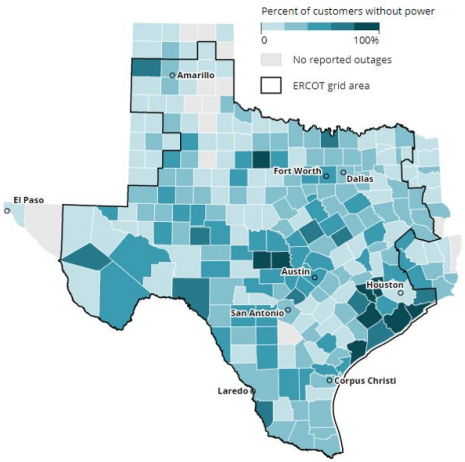
Dwight Look College of
ENGINEERING
TEXAS A&M UNIVERSITY

ECEN 404 Demo
Team #28: Power Outage Education App

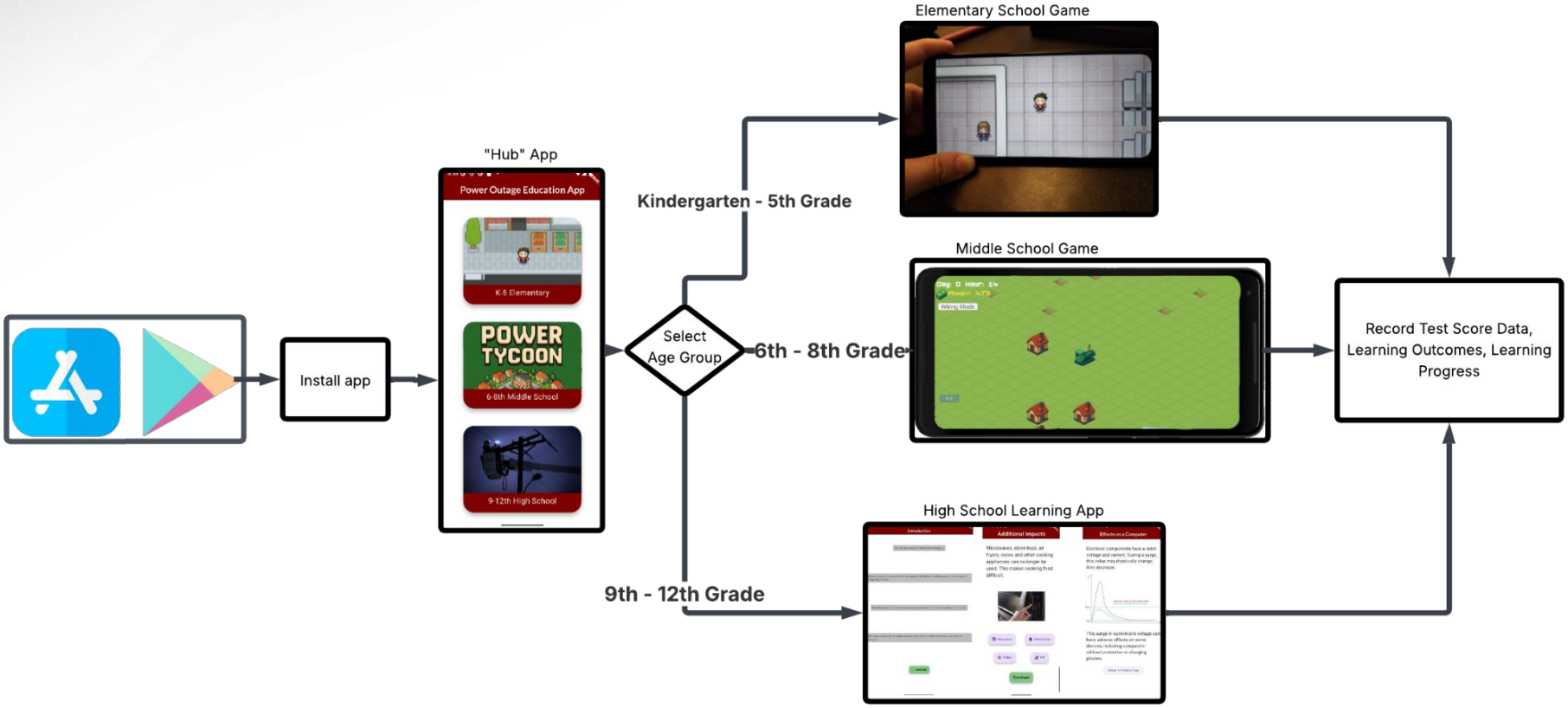
Jackie Villanueva, Aidan Petropoulos, Joey Raphael
Sponsor: Dr. Mladen Kezunovic
TA: Swarnabha Roy

Problem Overview

- Problem statement: “People are often uninformed about the best course of actions to take before and during a power outage.This lack of knowledge can result in lack of preparation and uninformed decision-making that can cause further harm.”
  - The 2021 Texas Freeze
  - Outages caused by natural disasters
- Solution: Develop an app that provides different age-specific knowledge to educate students about the impact of power outages utilizing engaging, interactive apps. By informing people about what measures to take before, during, and after a power outage, the harm caused by power outages can be mitigated.



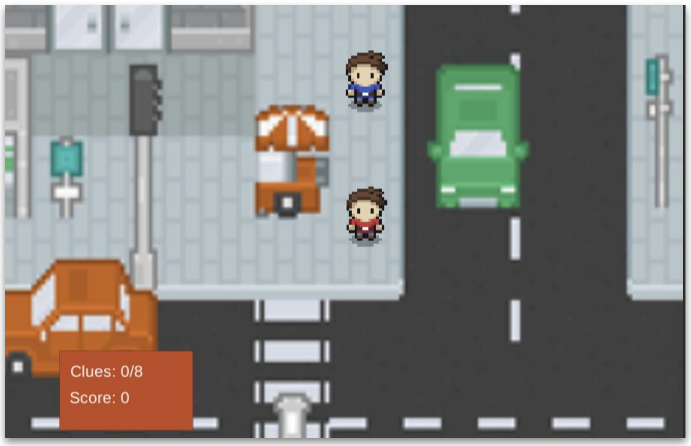
Integrated System Diagram

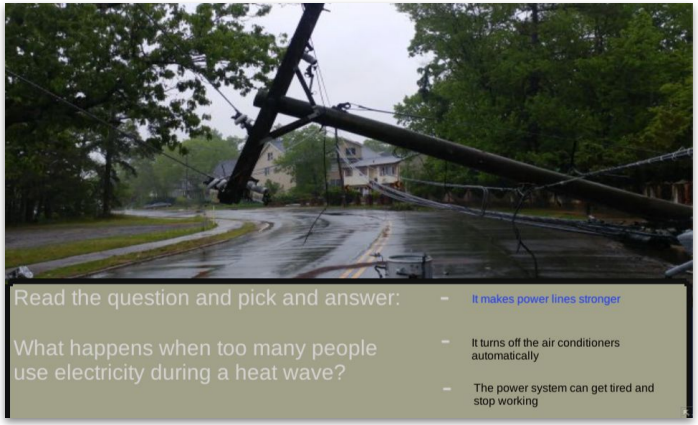


Engineering Design Accomplishments

Aidan Petropoulos

- Developed an app meant for K-5th graders on Unity
- Designed Features:
  - Compatible with PC/Android devices
  - Designed interactive map that mirrors real-world Doseum
  - Integrated pre- and post-quizzes for data collection
  - Built node-based clue system
  - Implemented clue-tracking system that rewards correct interactions





Code-Game Controller

- Dictates the game mode and the which modes to change
- Different modes:
  - freeRoam
  - Dialog
  - Clue
  - Paused
- Stores the which clues have been solved under solved clues

CODE-Player Controller

- This is what takes in the player input of movement
- Touch input for mobile devices
- Uses the Character script to move the player
- Interacts with the NPCs and other objects
- Sees if player has stepped on a clue

CODE-Character

- Takes in the information from Player and does the movement
- Makes it so the player walks in center of tiles to make it look more natural
- Collision logical

CODE-NPCController

- Controls the NPC idle, walking, and dialog
- Has movement pattern
- Speaks with dialog manager to speak with display NPC dialog

CODE-DialogManager

- Displays the UI box and text
- Has player click next with touch or Z
- Has the different dialog

CODE-ClueSystem

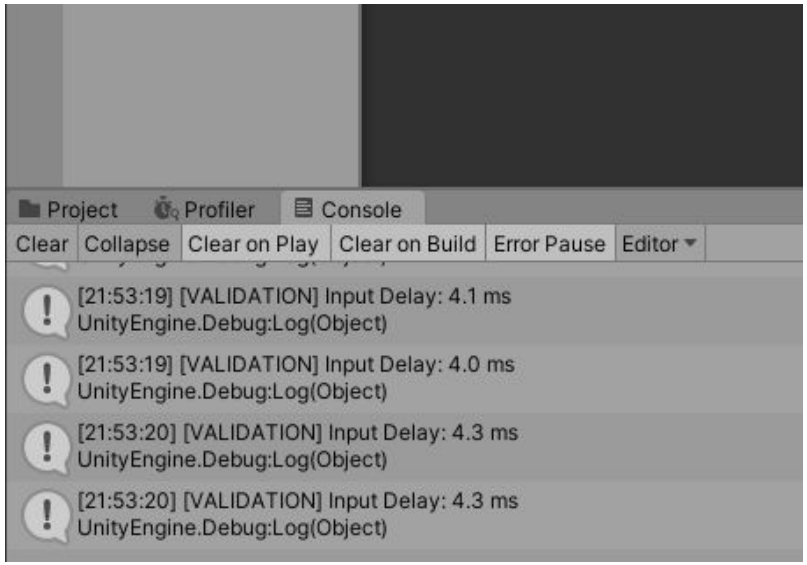
- Handles the quiz interactions
- Displays the questions and answer choices
- Prevents double taps
- Point manages
- Stores the solved clue IDs

Dwight Look College of ENGINEERING
TEXAS A&M UNIVERSITY

Engineering Design Accomplishments

Aidan Petropoulos

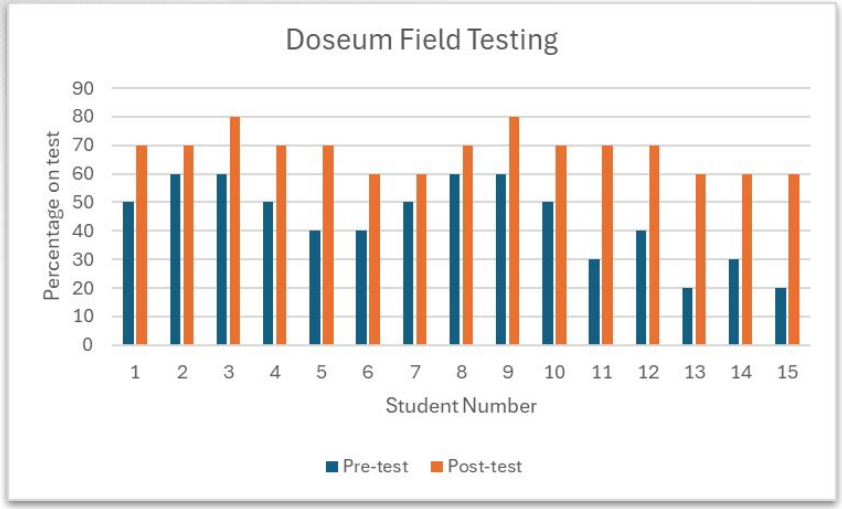
Specification	Measured Results
Average ≥ 30 FPS throughout play session	Average FPS: 40 FPS | Minimum: 28 FPS | Maximum: 52 FPS | 95% of frames < 33 ms
Input latency between user interaction and visual feedback ≤ 200 ms	Average: 4.2 ms | Min: 4.0 ms | Max: 4.3 ms
Game loads completely within 5 seconds (WebGL launch)	Average: 3.23 s | Min: 3.1 s | Max: 4.3 s
Memory usage ≤ 2 GB	Average: 2.05 MB | Max: 428.6 MB | Stable over 60 s
App reloads safely after temporary network loss	Recovery Time: 4.1 s | No crash
UI scales consistently across 8–12 inch displays	Readable and aligned



Dwight Look College of ENGINEERING
TEXAS A&M UNIVERSITY

Engineering Design Accomplishments

Aidan Petropoulos



Metric	Pre-Test (%)	Post-Test (%)	Change
Average	40.8	68.5	27.7
Minimum	20	60	40
Maximum	60	80	20
Standard Deviation	13.3	6.6	-6.7
Improvement (%)		68%	
Participants (n)	15	15	

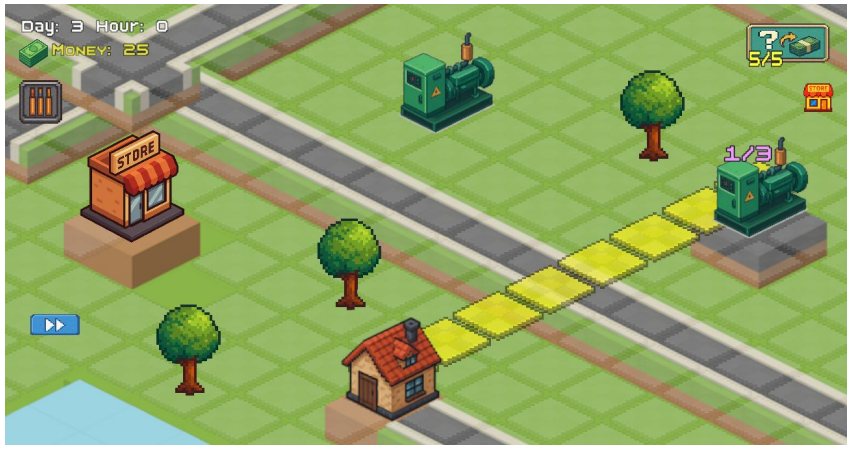
Validation Plan Doseum game

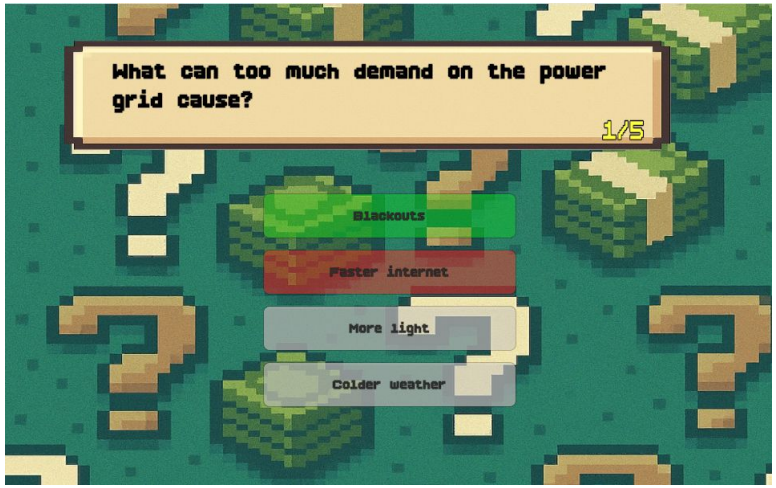
Paragraph #	Test Name	Success Criteria	Methodology	Status	Responsible Engineer(s)
3.1.0	Game Testing (LittleDoers)	Game is playable and the objective is accomplished.	Run game on Unity and complete a playthrough within a reasonable amount of time.	Fully Tested	Aidan Petropoulos
3.2.0 & 3.2.2	App/Game Perfromance	Each system of the app runs smoothly (at least 20 FPS). The app does not crash.	Simulate the app on computer and on a phone. Check that the application is able to 'run' each game/subsystem without crashes.	Fully Tested	Aidan Petropoulos
3.2.12	Offline and Online Modes	The app has different interfaces for if it is online or offline.	Run the software with and without wifi. Check if the interface differs between the two scenarios.	Fully Tested	Aidan Petropoulos
3.2.3	Learning Feedback System	Feedback can be provided based on the user's performance.	Run the app. Explore different scenarios and choices. Check that the app accepts the various scenarios and performs accordingly.	Fully Tested	Aidan Petropoulos
3.2.5	Interoperability	The app can be run on either IOS or Android.	Simulate the app on both types of devices. Check that the application does not crash and has equal performance quality.	Fully Tested	Aidan Petropoulos

Engineering Design

Joey Raphael

- Developed a Unity game targeted towards middle school students
- Main Features
  - Compatible with PC / Android devices
  - Integrated Quizzing System
  - Implemented A* algorithm for automated wire pathfinding
  - Developed grid-building system for tycoon-style game





Use of AI Disclosure

- Generative AI was used to create assets for the game (textures, UI, sprites, questions, etc.) and to assist in documenting code
- Generative AI was **not** used in implementing algorithms, programming game in general.

Tutorial!

Goal: Power all the buildings by connecting generators to them!

Wiring Button: Select a generator and building to create a wire between them, then click again to confirm selection. Powered buildings generate money so you can buy more generators! Deleting wires is done similarly.

However, wires cost money! $50 per tile, so be strategic in where you place generators and which houses you choose to connect to!

Answer questions about power outages to earn more money!

Buildings generate money per hour, but you must press this button to go through the hours!

General Gameplay / Experience

A* Algorithm

- Algorithm to find the shortest path between point A and point B
- 3 main variables
  - f-score = the sum of the g-score and h-score
  - g-score = cost to move from point A to point B
  - h-score = heuristic cost, cost of current shortest path, guess using manhattan distance $|x_1 - x_2| + |y_1 - y_2|$
- Create two lists, open list and closed list
  - Open list contains known nodes, but not explored
  - Closed list contains nodes already explored
- For every neighbor in a selected node
  - Calculate the g-score, f-score, h-score and add to open list if not in existing list
  - If neighbor node is in open list and has lower f-score, explore node
- Iterate, choose node with lowest f-score from open list, then move node to closed list
- Terminate for loop when path is found from goal node from open list

```
16      public class AStarManager : MonoBehaviour
25          public List<Node> GeneratePath(Node start, Node end) // Create of a list of nodes, determined the gScore for each node, determine the heuristic function
37
38              while(openSet.Count > 0)
39              {
40                  int lowestF = default;
41
42                  for(int i = 1; i < openSet.Count; i++)
43                  {
44                      if (openSet[i].FScore() < openSet[lowestF].FScore())
45                      {
46                          lowestF = i;
47                      }
48                  }
49
50                  Node currentNode = openSet[lowestF];
51                  openSet.Remove(currentNode);
52
53                  if(currentNode == end)
54                  {
55                      List<Node> path = new List<Node>();
56
57                      path.Insert(0, end);
58
59                      while(currentNode != start)
60                      {
61                          currentNode = currentNode.cameFrom;
62                          path.Add(currentNode);
63                      }
64
65                      path.Reverse();
66                      return path;
67                  }
68
69                  foreach(Node connectedNode in currentNode.connections)
70                  {
71                      float heldGScore = currentNode.gScore + Vector2.Distance(currentNode.position, connectedNode.position);
72
73                      if(heldGScore < connectedNode.gScore)
74                      {
75                          connectedNode.cameFrom = currentNode;
76                          connectedNode.gScore = heldGScore;
77                          connectedNode.hScore = Vector2.Distance(connectedNode.position, end.position);
78
79                          if (!openSet.Contains(connectedNode))
80                          {
81                              openSet.Add(connectedNode);
82                          }
83                      }
84                  }
85              }
86
87              return null;
88          }
```

Grid Building System

- Wiring Mode (enum)
  - 0 = null, 1 = wiring mode, 2 = wire cutter mode
- Place all interactable tiles beforehand, create nodes for each tile, use A* Star Manager to create paths
- When two buildings are connected, create a dictionary that uses the names of the two connected buildings as key, and the node path as the value
  - Replace all tiles on the path with wire tiles
- When deleting path of wires between two buildings, refer to dictionary, delete entry from dictionary, set tiles back to normal

```
387         if (wiringModeOn != 0)
388         {
389             if (Input.GetMouseButtonDown(0) && IsOverUIElement == false)
390             {
391                 //Debug.Log(cellPos + " lkjkdlaf");
392                 if (wiringModeOn == 1)
393                 {
394 
395                     if (selectedGameObjects.Count == 2)
396                     {
397                         Debug.Log("Two objects have been selected, generating path of wires!");
398                         Debug.Log($"Looking for Node {selectedGameObjectsPosition[0].x}, {selectedGameObjectsPosition[0].y}");
399                         Debug.Log($"Looking for Node {selectedGameObjectsPosition[1].x}, {selectedGameObjectsPosition[1].y}");
400                         Node startingNode = AStarManager.instance.FindNearestNode(new Vector2(selectedGameObjectsPosition[0].x, selectedGameObjectsPosition[0].y));
401                         Node endingNode = AStarManager.instance.FindNearestNode(new Vector2(selectedGameObjectsPosition[1].x, selectedGameObjectsPosition[1].y));
402                         List<Node> pathOfNodes = AStarManager.instance.GeneratePath(startingNode, endingNode);
403 
404                         //Debug.Log(selectedGameObjects[0].GetComponent<Building>().income + " dafadf");
405                         int numberOfWires = 0;
406                         if (selectedGameObjects[0].GetComponent<Building>().nodeConnectionsDictionary.Count  == selectedGameObjects[0].GetComponent<Building>().maxAllowedConnections)
407                         {
408                             Debug.Log("Max amount of connections!");
409                             GameObject.Find("CoreGameCanvas").GetComponent<UI_Manager>().SendNotification("No more available connections!", "alert", Color.red);
410                         }
411                         else if (pathOfNodes.Count * 50 <= playerManager.money)
412                         {
413                             foreach (Node node in pathOfNodes)
414                             {
415                                 Debug.Log(node.name + " HI!");
416                                 WireTilemap.SetTile(Vector3Int.FloorToInt(node.position), wireTile);
417                                 numberOfWires++;
418                             }
419                             Debug.Log($"Number of wires: {numberOfWires}");
420                             selectedGameObjects[0].GetComponent<Building>().nodeConnectionsDictionary.Add(new List<string>() {selectedGameObjects[0].name, selectedGameObjects[1].name}, pathOfNodes);
421                             selectedGameObjects[1].GetComponent<Building>().nodeConnectionsDictionary.Add(new List<string>() {selectedGameObjects[0].name, selectedGameObjects[1].name}, pathOfNodes);
422                             selectedGameObjects[0].GetComponent<Building>().ConnectionText.GetComponent<TextMeshPro>().text = $"{selectedGameObjects[0].GetComponent<Building>().nodeConnectionsDictionary.Count}/{selectedGameObjects[0].GetComponent<Building>().maxAllowedConnections}";
423                             playerManager.AddMoney(-50 * numberOfWires);
424                             selectedGameObjects[0].GetComponent<Building>().isConnectedToPower = true;
425                             selectedGameObjects[1].GetComponent<Building>().isConnectedToPower = true;
426                         }
427                         else
428                         {
429                             Debug.Log("Player doesn't have enough money!");
430                             GameObject.Find("CoreGameCanvas").GetComponent<UI_Manager>().SendNotification("Not enough money!", "alert", Color.red);
431                         }
432 
433                         selectedGameObjects[0].transform.Find("Sprite").GetComponent<SpriteRenderer>().color = Color.white;
434                         selectedGameObjects[1].transform.Find("Sprite").GetComponent<SpriteRenderer>().color = Color.white;
435                         selectedGameObjects.Clear();
436                         selectedGameObjectsPosition.Clear();
437                     }
```

```
    else if (wiringModeOn == 2)
    {
        if (selectedGameObjects.Count == 2)
        {
            Debug.Log("Two objects have been selected, deleting path of wires!");
            // force player to select generator
            Node startingNode = AStarManager.instance.FindNearestNode(new Vector2(selectedGameObjectsPosition[0].x, selectedGameObjectsPosition[0].y));
            Node endingNode = AStarManager.instance.FindNearestNode(new Vector2(selectedGameObjectsPosition[1].x, selectedGameObjectsPosition[1].y));
            List<Node> pathOfNodes = AStarManager.instance.GeneratePath(startingNode, endingNode);
            //pathOfNodes.Remove(pathOfNodes[0]);

            foreach (var node in selectedGameObjects[0].GetComponent<Building>().nodeConnectionsDictionary.ToList())
            {
                Debug.Log($"node.Key = {node.Key[0]}, {node.Key[1]}");
                if (node.Key.Contains(selectedGameObjects[0].name) && node.Key.Contains(selectedGameObjects[1].name))
                {
                    int numberOfWires = 0;
                    foreach (Node nodeInDictionary in node.Value)
                    {
                        Debug.Log(nodeInDictionary.name + " BYE!");
                        WireTilemap.SetTile(Vector3Int.FloorToInt(nodeInDictionary.position), null);
                        //playerManager.AddMoney(50);
                        // delete connections, delete list, set connection to false
                    }
                    List<string> entryName = new List<string> {selectedGameObjects[0].name, selectedGameObjects[1].name};
                    selectedGameObjects[0].GetComponent<Building>().nodeConnectionsDictionary.Remove(node.Key);
                    selectedGameObjects[1].GetComponent<Building>().nodeConnectionsDictionary.Remove(node.Key);
                    Debug.Log(selectedGameObjects[0].GetComponent<Building>().nodeConnectionsDictionary.Count);
                    Debug.Log(selectedGameObjects[1].GetComponent<Building>().nodeConnectionsDictionary.Count);
                    //selectedGameObjects[1].GetComponent<Building>().nodeConnectionsDictionary
                    playerManager.AddMoney(10 * numberOfWires);
                    selectedGameObjects[0].GetComponent<Building>().ConnectionText.GetComponent<TextMeshPro>().text = $"{selectedGameObjects[0].GetComponent<Building>().nodeConnectionsDictionary.Count}/{selectedGameObjects[0].GetComponent<Building>().maxAllowedConnections}";
                }
            }

            selectedGameObjects[0].GetComponent<Building>().isConnectedToPower = false;
            selectedGameObjects[1].GetComponent<Building>().isConnectedToPower = false;
            selectedGameObjects[0].transform.Find("Sprite").GetComponent<SpriteRenderer>().color = Color.white;
            selectedGameObjects[1].transform.Find("Sprite").GetComponent<SpriteRenderer>().color = Color.white;
            selectedGameObjects.Clear();
            selectedGameObjectsPosition.Clear();
        }
    }
}
```

Quizzing System

- Define a list of questions and answers formatted in .json
- Parse into array for code to read
- Output UI for questions, answer choices, etc.

```
public void ReadJsonFile() {
    //Debug.Log("Size of list on start " + questionsJson.Count); //
    char[] charsToTrim = { '[', ']'};
    var testString = PrepCardQuestionsJSONFile.text;
    string result = testString.Remove(0, 1);
    string result2 = result.Remove(result.Length - 1, 1);
    //Debug.Log(result);
    //Debug.Log(result2);
    string[] subs = result2.Split('{', '}');
    foreach (var sub in subs)
    {
        //Debug.Log($"Substring: {sub}");
        if (sub.Contains('"') == true) {
            //Debug.Log($"Substring: {sub}");
            string subModified = sub;
            subModified += "}";
            subModified = subModified.Insert(0, "{");
            questionsJson.Add(subModified);

        }

    }

    //Debug.Log("Size of list now " + questionsJson.Count);
    //Debug.Log(questionsJson[0]);
}
```

```
public int GenerateQuestion()
{
    var randomQuestionNumber = UnityEngine.Random.Range(0, questionsJson.Count);
    Debug.Log("Random Question Number: " + randomQuestionNumber);
    JsonUtility.FromJsonOverwrite(questionsJson[randomQuestionNumber], this);
    //Debug.Log(this.answerA);


    buttonA.onClick.RemoveAllListeners();
    buttonB.onClick.RemoveAllListeners();
    buttonC.onClick.RemoveAllListeners();
    buttonD.onClick.RemoveAllListeners();

    QuestionText.GetComponent<TMP_Text>().text = this.question;
    buttonA.GetComponentInChildren<TMP_Text>().text = this.answerA;
    buttonB.GetComponentInChildren<TMP_Text>().text = this.answerB;
    buttonC.GetComponentInChildren<TMP_Text>().text = this.answerC;
    buttonD.GetComponentInChildren<TMP_Text>().text = this.answerD;

    if (this.answerC == "" && this.answerD == "")
    {
        Color col = Color.white;
        col.a = 0;
        buttonC.GetComponent<Image>().color = col;
        buttonD.GetComponent<Image>().color = col;
        buttonC.interactable = false;
        buttonD.interactable = false;
    }

    buttonA.onClick.AddListener(delegate { CheckIfCorrect(buttonA, this.correctAnswer); });
    buttonB.onClick.AddListener(delegate { CheckIfCorrect(buttonB, this.correctAnswer); });
    buttonC.onClick.AddListener(delegate { CheckIfCorrect(buttonC, this.correctAnswer); });
    buttonD.onClick.AddListener(delegate { CheckIfCorrect(buttonD, this.correctAnswer); });

    return randomQuestionNumber;
}
```

Subsystem Validation Plan

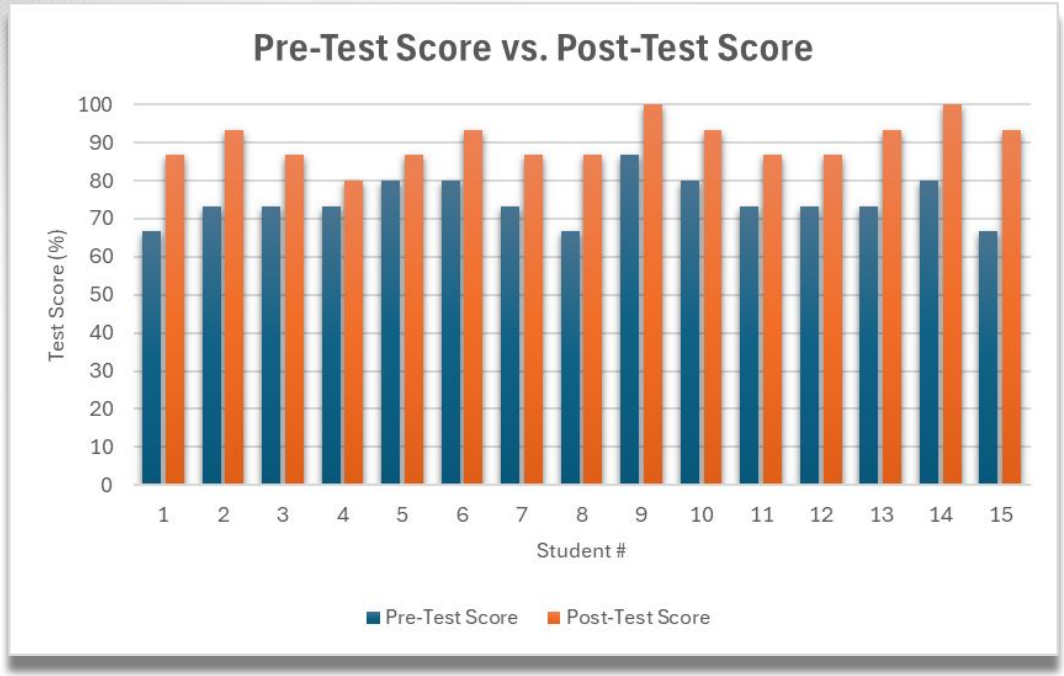
Joey Raphael

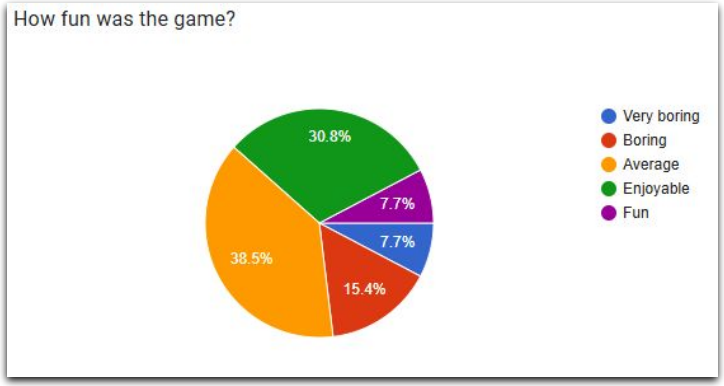
Specification	Methodology	Measured Results
Average ≥ 30 FPS throughout entire play session	Use Unity Performance profiler to track framerate over course of game.	Average FPS: 33 FPS Minimum FPS: 15 FPS Maximum FPS: 49 FPS 1% Low: 9 FPS
Input latency between UI interaction is ≤ 200 milliseconds	Implement Unity function that tracks time when screen is touched and stops when function is completed.	Average Latency: 28 ms Minimum Latency: 5 ms Maximum Latency: 137 ms
Game takes ≤ 15 seconds to load	Use stopwatch to track length of time between clicking "Load Game" button and having the game load.	Average Load Time: 8.9 s Minimum Load Time: 6.5 s Maximum Load Time: 15.8 s
Memory usage of game is ≤ 1 GB	Track memory usage using Unity Memory Profiler.	Average Memory Usage: 254 MBs Maximum Memory Usage: 437 MBs
Users can read text and visually identify UI elements from 20 inches, UI elements are not cropped or too small, and must support 16:9 aspect ratio	Utilizing test phones, have users interact with games at varying resolutions from ~20 inches. Consult with users to see if UI elements are readable.	Minimum Readable Resolution: 960 x 540 px
Users can complete instance of game.	With testing group, monitor number of students who can complete the game without external assistance. Verify that user does not encounter any bugs that prevent completion of game.	23 out of 25 students were able to successfully complete the game.
Average ≤ 3ms frame time	Use Unity Performance profiler to track frame time over course of game.	Average Frame Time: 1.7 ms

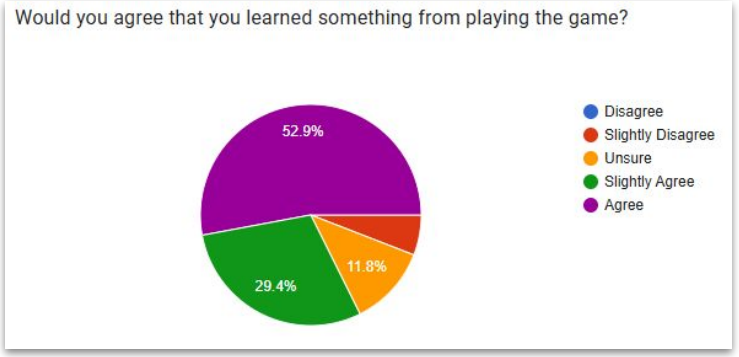
Dwight Look College of ENGINEERING
TEXAS A&M UNIVERSITY

User Test Data

Joey Raphael







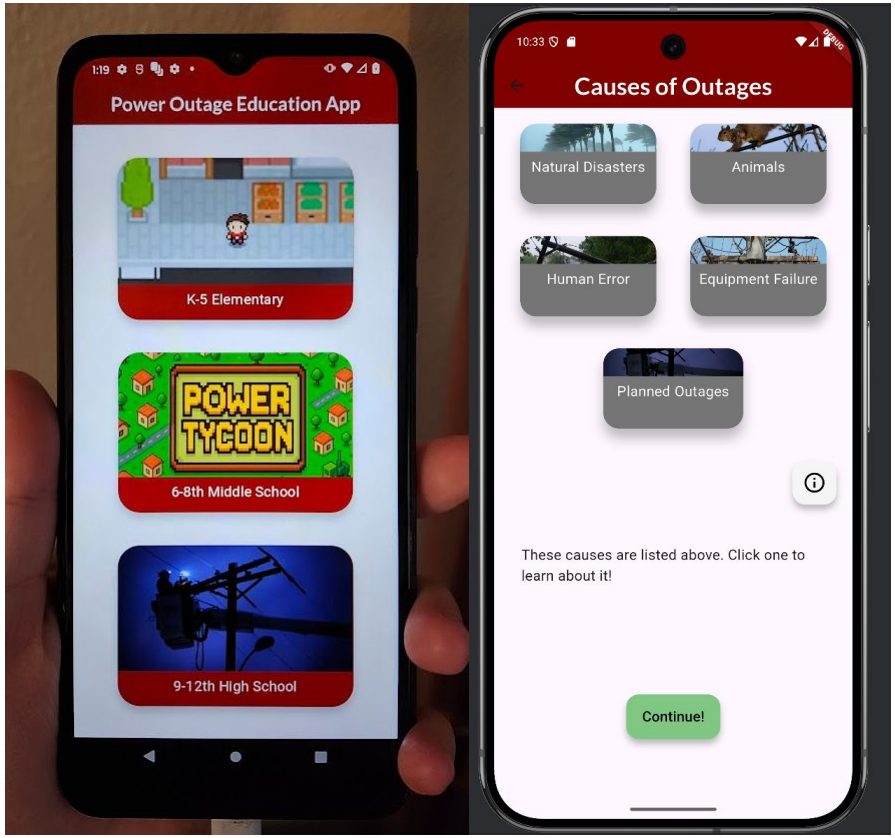
Average Pre-Test (%)	Average Post-Test (%)	Average Increase (%)	Low (%)	High (%)	Sample Size
75	90	16	80	100	25

Sample Size	Average Age	S.D of Pre-Test Score	S.D of Post-Test Score
25	13.6	5.6	5.4

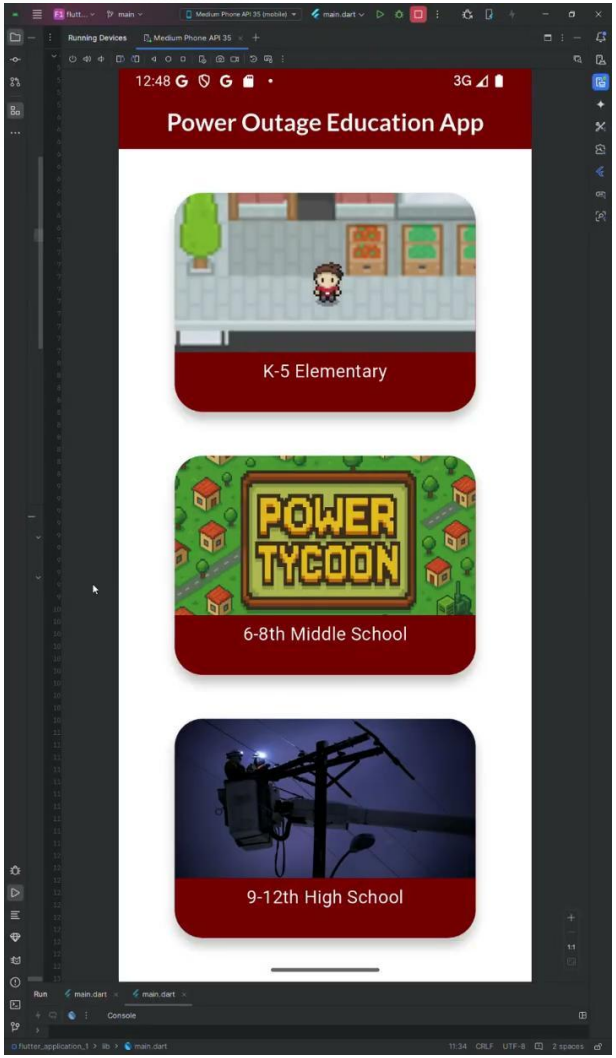
Flutter Base + High School Lesson

Jackie Villanueva

- Developed the 9th - 12 grade lesson and Flutter app page to navigate to each lesson
  - Outage Scenario
  - Causes of Outages
  - Mitigation Measures
  - During an Outage (Quiz)
- Expected outcome: teach high school students about power outages



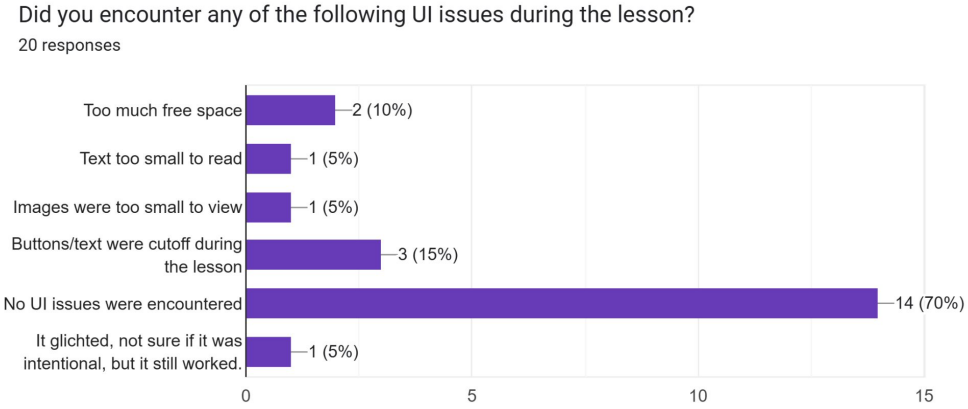
High School Lesson



Testing with High School Students

Testing with East Central High School on 11/10, with 20 students

- Average time for lesson: 6-10 minutes
- Effectiveness of lesson
  - Structure
  - Difficulty
- App Design
  - Issues with UI
  - Catch bugs



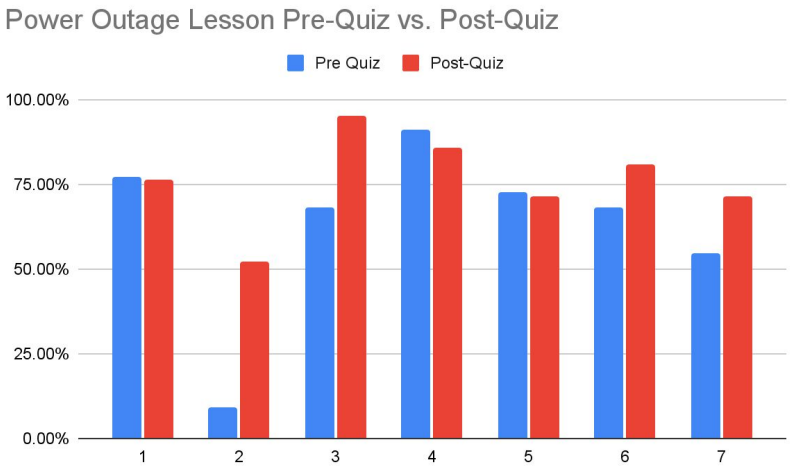
Testing with High School Students

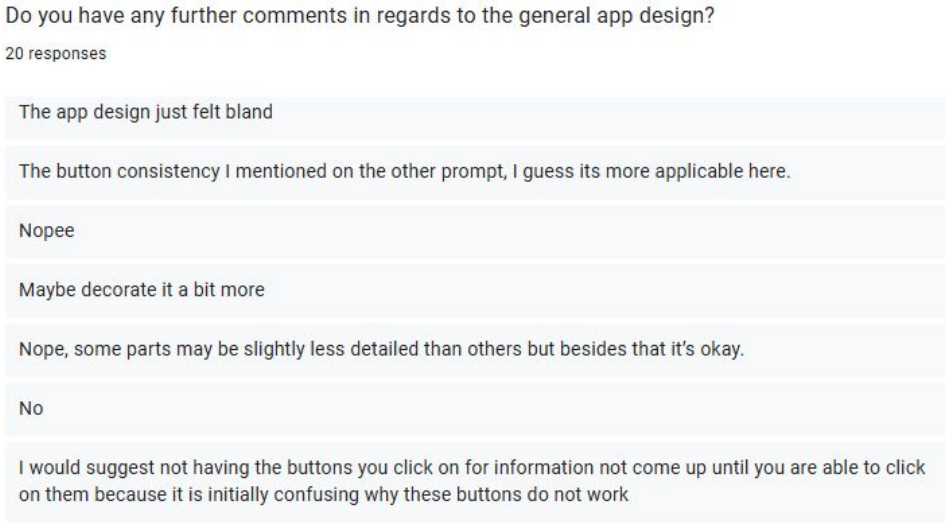
Jackie Villanueva

From testing with high school students:

- Lesson not concise and clear (decrease from pre-quiz to post-quiz)
- App was found to be bland along with other UI errors
- Consistency of the buttons on Causes of Outages page
- Info button caused confusion with some students

Feedback was applied to the final product





Code - High School Lesson

- To develop the high school lesson, Flutter was used.
  - UI software kit
  - Coding language Dart
- External Packages used:
  - Fonts
  - inAppWebView, url_launcher
- Use of Flutter's:
  - AppBar
  - Container, Column, Row
  - Text, Image
  - InkWell, Button
  - Navigator

```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await SystemChrome.setPreferredOrientations([ //preferred app orientation (portrait)
    DeviceOrientation.portraitUp,
    DeviceOrientation.portraitDown,
  ]);
  runApp(MaterialApp(title: 'Power Outage Education App', home: Home()));
}

class Home extends StatelessWidget { //Base page design,
  // Home buttons to navigate to each page
  // - DoSeum Game
  // - Power Tycoon
  // - High School Lesson
  const Home({super.key});
  @override
  Widget build(BuildContext context) {
    // Force Portrait for Home Screen
    SystemChrome.setPreferredOrientations([
      DeviceOrientation.portraitUp,
    ]);
    return Scaffold(
      appBar: AppBar(
        title: Text('Power Outage Education App', style: GoogleFonts.lato(textStyle: Te
        centerTitle: true,
        backgroundColor: Color(0xFF800000),
      ),  // AppBar
```

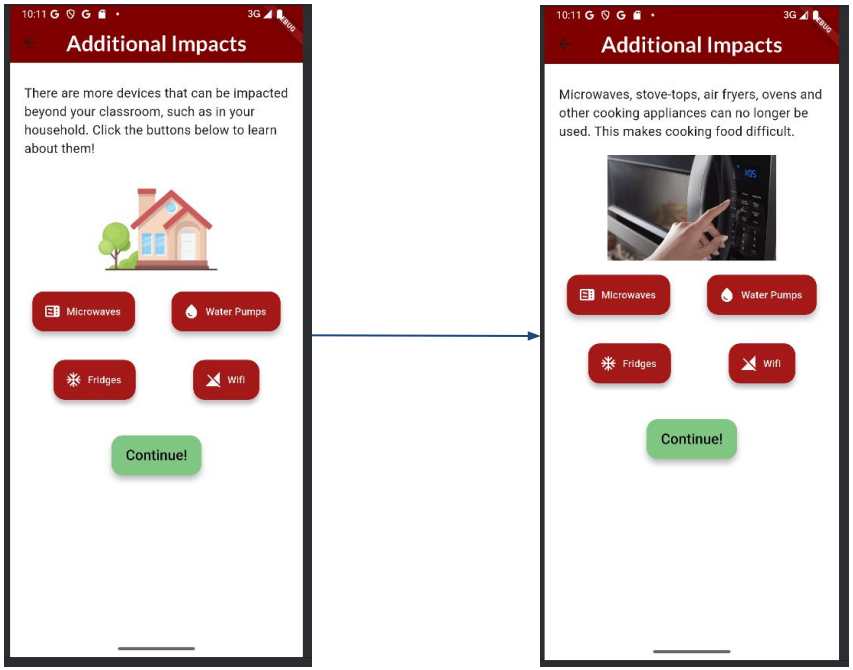
Code - High School Lesson

- The code uses Flutter's software UI kit to build the UI for the app.
- One button coded using Flutter
  - Elevated and rounded
  - InkWell for button function
  - Image to display a photo
  - Navigator to go to high school lesson
- Code repeats the use of this combination to build the UI
- Dart used to develop other functions

```
Flexible( //high school button
  flex: 5,
  child: Center(
    child: Material( //design
      color: Color(0xFF800000),
      elevation: 8,
      borderRadius: BorderRadius.circular(28),
      clipBehavior: Clip.antiAliasWithSaveLayer,
      child: InkWell( //button function -> high school page
        onTap: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => const HighSchoolPage()),
          );
        },
        child: Column( //text and image
          children: [
            Ink.image(
              image: AssetImage('assets/powerOutage.PNG'),
              height: 160,
              width: 300,
              fit: BoxFit.cover,
            ), // Ink.image
            SizedBox(height: 6),
            Text('9-12th High School',style: TextStyle(fontSize: 18, color: Colors.white)),
          ],
        ), // Column
      ), // InkWell
    ), // Material
  ), // Center
), // Flexible
```

Code - High School Lesson

- Dart used to develop other functions
  - Counters for dynamic text and images
  - Paired with StatefulWidget




```dart
Flexible(
  flex: 5,
  child: Container( //Textbox with dynamic/stateful text
    padding: const EdgeInsets.fromLTRB(10, 10, 10, 10),
    margin: const EdgeInsets.fromLTRB(10, 20, 10, 10),
    child: Text(lessonsText[lessonNum], style: TextStyle(fontSize: 18)),
  ),  // Container
),  // Flexible
Flexible(
  flex: 3,
  child: Center( //dynamic photo that changes per button
      child: Image(
        image: AssetImage(photo),
        height: 250,
        width: 250,
      )  // Image
  ),  // Center
),  // Flexible
```

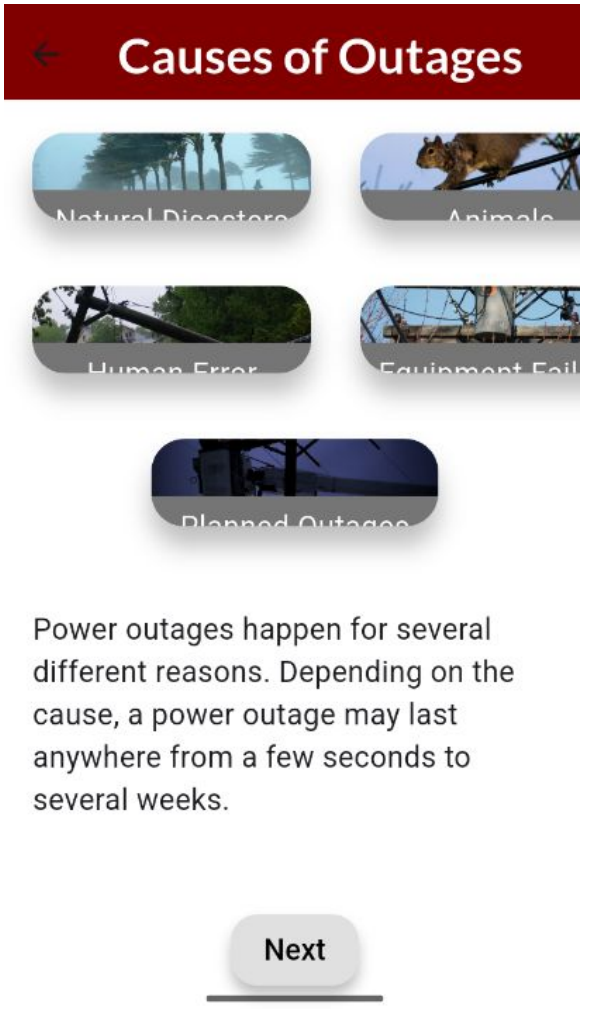
```dart
Container(
  padding: const EdgeInsets.all(10),
  margin: const EdgeInsets.fromLTRB(10, 10, 10, 10),
  child: FloatingActionButton.extended(
      label: Text('Microwaves', style: TextStyle(color: Colors.white)),
      icon: Icon(Icons.microwave, color: Colors.white),
      backgroundColor: Color(0xFFa61919),
      onPressed: () {
        setState(() { //send variables to change text/image
          lessonNum = 1;
          photo = 'assets/microwaveOff.PNG';
        });
      },
  ),  // FloatingActionButton.extended
),  // Container
```

Validation Plan - High School Lesson

Jackie Villanueva

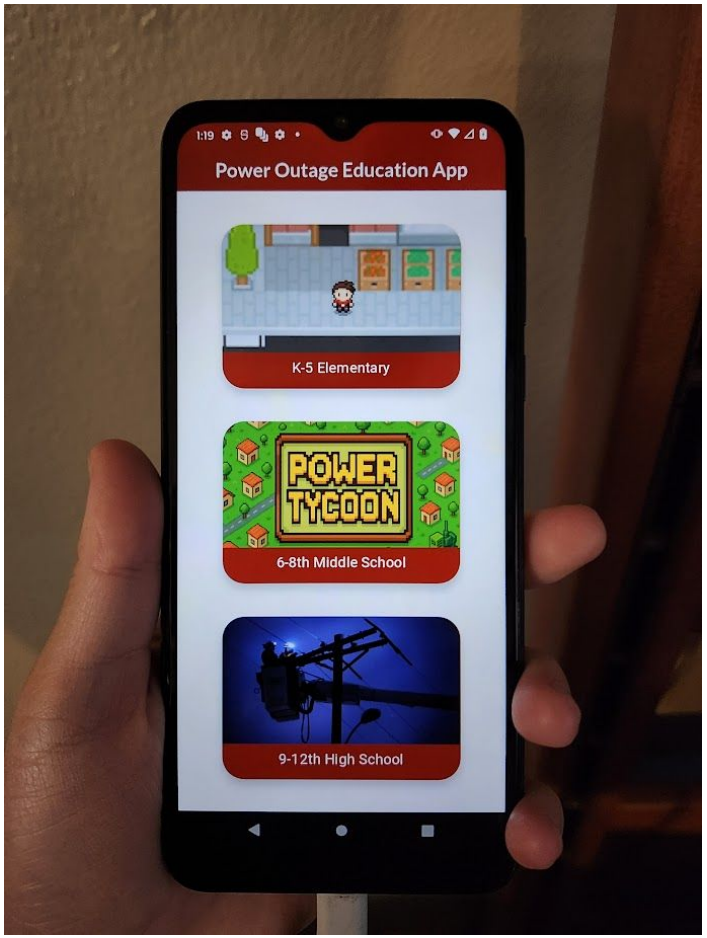
- Main issues with UI (Nonideal Case)
  - Used Flexible widget, but did not completely solve issue.
  - Solved error from testing: lesson can now be completed from any device

Requirement	Specifications	Data
App Loading Time	≤ 3s	Mean: 0.91s Min: 0.76s Max: 1.02s
Frames per second (fps)	≥ 30 fps	Mean: 54.5 fps, Min: 51 fps Max: 58 fps
Touch input delay	≤ 200ms	Range: 2-3ms
UI Consistency	Consistent on all devices	Functions as intended on most phones, UI is “squished” on small devices, but functions



Integrated System Testing

- Integrated system testing was performed on a Moto G Pure
- Testing included:
  - Running through each lesson in one instance
  - Navigating back to the main page
- Integration performed using InAppWebView

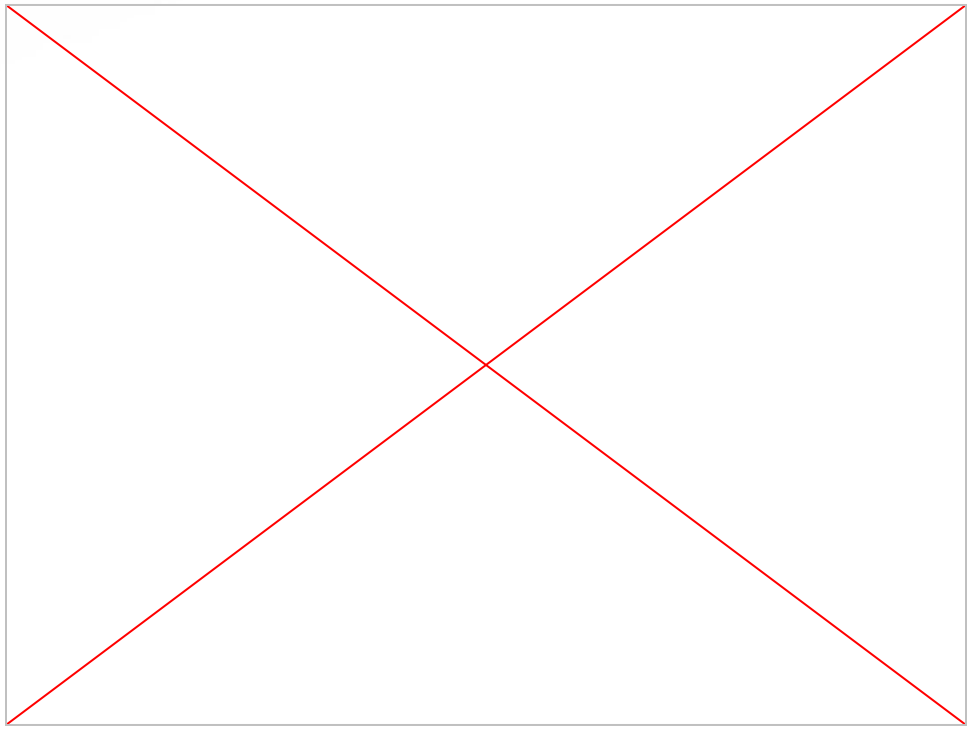


Integration - Flutter Base

- InAppWebView used for integration
  - Elementary and Middle School Lesson
- Code:
  - Checks platform (Web, IOS, or Android)
  - Extra tools for debugging, refreshing
  - Sets landscape orientation
  - Opens itch.io page

```
@override
Widget build(BuildContext context) { //set to preferred orientation: landscape
  SystemChrome.setPreferredOrientations([
    DeviceOrientation.landscapeLeft,
    DeviceOrientation.landscapeRight,
  ]);
  return Scaffold( //Layout, opens WebView page
    appBar: AppBar(title: const Text("Elementary School Game")),
    body: SafeArea(
      child: Column(
        children: <Widget>[
          Expanded(
            child: Stack(
              children: [
                InAppWebView( //Main WebView Widget
                  key: webViewKey,
                  webViewEnvironment: webViewEnvironment,
                  initialUrlRequest: URLRequest( //URL to open
                    url: WebUri("https://aidanpetro.itch.io/ecen404game"),
                  ),  // URLRequest
                  initialSettings: settings,
                  pullToRefreshController: pullToRefreshController,
                  onWebViewCreated: (controller) { //reload controller
                    webViewController = controller;
                  },
                  onLoadStart: (controller, url) { //updates URL and URLcontroller
                    setState(() {
                      this.url = url.toString();
                      urlController.text = this.url;
                    });
                  },
                  onPermissionRequest: (controller, request) async { //asks for permissions
                    return PermissionResponse(
                      resources: request.resources,
                      action: PermissionResponseAction.GRANT,
                    );  // PermissionResponse
                  },
                  shouldOverrideUrlLoading: ( //for all URLs
                      controller,
                      navigationAction,
```

Integrated App



Integrated System Results

- App successfully runs end-end on across all sections of the app

Requirement	Specifications	Measured Result
App Loading Time	≤ 3s	Mean: 2.584 s Min: 2.39 s Max: 2.92 s
Frames per second (fps)	≥ 30 fps	Mean: 48 fps Min: 28 fps Max: 55 fps
Touch input delay	≤ 100ms	Mean: 4.9 ms Min: 2.2 ms Max: 5.8 ms
UI Consistency	Consistent on all supported device resolutions.	Functions as intended on Moto G Phone. UI is cropped on small devices
Stability	App does not crash when under high memory load.	App performance remains stable when all 3 subsystems are running.
Seamless Navigation	User should be able to navigate through the 3 subsystems and back to the “hub” app.	User is able to access all necessary pages and return to the “hub” app.

System Validation Plan

Milestone #	Test	Detail	Validation Method
1	App Functionality Testing	App successfully opens and does not crash, lag, or glitch.	Boot up the app and navigate through all systems of the app.
2	Subsystem Objective Completion	Each subsystem can be played from start to end.	Run through each individual subsystem and reach the 'end.'
3	User Interface Consistency	Check that all buttons, menus, and text show up correctly and work the same way on both phones and tablets	Run the app on an Android phone and small phone. Confirm the correct functionality of each button.
4	Offline and Online Functionality	The app has different interfaces for if it is online or offline.	Run the software with and without wifi. Check if the interface correctly differs between the two scenarios.
5	Performance	The game runs at an acceptable performance: - Average of 30 FPS - Touch input delay of less than 100 ms - App loads within 3 seconds - App experiences minimal to no lag	- Load integrated app into test phone and must open within 3 seconds. - Modify code to log into console how long it takes for function to run after associated button is pressed on app. Button press to app action should not take longer than 100 ms for a response. - Monitor system resource usage using Flutter resource profiler - Plot RAM usage, CPU usage, frametimes, over time and label events such as app loading, game loading, game start, game end, etc. - Verify that usage of system resources does not exceed target device specs i.e game does not use more than 4 GBs of RAM, utilize more than 4 cores in standard 8-core CPU, average FPS is >30
6	Flutter and Unity	Test that the Unity WebGL game runs properly inside the Flutter app, responds to touch input, and displays the correct scenes, buttons, and animations without distortion or delay	Click Elementary and middle school tabs from Flutter base. Play through the individual games for both.
7	Learning Outcomes (Elementary)	Students demonstrate improved understanding of foundational concepts after playing.	Pre- and post-game quizzes, teacher feedback, observation.
8	Learning Outcomes (Middle School)	Students demonstrate improved understanding of targeted concepts after playing.	Pre- and post-game quizzes, teacher feedback, observation. - Adminster quiz before students play game, quiz will consist of questions from the game, record what questions were answered correctly or incorrectly on a per-student basis. - Adminster quiz after students play game, quiz will consist of same questions, record what questions were answered correctly or incorrectly - Determine overall accuracy of questions, identify which questions were difficult, determine "learning" by assessing if students were able answer question correctly after getting question initially wrong in pre-game quiz - Output bar graph of questions answered correctly, determine percentage of students who had increased test scores, determine average test score increase
9	Learning Outcomes (High School)	Students demonstrate improved understanding of advanced concepts after advancing through the curriculum.	Pre-content quiz, post content quiz, observation, survey response to app - Pre-quiz and post quiz consistent of the same asnswers. All students grades, on average, must improve following the lesson

System Validation Plan

Data	Status	Owner	Date of Completion
- Each game/app currently opens and does not crash when run.	Success	Jackie Villanueva	10/26/2025
- Each subsystem can be finished to completion on any device	Success	Aidan Petropoulos	10/26/2025
- Lettering is cut on small device for high school system - Buttons work as intended regardless of device	Partial Success	Joey Raphael	11/28/2025
- App continues to run with or without wifi and works when reloaded. - App expectedly cannot run Unity games without wifi, and throws an error when it cannot be run	Sucess	Aidan Petropoulos	10/26/2025
- Output graphs detailing frametime over frame, memory usage over time, CPU usage over time with key events labeled (game loading, game start, game end, app start, etc.) - Touch input latency were <100ms, averaged 4.9 ms - Game embedded in Flutter app utilized 467 MBs of RAM - Average FPS is 48 FPS - Average app loading time was 2.584 seconds	Success	Joey Raphael	10/26/2025
- Unity games take time to load, but all apps run with little to no issue on the phone. - Both systems can be completed from start to finish.	Success	Joey Raphael	10/26/2025
- All students improved post-quiz taking the lesson	Success	Aidan Petropoulos	11/10/2025
- Output accuracy data and test scores as .csv files - Process into bar graphs to compare testing accuracy among different questions, students, etc.	Success	Joey Raphael	11/10/2025
- Students, on average, improved following taking the lesson	Success	Jackie Villanueva	11/28/2025

GitHub for Power Outage App

Link: https://github.com/joeyraphael/ecen_403_flutter_app

Thank you! Any questions?