right Match, Wrong Match, Grab a Bucket
Extends:	
Cross-refs:	
Uses cases:	End the game and show score

Use Case Name:	Right match
Actors:	Program
Description:	Both the held piece and the selected bucket will be removed from the screen. The score is incremented by the appropriate amount.
Type:	Secondary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	None

Use Case Name:	Wrong match
Actors:	Program
Description:	The held piece is dropped and returned to the hand. A new row of buckets is inserted at the top of the playfield.
Type:	Secondary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	None

Use Case Name:	End the game and show score
Actors:	Program
Description:	When the bottom most row of buckets reaches the bottom of the playfield, the game ends and the total score is shown on screen.

Type:	Secondary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	None

Element Name		**Description**
Playfield		Contains Buckets, represents the "Game Board"
Attributes		
	SizeXY:Integer Tuple	Width and height of play field grid
	Bucket:Bucket[x][y]	2 dimensional array of bucket objects
Operations		
	Advance ():void	Advances the buckets 1 step downward
	Destroy (int x, int y):	Removes the bucket at the given coordinates.
Relationships	Contains a grid of Bucket objects.	
UML Extensions		

Element Name		**Description**
Bucket		Contains an equation
Attributes		
	Equation:String	Mathematical equation represented as a string (“3+1”, “1 + 1” etc.)
	Solution:Int	Integer solution to math problem used to compare with answer Pieces
Operations		
	IsSolution(Piece):Boolean	Returns true if the Piece object given is a solution to the Bucket.
		Description4
Relationships		
UML Extensions		

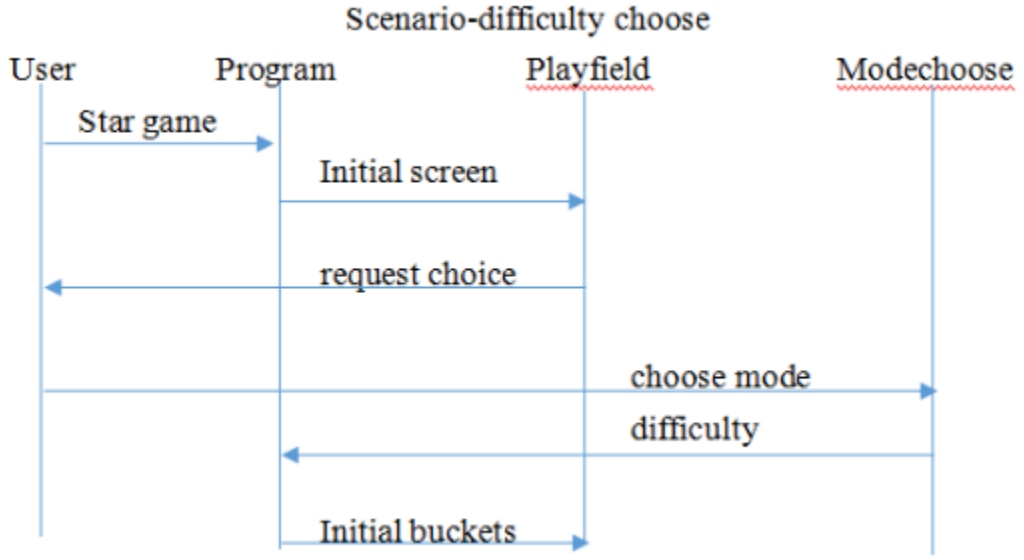
Element Name		**Description**
Hand		Represents the players hand of playable pieces.
Attributes		
	Pieces:Piece[]	Array of Piece objects.
	Size:int	Maximum hand size
Operations		
	AddPiece(int solution):void	Add piece with the given solution to the hand

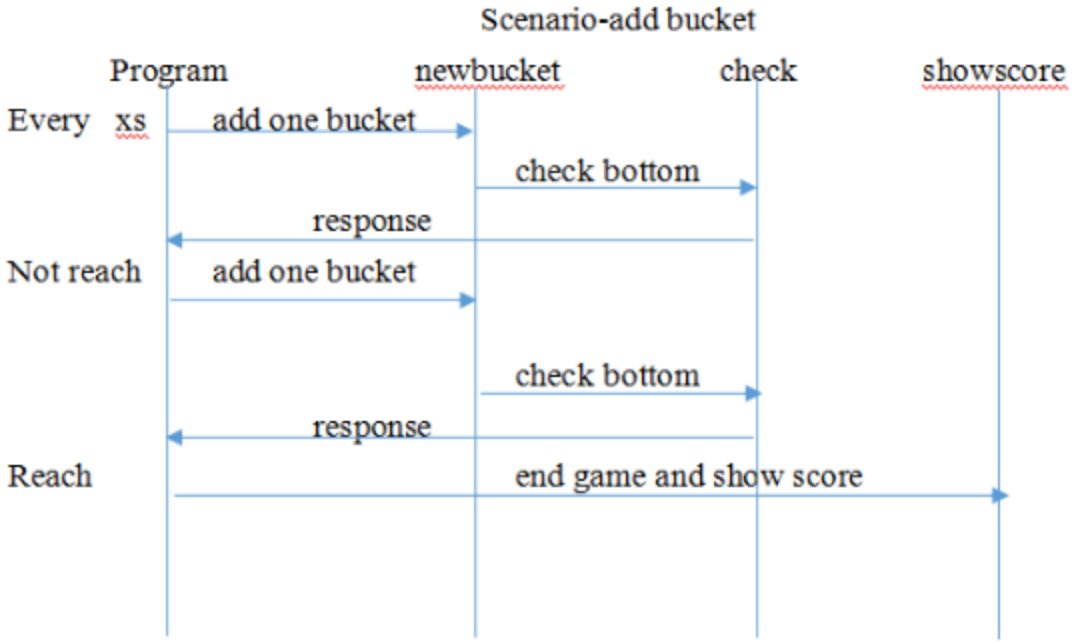
Relationships		
UML Extensions		

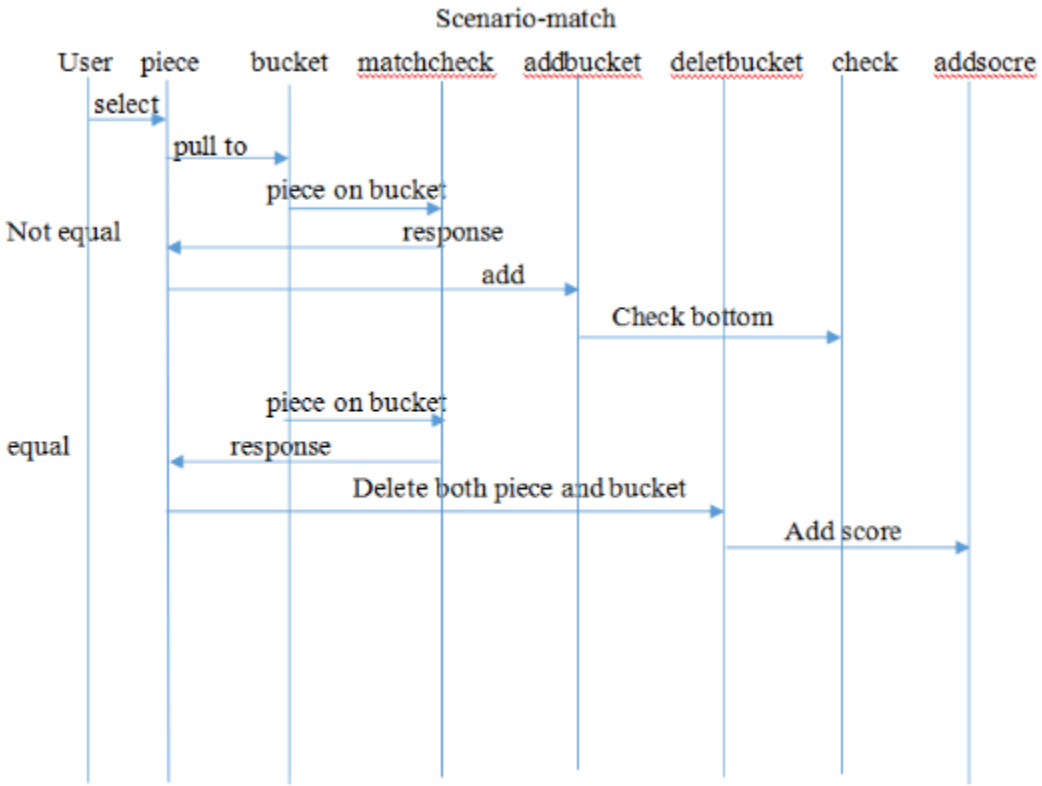
Element Name		**Description**
Piece		Represents a piece that can be played by a player
Attributes		
	Equation:String	Mathematical equation string
	Solution:Int	Integer solution to Equation for comparison with Bucket objects.
Operations		
Relationships		
UML Extensions		

Element Name	**Description**

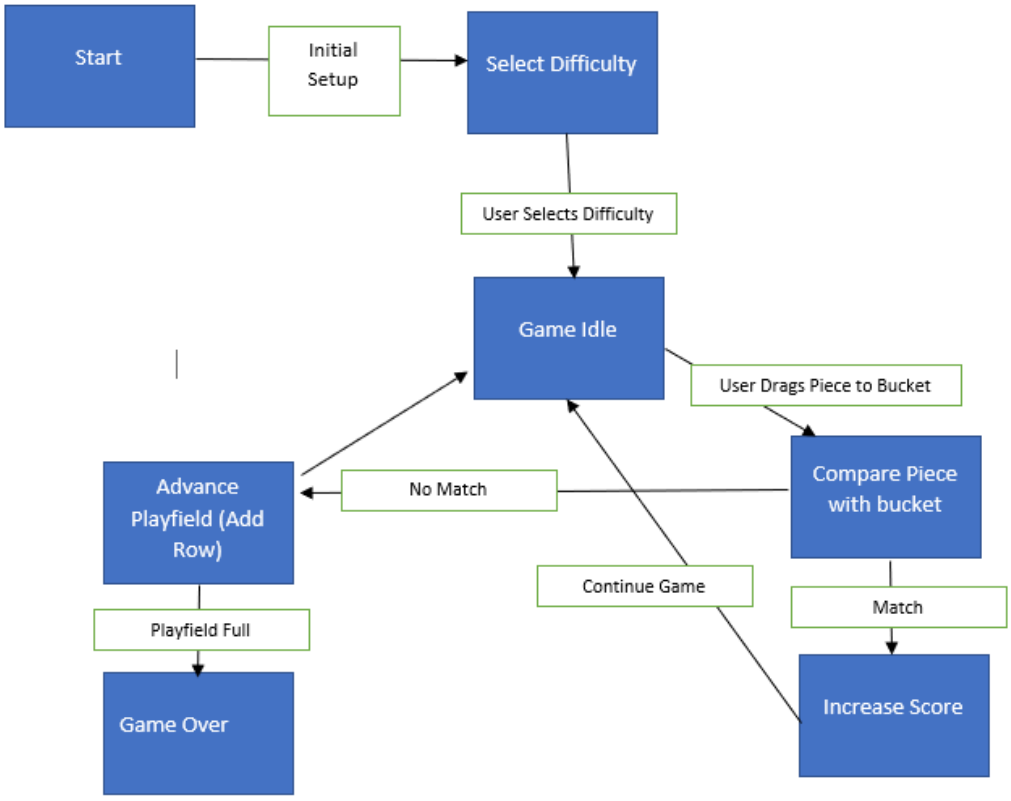
Game		Singleton object that keeps track of game state.
Attributes		
	Playfield:Playfield	Instance of Playfield Object
	Hand:Hand	Instance of Hand Object
	GameOver:Boolean	Set to true if a game over condition is met.
Operations		
Relationships		
UML Extensions		







State Machine:



8 Prototype

The Prototype shows a basic representation of the UI elements that will be present in the final version of the game, with some functionality. It demonstrates the process flow from starting the application and selecting a difficulty from the startup menu, completing a simplified matching puzzle, leading to a simple end game screen. This prototype includes the functionality to drag and match pieces with buckets, as well as other minimal functionality required to demonstrate the game from start to finish. This prototype also includes nonfunctional UI components as placeholders for the score counter, and timer components that will be present in a later version.

9 How to Run Prototype

This prototype requires a Python 3.0 installation. Documentation on installing python and its dependencies can be found on the official python website (https://www.python.org/download/releases/3.0/). Once python is installed, the required dependencies for the prototype can be installed using the pip3 command in your operating systems command interpreter:

pip3 install pygame

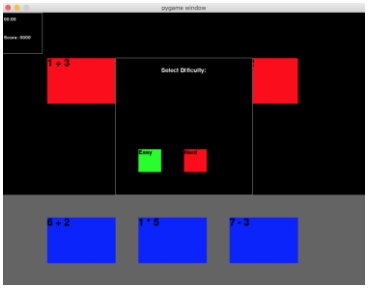
Pygame is compatible with any of the major operating systems (Windows, OSX, Linux). Once the above requirements are met, the prototype can be cloned or downloaded from the following repository:

https://github.com/alexmeier2828/SoftwareEngineering

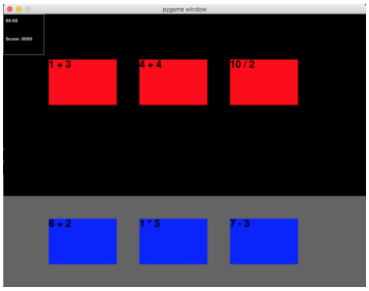
To run the prototype, navigate to the SoftwareEngineering/Prototype folder, then run the following command:

$python3\ main.py$

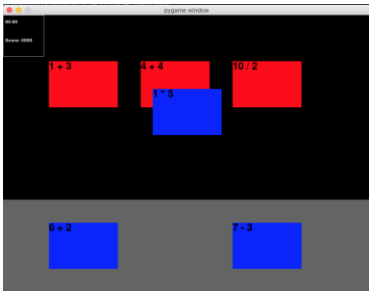
10 Sample Scenarios



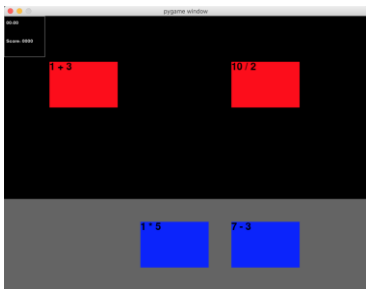
1

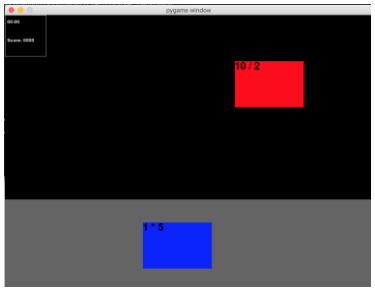


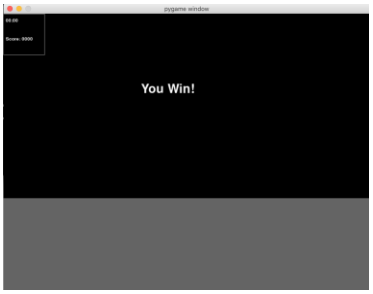
2



3



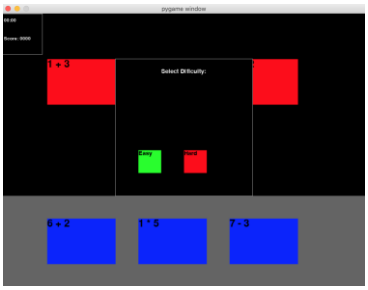


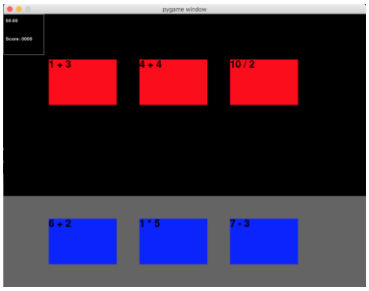


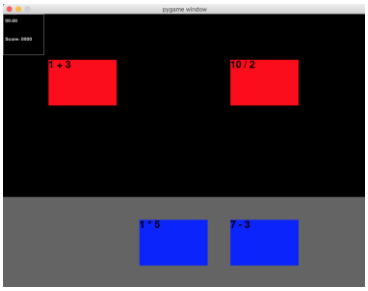
4 5 6

1. Game loads and user selects easy mode(green button)
2. User begins to do the math to calculate each piece
3. User drags middle piece to middle bucket but answer is incorrect, piece returns to hand(same screen as 2)
4. User recalculates each piece and bucket then matches first piece to middle bucket and both disappear
5. User matches third piece to the first bucket and both disappear
6. User matches middle piece to last bucket and both disappear. The game ends.

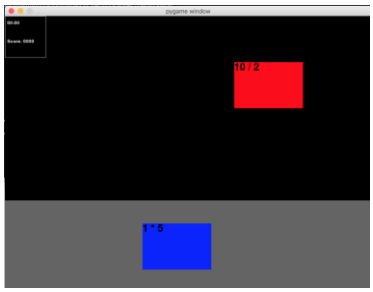
11 Sample Scenarios

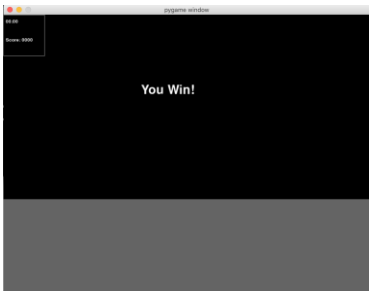






1 2 3





4 5

1. Game loads and user selects hard mode(red button)
2. User begins to do the math to calculate each piece
3. User then matches first piece to middle bucket and both disappear
4. User matches third piece to the first bucket and both disappear
5. User matches middle piece to last bucket and both disappear. The game ends.

12 References

[1] Bartłomiej, Burek “Python-Examples”, n/a , n/a, October 2019. https://github.com/furas/python-examples/tree/master/pygame

13 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.