Leslie Hoyt
COMP 465 - MON/WED
Professor Karamian
10/8/2019

Project 1 - Lego Blocks

Table of Contents

1 Introduction

Project 1 creates a three-dimensional tetris-type game with Lego blocks as the shapes, using the Unity game engine. The program has two main modes: simulation, wherein the user merely spectates and can move a camera, and a playable game mode, wherein the user can move the Lego blocks as well. In both the simulation and playable game modes, Lego blocks fall toward a platform and stack on top of other fallen blocks. The program has a user interface with menus and adjustable settings for the user to choose a simulation mode or a playable game mode, adjust the size of the platform, the difficulty level (rate of speed of falling blocks), and whether the blocks break on impact.

2 Design Goals and Objectives

The project implemented the following core requirements:

1. Dynamically generate the platform where the Lego blocks land. The user will provide the dimensions of the platform when they run the application.
2. Dynamically instantiate one of the many Lego blocks at random.
    a. The instantiated Lego block will have a random position on the xz-plane.
    b. The instantiated Lego block will have a random orientation.
        i. Orientation for Lego block restricted to the angles 0, 90, 180, 360.
    c. The instantiated Lego block will have a random color assigned to it.
3. User can move the falling Lego block on the xz-plane.
    a. Using the following keys for the movement: A, W, S, D.
4. User can re-orient the falling Lego block using the same constraints listed above.
    a. The R key rotates the block.
5. The falling block will stop when it has landed either on the platform or on another existing block.
6. The cycle will begin all over again until the user quits the application.
7. The user can terminate the application by pressing the Q button.

The project also implemented the following requirements for camera movement:

1. Rotate the camera around the platform, always looking at the center of the platform
    a. The left and right arrow keys rotate the camera around the platform.
2. Move the camera Up and Down on the Y-Axis
    a. The up and down arrow keys elevate the camera position or lower it.
3. Zoom In or Out the camera for closer look at the platform
    a. The Z and X keys zoom the camera in and out, respectively.

The project has two distinct scenes:

Back To Top

1. GUIMenu
2. LegoScene

The simplified flow of the code goes as follows:

1. Functions in the UIManager script get called by buttons in the GUIScene
    a. Start button, from Main Menu, calls butStartClicked():
    → Deactivate visibility of the Main Menu panel, activate the visibility of the Menu button, and load the LegoScene.
    b. Setup button, from Main Menu, calls butSetupClicked():
    → Deactivate visibility of the Main Menu panel, activate the visibility of the Setup Menu panel.
    c. Exit button, from Main Menu calls butExitClicked():
    → Close application.
    d. OK button, from Setup Menu, calls butSetupOkClicked():
    → Get user's choice of board size from input fields and save in PlayerPrefs, get user's choice of Simulation Mode or Game Mode and save in PlayerPrefs, get user's choice of difficulty and save in PlayerPrefs, get user's choice of breakable blocks and save in Player Prefs. Then deactivate visibility of setup panel and activate visibility of Main Menu panel.
    e. M button, which is visible only in LegoScene, calls butMenuClicked():
    → Check if Main Menu panel has activated, if so deactivate its visibility. If instead, Setup panel has activated, deactivate it. If neither Main Menu nor Setup Menu has active visibility, activate the visibility of Main Menu.
2. The EventManager script handles the listening of the Spawn event that the Game Mode of the program triggers, using UnityEvent objects and Dictionary objects.
3. Within the GameMaster script, the functions OnEnable() and OnDisable() start listening for the event Spawn and stop listening for the event Spawn, respectively. The Start() function generates the platform based on the data saved in PlayerPrefs. Also, based on PlayerPrefs, it calls the Spawn() function if the user selected Game Mode.
    a. The Spawn() function handles the generation of a single Lego block, giving it a random size, random position in the air, and random orientation.
        i. It passes the instantiated GameObject representing the Lego block to the function OutOfBounds to determine if the block's position exists beyond the platform.

Back To Top

            1. If the position lies outside the boundaries, then it gets passed to the Destroy function and Spawn gets called again to generate another block.
            2. If the block lies within the platform boundaries, then it gets a random Material assigned to it.
                a. If the user selected breakable mode for the blocks, Spawn() iterates through each child with a Box Collider (the block's child cubes) and assigns to each a BlockCollision script and a BlockMovement script. The first child cube receives the tag "block_tag", while all other child cubes get the tag "child_tag".
                b. If the user did not select breakable mode for the blocks, Spawn() assigns the BlockMovement and BlockCollision script to the parent object. The parent object also receives the tag "block_tag".

    b. The Update() function gets called each frame, wherein it checks if the user pressed the Q key. If so, the application quits. It also checks if the user selected Simulation Mode, and if so calls the SimulationMode() function.
    c. The SimulationMode() function generates a Lego block based on the reset time.
        i. Each Lego block has a random size, random position and random orientation.
        ii. It passes the instantiated GameObject representing the Lego block to the function OutOfBounds, same as the function Spawn() does, to determine if the block lies inside or outside the boundaries.
            1. If the block lies outside of the boundaries, then it gets passed to the function Destroy and control returns to Update() which will call SimulationMode() again.
            2. If the block lies within the boundaries, then it gets a random Material and the script BlockObject.
    d. The OutOfBounds function takes a GameObject as its only parameter and returns a boolean variable. It iterates through each Transform child in the GameObject and checks for a Box Collider to act only one the child cubes of the Lego block. It then checks each child cube's position against the boundary positions to determine whether the child cube lies inside or outside the boundary.

4. Within the BlockObject script's Start() function, it checks if the user selected breakable mode for the Lego blocks.

Back To Top

   a. If the user selected breakable mode, then each child cube in the block gets its own Rigidbody. Then it sets the drag of that Rigidbody to the difficulty setting the user chose.
   b. If the user did not select breakable mode, then the parent GameObject of the Lego block has a Rigidbody attached and the drag of that Rigidbody gets set to the difficulty setting of the user's choice.

5. The BlockCollision script's Start() function initializes a boolean variable allowMovement to true, representing the user's ability to move a block initially. It's OnCollisionEnter function gets called whenever a Lego block collides with something.
   a. OnCollisionEnter must check the tag of the colliding objects to prevent the Lego block from acting on repeated collisions with the same object or with its own child cubes (in the case of breakable mode). It does this with looking for the case where the colliding object does not have a tag of "block_tag" nor "child_tag". This allows collision only with object's containing the tag "base_tag" for plates making the platform and for those labeled "collision_tag".
      i. If this check passes, it checks the owning object's own tag for "block_tag" or "child_tag".
         1. If "block_tag", then this means the owning object has never collided with anything before. The owning object has its tag changed to "collision_tag", allowMovement becomes false to prevent movement of this block, and it triggers the event Spawn, to enable the generation of the next block.
         2. If "child_tag", then the tag changes to "collision_tag" and allowMovement becomes false, but no event triggers.
6. The BlockMovement script's Update() function checks if the user entered the appropriate movement keyboard keys, each frame.
   a. If "R" pressed
      i. Call Rotate()
         1. Checks if the object the script belongs to can move. If so, it calculates the degrees of rotation to rotate to. In either the breakable blocks mode or not, the parent object rotates. Rotate() performs a boundary check, by passing this object (or parent in the case of breakable blocks) to the OutOfBounds function in the GameMaster script.
            a. If out of bounds, then the object reverts to its original rotation, otherwise the object can keep this new rotation.

b. If “W” pressed
    i. Call MoveForward()
        1. Checks if the object the script belongs to can move. If so, it moves the block forward. Then it performs a boundary check, by passing this object (or parent in the case of breakable blocks) to the OutOfBounds function in the GameMaster script.
            a. If out of bounds, then the object reverts to its original position, otherwise the object can keep this new position.

c. If “S” pressed
    i. Call MoveBack()
        1. Checks if the object the script belongs to can move. If so, it moves the block backward. Then it performs a boundary check, by passing this object (or parent in the case of breakable blocks) to the OutOfBounds function in the GameMaster script.
            a. If out of bounds, then the object reverts to its original position, otherwise the object can keep this new position.

d. If “D” pressed
    i. Call MoveRight()
        1. Checks if the object the script belongs to can move. If so, it moves the block to the right. Then it performs a boundary check, by passing this object (or parent in the case of breakable blocks) to the OutOfBounds function in the GameMaster script.
            a. If out of bounds, then the object reverts to its original position, otherwise the object can keep this new position.

e. If “A” pressed
    i. Call MoveLeft()
        1. Checks if the object the script belongs to can move. If so, it moves the block to the left. Then it performs a boundary check, by passing this object (or parent in the case of breakable blocks) to the OutOfBounds function in the GameMaster script.

Back To Top

            a. If out of bounds, then the object reverts to its original position, otherwise the object can keep this new position.

7. The CameraManager's Start() function gets the point in the center of the platform and makes the camera look at this point, position the camera a variable distance away from the platform (depending on size of platform), and angles it 45 degrees to look somewhat isometric. The Update() function checks every frame for the appropriate user keyboard input.
    a. If right arrow pressed or held
        i. The camera's transform calls RotateAround to rotate to the right, around the center point of the platform.
    b. If left arrow pressed or held
        i. The camera's transform calls RotateAround to rotate to the left, around the center point of the platform.
    c. If up arrow pressed or held
        i. The camera's position increases in the y direction.
    d. If down arrow pressed or held
        i. The camera's position decreases in the y direction.
    e. If Z key pressed or held
        i. The camera's position changes in the positive direction of its forward vector.
    f. If X key pressed or held
        i. The camera's position changes in the negative direction of its forward vector.

3 System Behavior

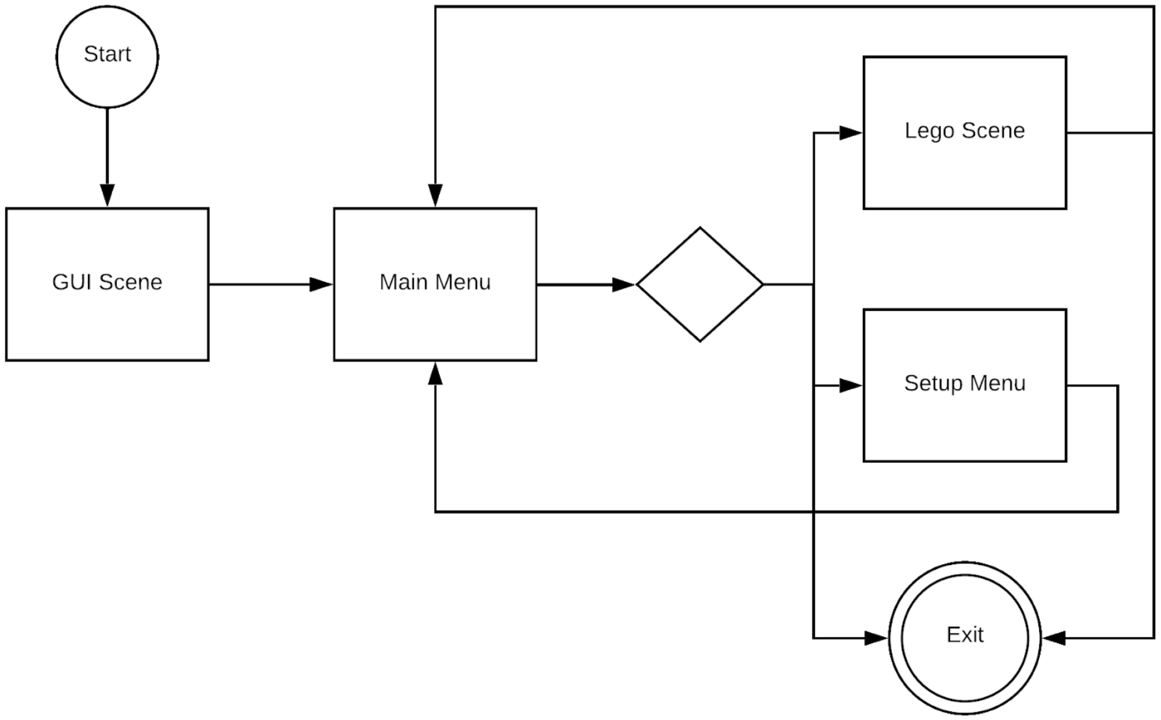
3.1 System Components

The user has varying available controls based on the scene.

1. "GUIScene":
   This scene activates when the application first runs. The application presents the user with three buttons, which allows them to start the scene based on settings saved in Player Preferences, change those settings, and to exit the application.
2. "LegoScene"
   This scene activates when the user clicks on the Start button from the Main Menu. It generates the platform and the Lego blocks. Regardless of game mode chosen, the user can move the camera using the keyboard. The user also has access to a Menu button, labeled M, that lets them access the Main Menu and

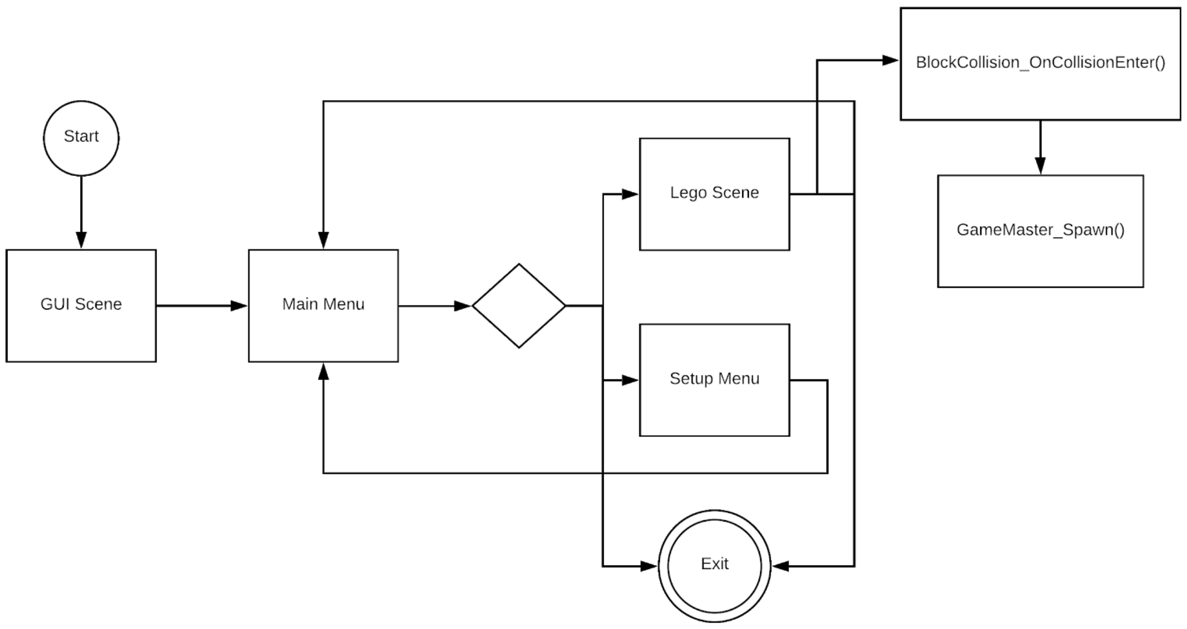
Back To Top

from that the Setup Menu to change settings. If the user chooses the playable Game Mode, then they can move each block based on keyboard input. At all times, the user can close the application based on keyboard input.



Back To Top

3.2 System Components with Events



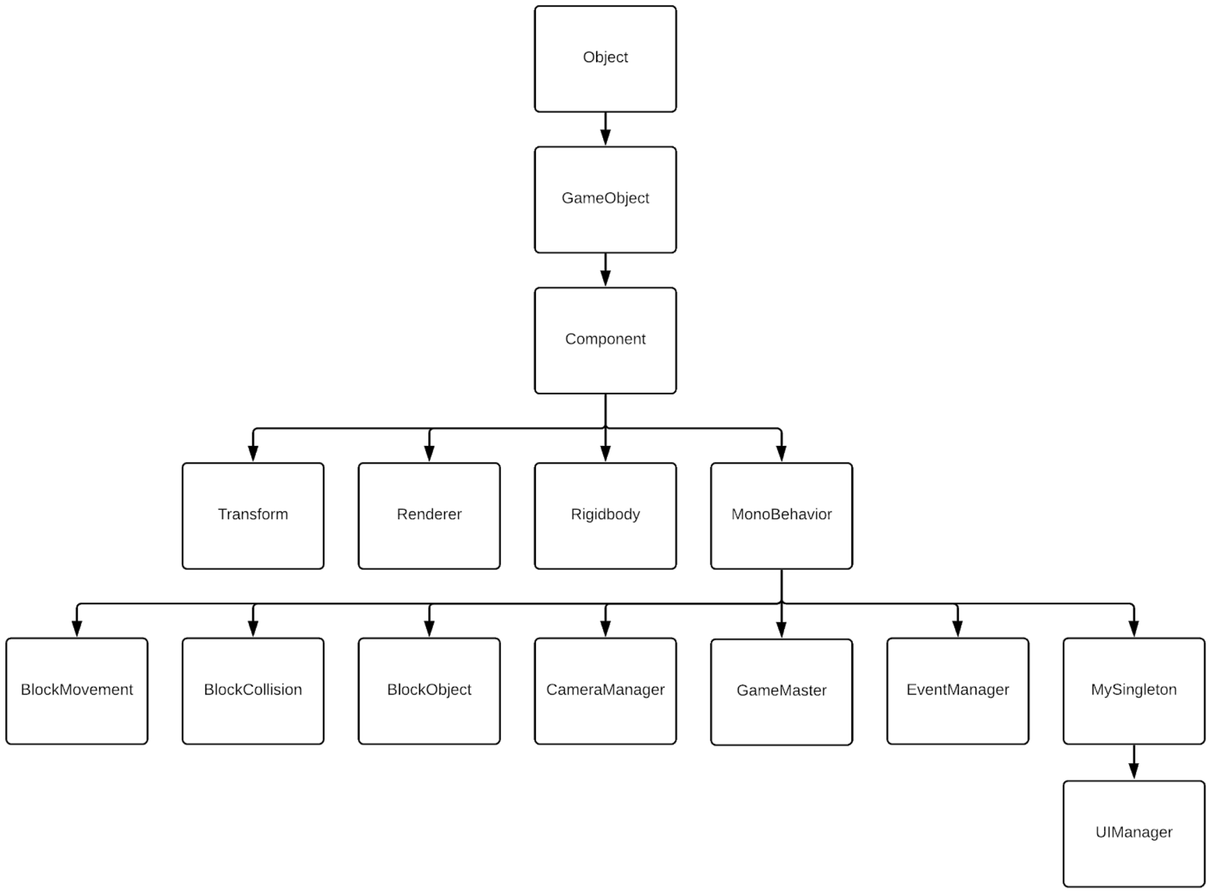
Back To Top

4 Logical View

The logical view describes the main functional components of the system.

4.1 Unity Class Hierarchy

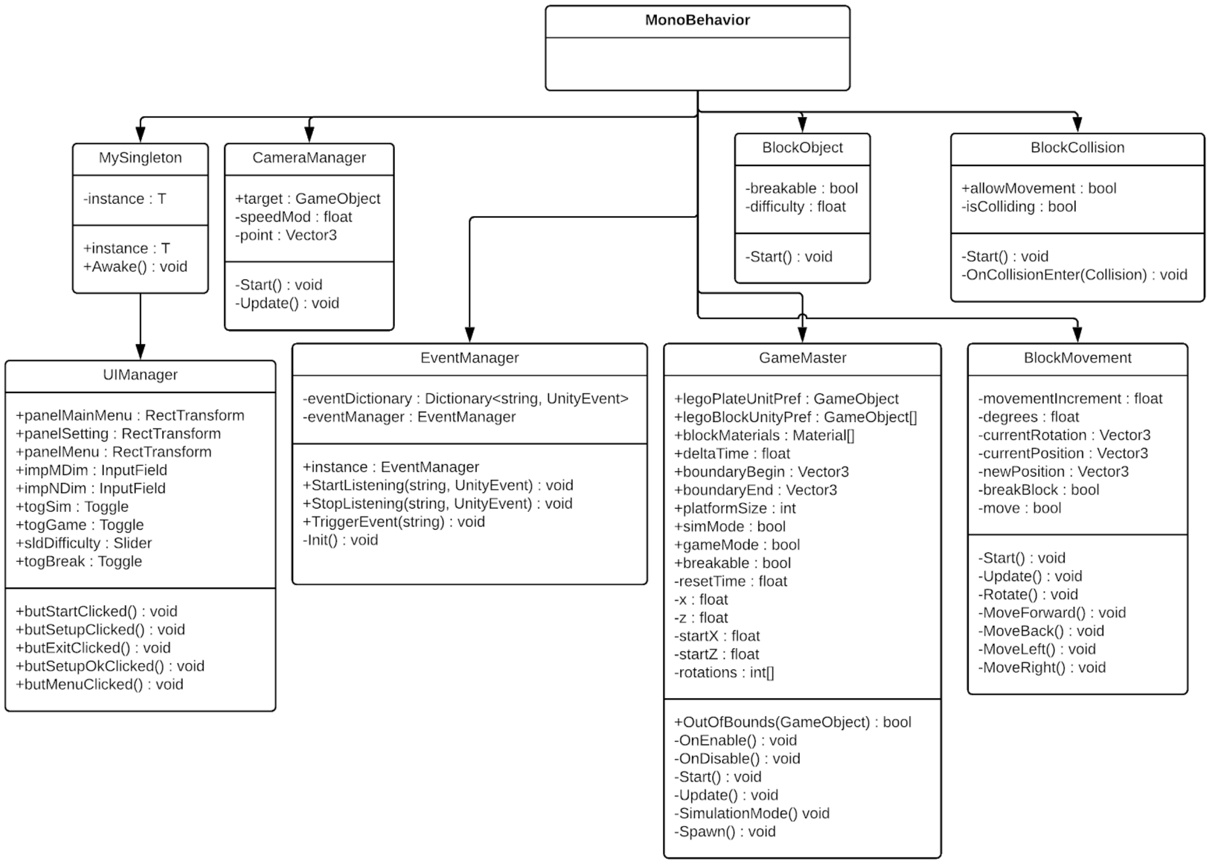
The Unity Class Hierarchy displays the relationship of classes within the Unity framework. In Unity, all user defined components inherit from the GameObject class.



Back To Top

4.2 Detailed Class Design

A detailed UML diagram of the scripts in the program.



5 User Interaction

a) Clickable Buttons
   i) Start: Used to load the Lego Scene
   ii) Setup: Used to activate the Setup Menu
   iii) Exit: Used to quit the application
   iv) Ok: Used to save the changes made to settings in the Setup Menu
   v) M: Used to activate/deactivate the Main Menu from the Lego Scene, depending on whether a menu is already active or not

b) Input Fields
   i) Enter N: Used to type the number of Lego plates for the platform along the x-dimension
   ii) Enter M: Used to type the number of Lego plates for the platform along the z-direction

c) Sliders

Back To Top

        i) Difficulty: Used to set the drag of the falling Lego blocks. By default, it is set to Hard (drag = 0).

d) Toggles
    i) Simulation: Used to set the mode of play to Simulation, wherein the Lego blocks will fall without user movement
    ii) Game: Used to set the mode of play to Game, wherein the user can move the Lego blocks
    iii) Breakable: Used to set whether the Lego blocks break upon impact or not.

e) Keyboard Input
    i) W Key: Moves the Lego block forward, if in Game mode
    ii) A Key: Moves the Lego block left, if in Game mode
    iii) S Key: Moves the Lego block down, if in Game mode
    iv) D Key: Moves the Lego block right, if in Game mode
    v) Z Key: Zoom camera in
    vi) X Key: Zoom camera out
    vii) Up Arrow: Move camera up
    viii) Down Arrow: Move camera down
    ix) Left Arrow: Move camera left
    x) Right Arrow: Move camera right

6 User Interface

A separate scene, called GUIScene, holds all of the User Interface elements. This scene contains the main menu, the setup menu, and a button to open the main menu. An empty game object acts as the parent to all of these user interface elements, including an EventSystem. Additionally, the panels that hold the main menu elements, settings elements, and menu button have a Canvas as a parent.

The main menu simply contains three buttons: start, setup, and exit. The start button loads the LegoScene which serves as the scene for the main game/simulation. The setup button disables visibility of the main menu and enables visibility of the setup menu. The exit button terminates the game and returns to the Unity editor (if the Unity editor started the game).

The setup menu contains several settings that change the functionality of the game/simulation. First, the user can enter a size for the board/platform, with input fields for the x dimension and y dimension. Second, the user can select Simulation mode or Game mode. Toggles handle this selection, with a toggle group that prevents the user selecting both modes or deselecting both modes. By default, Simulation mode is active.

Back To Top

Third, the user can change the difficulty of the simulation or game. A slider captures how easy/hard the user wants, which essentially serves as the value of the blocks' rigidbody drag. The closer to the easy end of the slider the user chooses, the greater the drag and the slower the blocks will fall. Conversely, the closer to the hard end of the slider the user chooses, the lower the drag and the faster the blocks will fall. By default, the slider starts at the Hard end of the slider, which equates to a drag value of 0. Lastly, the user can choose whether the blocks will break or not. A single toggle captures this choice, with a default setting of Yes (break apart blocks). Finally, a button labeled ok allows the user to save these settings and apply them.

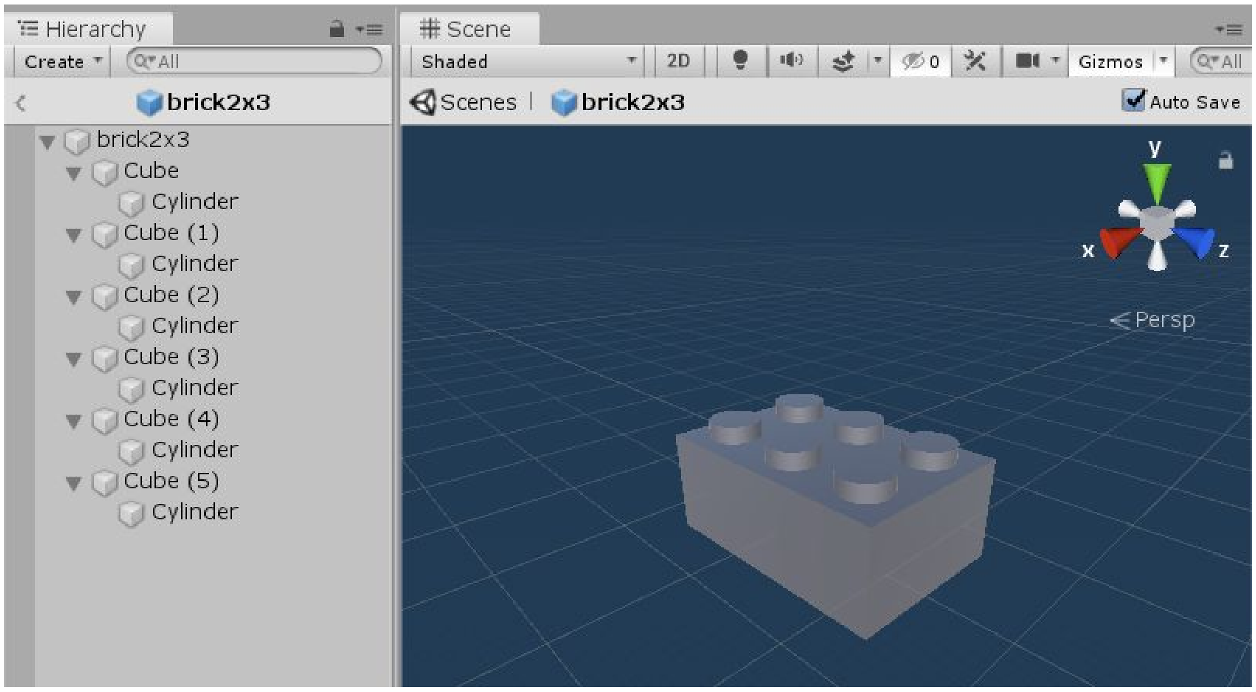
The menu button only becomes visible in the LegoScene, once the user clicks the start button from the main menu. It gives the user the option to open the main menu from the LegoScene, for the purpose of changing the settings without quitting and restarting the game. For these changes to apply, the user must click the start button to reload LegoScene.

The script UIManager handles all of the functionality of the user interface. As a Singleton, it maintains existence across scenes. The function butStartClicked() executes when the user clicks on the start button. It sets the panel containing the main menu as inactive (not visible), the panel containing the menu button as active (visible), and loads the LegoScene. The function butSetupClicked() executes when the user clicks on the setup button. It simply deactivates (makes invisible) the panel containing the main menu and activates (makes visible) the panel containing the setup menu. The function butExitClicked() executes when the user clicks on the exit button. It checks if the Unity Editor is playing or not. If it is, then UnityEditor.EditorApplication.isPlaying is set to false; if not, then Application.Quit() gets called. The function butSetupOkClicked() executes when the user clicks the OK button from within the setup menu. It saves the user's choices of board dimensions, simulation mode vs game mode, difficulty level, and whether to break the blocks, in the Player Prefs. It also deactivates the panel containing the setup menu and activates the panel containing the main menu. The function butMenuClicked() executes whenever the user clicks the menu button. It simply activates the panel containing the main menu, but only if the panel containing the setup menu is not active, and deactivates the panel containing the main menu if the main menu is currently visible.

Back To Top

7 Screenshots

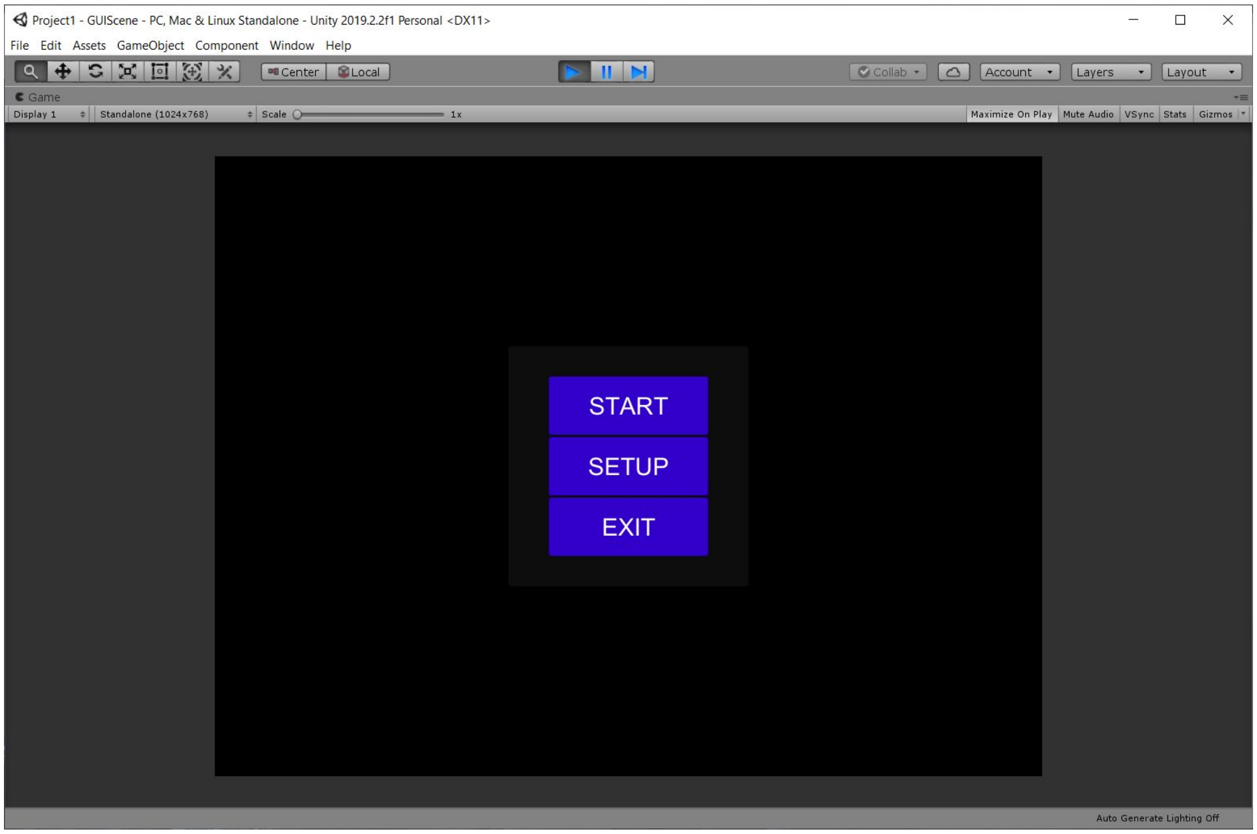
1) Example Lego block prefab, in this case a 2x3 brick, to show how the modeling of the blocks.



brick2x3
- Cube
  - Cylinder
- Cube (1)
  - Cylinder
- Cube (2)
  - Cylinder
- Cube (3)
  - Cylinder
- Cube (4)
  - Cylinder
- Cube (5)
  - Cylinder

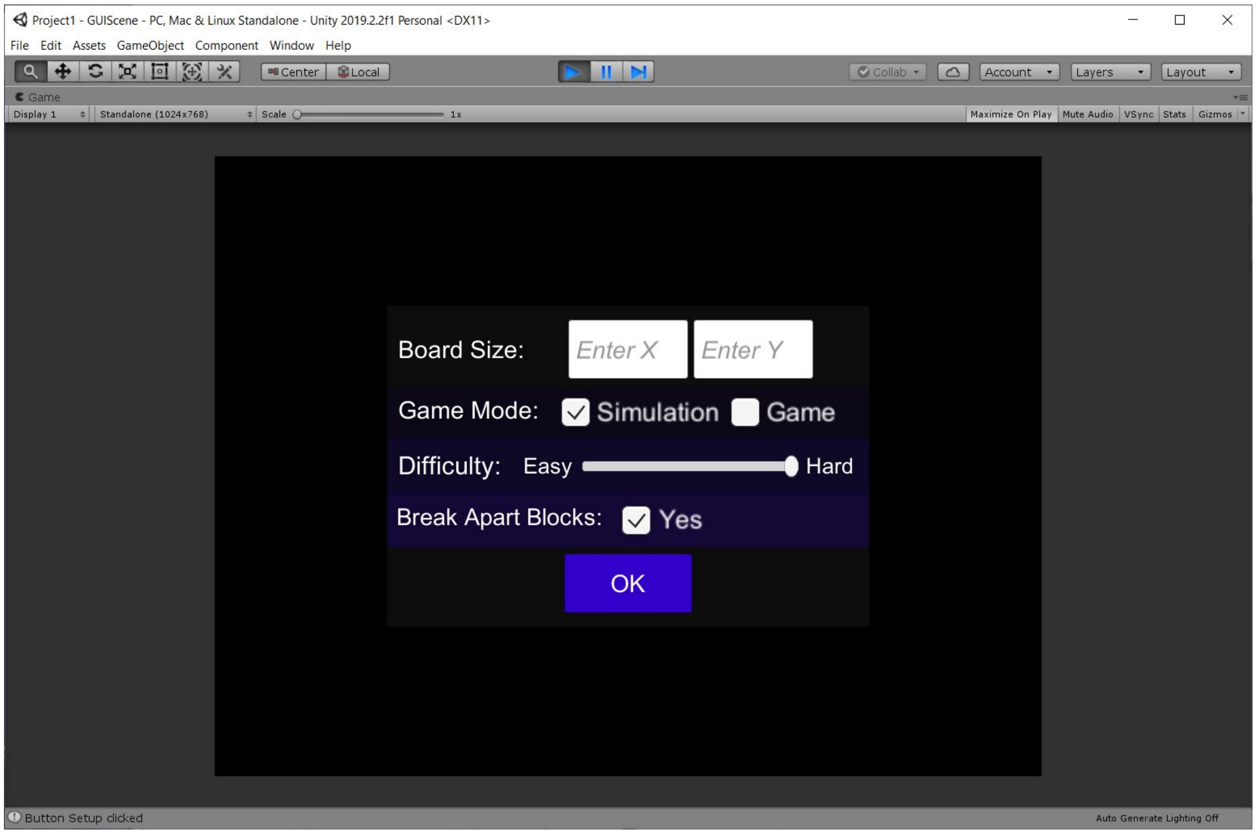
Back To Top

2) The starting scene of the application, the Main Menu from the GUIScene


Project1 - GUIScene - PC, Mac & Linux Standalone - Unity 2019.2.2f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Center
Local
Collab
Account
Layers
Layout
Game
Display 1
Standalone (1024x768)
Scale 1x
Maximize On Play
Mute Audio
VSync
Stats
Gizmos
START
SETUP
EXIT
Auto Generate Lighting Off

Back To Top

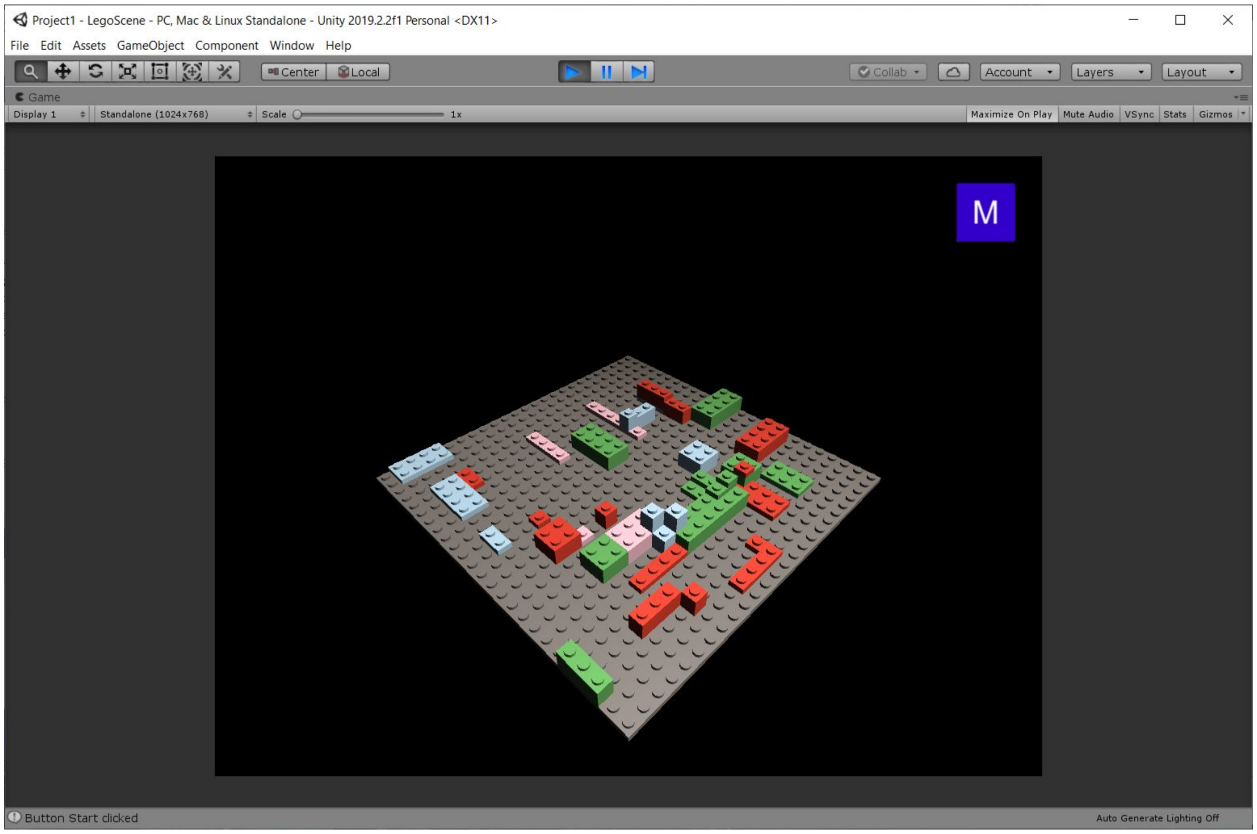
3) After clicking the “Setup” button, the Setup Menu becomes visible.



Button Setup clicked

Back To Top

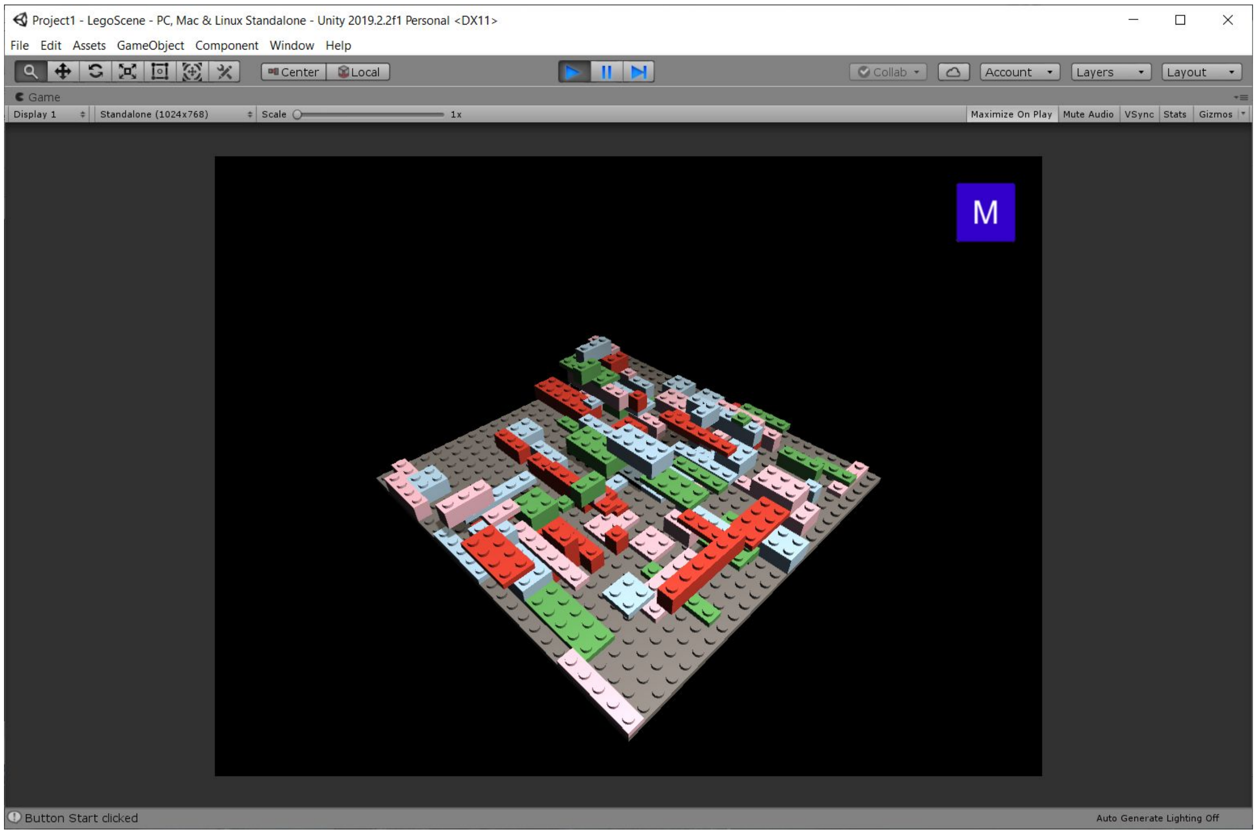
4) Now in the “LegoScene” scene after clicking the “Start” button, and selecting to break apart blocks


Project1 - LegoScene - PC, Mac & Linux Standalone - Unity 2019.2.2f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Game
Display 1 Standalone (1024x768) Scale 1x
Maximize On Play Mute Audio VSync Stats Gizmos
M
Auto Generate Lighting Off

Button Start clicked

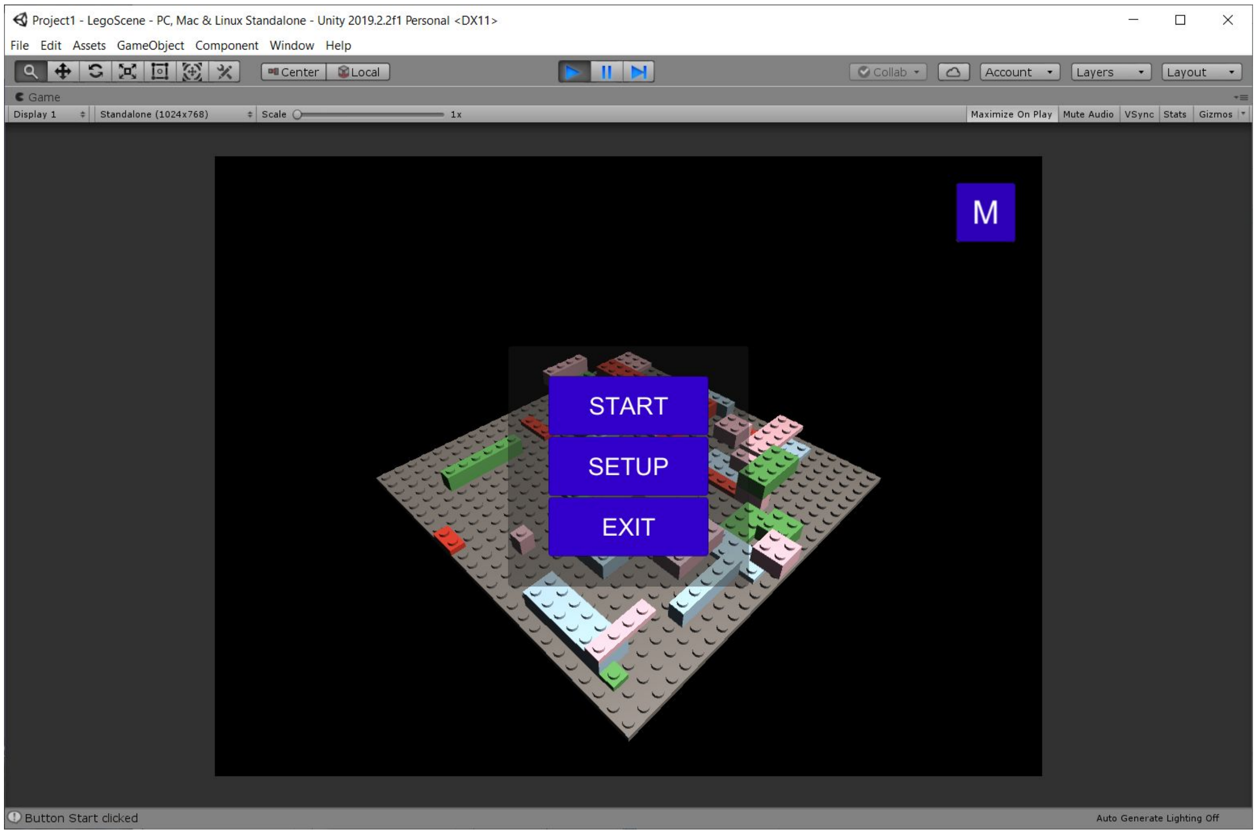
[Back To Top]

5) Now in the “LegoScene” scene after clicking the “Start” button, and selecting to not break apart blocks


Project1 - LegoScene - PC, Mac & Linux Standalone - Unity 2019.2.2f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Game
Display 1 Standalone (1024x768) Scale 1x
Maximize On Play Mute Audio VSync Stats Gizmos
M
Button Start clicked
Auto Generate Lighting Off

Back To Top

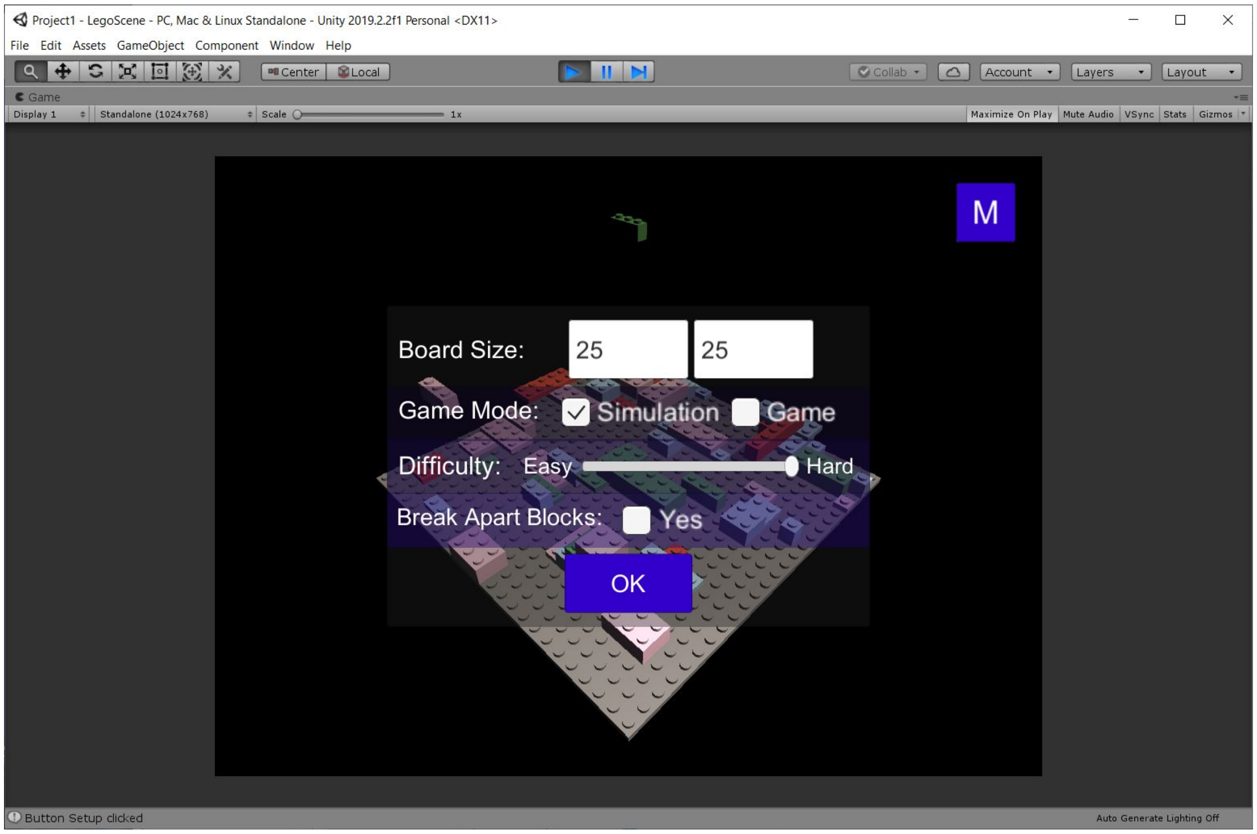
6) Main Menu, as accessed from within LegoScene from clicking the M (Menu) button.


Project1 - LegoScene - PC, Mac & Linux Standalone - Unity 2019.2.2f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Center
Local
Collab
Account
Layers
Layout
Game
Display 1
Standalone (1024x768)
Scale 1x
Maximize On Play
Mute Audio
VSync
Stats
Gizmos
M
START
SETUP
EXIT
Auto Generate Lighting Off

Button Start clicked

Back To Top

7) Setup Menu, as accessed from within LegoScene.


Project1 - LegoScene - PC, Mac & Linux Standalone - Unity 2019.2.2f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Game
Display 1 Standalone (1024x768) Scale 1x
Maximize On Play Mute Audio VSync Stats Gizmos
M
OK
Auto Generate Lighting Off

Board Size: 25 25

Game Mode: [✓ Simulation] Game

Difficulty: Easy Hard

Break Apart Blocks: Yes

Button Setup clicked

8 Lessons Learned

There remains one issue I could not fix before the project deadline. In game mode, with breakable blocks, the blocks move too far if they are greater in size than 1x1. Apart from this one issue, this program meets the requirements of the assignment.

I found the greatest challenge, in this project, with getting the blocks to break on impact in the playable game mode. In my first attempt, I used Coroutines to iterate each drop of a block. While this worked with the uniform model of the blocks, it produced erroneous behavior for the breakable blocks. At first it performed well, but after running the program several more times, it stopped working as desired. The breakable blocks would generate multiples at a time, or sometimes only at most four would ever drop. I still do not have an explanation for why that happened. Instead, I decided to research Unity Events, listeners, and event triggers. This ended up solving nearly all of my problems.

I also learned that Unity does not export/import any object tags not assigned manually to any object. Since, I employ tags, in my collision detection logic, and assign them at runtime to the Lego blocks, I had to find a workaround to this problem. For my solution, I

Back To Top

made a copy of a brick prefab named test_brick and assigned the tags I wished to have exported/imported on the test_brick's parent object and child cubes. After testing with exporting and importing the Unity project, I can confirm that this solves this problem, although not in the most elegant way.

Overall, with this project, I learned a considerable amount on moving cameras, collision detection (and the usefulness of GameObject tags), user interfaces, keyboard input, Singletons, and Unity Events.

9 Summary

The overall implementation of Project 1 proved successful. The project employs several scripts to break up the program logic in a readable and organized manner. Each script uses its Start() function to initialize its attributes, and in the case of GameMaster, builds the platform. Each script employs their respective Update() functions for aspects that need changing each frame. Collision detection and Events provide the triggering of each block to spawn in the playable Game Mode, since code within an Update() function alone could not handle this functionality.

Apart from one minor issue, as detailed in the section on Lessons Learned, this implementation meets all of the outlined objectives. The user interface allows the user to start the Lego Scene and adjust important settings such as how large to make the platform, to spectate with Simulation Mode or move blocks with the playable Game Mode, to adjust the difficulty (rate of fall of the blocks), and to make the blocks break upon impact or not. The user interface also allows the user to exit the application with a button. The user can successfully move the camera around in all modes of play in the Lego Scene, as desired. In the playable Game Mode, the user can also move the blocks before the block hits either the platform or another block.

Back To Top