Cornell University

SYSEN 5900: Systems Engineering Design Project Report

Project: Engineering to Save the World

Brighid Meredith (jaw465), Ryan Lo (rjl296), Yong Ha (wh346)

Systems Engineering, M.Eng Fall 2017

Contents

Introduction 2
Log Line 2
Customer Value Proposition 2
Context Diagram 3
Annotated Concept Sketch 3
Original Structural Concept Sketch 4
Updated Structural Concept Sketch 5
Updated Functional Concept Sketch 6
Platform 6
Event System 19
Introduction 19
Description 20
Functional Decomposition of Event System 20
Activity Diagram of Event System 22
Activity Diagram of Reward System (Current State) 23
Activity Diagram of Reward System (Planned Expansion) 24
Sequence Diagram Event System 25
Drills 26
Description of Drill Engine 26
New Drills 26
Sequence Diagram for Reliability and Cost Drill 28
Sequence Diagram for User Interaction Diagram Drill 29
Sequence Diagram for Assignment Drills 30
Script Modifications to Main Game 31
GameControllerScript 31
Mini Games 34
Mastermind 34
Guessing Game 37
References 38

Introduction

The following is the Project Report for Engineering to Save the World (The System, E2SW or System). The Project is the brain child of Professor David R. Schneider who saw the value in developing an educational game that would teach the complexities and discipline and knowledge of System Engineering to users as young as students in high schools. The project began in fall 2016 and is currently still in development as the writing of this paper (Fall 2017).

Log Line

Engineering to Save the World is an accessible interactive game system that teaches the player System Engineering concepts and offers a simulated, reliable and realistic, experience as a System Engineer where the player deals with the management of complex projects and dealing with decisions that involve the tradeoffs of resources, time and priority.

Customer Value Proposition

Engineering to Save the World (the System) is designed to educate and introduce System Engineering concepts to students ranging from high school to undergraduate studies. To be accessible, the System will be designed and tested for a Windows PC environment for optimum playing performance.

The prime educational principle taught by the System will be to expose the player to a realistic System Engineering experience. The System will do this by simulating the management of a complex project for a client and tradeoff analysis and decision making between System Engineering Resources of labor, time and budget. A turn base rule system will be used to represent time movement.

The System will start the user with a predetermined set of budget and labor. To simulate the realistic encounter with clients, the System will feature a “win” condition where the objective is to meet the client’s expectation via performance metrics. To simulate the ambiguities dealing with clients, the client’s performance metrics will be hidden, and it will be up to the player to strategize on how to uncover the true client metrics. The user will play Mini Games to guess the client’s performance metrics, and the System will also feature “testing.”

The user will affect their metrics by selecting nodes. Nodes represent tasks or objectives with their own contribution to the performance metrics of the project. To activate a Node, players will spend labor and funds. To represent the interconnectedness of tasks within a project, some nodes will remain hidden until another node(s) are activated. To further increase the educational value of the system and to provide more resources to the player, a tutorial feature will be available, and an Event System will challenge the user to complete drills, where rewards and penalties affect the users performance metrics.

Context Diagram

The following Context Diagram defines the System's boundaries as well as the environment, actors and stakeholders that interacts with the System as a whole.

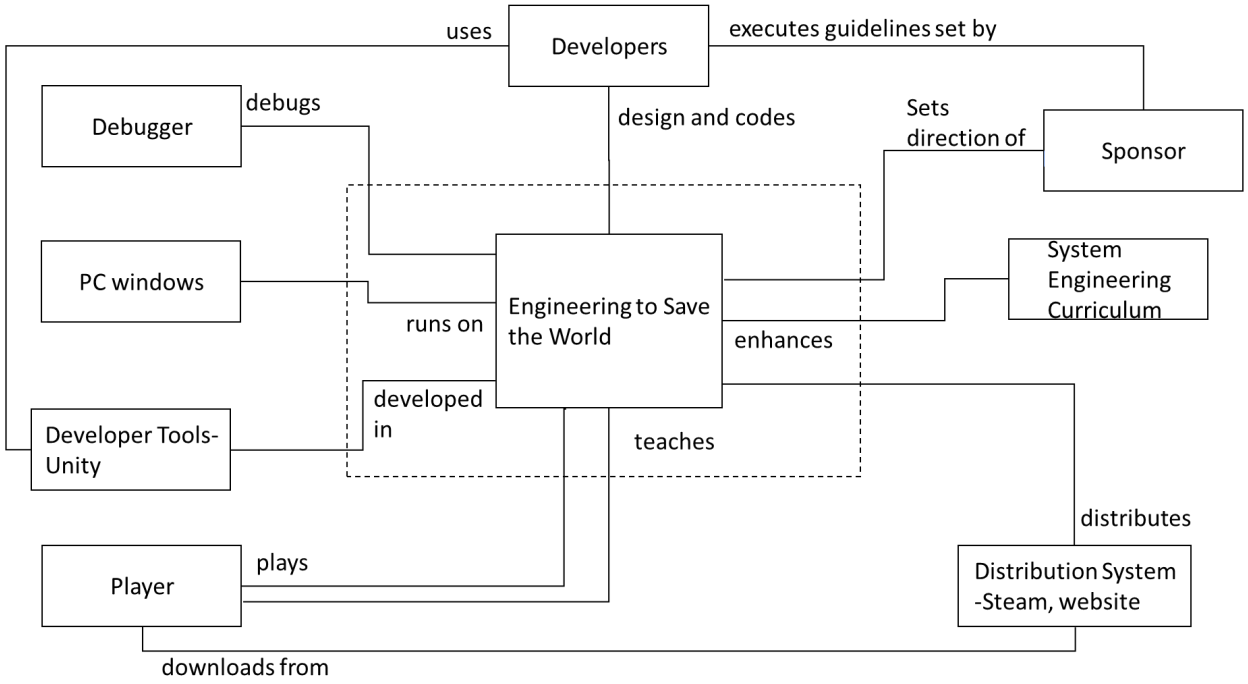


Figure 1: Context Diagram of the System

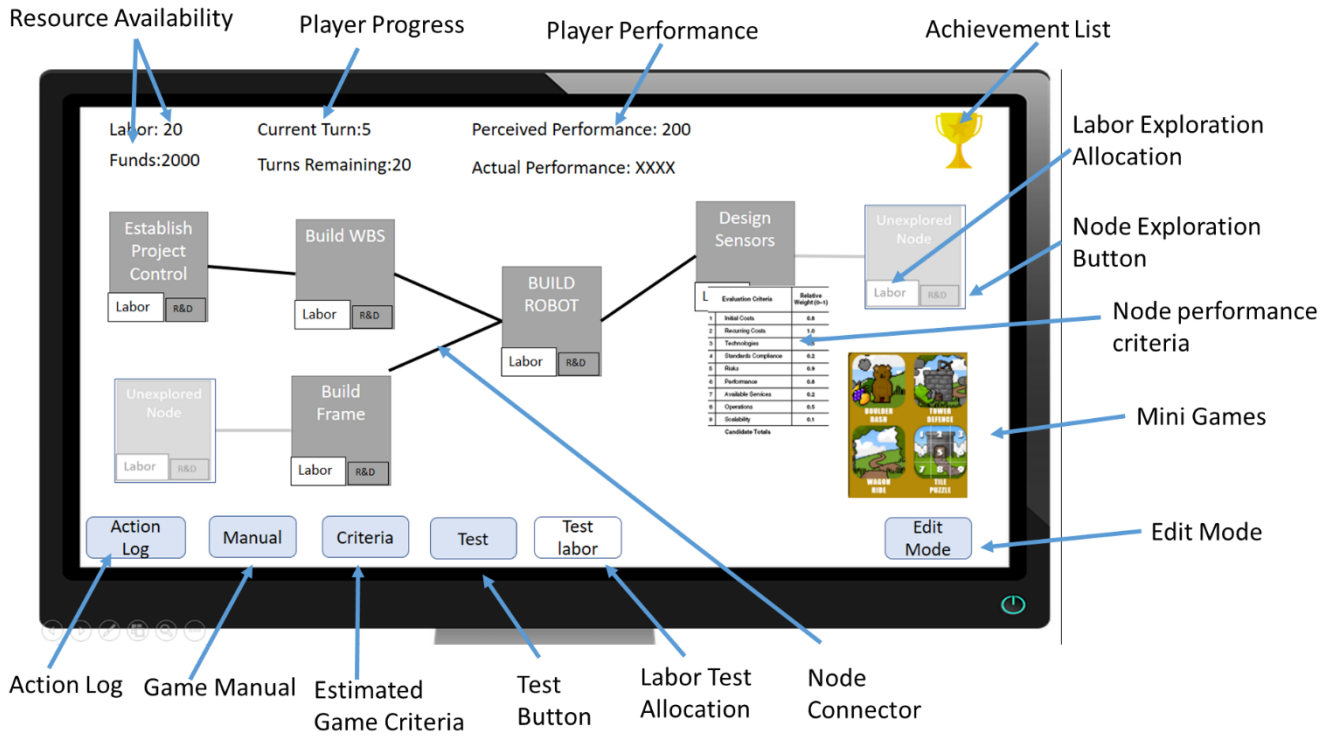
Annotated Concept Sketch

From the CVP and Log Line the team developed an initial concept sketch. In order to guide design and solicit better feedback from our users a functional and structural version were made.

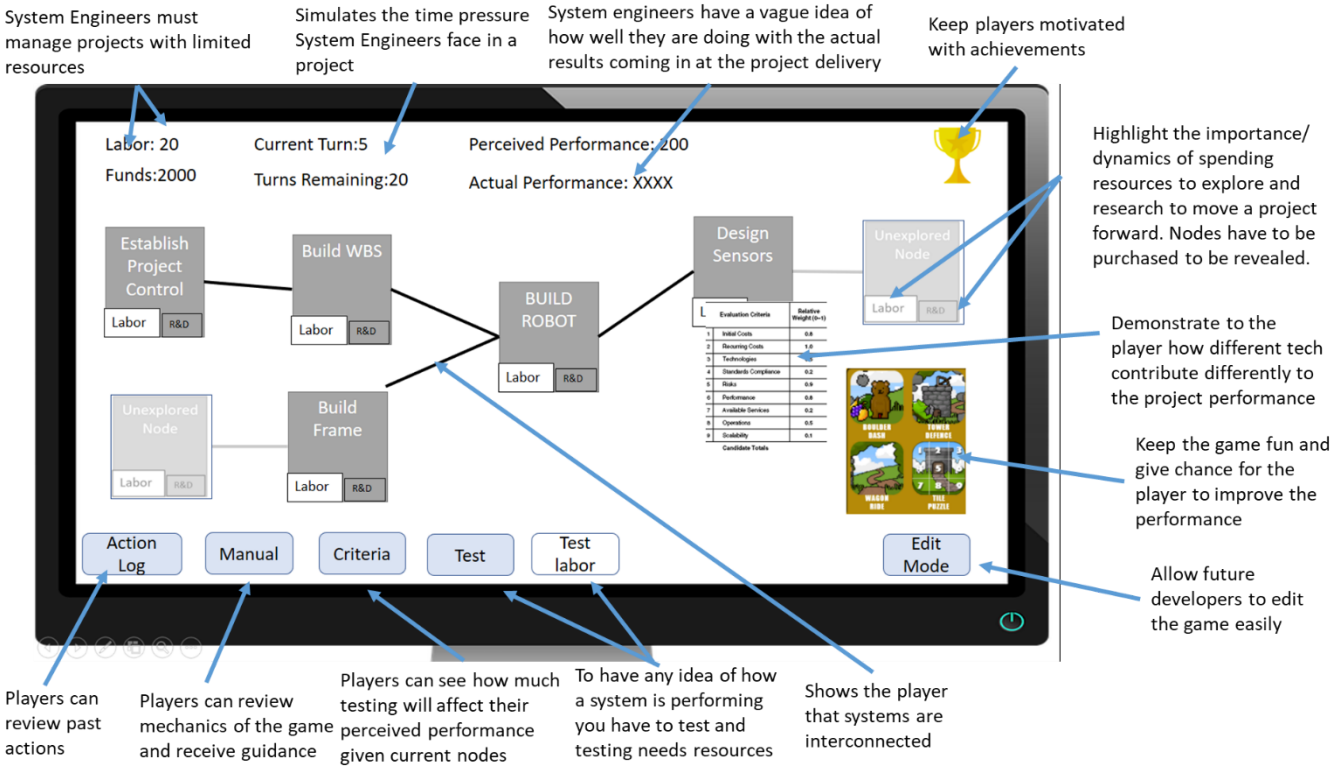
The benefit of Structural Annotated Concept Sketch is that it displays the system as a series of functions and parts that will meet the needs of the user. However, it tends to be extremely specific and can fail to speak directly to the user.

The functional Annotated Concept sketch explains the system diagrammatically through the user needs. It shows what part of the system will address the user requirements. It can be very generic and is best used to help develop more specific functions.

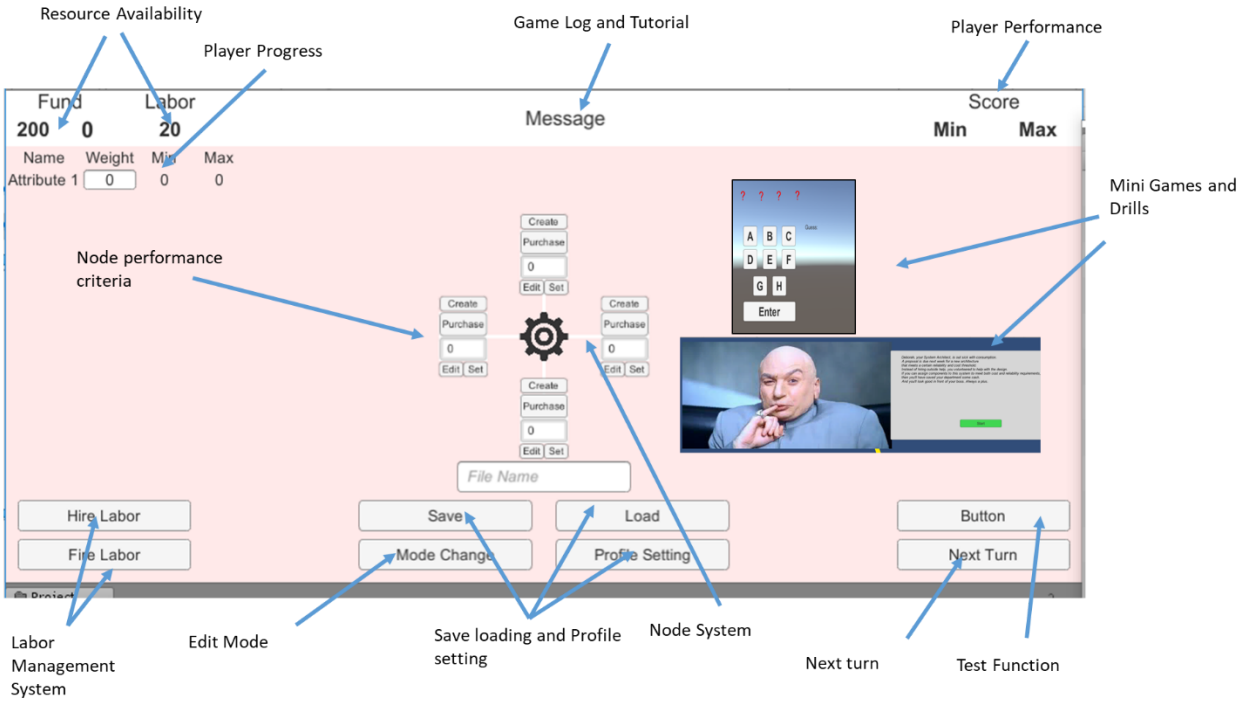
Original Structural Concept Sketch (Allen Liu, 2017)



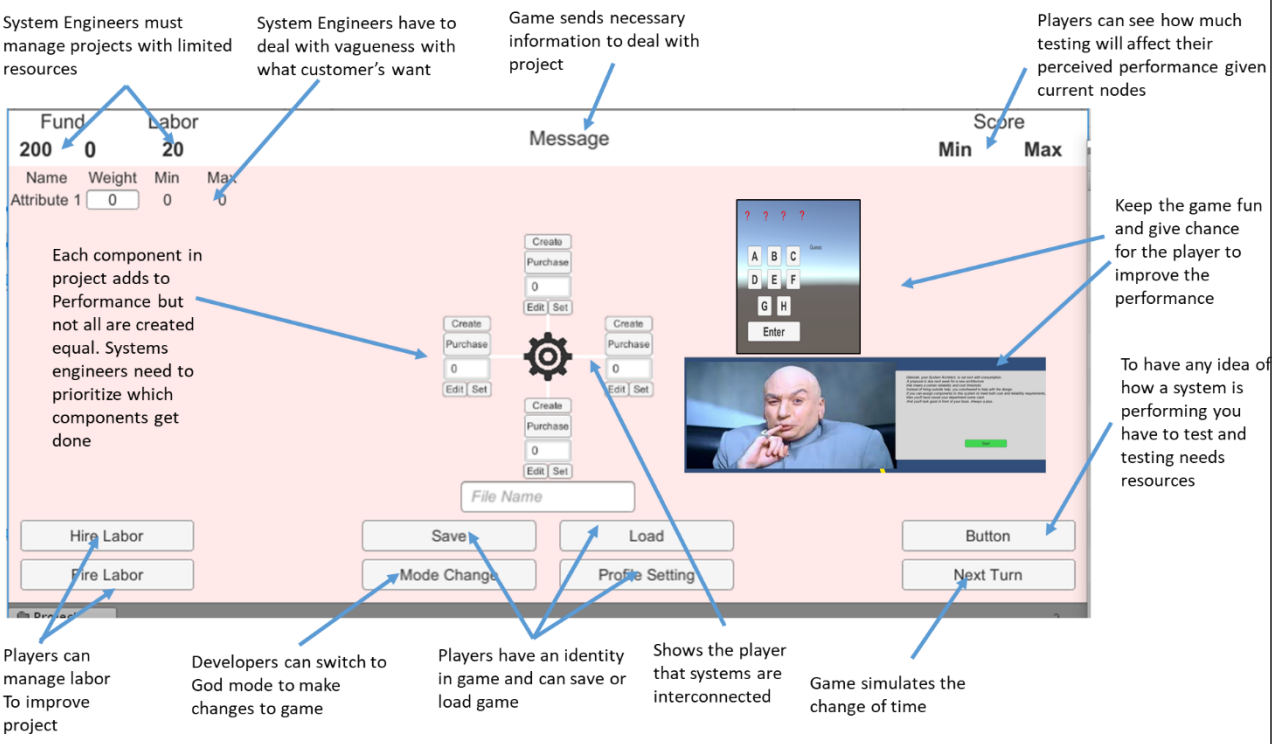
Original Functional Concept Sketch (Allen Liu, 2017)



Updated Structural Concept Sketch



Updated Functional Concept Sketch



Platform

Engineering to Save the World is required a platform for create and play the game. The meaning of platform is not just playground of the game. Since the Game design as customizing and playing by player, the game should be very flexible and accessible. For fit on those features, the generalize and customize game is necessary.

Game Controller

Initial Game

Upon initialization, the game must auto create and populate the default settings and let user play immediately. In order to initialize the game, Game Controller must have full access and secure all of the data from the game playing. Therefore, the Game Controller contains live updated node list, fund, fund per turn, and labor; all the data except the node list is saved in the format of ProfileInfo.

```
public static GameControllerScript InstanceGameCornoller;
public int initialPos;
public int nodeTrim;
public Text fundText;
public Text fundTurnText;
public Text laborText;
public GameObject node;
public GameObject centerNode;
public GameObject nodeConnection;
public GameObject labor;
public List<NodeInfo> nodeList = new List<NodeInfo> ();
public List<string> nodeNameList = new List<string> ();
public PlayerInfo playerInfo;

private NodeInfo currentNodeInfo;
private Transform nodeParentGroup;
private Transform laborParentGroup;
private GameObject centerNewNode;
private GameObject newNode;
private GameObject newConnection;
private GameObject newLabor;
```

Figure 2: Example of GameControllerScript Global Variables

Pre-generating Node

Pre-generation of the node is only for the initializing the game. Before player starts a game, the player may not have preset game. In this situation, player must customize the game. Pre-generating Node can occur in one of two ways: generating the first 4 nodes or loading generation.

First 4 nodes represent the 4 different ways that player can spread the node which mean 4 different story lines. Since the game is 2D and recognize only grid position, it has to be restricted by only 4 stories.

Loading methods only active when size of preset nodeList is bigger than 1. Loading methods call general generating node method.

General Generating Node

Since the C# instruction for Unity3D shows “GameObject.Find” (Unity, 2017) function as unstable, developers should avoid relying upon the function in scene updates. The function will only return “Active” Game Objects, and the function may return the wrong Game Object if multiple Game Objects exist with the same name. Therefore, the developers invested in a recursive loading function.

The connection in the node tree system has to include two game objects inside which mean the variable itself is a Game Object, not a certain kinds of fixed variable type. If developer uses the “GameObject.Find” function on the generating node methods and for connection, one can recognize the non-active game object cannot be find and certain bug keep occurring on the finding method.

To solve the problem of “GameObject.Find”, developer has to not using “GameObject.Find” method at all. Therefore, developer has to use recursion method.

```
public GameObject generateNode(GameObject _node, bool isShowAll)
```

The generateNode method is keep returning the last node’s game object, so that the function adds all connection at the end.

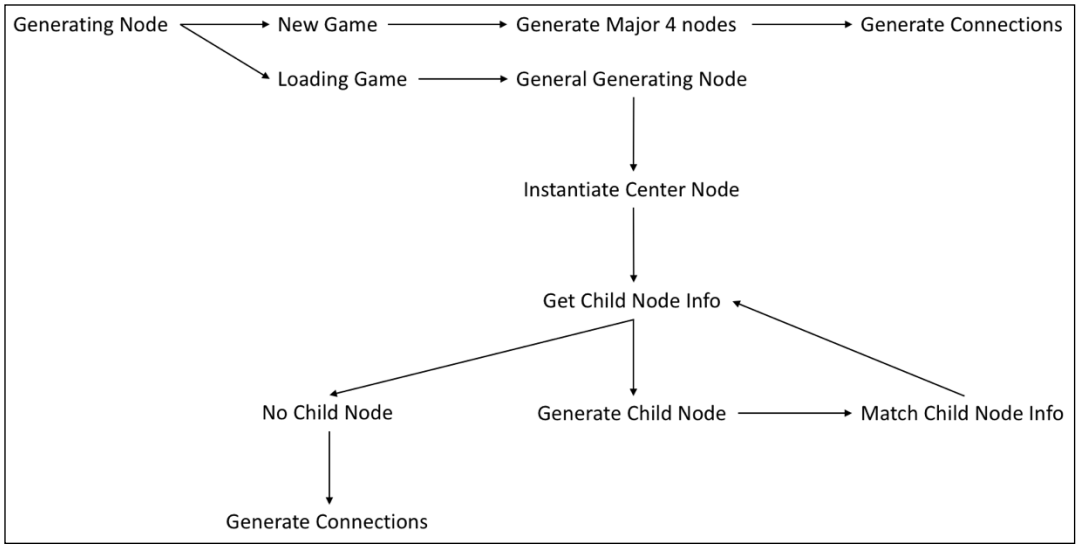


Figure 3: The flow chart above shows the brief method flowing for the generating node. The recursion for general generating is the best way to precise and safest method to loading the game.

Data Structure

Profile – ProfileInfo.cs

The profile is a major game data which includes profile, resource, and node data as well. Profile itself has several unique features: funds, funds per turns, number of turns, and message logs etc.

```
public string name;
public int fund;
public int fundTurn;
public int labor;
public List<NodeInfo> playerNodeList = new List<NodeInfo> ();
public List<AttributeInfo> playerAttributes = new List<AttributeInfo> ();
```

From the code global variable, there are the variables name, fund, fundTurn, labor, playerNodeList, and playerAttributes. All of which are saved at the certain moment of the turn. ProfileInfo is aimed for saving and loading file. Since the C# only can make one object or type of variable to binary file for converting, it is necessary save all the data on one object: for this game is ProfileInfo.

Resource – AttributeInfo.cs

Engineering to Save the world's major goal of the game is getting a high score when the player reaches the certain final node. The score is comprised of a multiplication of performance metrics and weights and a summation of unspent resources. The player needs to figure out the weight by playing mini games, but player has to consider more about the concept that the resource is major structure of the score.

```
public int ID;
public string name;
public int value;
public bool isActive;
```

AttributeInfo consists with ID, name, value, and isActive variable.

An ID is unique 6-digit number of integer that recognize the attribute. ID can help to recognize whether the attribute is newly created or edited. Each attribute is also used in the node resource, therefore, destroy the object and create new object is not possible.

Name and value are basic information of the attribute. Name is the name of the resource and value is a weight of the resource. Even the player creates an attribute, one can activate or deactivate the attribute which will not count as final score.

Name	Weight	Min	Max
Att 1	0	0	0
Att 2	0	0	0
Att 3	0	0	0

Attributes will be shown left corner of the main game scene. Since the player will not able to know the actual weight of each resource, weight column is input field in order to player can guess.

Node – NodeInfo.cs

Node is a key element for Engineering to Save the World. By purchasing and labor setting, the player can move forward the game flow.

```
//Node Identification
public int direction; //N = 0, E = 1, S = 2, W = 3, Center = 4;
public int level;
public int order;
public string name;

//Node information
public bool isHidden; //Hidden = 1 or true, Not Hidden = 0 or false;
public bool isPurchased;
public bool isLaborSet;
public bool isNewAttribute;
public bool isTested;

public int laborSet;

//Resources
public bool isFinalNode;
public int minimumFund;
public int minimumLabor;
public List<AttributeElementInfo> attributeElementInfoList;

//Connection information
public List<string> childNodeName;
```

Figure 4: Example of Script from NodeInfo, showing Global Variables

The nodeInfo has four major distributions: node identification, node information, node resources, and connection information.

In the node identification, there are three key information that refers the position of the node: direction, level, and order. The direction, describes above, 0 is upside, 1 is right, 2 is downside, and 3 is left side from the center node. The center node is referred as 4. The node spread is based on the direction. Therefore, each direction cannot be overwrap which means each direction node cannot have connected each other. The separation of the direction can make completely different story line and be avoid the confusion of the structure.

The level and order refer the position that according to the direction. From the center node, the gap between current node refers the level, for example, since the center node is called always “4 0 0”, the right side without any gap node will be call “1 1 0”. Order refers a distance from the horizontal or vertical axis, for example, if there is a node call “1 1 0”, the name of right side one node down call “1 2 -1”.

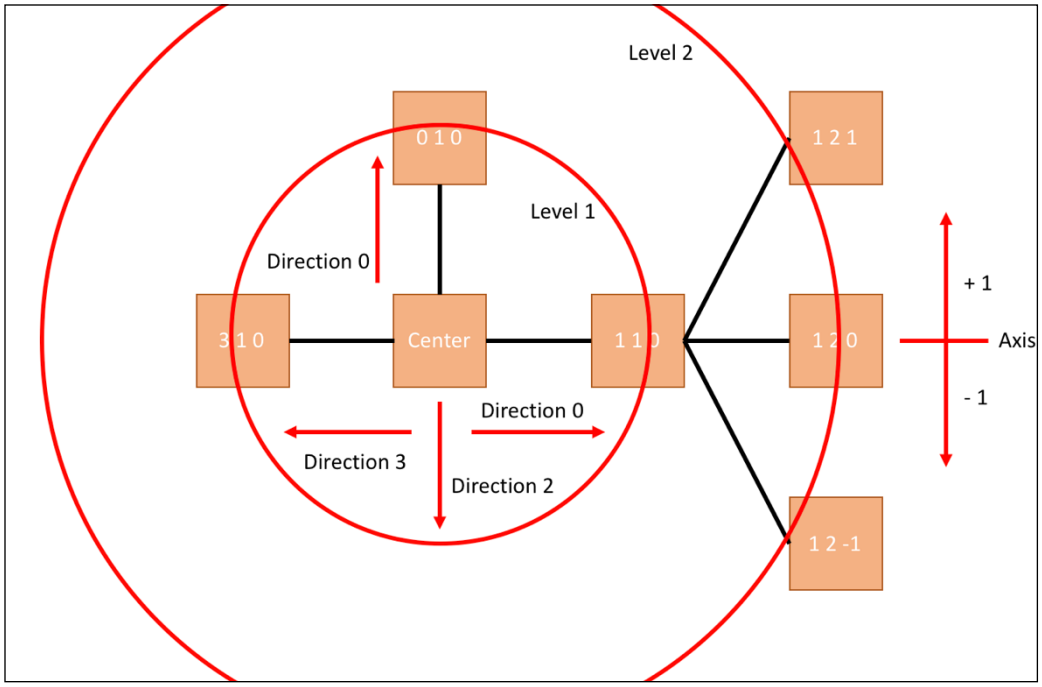


Figure 5: Node Position

The above diagram is a concept of the position design. In the direction 1, right side, the order increase when the node above the axis but in the direction 3, left side, the order will decrease then the node above the axis. The reason the order reverse upon the direction is the game controller will consider the direction is actually rotation of the certain node spread system which means since the direction 1 and direction 3 is opposite way, the order will be also opposite.

After saving the position information, the node still has to recognize the information for the game playing: Boolean value for purchase, hidden, labor set, and test and custom value for purchase fund, minimum labor, and resource list.

One of the key features that each node should save is child node. Each node may have a child node even more than one.

Node Methods

Node Creation and Node Edit

To make a custom game, the player must be able to control the node creation. The node creation and edit are based on the drag and drop. For activate the drag and drop, class must contain IBeginDragHandler, IDragHandler, and IEndDragHandler.

```
class ButtonCreate_NewNode : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
```

Since the position of the node is based on the grid system, the positions will not be the random place on the main game scene. Therefore, the finding position methods are necessary. The methods can find the possible position then it checks the node is already on the position or not.

```
public static string[] validCreatePosition(NodeInfo _node)
```

The method returns the name is position that valid for creation and edit.

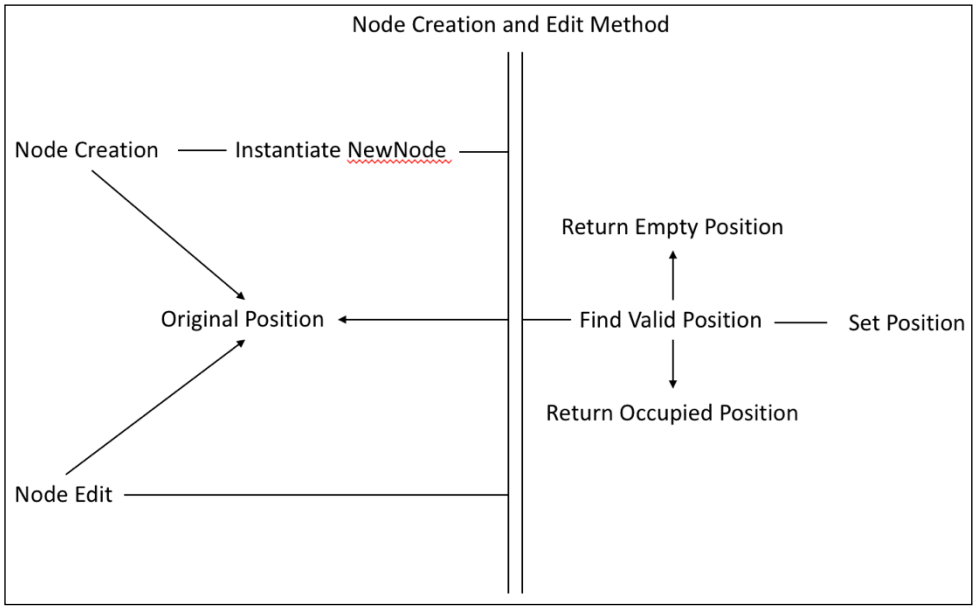


Figure 6: Methods of Creation and Edit Mode

The above diagram shows that the methods flow of the creation and edit method. The original position will use when the player did not place the proper position what creation and edit function given.

Each mode will show the possible position with above images: the left box means the possible new position; the right box means combine the nodes. The square boxes is a possible position that player can place the original node. If player drags the node close enough which means the cursor is inside of the square boxes, the player can see the node will auto fit on the boxes.

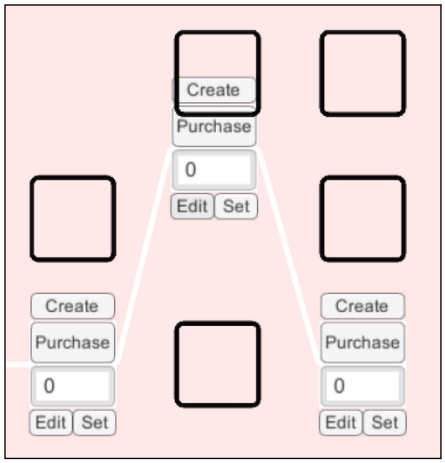


Figure 7: Node Edit Mode

The above picture refers an example of node edit mode. In edit mode, the player can drag the node to possible position. Player may drag the node to other position then the node will change the position as well.

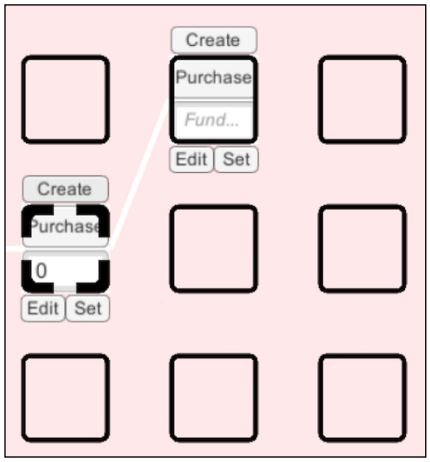


Figure 8: Node Creation Mode

The above picture refers an example of node creation mode. In node creation, there are two possibilities: create a new node and combine nodes. Create a new node is a literal meaning of create a new node. A new node will create when the position is a new. But when player drags to position that already occupied, player may can combine the nodes which means create a new connection only. At this time, the node will destroy but the new connection between original node and targeted node will create.

All of the above methods are saved in the nodeList in the GameControllerScript. Therefore, the undo will not be able to occur; however, since all data is in the GameControllerScript, it will be automatically saved on the saving file.

Node Setting

According to the NodeInfo, there are lots of features that player can customize. Player can delete the node, change the minimum fund to buy, minimum labor to explore, and resource maximum score. Both resource and purchased check box will not be able to click since those variables will set by profile setting and player playing.

Functions

Figure 9: Example of Node Functions

Delete Node

Delete Node button is a function for deleting the current node. The deleting of the node is simply using the “destroy” function for Unity Engine library.

Cancel

Player may change the mind that does not want to change the feature anymore. Therefore, the cancel button will close the node setting window without save any part.

Apply

When player decides to change the feature, the player may hit the apply button. Applying on the node, the system finds the node info and save all the changes. Basically, the system has to recognize the feature is new or edit, therefore, it will not cause any further problem on other methods.

Purchased

The purchased toggle shows whether the node has been purchased or not. Player may can decide to change the status of the node. If the purchased button is unboxed by player

customize, the story flow may contradict. Purchasing means exploring in the game as well, therefore, the connection may break down.

Profile Methods

Profile Setting

Player can fix the name, fund, and labor of the game. Profile setting is an overall game component setting. The profile setting is more directly influence the current game flowing.

Figure 10: Profile Setting Operations

Name

Profile name is a player name or story name that currently playing. Profile name does not have any meaning. The name will show on the main game scene.

Fund

Changing fund is design for initialize the game but player may can change it middle of the game if player want to see the ending part of the game.

Labor

Even labor recharging every turn but still player can change the current possession of the labor to change the difficulty of the game.

Resource Setting

Resource setting is also on the profile setting. The resource setting on the profile setting affects overall game, therefore, player may care more to fix the resource attributes.

Attribute Name

Basically, changing the name is fixing the attribute name showing to the player. Player may can have same name of the resource multiple times.

Actual Weight

Each resource has to have certain amount of weight for final score. This weight is directly connecting to the final score and mini games. The player may not be able to know until one playing the mini games to guess. The range of the weight is 0 to 999.

Activation

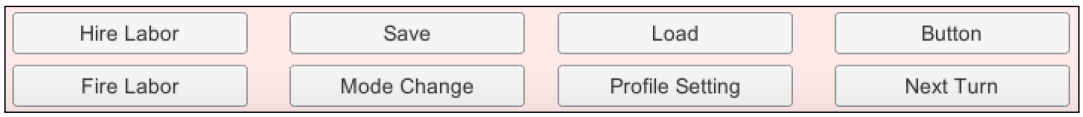
Even player assigns the attribute, player may not want to consider the resource because of difficulties or other reason. But if player does not want to erase the attribute, player just needs to deactivate the attribute. Unboxing the toggle will deactivation of the attribute. It will be still showing on the node setting but it will not be showing on the playing mode.

Delete

The red button represents the deleting of the attribute. Once the attribute deletes, it is impossible to restore it. Also, even player made exactly game name of the attribute, player must reenter all the weights and score for not only in attribute setting in profile setting but also in attribute setting in node setting.

Utilities Methods

Below the buttons are consider as utilities of the game, like menu. Player can fix or test, even change the turns of the game.



Game utilities is also using lots of variable for processing. Since the game major changes and story flow is upon the “Test” and “Next Turn”, it is as important as Game Controller.

```
public string currentMode;
public Text MessageBox;
public Text Score_Min;
public Text Score_Max;
public Button TestButton;
public GameObject profileSetting;
public GameObject resourceLabel;
public Transform resourceElementGroup;

private int testPrice;
private string nodeName;
private Vector3 pos;
private NodeInfo nodeInfo;
private GameObject newProfileSetting;
private GameObject newResourceElement;
private GameObject[] ManualGroups;
private List<GameObject> TestNodes = new List<GameObject> ();
private List<AttributeInfo> playerAttributeList = new List<AttributeInfo>();

private string MESSAGE_SPACE = " ";
private string MESSAGE_NOTENOUGHFUND = "Not enough fund";
private string MESSAGE_FORTEST = "for testing";
```

Figure 11: Example of Global Variables from Utility Script

Each of the game movement the game utility method has to notify the player therefore, the messaging function.

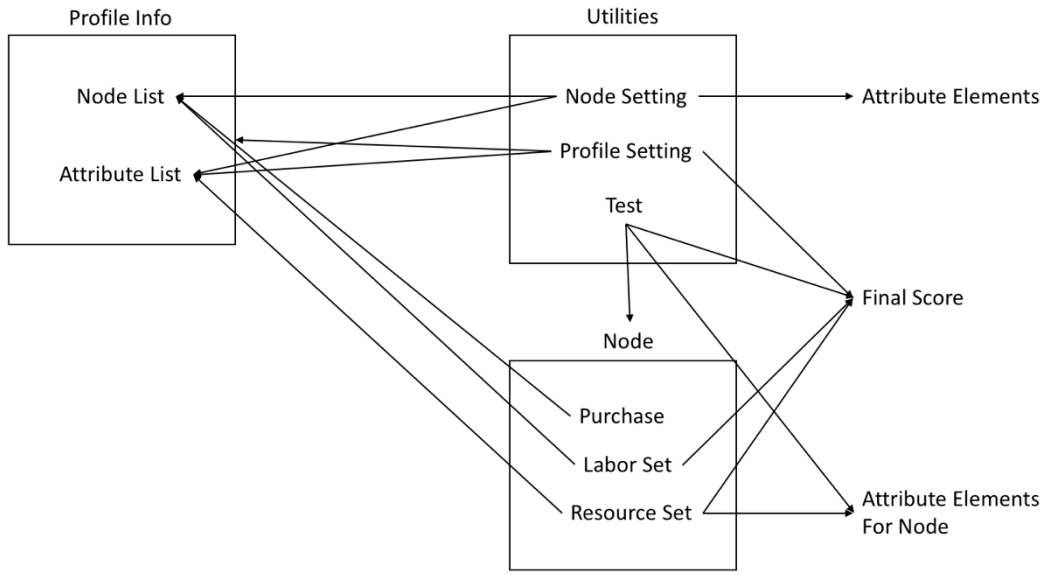


Figure 12: Game flow of Utilities, Node, and Profile

The above diagram is major contribution of the game utilizes to overall game flow. Game utilities is not only highly influence on the final score but also game playing. Since the “Test” and “Next Turn” functions are highly effect on the game playing it is separate from the GameUtilityScript.

Test (Button)

Test button is one of the key function of the game. It will determine each node resource score depends on frequency of the testing, depth, and labor setting.

Next Turn

The player may can limit the turns that player can play on the playing mode. Turn is one of the wining condition that player must finish the game before the limitation of the turn or player must achieve the score in certain wining condition.

Mode Change

The game has two modes: playing and manual mode. The playing mode is literally the player playing mode which means player cannot fix any of the component nor change the position of the node. Besides, the manual mode is editing mode that player can fix all kinds of the feature on not only node information (node setting) but also game general feature (profile setting).

Hire and Fire Labor

Future component: premade.

> Methods that how to hire and fire the labor can restrict the labor usage, therefore, it can control the difficulty of the game.

Saving and Loading

The major function that used for saving and loading methods is binary formatter. One of C# IO functions that easily build a binary file (.dat) in order to save the game data.

```
binaryFormatter.Serialize
```

Unlikely other programming language, C# has function called “binaryFormatter” which can change any type of variable to binary code, so that it can be saved on data file (.dat). Since all the data saved in the ProfileInfo, this is the only object which needs to be saved.

Event System

Introduction

The Event System is in charge of generating events during main game play. The Events are currently limited to a list of 10 drills, but the Events could also be expanded to a number of random prompts. The events are triggered based on chance and user activity in the main game. The Event System includes a Scene Manager, a probability generator, a Reward System, and a link to all possible drills and events.

Drills available to play during Event System:

1. Reliability and Cost Levels 1, 2, 3, 4, 5
2. Good Requirement Bad Requirement
3. Technical Readiness Level Assignment
4. User Interaction Diagram
5. Grandfathered drill from previous semester (Drill 1 & Drill 2)

A new Drill Engine was introduced to include drag and drop mechanics, allowing the creation of several interactive drills (Reliability and Cost, Good Requirement Bad Requirement, Technical Readiness Level Assignment, User Interaction Diagram). The script allows blocks to be dragged and placed into regions by the user. If the block is placed in a region, then the block's label is compared against the block's label and certain game mechanics can be programmed in. For instance, if assignment, the drill can detect if the block should or should not be assigned, and apply a delta score accordingly.

For the first time, a reward system was integrated into the main game. Currently the reward system is limited to modifying player funds, but significant expansion is planned to include modifying labor and node parameters, as well as introducing timers for some boons (such as special nodes becoming visible)

The Event System required significant changes to the main game. Singleton Scripts were created and appended to each element of the main game required to persist through scene changes. The Event System was appended to the GameController.GameControllerScript. A Scene Manager was introduced (to switch between drills and the main game. And scripts were bootstrapped into the Event System to hide or show the main game CanvasGroups.

The architecture of the main game was less than ideal for incorporating an Event System. For comparison, professional games typically have a main menu with a game controller which contains all variables and persistent memory. When scenes are changed, the game controller and memory are preserved, while the canvas elements are destroyed. Subsequent scenes will read the persistent memory, load the appropriate canvas elements, and modify the persistent memory based on game mechanics. Currently, our main game has a single scene which stores the persistent memory meshed with canvas elements. If the canvas elements are not preserved along with the persistent memory, then

the canvas elements will be lost upon return to the main game. This made integration with an Event System a bit 'hacky.' A singleton script was appended to each GameObject, along with a CanvasGroup. When a drill is loaded, the main game CanvasGroups are hidden, and when reloaded, the CanvasGroups are shown.

Future work for this project will include the following:

1. Significant additions to the Reward System
2. New Drills tied into Event System
3. Improvements to existing drills (aesthetics, start text, randomized parameters)
4. Creation of a memory system apart from the main scene, which the main scene controller reads to populate Canvas elements
5. A Singleton template which is extended by persistent memory scripts

Description

The Event System plugs drills into the main game. The Event System is triggered every time the user selects the primary node button. The Event System generates a random number, and if the random number falls below a certain threshold (10% for example), then a drill is initiated. The Event System then determines which drill to play based off a list of drills available to play, and a list of drills already played. If all drills have been played at least once, then the drill is chosen at random from the list of available drills. Otherwise the drill is chosen in the sequence of drills on the available drill list.

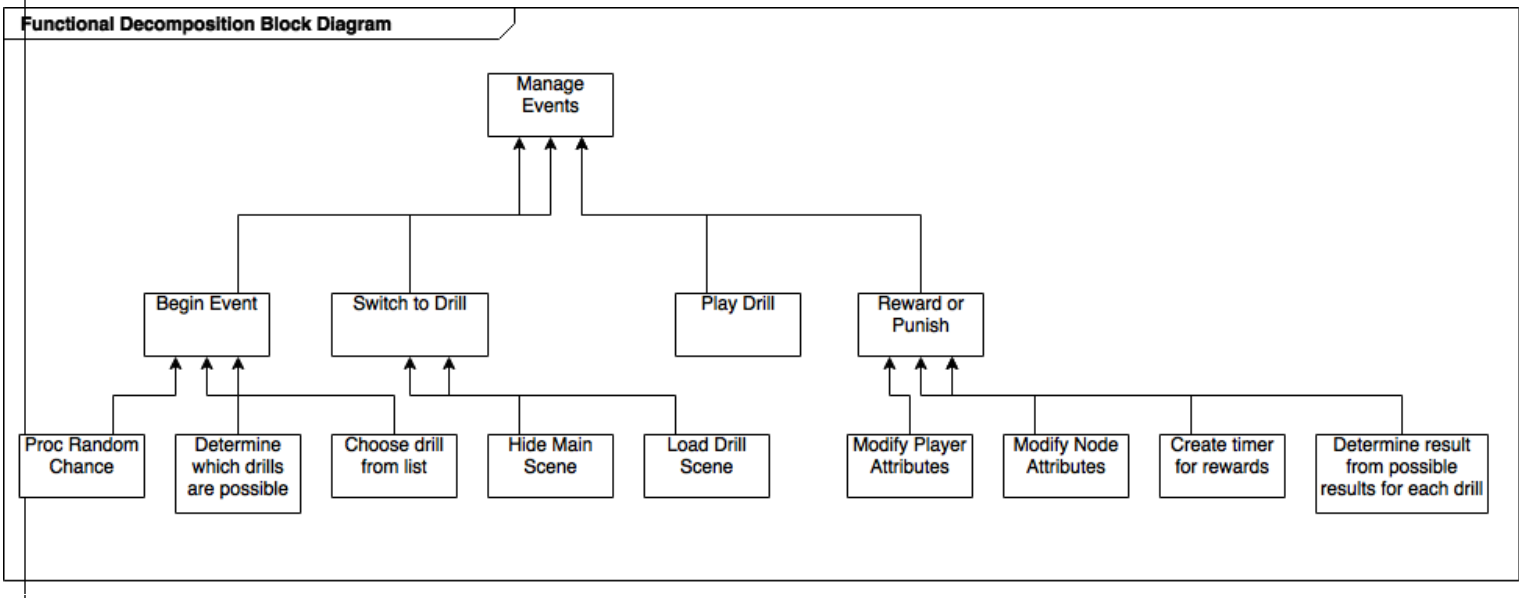
After the drill is selected, Event System hides the Main Scene CanvasGroup and loads the drill. After the user completes the drill, a button is displayed to return to the main game, along with a score and a pass or fail text. If the user finishes the drill prematurely, the user may opt to end the drill prior to the time limit expiring. The drill controller sends the pass or fail message back to the Event System. The Event System regenerates the main scene and triggers the Reward System.

The Reward System receives the pass or fail signal from the drill, along with the name of the drill. Currently, the reward system is limited to modifying the player funds based on the pass or fail signal. If the user failed to accomplish the drill, then funds are subtracted. If the user succeeded, then funds are added. However, substantial expansion is planned for the Reward System next semester.

Functional Decomposition of Event System

At the highest level, the Event System manages Events. At the first level, the Event System begins an event, switches to a drill, allows the user to play a drill, and then assigns a reward or punishment based on the user's performance. The Event System procs a chance for an event every time a node button is pressed. The Event System determines which drills are possible to play, and then selects the drill based on the drill selection logic. When switching to a drill, the main scene is hidden, and the drill scene is loaded. In the reward system, player or node attributes are modified based on user performance. Each

drill will have a list of rewards or penalties. The Reward System will select the penalty or reward, scale based upon performance, and randomly assign a timer if applicable.



Activity Diagram of Event System

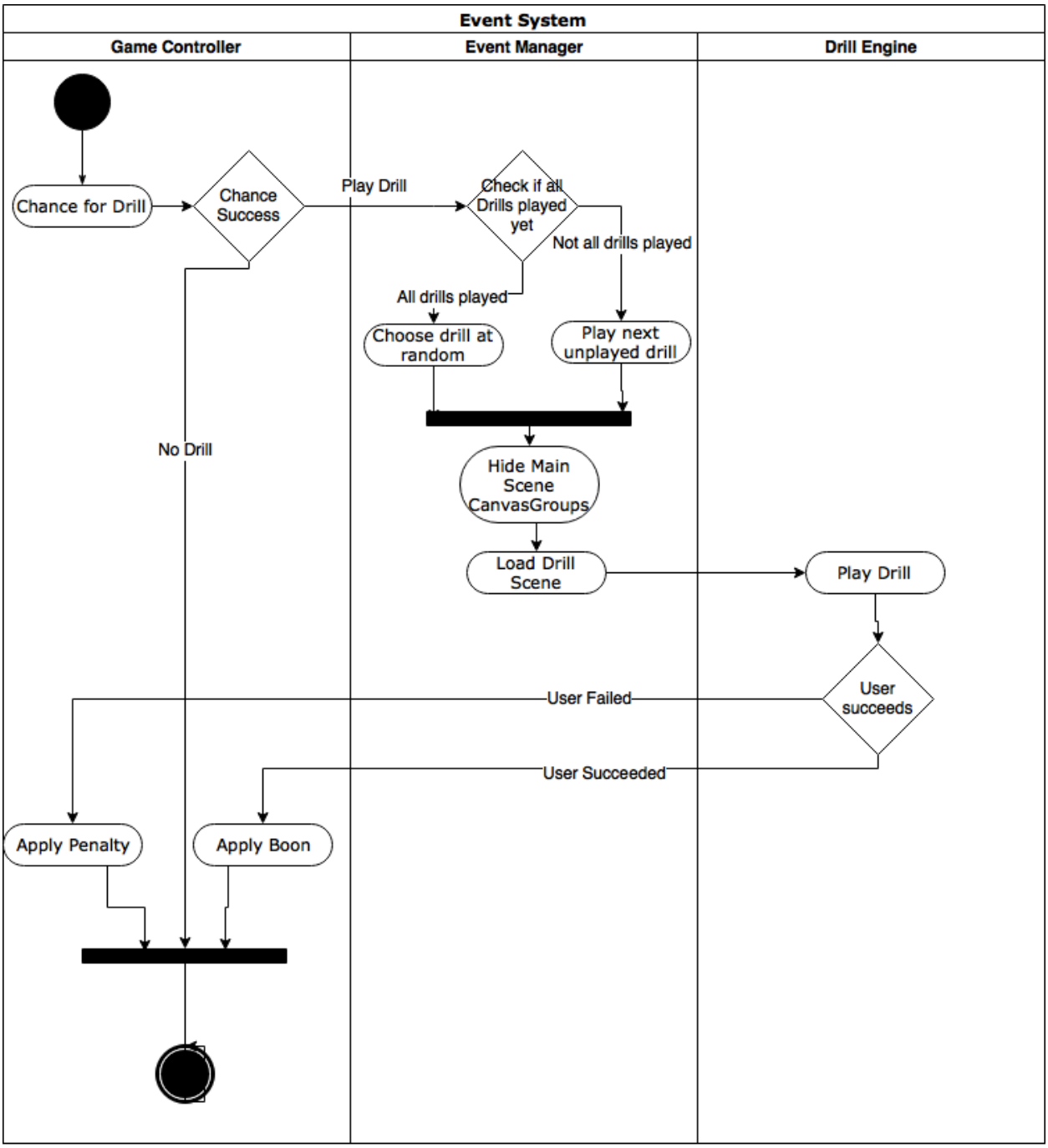


Figure 14: Event System Activity Diagram

When the Event System is activated, it first decides if a drill is applicable. If a drill is not applicable, then the Event System terminates. If a drill is applicable, then a list of playable drills is compared to a list of un-played drills. Drills yet to be played are preferred, in the order that they are found on the list of playable drills. If all drills have been played, then a drill is selected at random. The Event System hides the main game, loads the drill scene, and the user plays the drill. At the end of the drill, the drill returns

to the Event System (in the Game Controller), along with a pass or fail message. At this point a penalty or reward is applied. The Event System then terminates.

Activity Diagram of Reward System (Current State)

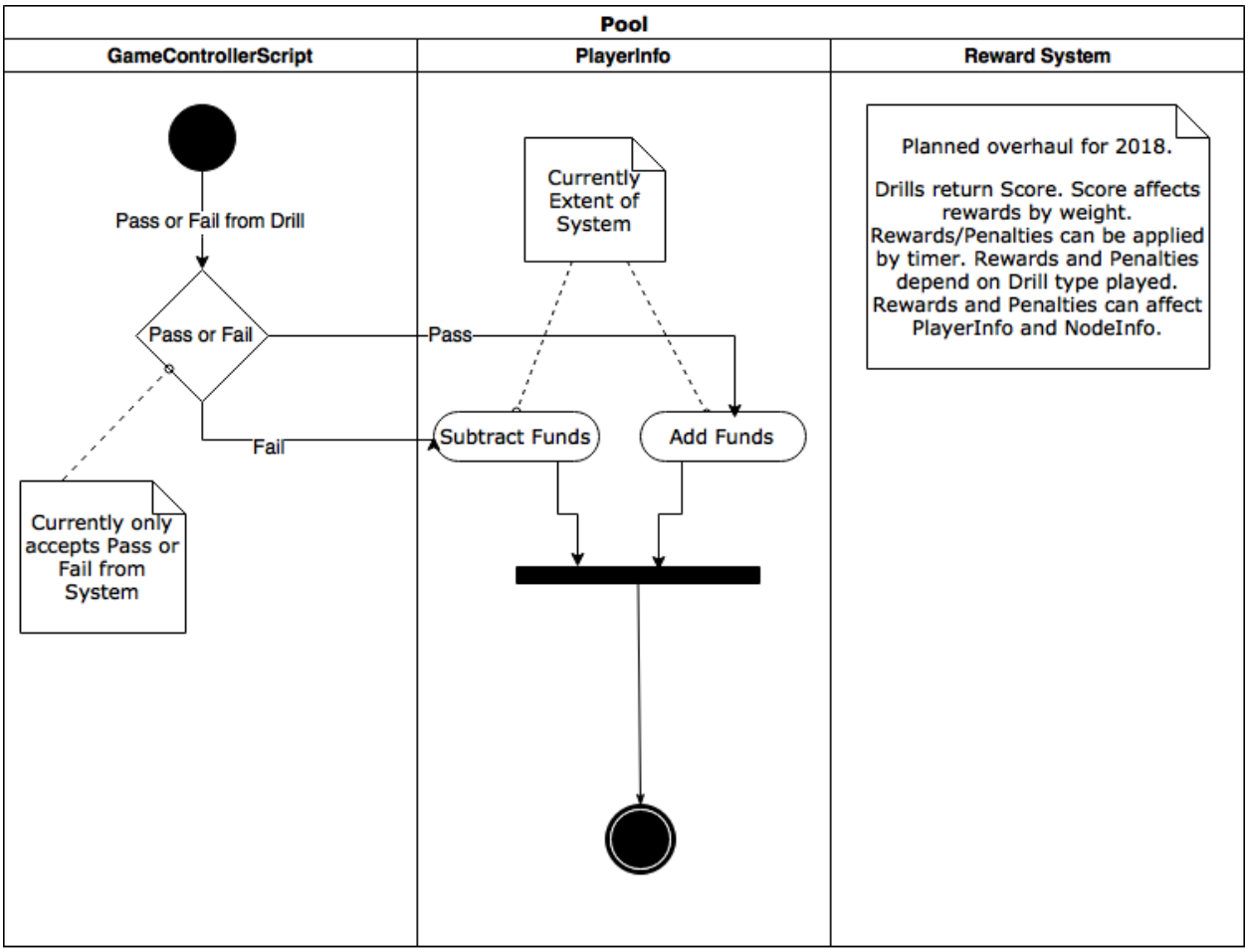


Figure 15: Reward System Activity Diagram

The Reward System is called along with the name of the drill played, and a pass or fail signal. If the user passed the drill, then funds are added to the player. If the user failed, then funds are subtracted. The planned expansion can be seen below.

Activity Diagram of Reward System (Planned Expansion)

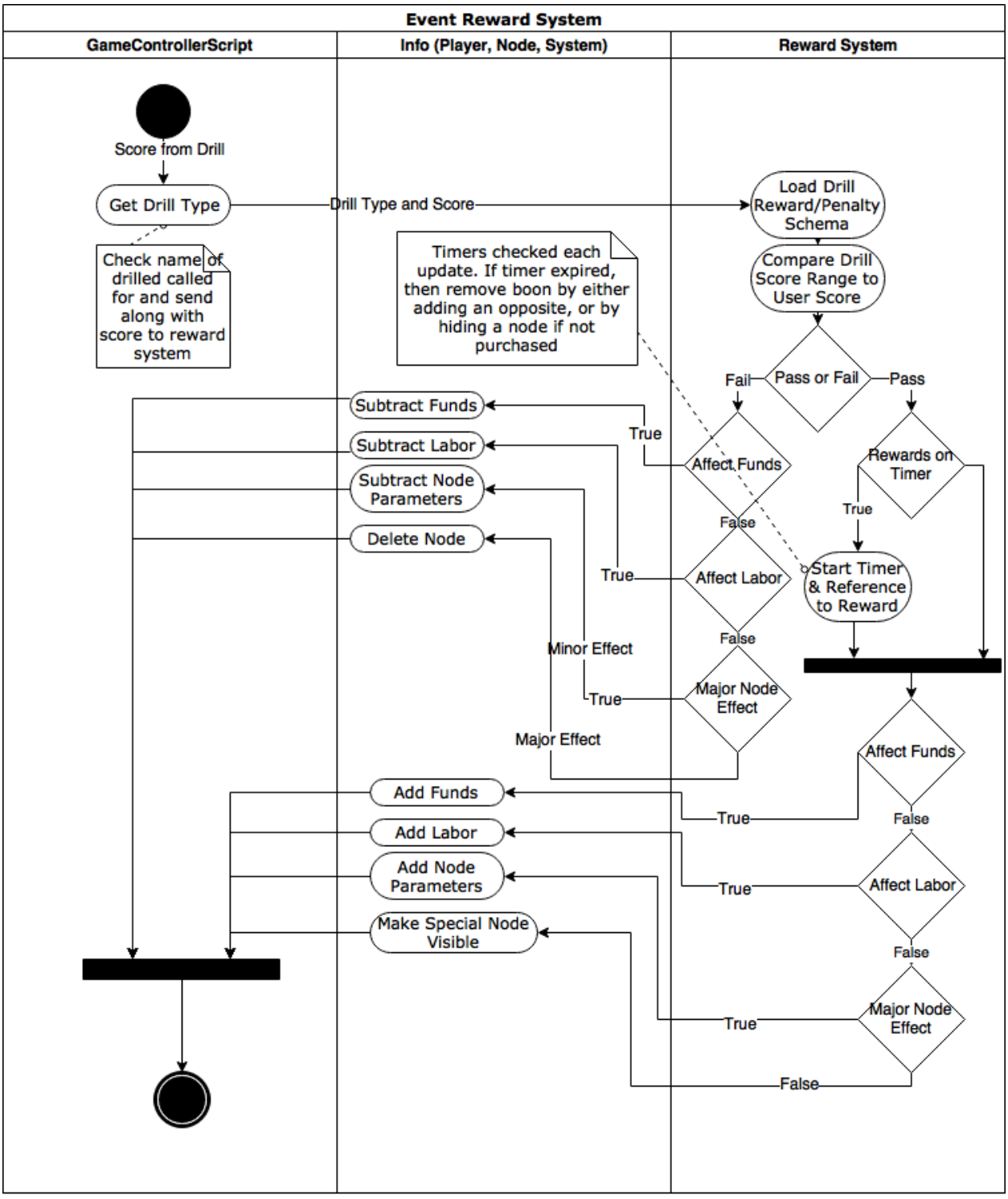


Figure 16: Reward System Planned Expansion

The planned Reward System works as follows: The drill name and score are sent to the Reward System. The Reward System looks up the reward/penalty schema for the given drill, and compares the user's

performance with the ranges provided in the schema (different ranges have different rewards and penalties applies). Based on the user's performance, the Reward System determines if a reward or penalty is applied. If a penalty is applied, the Reward System determines which of the following occur (based on performance and schema): Affect Player Funds, Affect Player Labor, Affect Node Parameters (subtract from purchased nodes), or Delete a purchased node. If a Reward is applied, first the Reward System determines if a timer is applicable (if so, then a timer is initialized and attached to the reward). Then the Reward System determines the type of reward: Add to Funds, Add to Labor, Add to existing Node Parameter, or make a new special node visible.

Sequence Diagram Event System

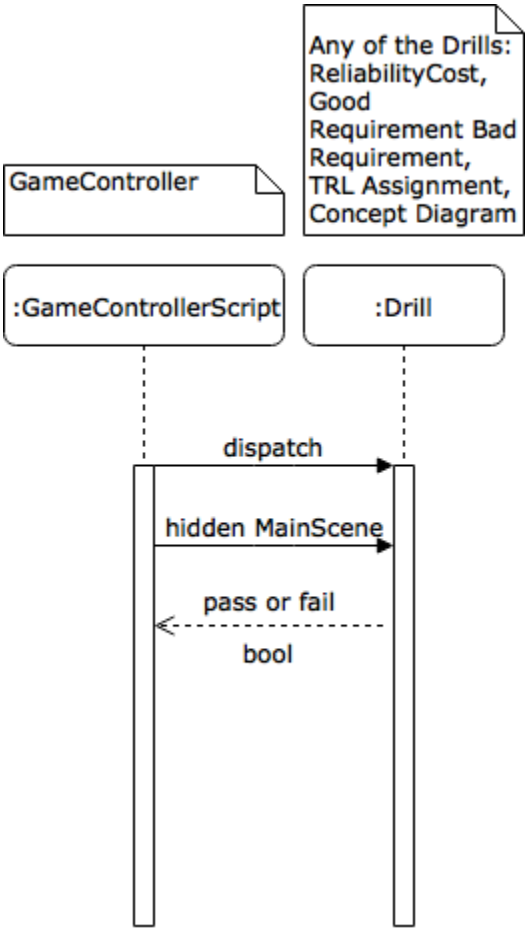


Figure 17: Sequence Diagram for Event System

The Game Controller dispatches the drill and sends the main game elements (hidden) to the drill. The Drill returns the pass or fail signal (current state) and eventually as score (future expansion).

Drills

Description of Drill Engine

A new drill engine was introduced this semester. The engine assigns a labels to Quads, allows the user to drag and drop playable blocks, and detects when a block is dropped into an non-playable Quad (Region). The Engine includes a timer, a score and score display mechanism, and the ability to insert game logic, depending on how the blocks and regions are to be used. Additionally, triggers are available for when a block is dropped into a region, and for when blocks are removed from a region. Hidden blocks can be made visible on a trigger (TRL drill for example), new aspects can be unlocked and become playable (user interface drill for example), or calculations may be performed (reliability cost drills for example). The drill engine includes a function to end the drill prior to the time-limit ending, and the engine also includes a function to return to the main game.

New Drills

The Reliability and Cost Drills are the largest expansion, with five different levels introduced, each level with increasing difficulty. The premise of the drill: The user is given multiple components of varying cost and reliability, and the user is given an architecture to drop the components into. As the user drops components in to the architecture, the reliability and cost of the system are calculated. The user's goal is to achieve a target reliability and cost, where a lower cost and an increased reliability are desired. Currently a simple pass or fail signal is generated at the end of the drill, however, in the future this could be expanded to a percentage exceeded or percentage failed.

The User Interaction Drill is the most complicated drill introduced this semester, with the most possibility for customization and optimization. The premise is that a given system is under investigation. The user must assign system-users and select the relation between the system-users and other system-users and the system itself. The user drags system-user blocks onto the map and into system-user regions. When regions are populated, a connection becomes available between two populated system-user regions. When the connection is selected, labels become available. At the end of the game, a score can be applied by how well the user-interaction diagram matches with the initial prompt and description.

The Good Requirement Bad Requirement Drill requires the user to read a prompt and determine the appropriate requirements (from a list of possible requirements). The user drags the different possible requirements into one of two regions, a good requirement region, or a bad requirement region. A score is applied based on the number of appropriately categorized requirements. Future expansion will include a buzzer sound to play on each incorrect answer, and a message showing an explanation for the incorrect answer.

The Technical Readiness Level (TRL) Assignment Drill gives 9 blocks detailing the readiness level, and 9 regions labeling the readiness level. The user reads the description and drags and drops the TRL block into the appropriate region. At the end of play, the incorrectly labeled regions have the correct labels applied, helping the user to learn TRL.

Both the Good Requirement and TRL Assignment Drills can be expanded to quiz users on most aspects of project management from a theoretical perspective.

Sequence Diagram for Reliability and Cost Drill

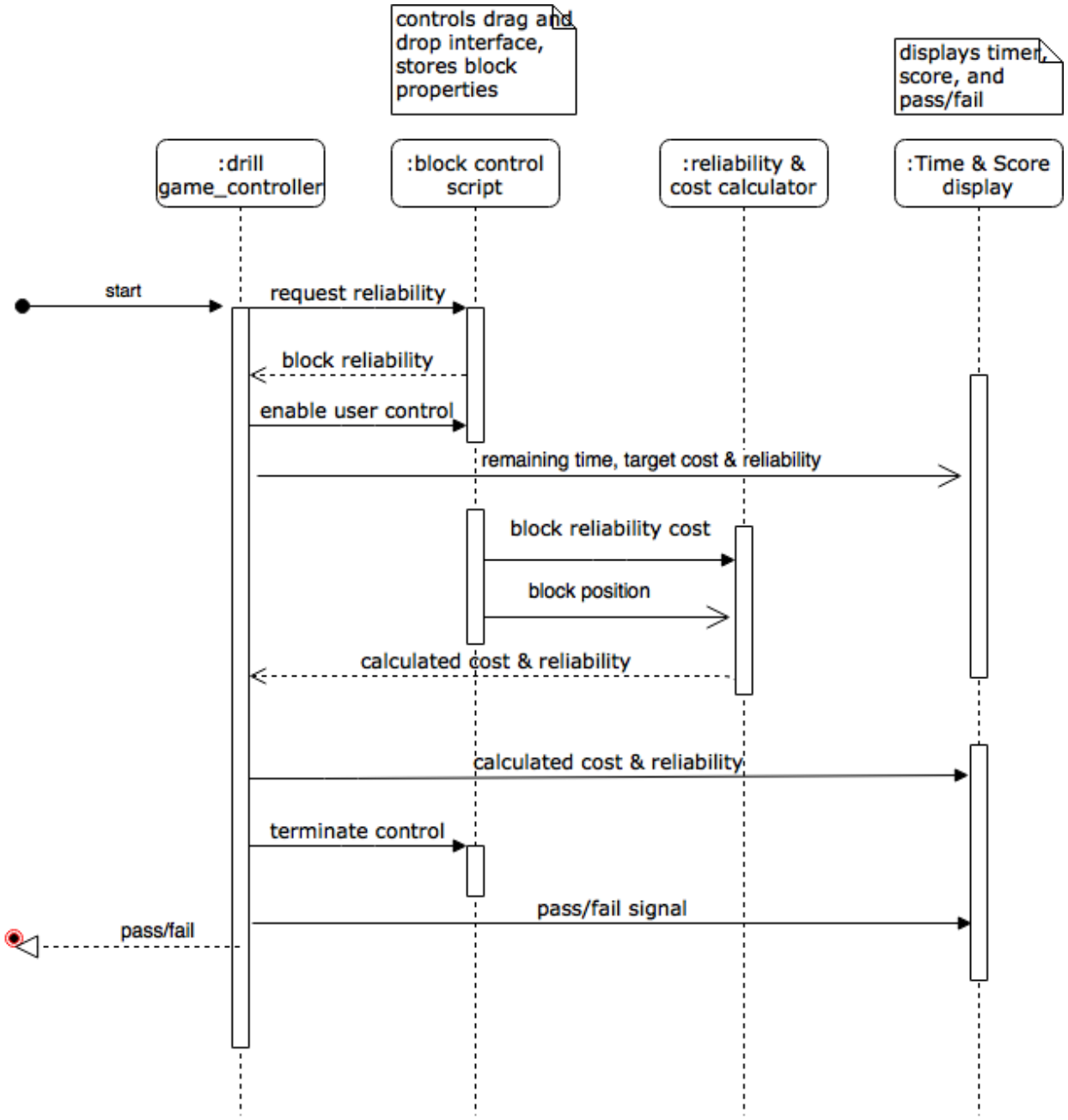


Figure 18: Sequence Diagram for Reliability and Cost Drill

Explanation of Sequence Diagram: The drill is initialized by the Event System. The drill game controller reads each component block for a reliability and cost, and then assigns the appropriate color and text to each component block. User control is then enabled, allowing the user to drag and drop each component. The timer is initialized along with the target cost, reliability, and initial cost and reliability.

Reliability and Cost are updated continuously as calculated based on user's interactions with blocks. Blocks send reliability and cost information to reliability and cost calculator as the blocks are dropped into the architecture. The calculations are returned to the game controller, which then updates the displays. The game controller terminates block control (ending play upon either timer expiration or user's choice to end drill), and the final score along with kill signal are sent to the displays, which then enables the user to exit back to the main game. The final pass/fail results are returned to the main game.

Sequence Diagram for User Interaction Diagram Drill

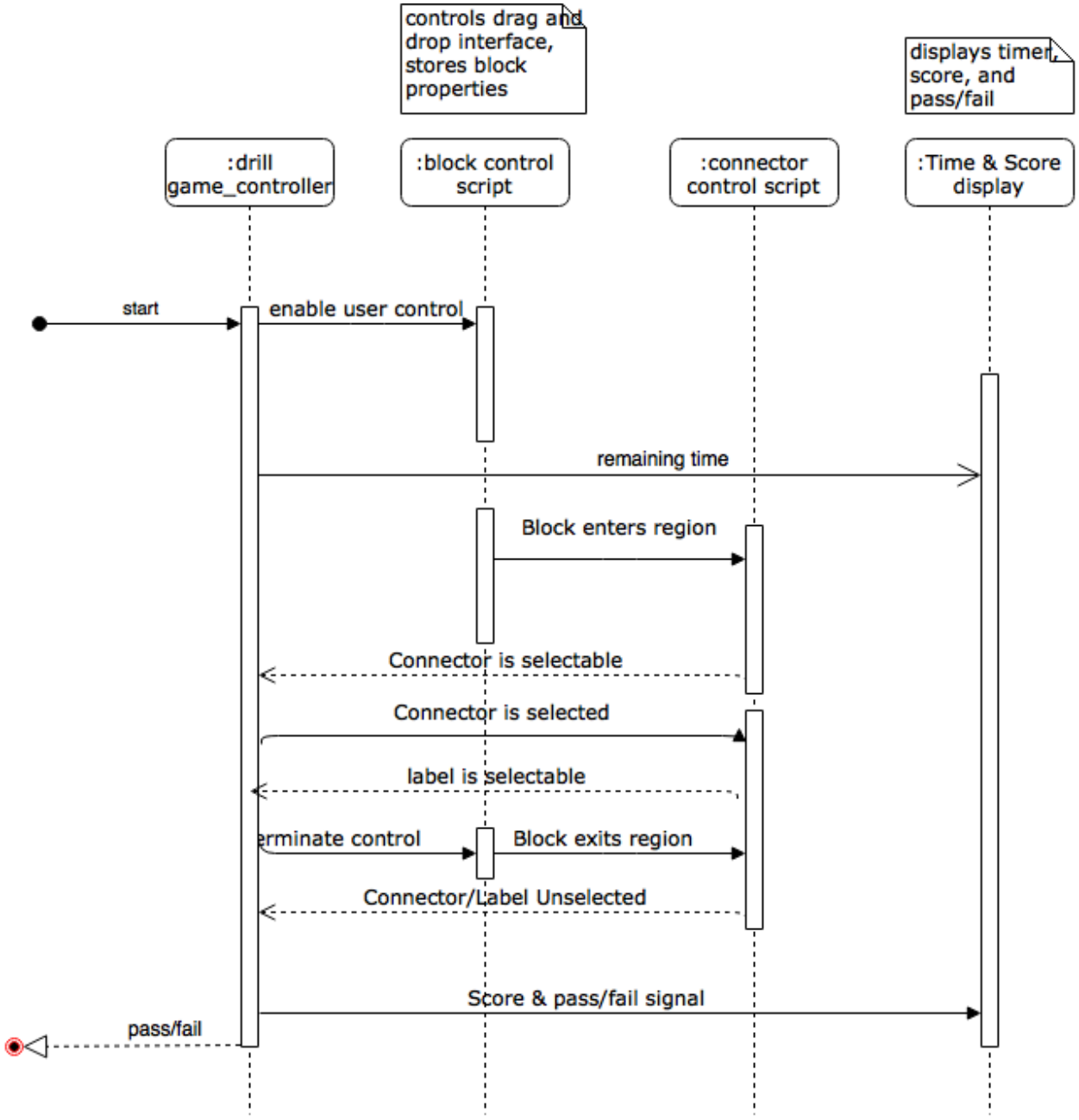


Figure 19: Sequence Diagram for Concept Diagram Drill

Explanation of Sequence Diagram: The drill is initialized by Event System. The game controller enables block script (allowing user to drag and drop blocks). The remaining time is sent to the timer, initializing the timer if not initialized already. When a block enters a region, a connector script is triggered, displaying possible connections. If a connection is selected, then a label becomes selectable. Upon end of play (per timer expiration or user decision), the game controller disables the blocks drag and play, and sends the final score to the display, enabling return to main game button. Upon return to main game, as pass or fail signal is returned.

Sequence Diagram for Assignment Drills

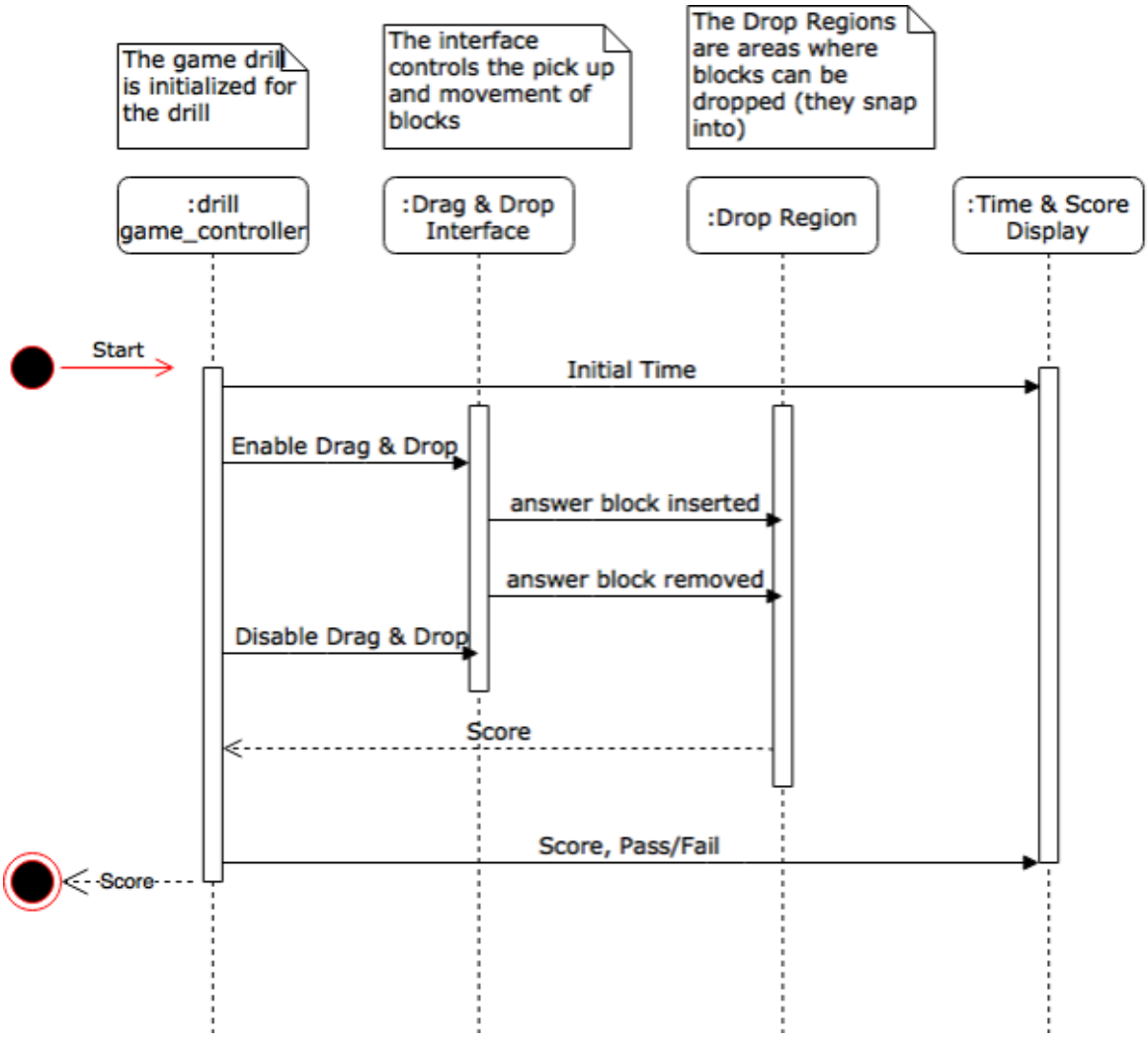


Figure 20: Sequence Diagram for Assignment Drills

Explanation of Sequence Diagram: This sequence diagram describes two drills: The Technical Readiness Level Assignment Drill and the Good Requirement Bad Requirement Drill. The Event System starts the drill. An initial time is sent to the display and timer. The blocks are enabled. When a block is inserted

into a drop region, the region gets the block information from the block. When a block is removed, the drop region resets. Upon end of play, the blocks are disabled, a score is returned to the game controller from the drop region calculator, and the score is sent to the display and the main game upon exit.

Script Modifications to Main Game

GameControllerScript

Global Variables

These variables were introduced to the GameControllerScript to enable the Event System. They include the chance for Event (used every time there is a chance for event, currently set to .5 = 50%). A list of drills already ran (appended each time a drill is ran), and a list of possible events (each string matches the SceneName in the build settings).

```
private double EventChance = .5;
public List<string> drills_run;
private string[] list_of_drills = {"reliability_cost_drill_level_1","reliability_cost_drill_level_2",
"reliability_cost_drill_level_3","reliability_cost_drill_level_4","reliability_cost_drill_level_5",
"concept_sketch_interface","drill_1","drill_2",
"good_requirement_bad_requirement","technical_readiness_level"};
```

Chance for Drill

This function is called whenever a chance for a drill occurs. A random number is generated, and compared to the chance for a drill. If the random number is less than chance for drill, then a drill is initiated. The function then determines which drill to call, and implements SceneManager to load the drill. Currently the MainGame CanvasGroups are required to be hidden, due to the fact that CanvasGroups have a “DontDestroyOnLoad” flagged, which is required as game memory is meshed with the canvas elements.

```
public void Chance_For_Drill(){

        Debug.Log ("Chance for Event");
        int randomNumber = Random.Range(0, 100);
        Debug.Log(randomNumber);
        string scenename = "";
        if (randomNumber < EventChance * 100){
                Debug.Log("Event System Activated.");

                if (drills_run.Count == list_of_drills.Length){
```

```
                randomNumber = Random.Range(0, list_of_drills.Length-1);
                scenename = list_of_drills[randomNumber];
            }else{

                scenename = list_of_drills[drills_run.Count];
            }
            Append_Drills(scenename);

            GameObject c = GameObject.Find("NodeGroup");
            CanvasGroup cg = c.GetComponent<CanvasGroup>();
            cg.alpha = 0f; //this makes everything transparent
            cg.blocksRaycasts = false;

            c = GameObject.Find("MainScene");
            cg = c.GetComponent<CanvasGroup>();
            cg.alpha = 0f; //this makes everything transparent
            cg.blocksRaycasts = false;
            Camera.main.gameObject.active = false;

            SceneManager.LoadScene(scenename, LoadSceneMode.Additive);
        }
    }
```

Reward System:

This method is called every time a drill returns to the main game. Current state, a simple boolean object is sent (pass or fail). In the future, this method will be expanded to allow for a double (score), and the title of the scene loaded. This method will then apply a penalty or boon per proposed Activity Diagram.

```
    public void ReturnToMainGame(bool success){
        Debug.Log("controller..."+success);

        if(success){
            playerInfo.fund = playerInfo.fund + 10;
        }else{
            playerInfo.fund = playerInfo.fund - 20;
        }


        preForNodeGeneration();

    }
```

Singleton Script

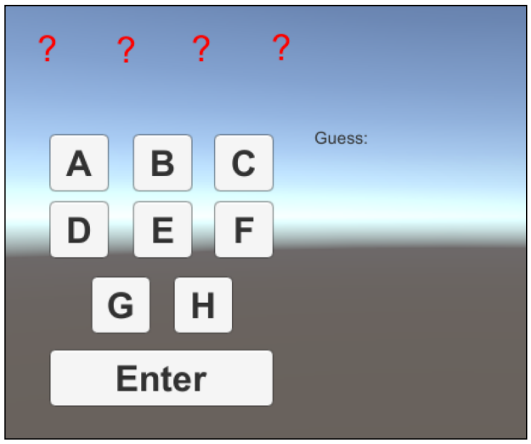
The Singleton Script was added to each necessary main game game object. A unique singleton script is required for each game object, as the singleton script creates a unique instance of itself (to prevent duplicates of the game object).

```
public class Persist_A : MonoBehaviour
{
  private static Persist_A instance = null;
  private int switch_count = 0;
  public static Persist_A Instance
  {
    get
    {
      if(instance == null)
      {
        instance = FindObjectOfType<Persist_A>();
        if( instance == null)
        {
         GameObject go = new GameObject();
          go.name = "Persist_A";
          instance = go.AddComponent<Persist_A>();
          DontDestroyOnLoad(go);
        }
      }
     return instance;
     }
  }
  void Awake()
  {
    if(instance == null )
    {
      instance = this;
      DontDestroyOnLoad(this.gameObject);
    }
   else
    {
      Destroy(gameObject);
      switch_count += 1;
      Debug.Log("switch_count"+switch_count);
    }
  }
}
```

Mini Games

Mastermind

Mastermind Mini-game screen



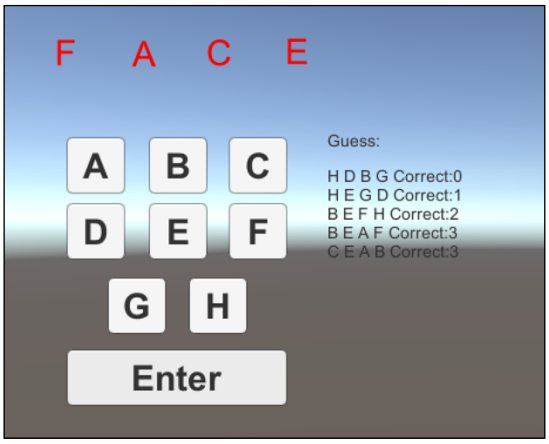


Figure 21: Screen Capture of Mastermind Mini-Game (1)

Figure 22: Screen Capture of Mastermind Mini-Game (2)

The mini- game 'Mastermind' pops-up as the player purchases nodes. The purpose of the mini-game is to help the player decide which performance criteria is important and which are not valued by the 'customer'. The game is based off the board game version of Mastermind where the user inputs a selection of guesses and the game will tell the user which guesses out of the selection are correct or incorrect.

The user input and the game output is displayed. The user inputs can be seen in red and the game output can be seen in a panel to the right of the main game interface. The game gives the user a selection of criteria listed from A-H which represent the criteria in the game. The user chooses any combination of criteria and clicks on 'Enter' for the game to process the entry and return a 'clue'. The clue shows the previous combination attempt and how many criteria were guessed correctly. In this case, the correct combination is "F A C E" in no particular order. The user can keep making attempts until a correct combination is made. There is no limit on how many guesses the user can make but there is a cost of labor for every attempt. The user may close the game at any time and return to the main interface of the game.

Mini-Game Scripts

```
public void n8()
{
    num8 = "H";
    if (count == 0) {
        ask1.text = num8.ToString ();
    } else if (count == 1) {
        ask2.text = num8.ToString ();
    } else if (count == 2) {
        ask3.text = num8.ToString ();
    } else {
        ask4.text = num8.ToString ();
    }
    count++;
}
```

Figure 23: C# code for each criteria button

The above code checks to see which position of the four inputs is currently empty and prints the respective criteria into the input window in the next available position. Currently, there is no option to cancel or remove a specific input. Once the user has clicked on a criterion, it will be printed into the input box above the buttons. The C# code above is applied to all the buttons from A-H.

```
public void enter()
{
    string c1, c2, c3, c4;
    c1 = (ask1.text);
    c2 = (ask2.text);
    c3 = (ask3.text);
    c4 = (ask4.text);
    if (checkWinner (c1, c2, c3, c4)) {
        Debug.Log("winner");
        //Application.LoadLevel ("winner");
    } else {
        ResetAll ();
    }
}


bool checkWinner(string num1, string num2, string num3, string num4)
{
    if (num1.Equals (ranNum1) && num2.Equals (ranNum2) && num3.Equals (ranNum3) && num4.Equals (ranNum4))
        return true;
    else
    Correct ();
    log.text += ask1.text + " " + ask2.text + " " + ask3.text + " " + ask4.text + " " + "Correct:" + cnt + "\n";
        return false;
}

void Correct()
{
    cnt = 0;
    test = ask1.text + ask2.text + ask3.text + ask4.text;
    foreach (char c in test) {
        if (c == 'F') {
            cnt++;
        } else if (c == 'A') {
            cnt++;
        } else if (c == 'C') {
            cnt++;
        } else if (c == 'E') {
            cnt++;
        }
    }
}
void ResetAll()
{
    ask1.text = "?";
    ask2.text = "?";
    ask3.text = "?";
    ask4.text = "?";
    count = 0;
    cnt = 0;
}
```

Figure 24: Script for checking User Input

The code above shows the code for comparing the user input to the correct answer. Once the user clicks ‘Enter’, the function ‘checkWinner’ will check the input string to the correct combination stored in the script. If there is a match in answer regardless of position, the function will print the current attempt with the number of correct criteria. The function ‘ResetAll’ will clear the inputs and allow the user to enter a new combination of guesses every time the ‘Enter’ button is clicked.

When the user inputs the correct combination of criteria, the user can play the clue guessing game which gives the user clues on the order of importance of the criteria. Both the mastermind game and the guessing game can be played at the same time as they show up in the same window side by side. The guessing game will be explained later in the report.

Guessing Game

This is the 2nd mini game incorporated into the main game. The objective of the game is to give an interactive method for the player to learn the most important criteria of the "customer". The user, at a cost of labor, chooses to compare two criteria. Once the player makes the choice, a clue will be generated. The clue generate takes in the form of >,<, to indicate moderately greater than / less than, >>,<< to indicate significantly greater than /less than and ~ to indicate approximately close .

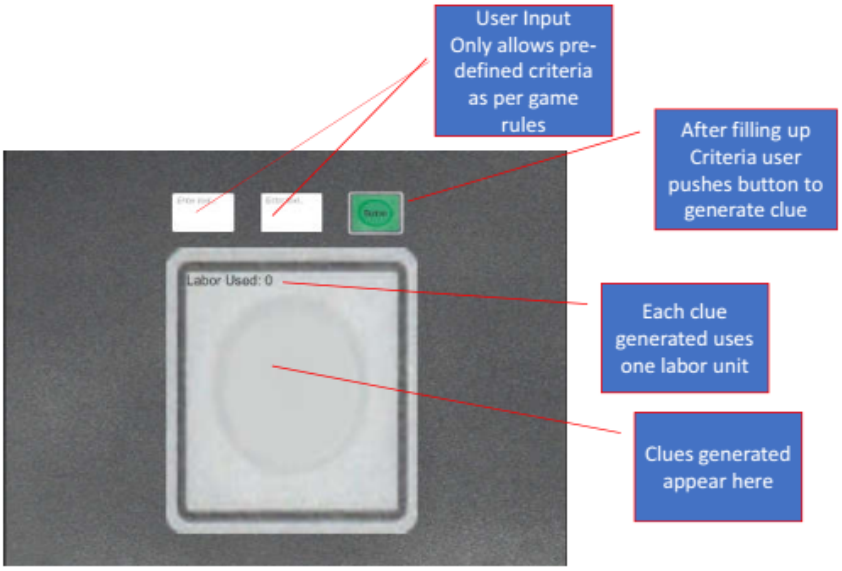


Figure 25: Guessing Game GUI

Currently the code is such that if the relative weight is greater +/-10% it will considered indicate significantly greater than /less than, if the relative weight is between +/-5% to +/-10% it will be classified moderately greater than / less than. And between 0 and +/-5% will classified approximately.

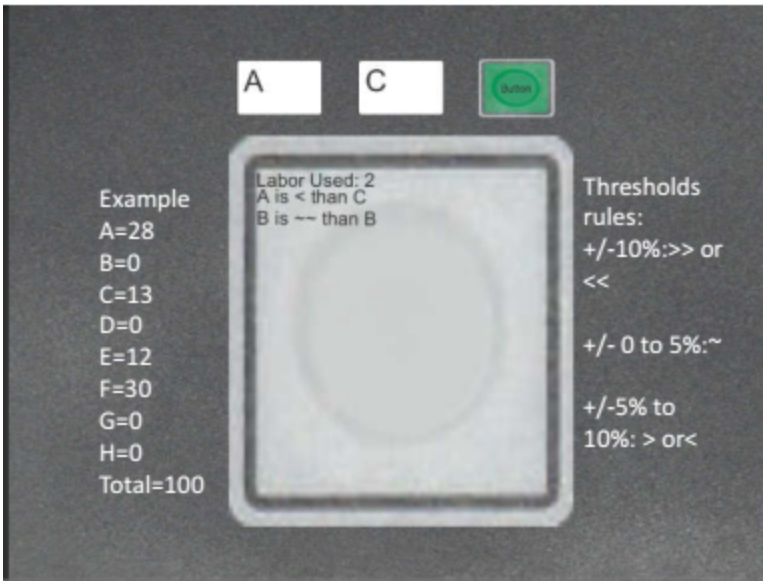


Figure 26: Guessing Game--Clue Generated

References

Allen Liu, C. P. (2017). *Systems Engineering, M.Eng.* Ithaca: Cornell.

Unity. (2017). GameObject.Find. *Unity Script References*, 1.