Testing Document
December 18, 2017

Delta: Introduction to Programming

Team 2

SW301/401 – Software Design Methods
Fall 2017
Date: December 18, 2017

Team Members: John Crowley
Peter Julian
Christopher Kelly
Matthew Richardson
Nick Zazula

Table of Contents

Introduction

This is the official testing document for SW301 Team 2. The testing of our system is based on the Requirements and Design document for the same program. We were able to utilize both White Box style and Black Box style testing on our project, allowing us to thoroughly check our work. We were able to ensure that our program would not crash or fail due to user inputs that may be unknown. We used edge and node testing during the White Box testing to make sure the program was running properly. And then created a small correction plan at the conclusion of this document to go over future goals, corrections, and addons to the game.

Function Testing with Black Box Style

Start Menu

Test Case Input	Expected Output	Correctness
Start clicked	Game starts	Accepted
Quit clicked	Game quits	Accepted

Home Town

Test Case Input	Expected Output	Correctness
Up arrow	Move North	Accepted
Down arrow	Move South	Accepted
Left arrow	Move West	Accepted
Right arrow	Move Right	Accepted
a	no movement	Accepted
s	no movement	Accepted
d	no movement	Accepted
w	no movement	Accepted
Collision Detection (Walls)	no movement	Accepted
ESC	Game exits	Accepted

Level 1 White box testing

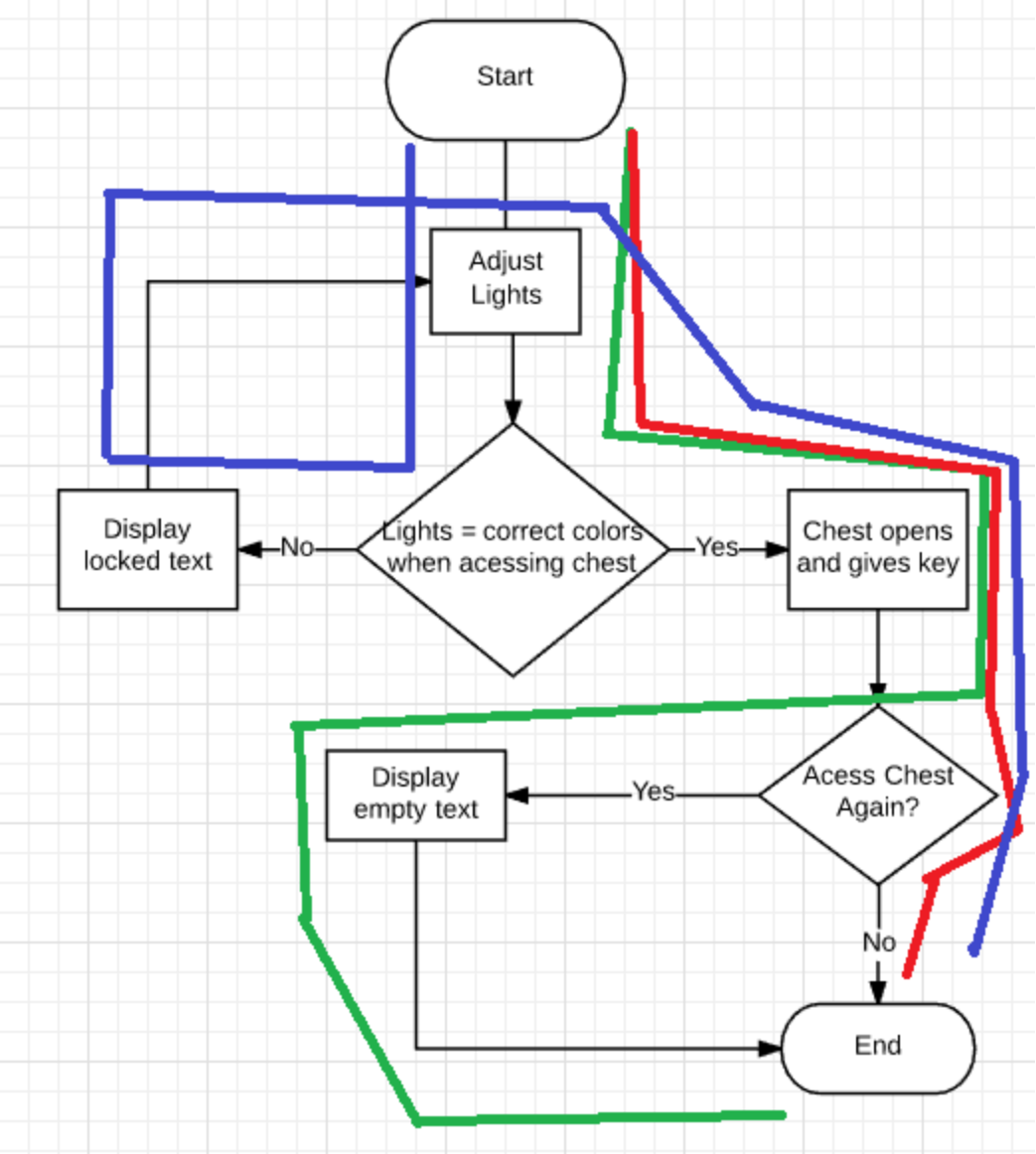
```
public class ChestInteraction : MonoBehaviour
{
   private DialogueManager dialogueManager;    // Allows editing of DialogeManager Class.
   public string[] dialogueLinesLocked;
   public string[] dialogueLinesKey;
   public string[] dialogueLinesEmpty;
   public Light blueRock, redRock, yellowRock;
   private bool isLocked;
   public bool isEmpty;

   // Use this for initialization
   void Start()
   {
      dialogueManager = FindObjectOfType<DialogueManager>();
      isLocked = true;
      isEmpty = false;
   }

   // Update is called once per frame
   void Update() //checks to see if lights are correctly colored
   {
      if (blueRock.color == Color.blue
        && redRock.color == Color.red
        && yellowRock.color == Color.yellow)
      {
         isLocked = false;
      }
      else isLocked = true;

   }

   void LockedText() // displays text for locked chest
   {
      if (!dialogueManager.dialogActive)
         // Point B
         {
```

```
            dialogueManager.dialogLines = this.dialogueLinesLocked;
            dialogueManager.currentLine = 0;
            dialogueManager.ShowDialogue();
        }
}

void KeyText() // displays text for finding the key
{
    if (!dialogueManager.dialogActive)
        // Point B
        {
            dialogueManager.dialogLines = this.dialogueLinesKey;
            dialogueManager.currentLine = 0;
            dialogueManager.ShowDialogue();
        }
}

void EmptyText() //d isplays text for empty chest
{
    if (!dialogueManager.dialogActive)
    // Point B
    {
        dialogueManager.dialogLines = this.dialogueLinesEmpty;
        dialogueManager.currentLine = 0;
        dialogueManager.ShowDialogue();
    }
}

void OnTriggerStay2D(Collider2D col)
/* Input: N/A (Runs in place of Update.  Will test once per frame).
 * Purpose: Will test for collision from gameObject Avatar (Point A).  Once true it ensures
 * that the DialogueManager is not currently active (Point  B).  If that says false (inactive
 * DialogueManager) then it will send over the dialogueLines to DialogueManager's
 * dialogueLines for displaying to the user.
 */
{
    if (col.gameObject.name == "Avatar")
    {
```

```
        if (Input.GetKeyDown(KeyCode.Return))
        // Point A
        {
            if (!isEmpty)
            {
                if (isLocked)
                {
                    LockedText();
                }
                else
                {
                    KeyText();
                    isEmpty = true;
                }
            }
            else EmptyText();
        }
    }
}


public class TownExit : MonoBehaviour {
    public ChestInteraction chest;
    private DialogueManager dialogueManager;
    public string[] dialogueLinesStuck;

    // Use this for initialization
    void Start ()
    {
        dialogueManager = FindObjectOfType<DialogueManager>();
    }

         // Update is called once per frame
         void Update () {

         }
```

```
    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name == "Avatar")
        {
            if (chest.isEmpty)
            {
                SceneManager.LoadScene("HomeTown");
            }
            else
            {
                dialogueManager.dialogLines = this.dialogueLinesStuck;
                dialogueManager.currentLine = 0;
                dialogueManager.ShowDialogue();
            }
        }
    }
}


public class LightSwitcher : MonoBehaviour {
    public new Light light;


            // Use this for initialization
            void Start () {
        light = GetComponent<Light>();
        light.color = Color.clear;
            }

            // Update is called once per frame
            void Update () {

            }

    void OnTriggerEnter2D(Collider2D col)
    /* Input: N/A (Runs in place of Update.  Will test once per frame).
     * Purpose: Will test for collision from gameObject Avatar (Point A).  Once true it ensures
     * that the DialogueManager is not currently active (Point  B).  If that says false (inactive
```

```
     * DialogueManager) then it will send over the dialogueLines to DialogueManager's
     * dialogueLines for displaying to the user.
     */
    {
        if (col.gameObject.name == "Avatar")
        {
            LightChanger();
        }

    }

    void LightChanger()
    {
        //if (Input.GetKeyDown(KeyCode.Return))
        // Point A
        {
            if (light.color == Color.clear)
            {
                light.color = Color.yellow;
            }
            if (light.color == Color.yellow)
            {
                light.color = Color.blue;
            }
            else if (light.color == Color.blue)
            {
                light.color = Color.red;
            }
            else if (light.color == Color.red)
            {
                light.color = Color.yellow;
            }
        }
    }

}
```



This is a set of three different test

Test A_1 (Red) : Lights: true, AcessEmptyChest: false, end
Test A_2 (Blue) : Lights: false, Lights2: true, AcessEmptyChest: false, end
Test A_3 (Green) : Lights: true, AcessEmptyChest: true, AcessEmptyChest2: false, end

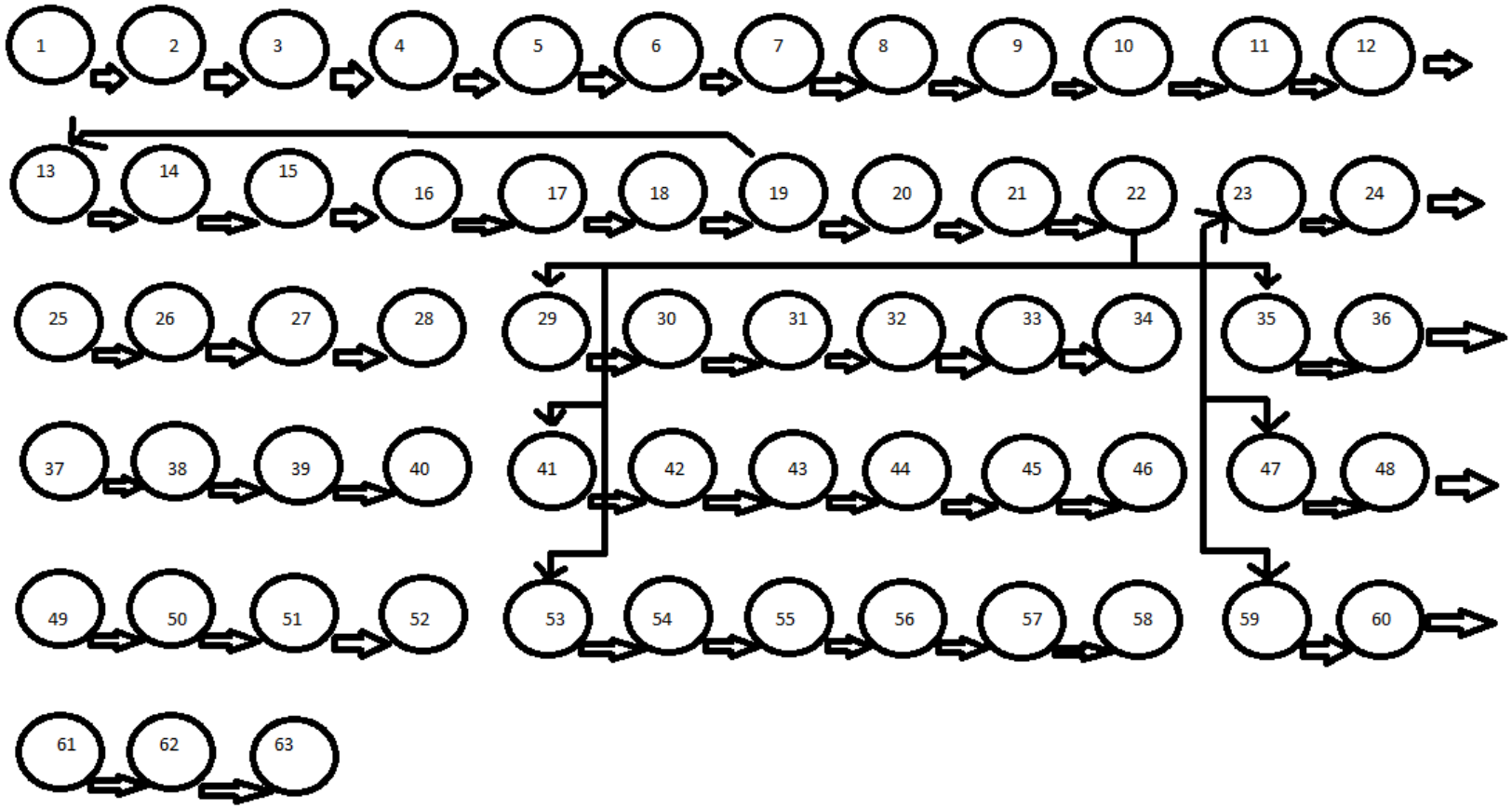
TestB_1(not shown): Changed lights after accessing chest to see that chest is still open after key collection. Results are that chest is open after key collection regardless of light configuration

Level 2 White Box Testing

ArrayController Testing - Node Testing

```
1.  public class ArrayController : MonoBehaviour
2.  {
3.      //public bool inArea = false;
4.      private ArrayManager arrayManager;
5.      private AvatarController avatarController;
6.      private SpriteRenderer spriteRenderer;
7.      void Start()
8.      {
9.          avatarController = FindObjectOfType<AvatarController>();
10.         arrayManager = FindObjectOfType<ArrayManager>();
11.         spriteRenderer = FindObjectOfType<SpriteRenderer>();
12.     }
13.     private void OnTriggerStay2D(Collider2D collision)
14.     {
15.         if ((Input.GetKey(KeyCode.P)) && !arrayManager.unlockBox)
16.         {
17.             MoveBox();
18.         }
19.     }
20.     private void MoveBox()
21.     {
22.         arrayManager.arrayMoved[5] = true;
23.         if (arrayManager.arrayMoved[0])
24.         {
25.             arrayManager.unlockBox = true;
26.             transform.position = new Vector2(-2.96f, -1.64f);
27.             print(transform.position.x);
28.         }
29.         else if (arrayManager.arrayMoved[1])
30.         {
31.             transform.position = new Vector2(-5.57f, -5.25f);
32.             print(transform.position.x);
33.             arrayManager.arrayMoved[0] = true;
34.         }
```

```
35.       else if (arrayManager.arrayMoved[2])
36.       {
37.          transform.position = new Vector2(-8.1f, -5.25f);
38.          print(transform.position.x);
39.          arrayManager.arrayMoved[1] = true;
40.       }
41.       else if (arrayManager.arrayMoved[3])
42.       {
43.          transform.position = new Vector2(-2.95f, -5.25f);
44.          print(transform.position.x);
45.          arrayManager.arrayMoved[2] = true;
46.       }
47.       else if (arrayManager.arrayMoved[4])
48.       {
49.          transform.position = new Vector2(-8.1f, -1.64f);
50.          print(transform.position.x);
51.          arrayManager.arrayMoved[3] = true;
52.       }
53.       else if (arrayManager.arrayMoved[5])
54.       {
55.          transform.position = new Vector2(-5.6f, -1.64f);
56.          print(transform.position.x);
57.          arrayManager.arrayMoved[4] = true;
58.       }
59.       else
60.       {
61.       }
62.    }
63. }
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63

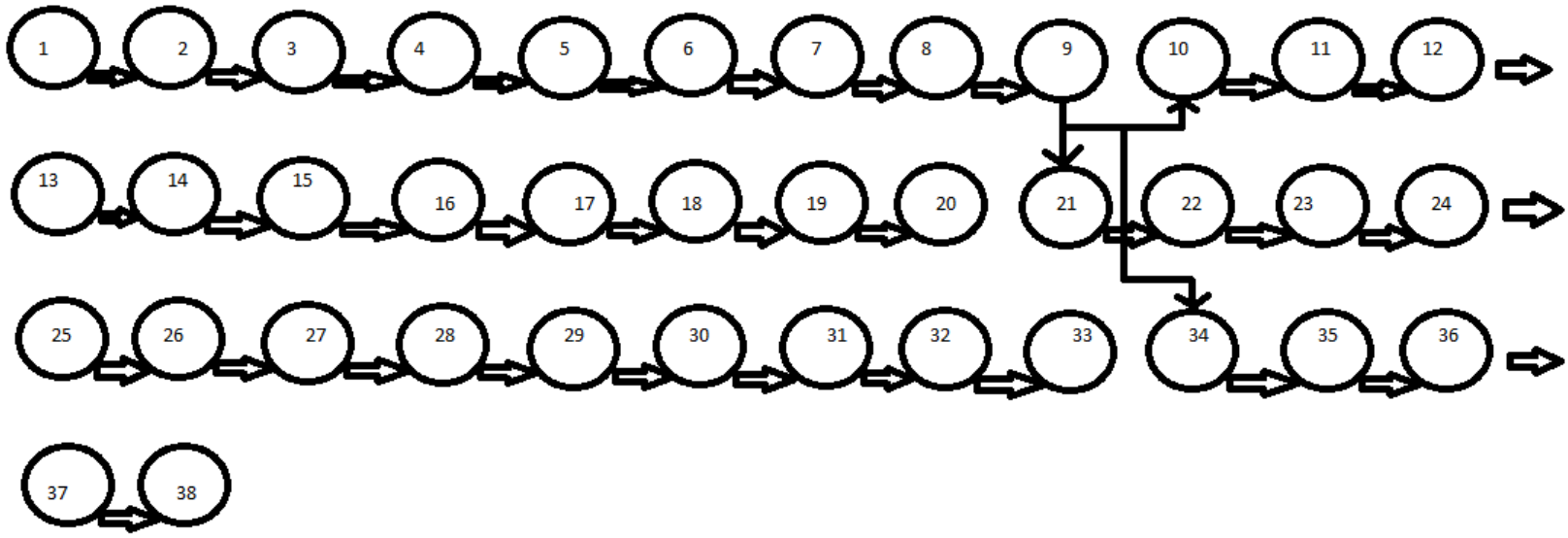
ArrayManager Testing - Node Testing

```
1. public class ArrayManager : MonoBehaviour
2. {
3.     public bool[] arrayMoved = new bool[6] { false, false, false, false, false, true };
4.     public bool unlockBox = false;
5. }
```

Exit Testing - Node Testing

```
1.  public class Exit : MonoBehaviour
2.  {
3.      private ArrayManager arrayManager;
4.      private AvatarController avatarController;
5.      private void Start()
6.      {
7.          arrayManager = FindObjectOfType<ArrayManager>();
8.          avatarController = FindObjectOfType<AvatarController>();
9.      }
10.     public void OnTriggerEnter2D(Collider2D collision)
11.     {
12.         avatarController.inArea = true;
13.         if(Input.GetKey(KeyCode.O))
14.         {
15.             if (arrayManager.unlockBox)
16.             {
17.                 avatarController.keyAccess = true;
18.             }
19.         }
20.     }
21.     public void OnTriggerStay2D(Collider2D collision)
22.     {
23.         if(avatarController.inArea)
24.         {
25.             if (Input.GetKey(KeyCode.O))
26.             {
27.                 if (arrayManager.unlockBox)
28.                 {
29.                     avatarController.keyAccess = true;
30.                 }
31.             }
32.         }
33.     }
34.     public void OnTriggerExit2D(Collider2D collision)
35.     {
36.         avatarController.inArea = false;
```

```
37.     }
38. }
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38

Level 3 White Box Testing

BoxController Class

```
1. public class BoxController : MonoBehaviour
2. {
3.     public int boxNumber;
4.     private BoxManager boxManager;
5.     private AvatarController avatarController;
6.     private SpriteRenderer spriteRenderer;
7.     void Start ()
8.     {
9.         avatarController = FindObjectOfType<AvatarController>();
10.        boxManager = FindObjectOfType<BoxManager>();
11.        spriteRenderer = FindObjectOfType<SpriteRenderer>();
12.    }
13.    void Update ()
14.    {
15.        // Do Nothing.
16.    }
17.    private void OnTriggerStay2D(Collider2D collision)
18.    {
19.        if(Input.GetKey(KeyCode.P))
20.        {
21.            MoveBox(this.boxNumber);
22.        }
23.        else if(Input.GetKey(KeyCode.O))
24.        {
25.            ReturnBox(this.boxNumber);
26.        }
27.    }
28.    private void MoveBox(int boxNumber)
29.    {
30.        boxManager.boxMoved[5] = true;
31.        if (boxManager.boxMoved[0])
32.        {
33.            transform.position = new Vector2(4.79f, 2.05f);
34.            print(transform.position.x);
```

```
35.         boxManager.boxMoved[0] = false;
36.         spriteRenderer.sortingOrder = boxManager.layerOrder[0];
37.     }
38.     else if(boxManager.boxMoved[1])
39.     {
40.         transform.position = new Vector2(4.79f, 0.39f);
41.         print(transform.position.x);
42.         boxManager.boxMoved[0] = true;
43.         spriteRenderer.sortingOrder = boxManager.layerOrder[1];
44.     }
45.     else if (boxManager.boxMoved[2])
46.     {
47.         transform.position = new Vector2(4.79f, -1.24f);
48.         print(transform.position.x);
49.         boxManager.boxMoved[1] = true;
50.         spriteRenderer.sortingOrder = boxManager.layerOrder[2];
51.     }
52.     else if (boxManager.boxMoved[3])
53.     {
54.         transform.position = new Vector2(4.79f, -2.82f);
55.         print(transform.position.x);
56.         boxManager.boxMoved[2] = true;
57.         spriteRenderer.sortingOrder = boxManager.layerOrder[3];
58.     }
59.     else if (boxManager.boxMoved[4])
60.     {
61.         transform.position = new Vector2(4.79f, -4.45f);
62.         print(transform.position.x);
63.         boxManager.boxMoved[3] = true;
64.         spriteRenderer.sortingOrder = boxManager.layerOrder[4];
65.     }
66.     else if (boxManager.boxMoved[5])
67.     {
68.         transform.position = new Vector2(4.79f, -6f);
69.         print(transform.position.x);
70.         boxManager.boxMoved[4] = true;
71.         spriteRenderer.sortingOrder = boxManager.layerOrder[5];
72.     }
```

```
73.       else
74.       {
75.          // Do Nothing.
76.       }
77.    }
78.    private void ReturnBox(int boxNumber)
79.    {
80.       boxManager.boxMovedBack[5] = true;
81.       if (boxManager.boxMovedBack[0])
82.       {
83.          avatarController.keyAccess = true;
84.                transform.position = new Vector2(-2.96f, -1.64f);
85.          print(transform.position.x);
86.          boxManager.boxMovedBack[0] = true;
87.       }
88.       else if (boxManager.boxMovedBack[1])
89.       {
90.          transform.position = new Vector2(-5.57f, -5.25f);
91.          print(transform.position.x);
92.          boxManager.boxMovedBack[0] = true;
93.       }
94.       else if (boxManager.boxMovedBack[2])
95.       {
96.          transform.position = new Vector2(-8.1f, -5.25f);
97.          print(transform.position.x);
98.          boxManager.boxMovedBack[1] = true;
99.       }
100.       else if (boxManager.boxMovedBack[3])
101.       {
102.          transform.position = new Vector2(-2.95f, -5.25f);
103.          print(transform.position.x);
104.          boxManager.boxMovedBack[2] = true;
105.       }
106.       else if (boxManager.boxMovedBack[4])
107.       {
108.          transform.position = new Vector2(-8.1f, -1.64f);
109.          print(transform.position.x);
110.          boxManager.boxMovedBack[3] = true;
```

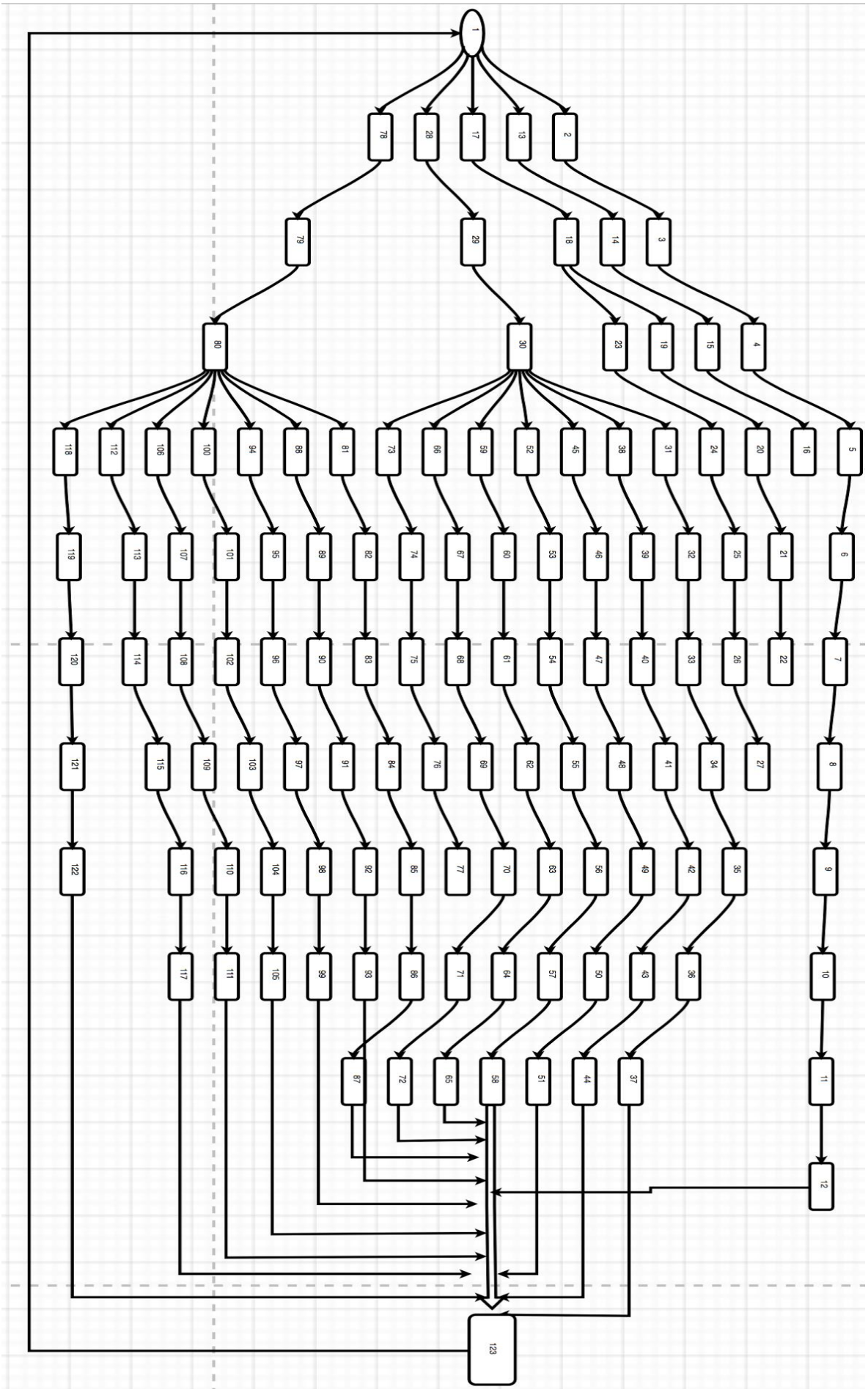
```
111.     }
112.      else if (boxManager.boxMovedBack[5])
113.      {
114.          transform.position = new Vector2(-5.6f, -1.64f);
115.         print(transform.position.x);
116.         boxManager.boxMovedBack[4] = true;
117.      }
118.      else
119.      {
120.         // Do Nothing.
121.      }
122.   }
123.}
```

Node Test Cases – Box Controller

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 123
- 1, 2, 3, 4, 5, 6, 13, 14, 15, 16, 123
- 1, 2, 3, 4, 5, 6, 17, 18, 19, 20, 21, 22, 123
- 1, 2, 3, 4, 5, 6, 17, 18, 23, 24, 25, 26, 27, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 38, 39, 40, 41, 42, 43, 44, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30,45, 46, 47, 48, 49, 50 ,51, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 52, 53, 54, 55, 56, 57, 58, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 59, 60, 61, 62, 63, 64, 65, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 66, 67, 68, 69, 70, 71, 72, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 73, 74, 75, 76, 77, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 88, 89, 90, 91, 92, 93, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 94, 95, 96, 97, 98, 99, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 100, 101, 102, 103, 104, 105, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 106, 107, 108, 109, 110, 111, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 112, 113, 114, 115, 116, 117, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 118, 119, 120, 121, 122, 123

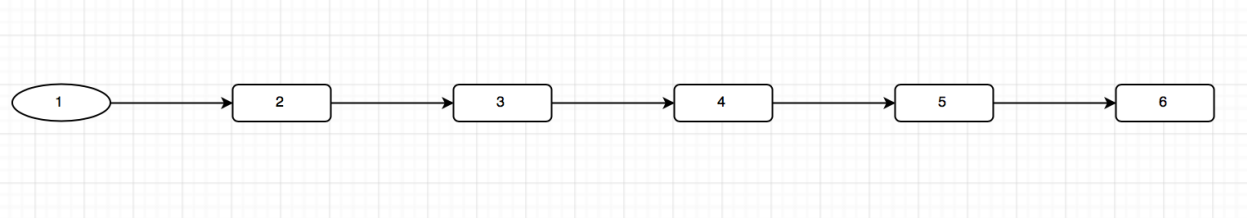
Edge Test Cases – Box Controller

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 123
- 1, 2, 3, 4, 5, 6, 13, 14, 15, 16, 123
- 1, 2, 3, 4, 5, 6, 17, 18, 19, 20, 21, 22, 123
- 1, 2, 3, 4, 5, 6, 17, 18, 23, 24, 25, 26, 27, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 38, 39, 40, 41, 42, 43, 44, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30,45, 46, 47, 48, 49, 50 ,51, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 52, 53, 54, 55, 56, 57, 58, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 59, 60, 61, 62, 63, 64, 65, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 66, 67, 68, 69, 70, 71, 72, 123
- 1, 2, 3, 4, 5, 6, 28, 29, 30, 73, 74, 75, 76, 77, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 88, 89, 90, 91, 92, 93, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 94, 95, 96, 97, 98, 99, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 100, 101, 102, 103, 104, 105, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 106, 107, 108, 109, 110, 111, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 112, 113, 114, 115, 116, 117, 123
- 1, 2, 3, 4, 5, 6, 28, 78, 79, 80, 118, 119, 120, 121, 122, 123


1
78
28
17
13
2
79
29
18
14
3
80
30
23
19
15
4
118
112
106
100
94
88
81
73
66
59
52
45
38
31
24
20
16
5
119
113
107
101
95
89
82
74
67
60
53
46
39
32
25
21
6
120
114
108
102
96
90
83
75
68
61
54
47
40
33
26
22
7
121
115
109
103
97
91
84
76
69
62
55
48
41
34
27
8
122
116
110
104
98
92
85
77
70
63
56
49
42
35
9
117
111
105
99
93
86
71
64
57
50
43
36
10
87
72
65
58
51
44
37
11
12
123

BoxManager Class

```
1. public class BoxManager : MonoBehaviour
2. {
3.    public bool[] boxMoved = new bool[6] {false,false,false, false, false,
  true };
4.          public bool[] boxMovedBack = new bool[6] { false, false, false, false,
  false, true };
5.    public int[] layerOrder = new int[6] { 1, 2, 3, 4, 5, 6 };
6. }
```



Node Test Cases – Box Manager

1, 2, 3, 4, 5, 6

Edge Test Cases – Box Manager

1, 2, 3, 4, 5, 6

Correction Plan

Our program ran without error for all of its tests. The only level with slight error would be the stacks level. It sometimes allows the user to pop items below the top most item. However, the level will not let the user out until all items have been popped from the stack. For future changes of this program we will look further into this issue. We also plan to complete the tree level which will focus on comments. We are very proud of the product we have created and think it can be a true asset to the community in teaching new programmer's logic behind programming. If a student can learn the logic of programming through this game and then learn the syntax in future lessons we know this will guarantee strong and effective programmers.