



SUPINFO
International University

INSTITUTE OF INFORMATION TECHNOLOGY

Class Group Project Development

Delivery

Rahul THAKOOR, Kaushik HURNAUM, Priyeshan SREENEEBUS,
Muhammad Umar CALLACHAND.
Documentation

Conditions d'utilisations : SUPINFO International University vous permet de partager ce document. Vous êtes libre de :

- Partager — reproduire, distribuer et communiquer ce document
- Remixer — modifier ce document

A condition de respecter les règles suivantes :

Indication obligatoire de la paternité — Vous devez obligatoirement préciser l'origine « SUPINFO » du document au début de celui-ci de la même manière qu'indiqué par SUPINFO International University – Notamment en laissant obligatoirement la première et la dernière page du document, mais pas d'une manière qui suggérerait que SUPINFO International University vous soutiennent ou approuvent votre utilisation du document, surtout si vous le modifiez. Dans ce dernier cas, il vous faudra obligatoirement supprimer le texte « SUPINFO Official Document » en tête de page et préciser notamment la page indiquant votre identité et les modifications principales apportées.

En dehors de ces dispositions, aucune autre modification de la première et de la dernière page du document n'est autorisée.

NOTE IMPORTANTE : Ce document est mis à disposition selon le contrat CC-BY-NC-SA Creative Commons disponible en ligne <http://creativecommons.org/licenses> ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA modifié en ce sens que la première et la dernière page du document ne peuvent être supprimées en cas de reproduction, distribution, communication ou modification. Vous pouvez donc reproduire, remixer, arranger et adapter ce document à des fins non commerciales tant que vous respectez les règles de paternité et que les nouveaux documents sont protégés selon des termes identiques. Les autorisations au-delà du champ de cette licence peuvent être obtenues à support@supinfo.com.

© SUPINFO International University – EDUCINVEST - Rue Ducale, 29 - 1000 Brussels Belgium . www.supinfo.com

Table of contents

1	GROUP SUMMARY	4
1.1	GROUP MEMBERS	4
	231580.....	4
	THAKOOR.....	4
	RAHUL.....	4
	231581.....	4
	HURNAUM	4
	KAUSHIK	4
	264144.....	4
	SREENEEBUS	4
	PRIYESHAN	4
	223174.....	4
	CALLACHAND	4
	MUHAMMAD UMAR.....	4
2	PROJECT REPORT.....	5
3	SOLUTION MANUAL	6
4	TECHNICAL DOCUMENTATION	15

1 GROUP SUMMARY

1.1 GROUP MEMBERS

Campus: **Mauritius**

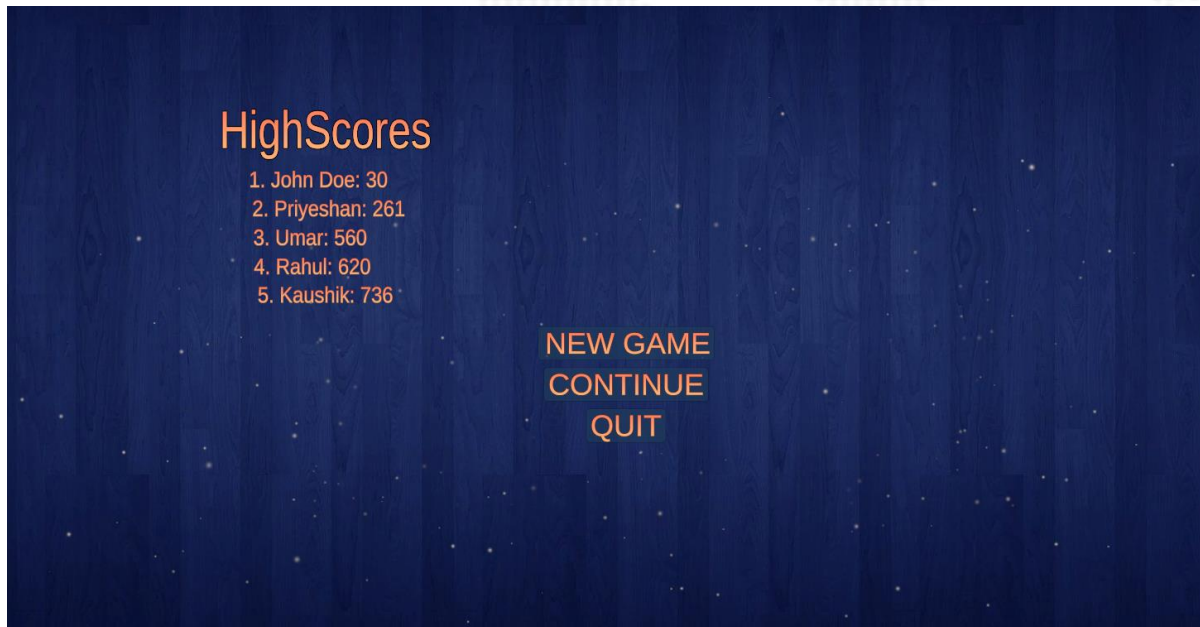
Class: **B2**

ID Open Campus	Last Name	First Name	Photo
<u>231580</u>	<u>THAKOOR</u>	<u>RAHUL</u>	
<u>231581</u>	<u>HURNAUM</u>	<u>KAUSHIK</u>	
<u>264144</u>	<u>SREENEEBUS</u>	<u>PRIYESHAN</u>	
<u>223174</u>	<u>CALLACHAND</u>	<u>MUHAMMAD UMAR</u>	

2 PROJECT REPORT

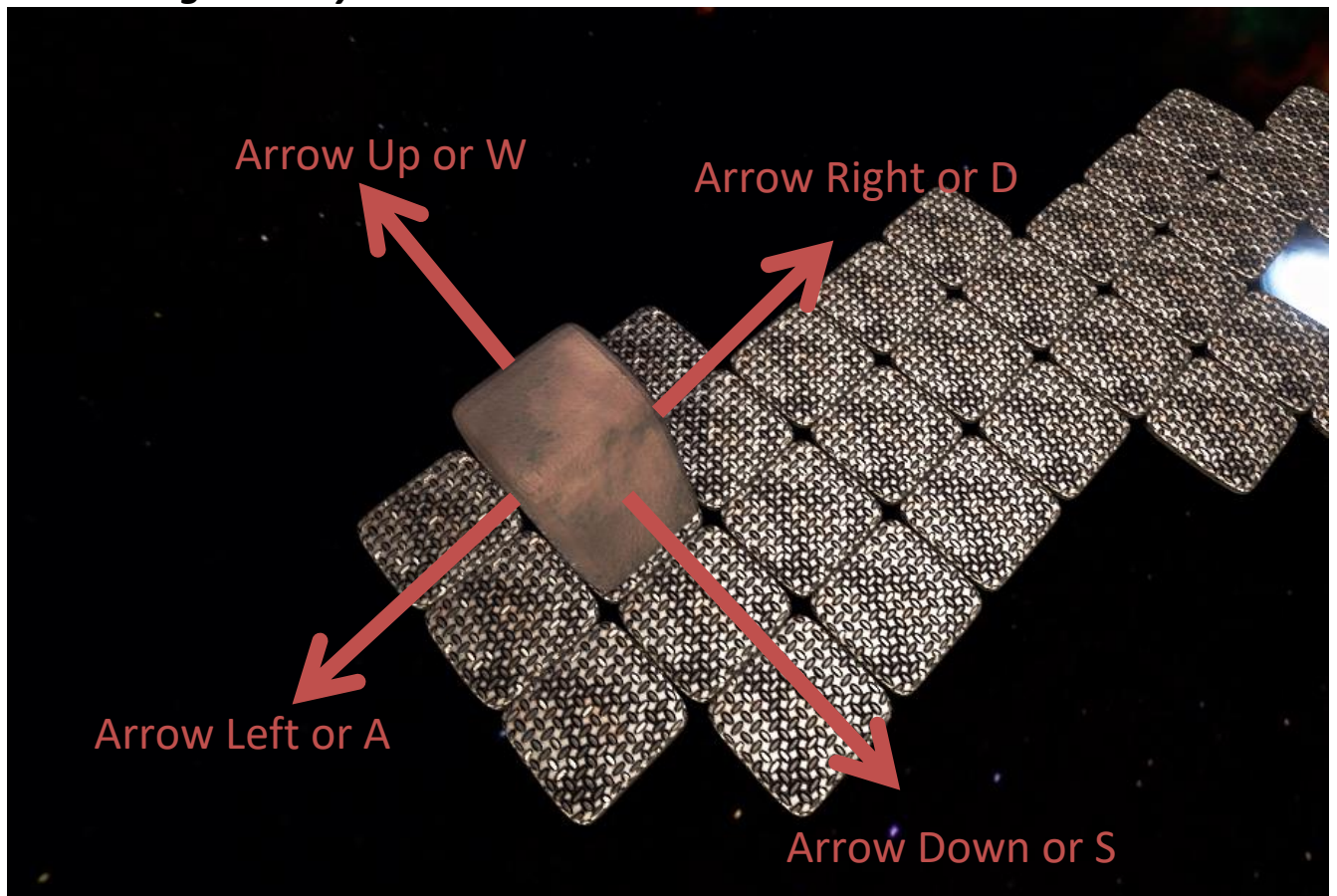
First Name	Tasks
Priyeshan	Animations, Teleportation, Sound, Cuboid Rotate, Save and Load
Kaushik	Score handling with SQLite, Save and Load, End Screen
Rahul	Level and Component Design, Timer, Wooden Tiles, Start Menu, Score handling
Umar	Switches and Bridges, Cuboid Rotation, Cuboid Falling

3 SOLUTION MANUAL



On the Start Screen, we can choose to Start a fresh game, load a previously saved game or Quit the game. The highscores of previously played games are displayed.

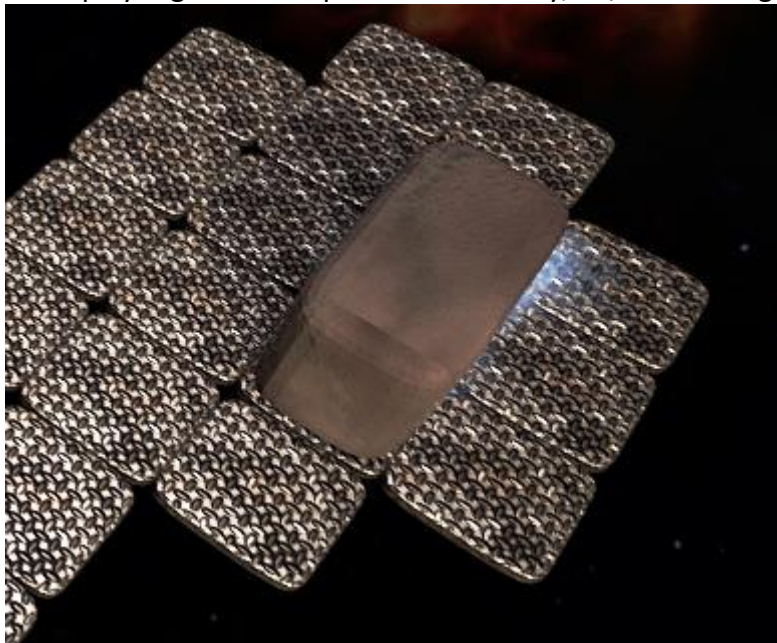
Controlling the Player



The Player should control the Cuboid and drop it vertically through the portal.



If the player gets on the portal horizontally, he/she won't get through next stage.



Furthermore, the player should be very cautious not to fall off either by moving the cuboid completely off the tiles or just half of the cuboid.

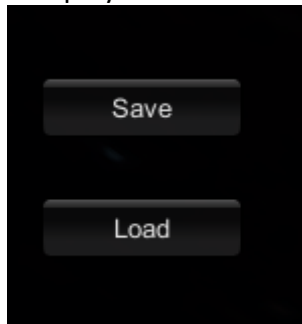


Score:

The score is incremented whenever the player moves the cuboid. The smallest number of moves, i.e., score, is the best score. It is displayed at the top left side of the screen.



The player can also save and load game by clicking on the buttons at the left side of the screen:

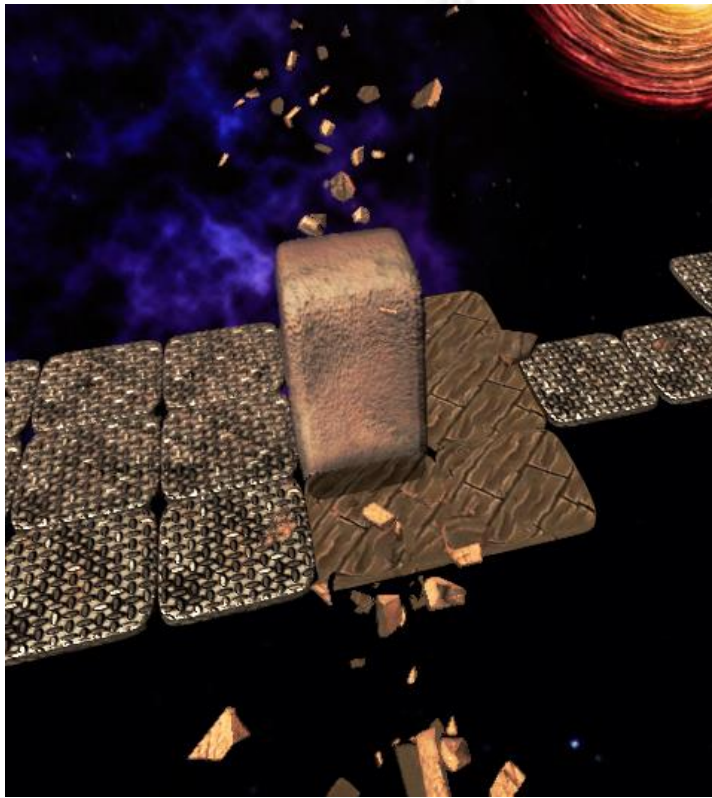


A countdown timer can be started whenever the player wishes to and if the timer ends before going through the portal, the level will restart. The countdown timer is set to 30 seconds.

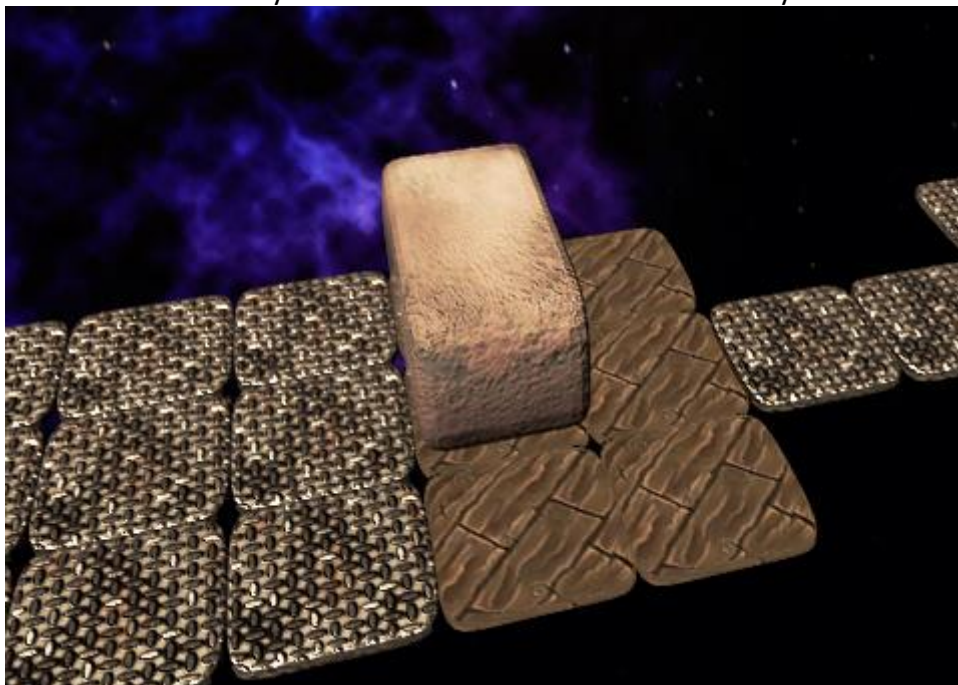


Wooden tiles

The wooden tiles are fragile and cannot support the weight of the whole cuboid. If the player stands vertically on a wooden tile, it will break causing the cuboid to fall.



The cuboid can safely move on the wooden tiles horizontally.

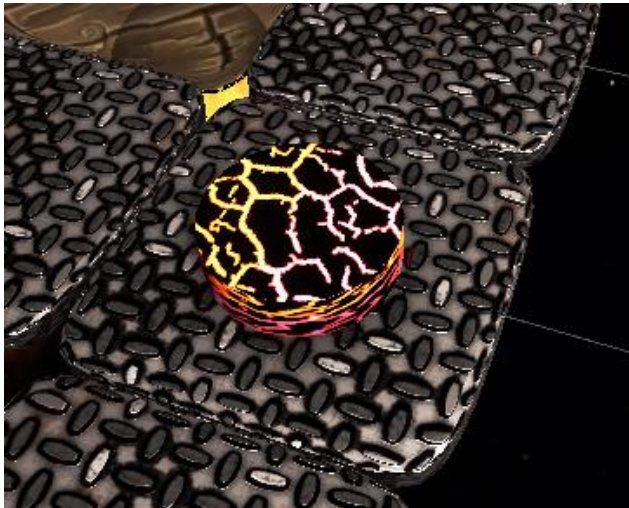


Switches and Bridges:

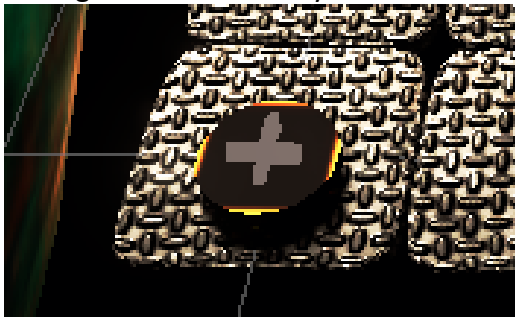
There are 2 types of bridges: normal switch and strong switch. Both are used to activate bridges so the cuboid can move to the portal.

The strong switch is only activated when the cuboid stands vertically on it.

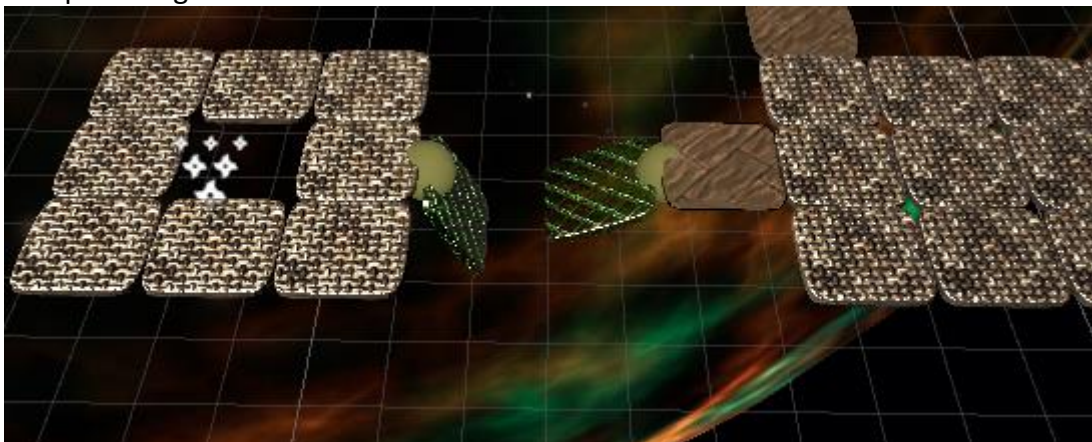
Normal Switch which is rounded:



Strong switch marked by an X:

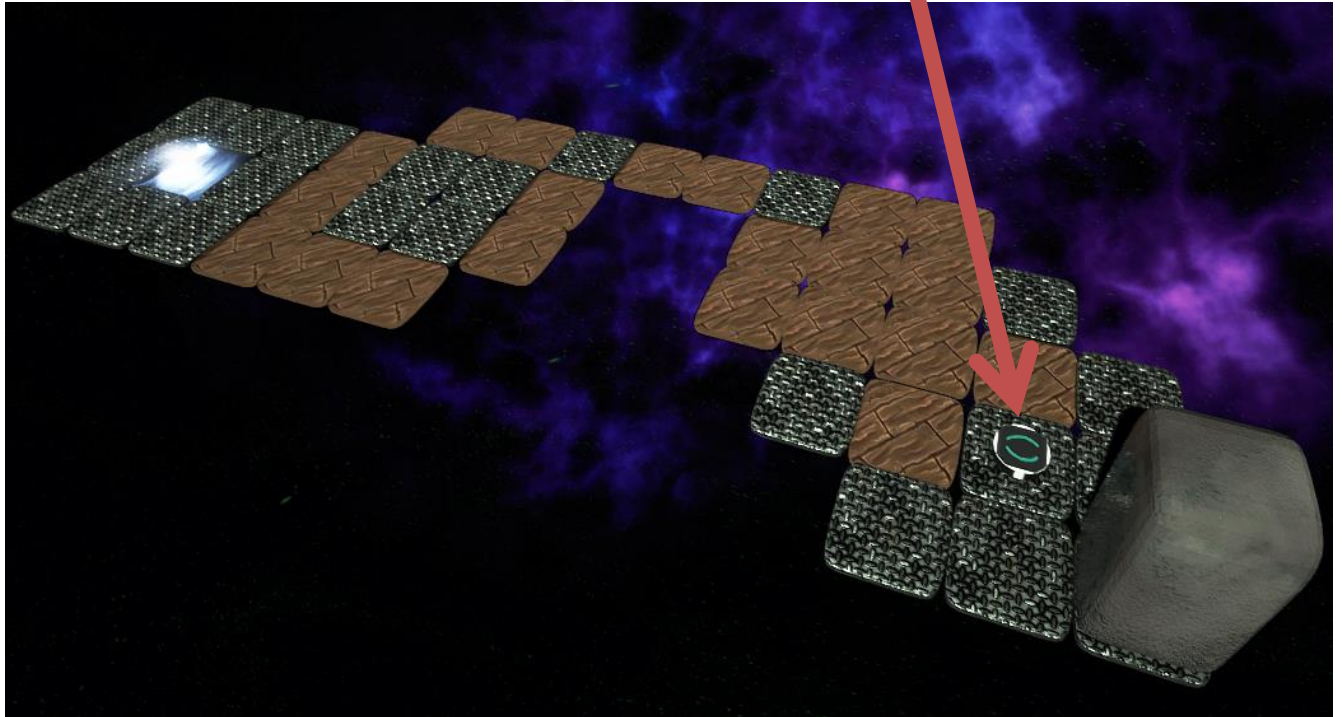


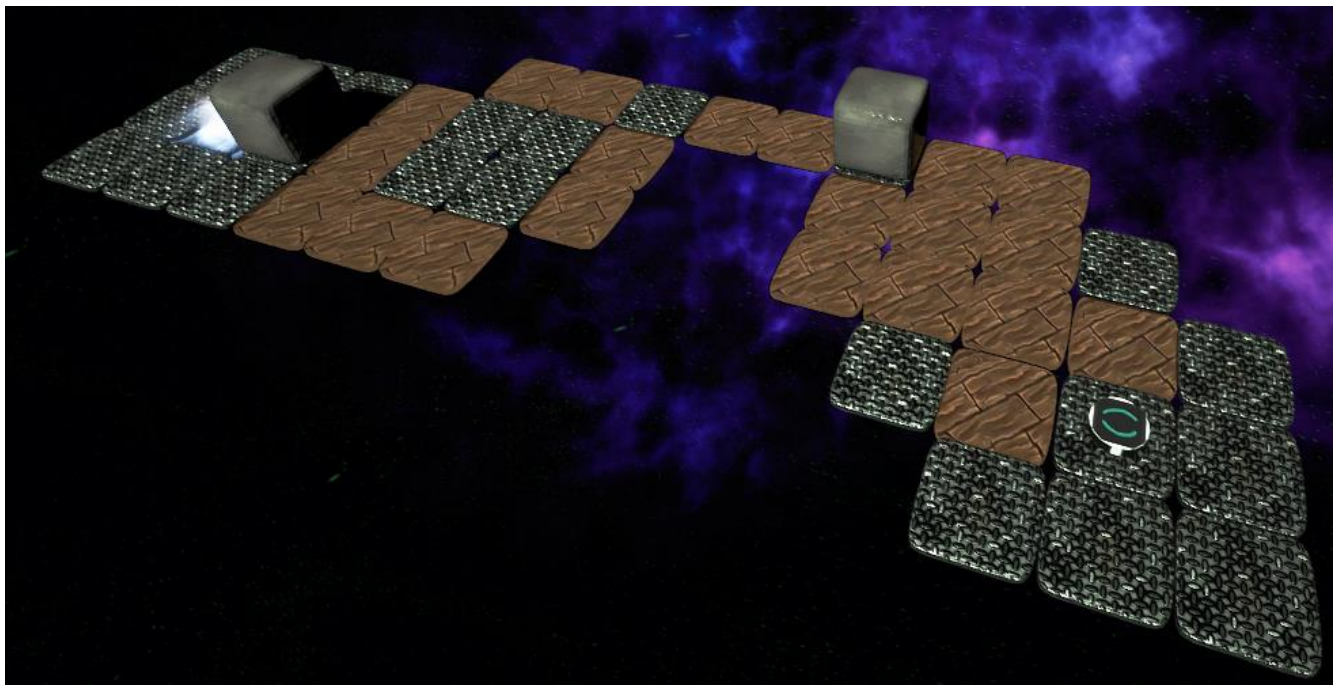
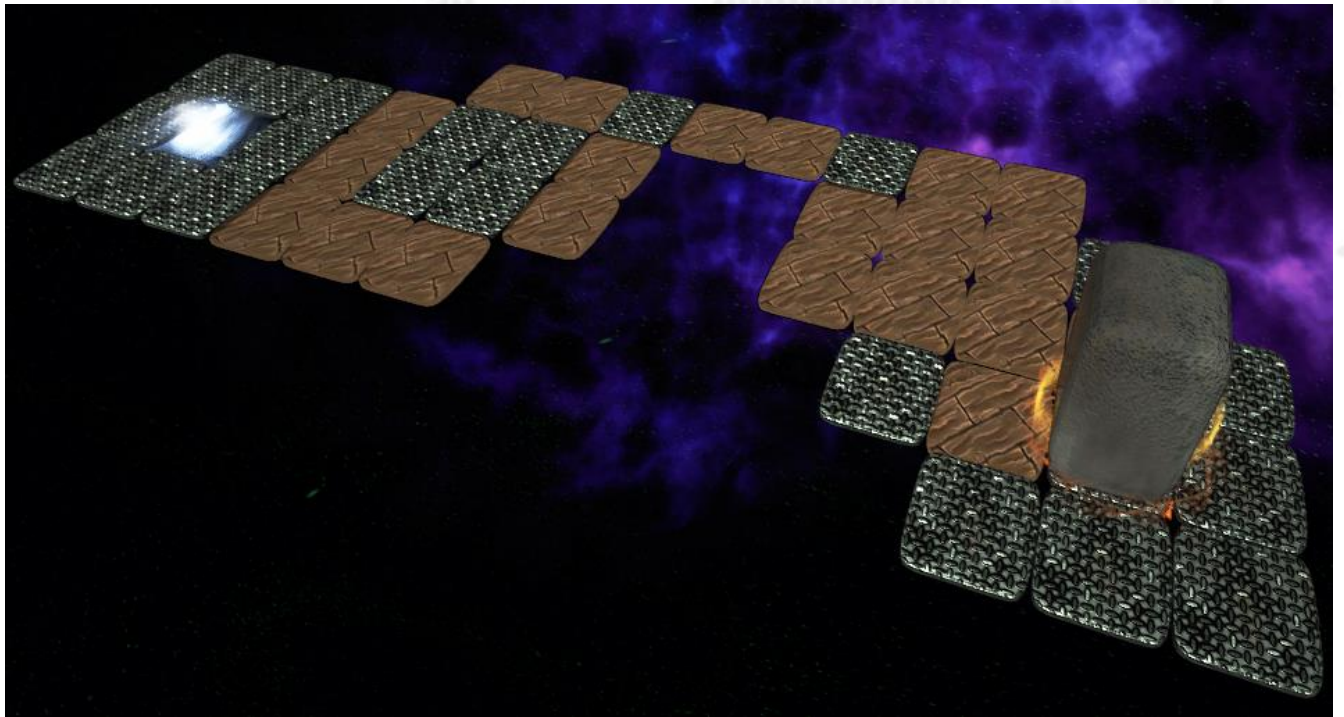
An open bridge:



Teleportation

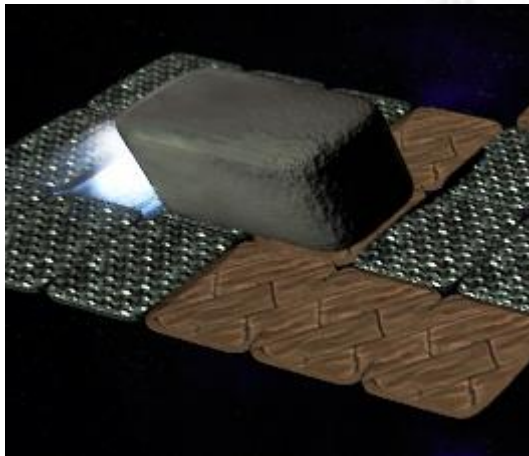
The cuboid can be teleported! If the cuboid stands on a teleportation switch (marked by ()), it splits into 2 cubes at 2 different places.



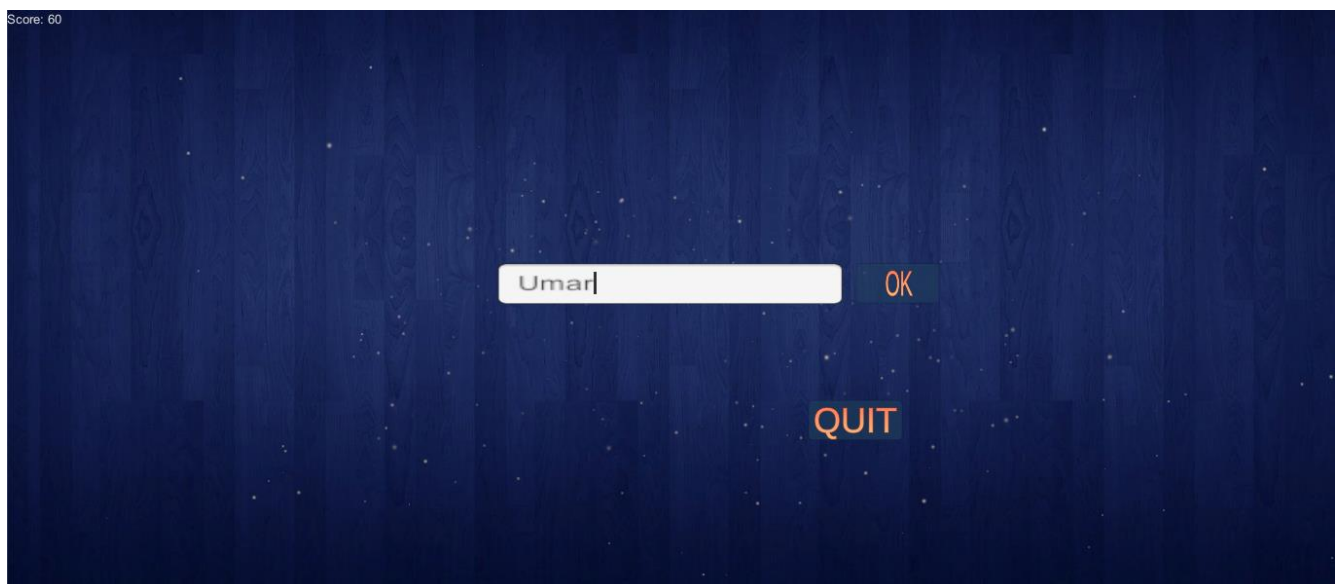


The cubes are moved separately, the player should press Tab key to switch cube.

When the 2 cubes touch each other, they are joined back to the cuboid:



At the end of the 10 stages, the player will submit his/her name and the top 5 high scores are displayed.



HighScores

1. John Doe: 30
2. Umar: 60
3. Priyeshan: 163
4. Rahul: 302
5. Kaushik: 512

QUIT

4 TECHNICAL DOCUMENTATION

Graphical Engine Used

The game engine Unity was used for making this game. Unity is a very powerful game engine highly praised for its ease of use. To display 3d objects on the screen, we simply need to drag and drop the models, here we used 3d objects designed from Blender, into the Scene view. Unity comes with various tools to enhance game visuals and also a service for collaborating online. No need for git! Most importantly, for scripting, Unity uses C# which is a powerful programming language. Unity is cross platform, one script to rule them all. Code once and deploy on Windows, linux and MacOS all at once.

Moves and Gravity

To move the cuboid, the player flips it with the four arrow keys. The whole base area of the cuboid must be on the grid, if not it falls.

```
if (_moving)
{
    float deltaRotation = rotationSpeed * Time.deltaTime;
    if (_totalRotation + deltaRotation >= 90)
    {
        deltaRotation = 90 - _totalRotation;
        _moving = false;
    }
    if ((_rotationDirection == Direction.West) || (_rotationDirection == Direction.North))
        transform.RotateAround(_pivot, _axis, deltaRotation);
    else transform.RotateAround(_pivot, _axis, -deltaRotation);

    _totalRotation += deltaRotation;
    //play audio
    if (_roll)
    {
        _audioManager.Play(_soundName);
        _roll = false;
    }
}

else if ((Input.GetKeyDown(KeyCode.D) || Input.GetKeyDown(KeyCode.RightArrow)) && IsEnable) Rotate(Direction.North);
else if ((Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.UpArrow)) && IsEnable) Rotate(Direction.West);
else if ((Input.GetKeyDown(KeyCode.A) || Input.GetKeyDown(KeyCode.LeftArrow)) && IsEnable) Rotate(Direction.South);
else if ((Input.GetKeyDown(KeyCode.S) || Input.GetKeyDown(KeyCode.DownArrow)) && IsEnable) Rotate(Direction.East);
```

If the player presses on S, the cuboid will rotate in the East Direction. Direction is an enum defined as shown:

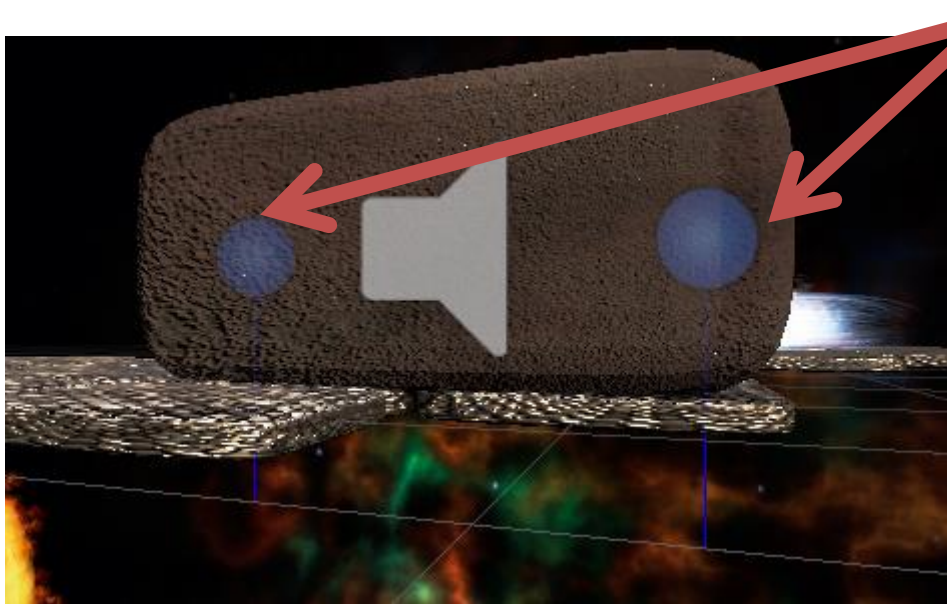
```
public enum Direction
{
    North,
    South,
    East,
    West,
    Up
}

void Rotate(Direction direction)
{
    _rotationDirection = direction;
    _moving = true;
    _totalRotation = 0;
    CuboidGameObject.transform.localPosition = Vector3.zero;
    _grounded = false;
    if (_rotationDirection == Direction.East)
    {
        if ((TopDirection == Direction.East) || (TopDirection == Direction.West))
        {
            TopDirection = Direction.Up;
            _pivot = transform.position + new Vector3(1, -.5f, 0);
        }
        else if (TopDirection == Direction.Up)
        {
            TopDirection = Direction.East;
            _pivot = transform.position + new Vector3(.5f, -1, 0);
        }
        else _pivot = transform.position + new Vector3(.5f, -.5f, 0);
        _axis = Vector3.forward;
    }
}
```

To move the cuboid, the Direction enumerator denotes the current orientation of the cuboid. A virtual pivot is used to rotate the cuboid on it's sides, depending on the player's input and the cuboids orientation.

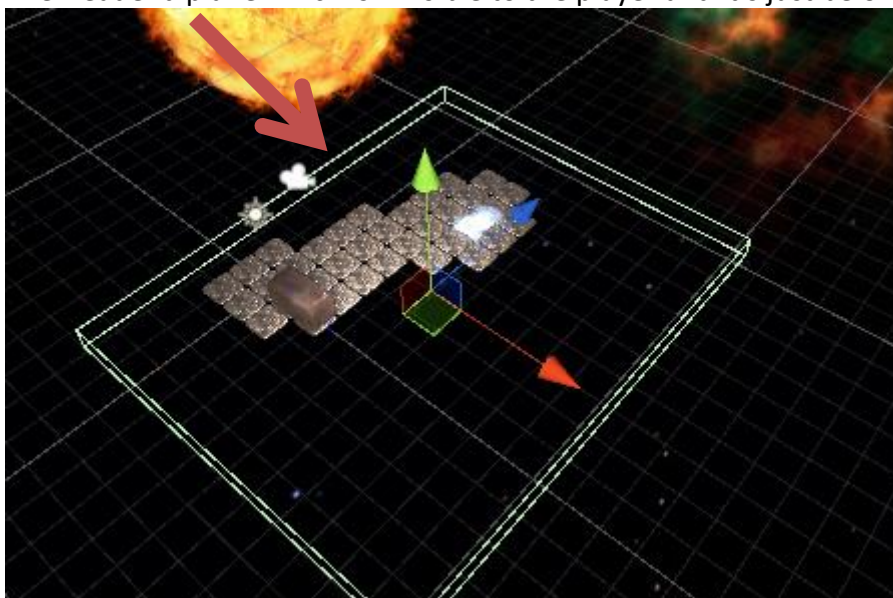
Falling

The cuboid has 2 Rays which will emit Raycast downwards. These Raycasts will detect if they have hit the deadend gameobject in the scene. If both have hit deadend, it means the cuboid is completely off the tiles whereas if only one Raycast has hit the deadend, it means only half the cuiboid is off the tiles.

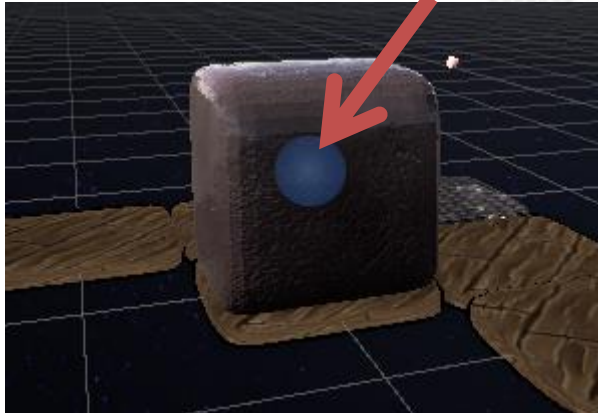


If the cuboid is completely on the tiles, both rays will hit the tiles.

The Deadend plane which is invisible to the player and it's just below:



Cubes have only one ray in the middle:



If both rays have hit deadend:

```
//both ray cuts through deadend
if (hit1.transform.CompareTag("Deadend") && hit2.transform.CompareTag("Deadend"))
{
    Debug.Log("hit");

    _grounded = true;
    //restart game after 1.5s
    GameObject.Find("Deadend").GetComponent<RestartLevel>().Restart();

    //make player fall
    gameObject.GetComponent<BoxCollider>().isTrigger = true;
    GameObject.Find("cuboid2").GetComponent<BoxCollider>().isTrigger = true;
    canUpdate = false;
}
```

This will make the cuboid falls and restart the Game after 1.5 seconds.

If only one ray hit deadend, another object is instantiated to mimic a realistic falling move of the cuboid with a force applied at the end.

```
else if (hit1.transform.CompareTag("Deadend"))
{
    Debug.Log("hit");

    _grounded = true;
    //restart game after 1.5s
    GameObject.Find("Deadend").GetComponent<RestartLevel>().Restart();

    PlayerFallOneHit(RayPosition[0].position);
}
else if (hit2.transform.CompareTag("Deadend"))
{
    Debug.Log("hit");

    _grounded = true;
    //restart game after 1.5s
    GameObject.Find("Deadend").GetComponent<RestartLevel>().Restart();

    PlayerFallOneHit(RayPosition[1].position);
}
```



```
private void PlayerFallOneHit(Vector3 positionOfForceVector3)
{
    //make object fall
    var newGameObject = Instantiate(_fallinGameObject, transform.position, transform.rotation);
    newGameObject.GetComponent<Rigidbody>().AddForceAtPosition(Vector3.down * 100, positionOfForceVector3);

    Destroy(gameObject);
    canUpdate = false;
}
```

Restarting Level

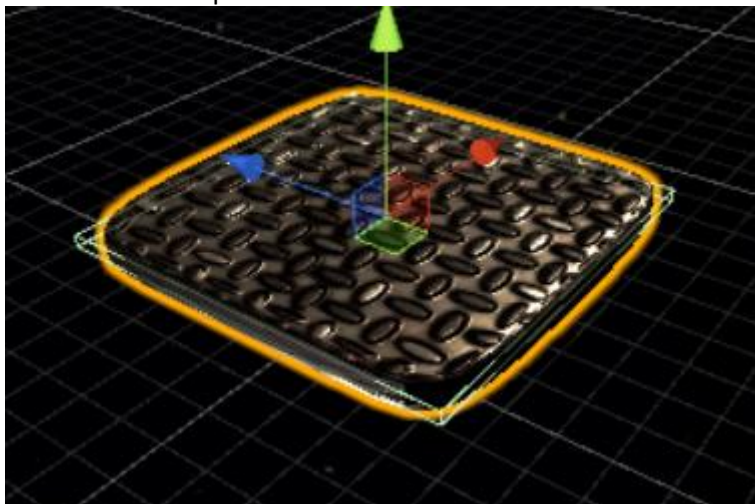
The game waits for 1.5 seconds before reloading the same scene from beginning.

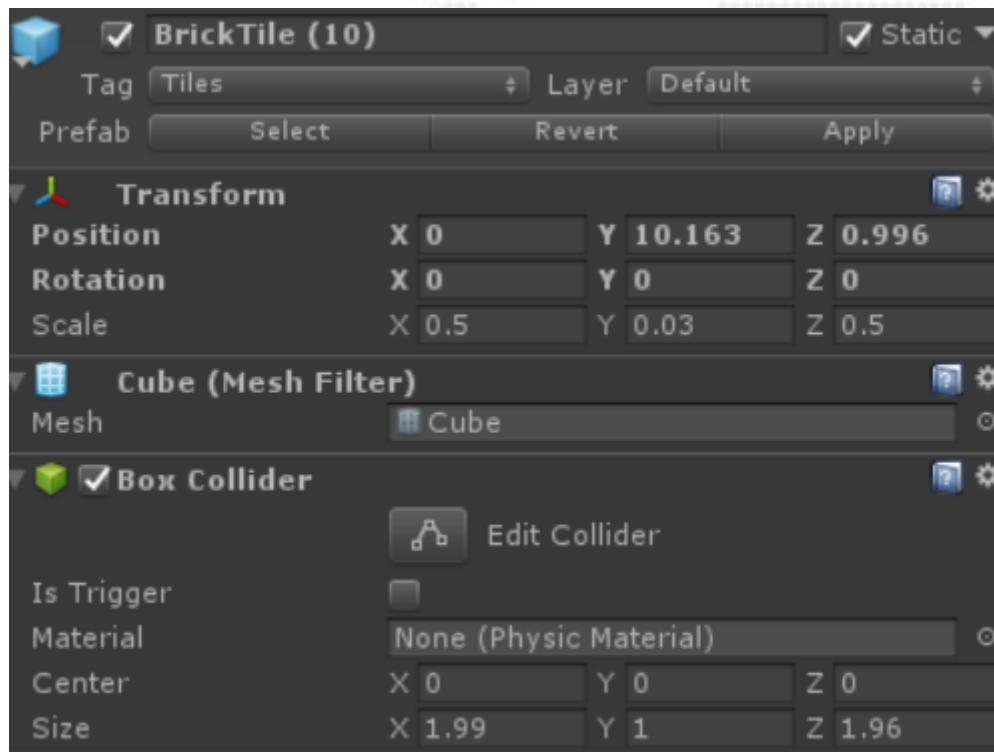
```
public void Restart()
{
    StartCoroutine(RestartGame());
}

IEnumerator RestartGame()
{
    yield return new WaitForSeconds(1.5f);
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
```

Squares

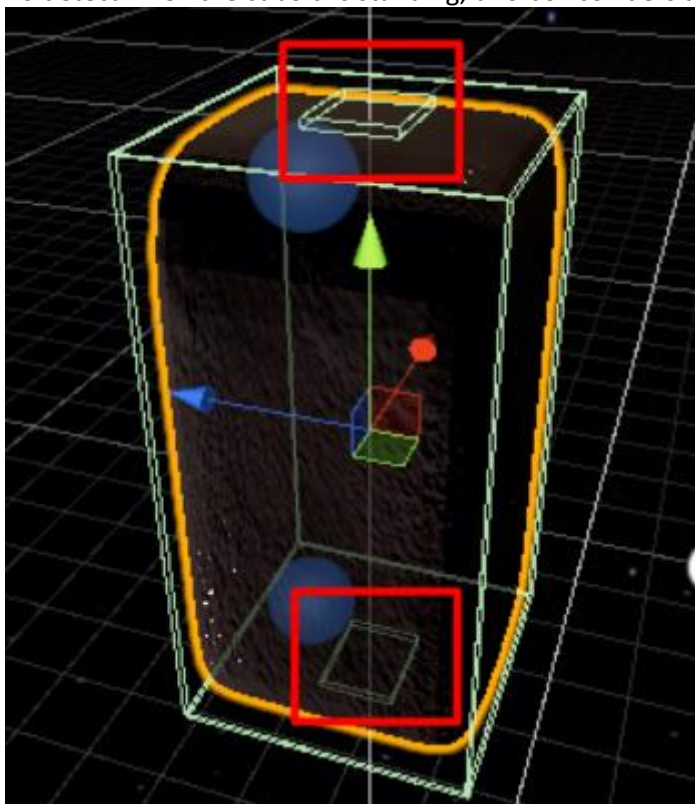
For brick square a simple 3d object made in blender is used. In unity, we used a box collider to allow the Cuboid rest on top of it.



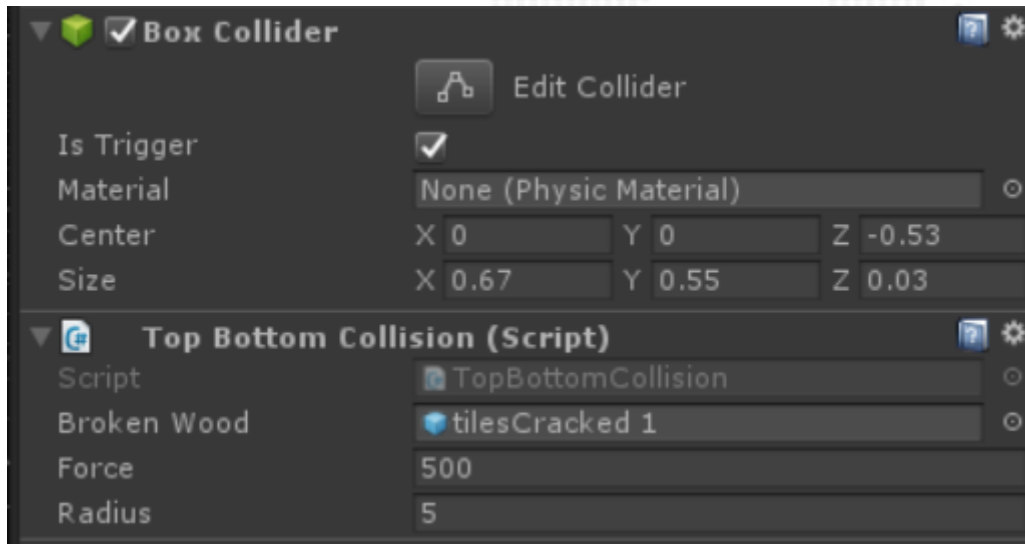


2) For the Wooden Tiles :- a script is used to detect when the Cuboid is standing on top of it then it will break causing the Cuboid to fall, and the game to restart as the player fails.

To detect when the cuboid is standing; two box colliders are used on the Cuboid as shown below



The 2 colliders used to detect when the player is standing is as shown above, in green. The 2 Colliders are marked as “Is Trigger” which allow and action to occur when it collides with a specific object.



Script used for the wooden tiles is as shown below:

```
public class TopBottomCollision : MonoBehaviour {

    public GameObject BrokenWood;
    public float Force;
    public float Radius;
    private GameObject _soundManager;

    0 references | prichiSupinfo, 2 days ago | 1 author, 2 changes
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Wood"))
        {
            _soundManager = GameObject.Find("Sound Manager");
            _soundManager.GetComponent<SoundManager>().Play("Wood break");
            var destroyed = Instantiate(BrokenWood, transform.position, transform.rotation);
            Destroy(other.gameObject);

            foreach (var d in destroyed.GetComponentsInChildren<Rigidbody>())
            {
                d.AddExplosionForce(Force, transform.position, Radius);
            }
        }
    }
}
```

When the colliders hit a wooden tile which is detected using the tag “Wood”, the script will play a wood breaking sound and play a wood breaking apart animation as scene in the script above. This cause the Cuboid to fall to its doom.

Save/Load Game

To save and load data, all the important information data such as position, score, orientation, level is stored in a “savefile.dat” file. A class called “SaveData” contains all the information needed as attributes, and is also serializable.

```
[Serializable]
3 references | Umar Callachand, 8 days ago | 2 authors, 2 changes
public class SaveData {
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public float PlayerX { get; set; }
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public float PlayerY { get; set; }
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public float PlayerZ { get; set; }
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public int Score { get; set; }
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public int CurrentLevel { get; set; }
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public float PlayerXRotation { get; set; }
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public float PlayerYRotation { get; set; }
    2 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    public float PlayerZRotation { get; set; }
    2 references | Umar Callachand, 8 days ago | 1 author, 1 change
    public Direction TopDirection { get; set; }
}
```

Another script called “SaveGame” is used to actually store and retrieve the data from the aforementioned file.

```
{
    FindPlayerObject();
    FindGameScoreObject();
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    FileStream file = File.Create(Application.persistentDataPath + "/saveFile.dat");

    PlayerX = _player.transform.position.x;
    PlayerY = _player.transform.position.y;
    PlayerZ = _player.transform.position.z;
    Score = GameScore.GetComponent<ScoreDB>().GetScore();
    CurrentLevel = SceneManager.GetActiveScene().buildIndex;
    PlayerXRotation = _player.transform.rotation.eulerAngles.x;
    PlayerYRotation = _player.transform.rotation.eulerAngles.y;
    PlayerZRotation = _player.transform.rotation.eulerAngles.z;
    TopDirection = _player.GetComponent<PlayerRoll>().TopDirection;

    _saveData = new SaveData
    {
        PlayerX = PlayerX,
        PlayerY = PlayerY,
        PlayerZ = PlayerZ,
        Score = Score,
        CurrentLevel = CurrentLevel,
        PlayerXRotation = PlayerXRotation,
        PlayerYRotation = PlayerYRotation,
        PlayerZRotation = PlayerZRotation,
        TopDirection = TopDirection
    };
    binaryFormatter.Serialize(file, _saveData);
    file.Close();
}
```

To save the player location, we have to store the x, y and z position separately as Vector3 type is not serializable. The same is done for the rotation. For the scene, the current build index is saved.


```
public void Load()
{
    FindPlayerObject();
    FindGameScoreObject();

    if (File.Exists(Application.persistentDataPath + "/saveFile.dat"))
    {
        var binaryFormatter = new BinaryFormatter();
        var file = File.Open(Application.persistentDataPath + "/saveFile.dat", FileMode.Open);

        var saveData = (SaveData) binaryFormatter.Deserialize(file);
        file.Close();

        PlayerX = saveData.PlayerX;
        PlayerY = saveData.PlayerY;
        PlayerZ = saveData.PlayerZ;
        Score = saveData.Score;
        CurrentLevel = saveData.CurrentLevel;
        PlayerXRotation = saveData.PlayerXRotation;
        PlayerYRotation = saveData.PlayerYRotation;
        PlayerZRotation = saveData.PlayerZRotation;
        TopDirection = saveData.TopDirection;

        if (SceneManager.GetActiveScene().buildIndex != CurrentLevel)
        {
            _isLoading = true;
            SceneManager.LoadScene(CurrentLevel);
        }

        else
        {
            LoadPlayerPosition();
        }
    }
}
```

Loading is just the reverse process taking the values back from the file and assigning it to the player.

```
private void LoadPlayerPosition()
{
    FindPlayerObject();
    FindGameScoreObject();
    _player.transform.position = new Vector3(
        PlayerX,
        PlayerY,
        PlayerZ
    );
    _player.transform.rotation = Quaternion.Euler(new Vector3(PlayerXRotation, PlayerYRotation,
        PlayerZRotation));
    GameScore.GetComponent<ScoreDB>().SetScore(Score);
    _player.GetComponent<PlayerRoll>().TopDirection = TopDirection;
    _isLoading = false;
}
```

First it finds the player and score objects from the scene, and then positions it back to where it was using the transform.position and transform.rotation attribute. The GameScore is set back to what it was and the orientation of the cuboid is loaded.

For loading and saving 2 GUI buttons are used as such:

```
public class SaveLoadButton : MonoBehaviour {
    0 references | prichiSupinfo, 28 days ago | 1 author, 1 change
    private void OnGUI()
    {
        if (GUI.Button(new Rect(20,140, 100, 30), "Save"))
        {
            SaveGame.saveGame.Save();
        }

        if (GUI.Button(new Rect(20, 200, 100, 30), "Load"))
        {
            SaveGame.saveGame.Load();
        }
    }
}
```

For the gameSave object, a singleton is used such that only a single instance is created everytime.

```
0 references | prichiSupinfo, 0 days ago | 1 author, 4 changes
private void Awake()
{
    if (saveGame == null)
    {
        DontDestroyOnLoad(gameObject);
        saveGame = this;
    }

    else if (saveGame != this)
    {
        Destroy(gameObject);
    }
}
```

Such that the save/load values can be permeated throughout the different scenes.

Switches and bridges

To activate bridges we have normal or strong switches:

Normal Switch:



```
public GameObject bridges;
private Animator anim;
public bool IsOn;
void Start()
{
    anim = bridges.GetComponent<Animator>();
    anim.SetBool("on", IsOn);
}

private void OnTriggerEnter(Collider other)
{
    IsOn = !IsOn;
    anim.SetBool("on", IsOn);
}
```

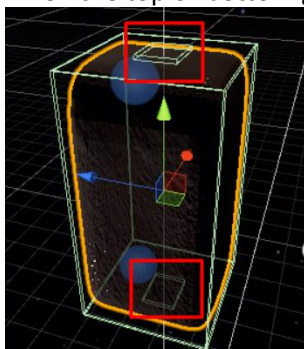
The switch will activate the bridges whenever an object collides with the switch.

Strong Switch:



```
private void OnTriggerEnter(Collider other)
{
    //only triggers when cuboid is standing on it
    if (other.CompareTag("TopBottomCollider"))
    {
        IsOn = !IsOn;
        anim.SetBool("on", IsOn);
    }
}
```

This one will only be triggered and activate the bridges associated with it when the cuboid stands on it, that is, when the top or bottom gameObjects collide with the switch:



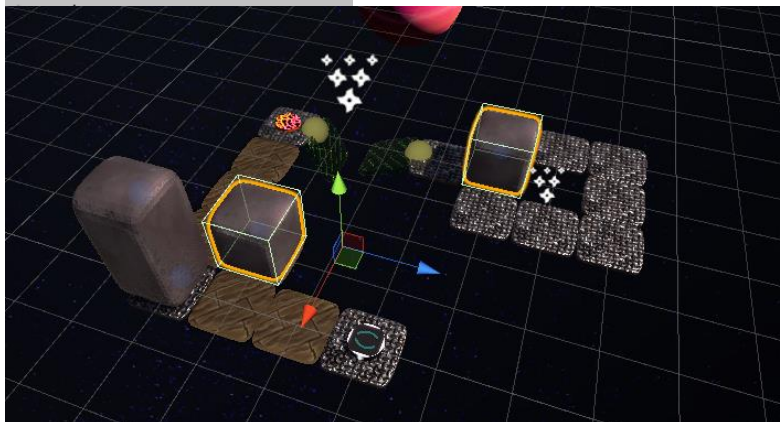
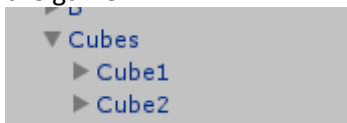
Teleportation

The teleportation switch is activated whenever the cuboid stands on it:



```
void Start()
{
    //saves the gameobject cube in a variable so that we can activate it again
    _cubes = GameObject.Find("Cubes");
    //deactivates at start of game
    _cubes.SetActive(false);
    _isTeleported = false;
    _player = GameObject.Find("Player Final");
}
```

At the start of the game, the 2 cubes are set inactive which means invisible to the player and they don't affect the game.



The cuboid is saved in a variable named `_player` for future use.

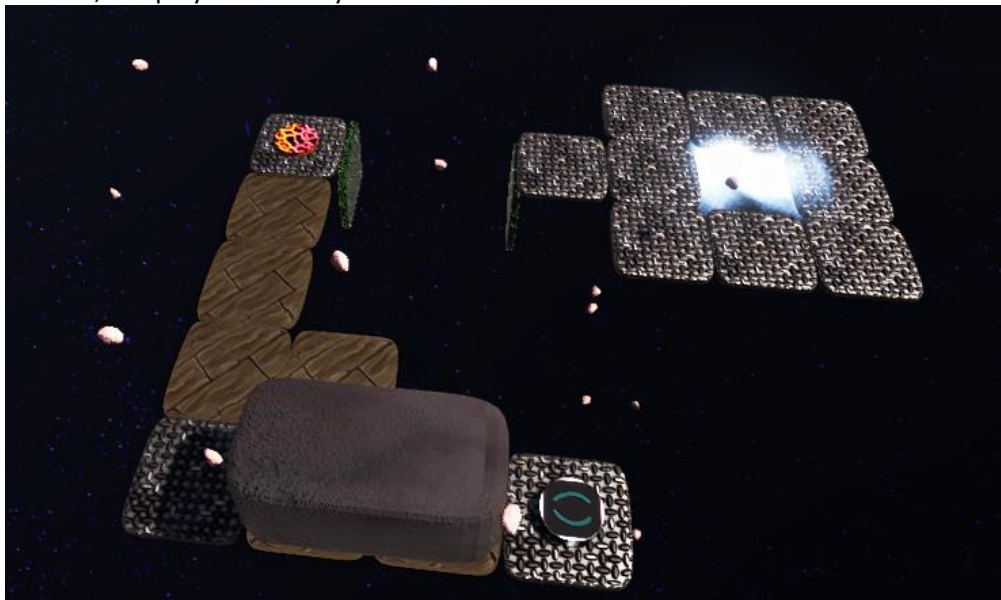
```
private void OnTriggerEnter(Collider other)
{
    //only triggers when cuboid is standing on it
    if (other.CompareTag("TopBottomCollider"))
    {
        StartCoroutine(Teleport());
    }
}

private IEnumerator Teleport()
{
    _isTeleported = true;

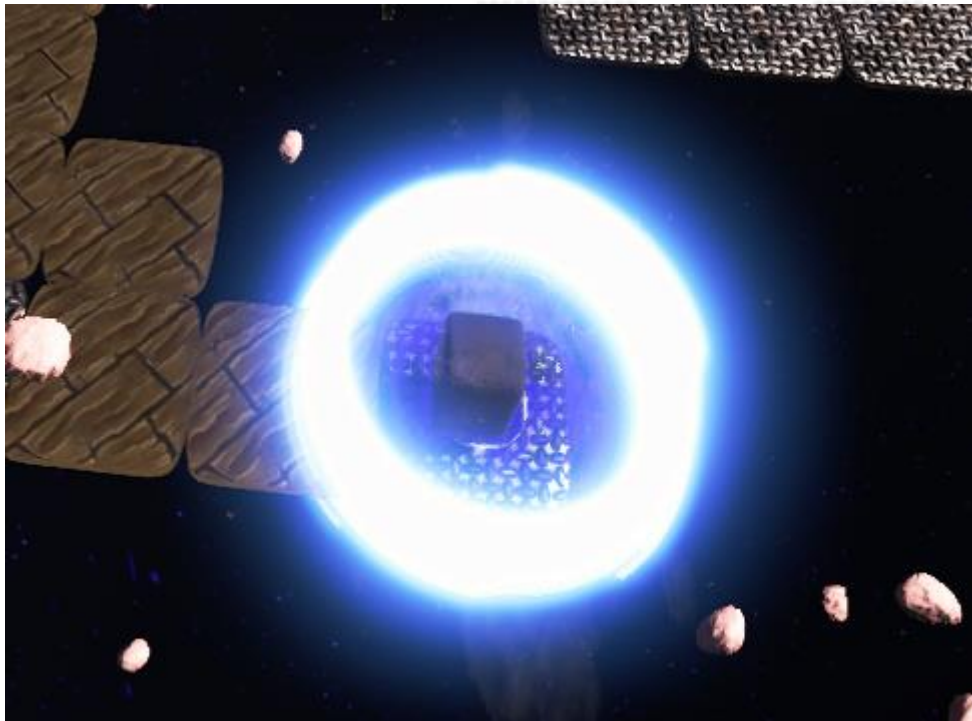
    var teleportLocation = transform.position + new Vector3(0, 0.2f, 0);
    var teleport = Instantiate(TeleportEffect, teleportLocation, transform.rotation);
    yield return new WaitForSeconds(2f);
    _isTeleported = false;
    GameObject.Find("Player Final").SetActive(false);
    Destroy(teleport);
    _cubes.SetActive(true);
    gameObject.SetActive(false);
}
```

Now when the cuboid stands on the teleportation switch, `Teleport()` method is called which will Instantiate an effect called `teleport` and waits for 2 seconds before destroying the cuboid and setting active the cubes.

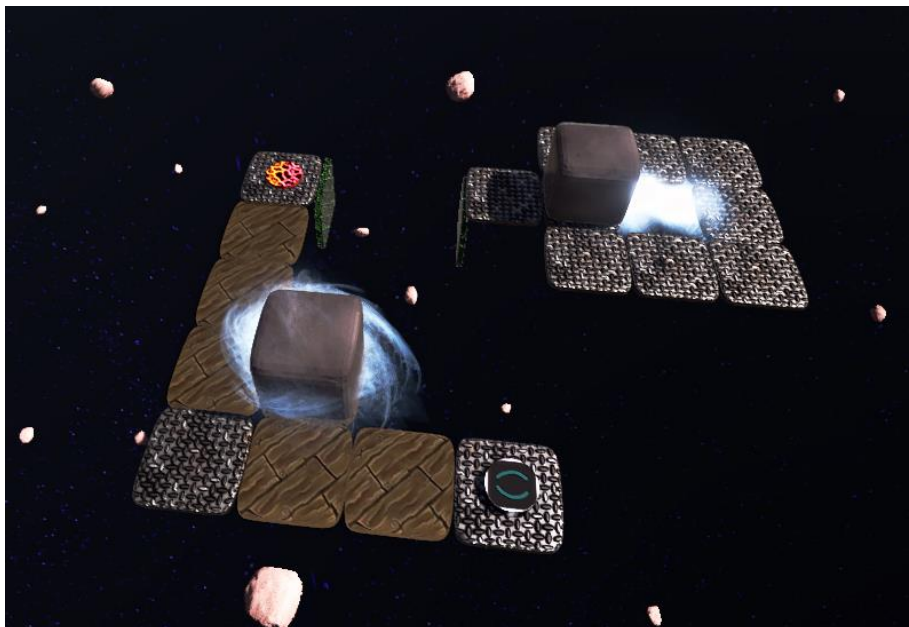
At start, the player sees only the cuboid:



When cuboid stands on animation switch, the animation starts:



The cubes become active:



The active cube is glowed.

The player now can control one cube at a time and decides to control the other by pressing Tab key.

```
//Awake is used to initialize any variables or game state before the game starts
private void Awake()
{
    _playerRollCube1 = GameObject.Find("Cube1").GetComponent<Cube_rotation>();
    _playerRollCube2 = GameObject.Find("Cube2").GetComponent<Cube_rotation>();
}

void Start ()
{
    _playerRollCube1.enabled = true;
    _playerRollCube2.enabled = false;
}

void Update ()
{
    if (Input.GetKeyDown(KeyCode.Tab))
    {
        _playerRollCube1.enabled = !_playerRollCube1.enabled;
        _playerRollCube2.enabled = !_playerRollCube2.enabled;
    }
}
```

Teleportation Joining:

```
private void LateUpdate()
{
    var offset = _cube1.transform.position - _cube2.transform.position;
    var modX = Mathf.Round(Mathf.Abs(offset.x));
    var modZ = Mathf.Round(Mathf.Abs(offset.z));
    Debug.Log(modX + " " + modZ);

    if(modX == 1 && modZ == 0 && !_canJoin)
    {
        FindXZCuboid();
        dir = Direction.West;
        rotationDir = new Vector3(-180, 90, 0);
        _canJoin = true;
    }
    else if (modZ == 1 && modX == 0 && !_canJoin)
    {
        FindXZCuboid();
        dir = Direction.South;
        rotationDir = new Vector3(0,0,0);
        _canJoin = true;
    }

    else _canJoin = false;
}
```

For joining the 2 cubes back together, the offset between the two cubes are calculated first. With the implementation of our game, the distance between any two adjacent tiles are always equal to one. Hence, when the 2 cubes are adjacent to each other, the absolute value for the offset in the x or z axis will be equal to 1. When either of these two values are equal to 1, but not both at the same time, the coordinates in between the two cubes are calculated as shown below :

```
private void FindXZCuboid()
{
    _cuboidX = (Mathf.Round(_cube2.transform.position.x) +
Mathf.Round(_cube1.transform.position.x)) / 2;
    _cuboidZ = (Mathf.Round(_cube2.transform.position.z) +
Mathf.Round(_cube1.transform.position.z)) / 2;
}

private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player_half"))
    {
        //sets player to active again so that Instantiate will create an active
gameObject and set it inactive just after
        _player.SetActive(true);

        _player.transform.localScale = new Vector3(0.5f,0.5f,1f);
        var newPlayer = Instantiate(_player, (new Vector3(_cuboidX, 2.75f,
_cuboidZ)), Quaternion.Euler(rotationDir));
        newPlayer.GetComponent<PlayerRoll>().TopDirection = dir;

        _player.SetActive(false);

        //set both cubes to inactive
        _playerSplitted.SetActive(false);
    }
}
```

These values are stored in a variable each, from there if the player chooses the merge the cubes together, a new cuboid object will be instantiated in the correct positions calculated and in the correct direction by using the Direction enumerator from the PlayerRoll script.

Score:

SQLite is used to store the players' high scores.

SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. SQLite is the most used database engine in the world.

The CuboidHighScore table is created with columns as shown:

```
public void InitializeDb()
{
    dbConnection = new SqlConnection(_connectionString);

    string commandText = "create table IF NOT EXISTS CuboidHighScore " +
        "(ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "Name VARCHAR(255), " +
        "Score INTEGER, " +
        "Date DATE);";

    if (dbConnection != null)
    {
        dbConnection.Open();
        try
        {
            using (IDbCommand command = dbConnection.CreateCommand())
            {
                command.CommandText = commandText;
                command.ExecuteNonQuery();
            }
        }
        catch (Exception ex)
        {
            Debug.Log(ex.Message);
        }
        dbConnection.Close();
    }
}
```

ScoreDB is a singleton class so that only one instance is created throughout all stages.


```
private void Awake()
{
    if (scoreDb == null)
    {
        DontDestroyOnLoad(gameObject);
        scoreDb = this;
    }

    else if (scoreDb != this)
    {
        Destroy(gameObject);
    }
}
```

Also, it contains a static int for storing the players score:

```
private static int _score = 0;
```

Now in playerRoll, whenever the player moves the cuboid, the score is incremented:

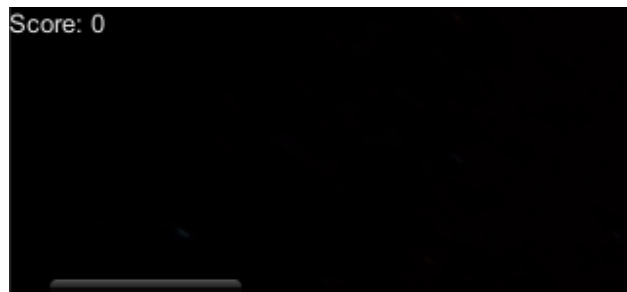
```
_score.UpdateScore();
```

In scoreDB:

```
public void UpdateScore()
{
    _score++;
}
```

The score is displayed on all scenes

```
private void OnGUI()
{
    GUI.Label(new Rect(0,0,100,100), "Score: " + _score.ToString() );
}
```



The score will be used mostly on end screen when the player submits his/her name and the top 5 high scores are displayed.



When the OK button is pressed and the input field isn't empty,

```
public void OkBtn()
{
    //GameScore.GetComponent<ScoreDB>().SetScore(5);
    Debug.Log(GameScore.GetComponent<ScoreDB>().GetScore());

    if (inputField.text != "")//ignore empty input
    {
        List<Score> scores = SaveScore();

        DisplayScores(scores);
    }
}
```

```
private List<Score> SaveScore()
{
    ScoreDB scoreDb = GameScore.GetComponent<ScoreDB>();

    Score score = new Score(_name, scoreDb.GetScore(), DateTime.Now);

    //scoreDb.InitializeDb();
    List<Score> scores = scoreDb.ReadingFromDatabase();

    foreach (Score score1 in scores)
    {
        if (score.ScoreValue < score1.ScoreValue || scores.Count < 5)
        {
            scoreDb.SaveScore(score.ScoreValue, _name);
            scores = scoreDb.ReadingFromDatabase(); // update scores

            //save only 5 best in db
            if (scores.Count > 5)
            {
                scoreDb.DeleteFromDb(scores[5].Id);
            }
            break;
        }
    }

    return scores;
}
```

A new Score is created for this particular player and all previous high scores are loaded from database.

If the player's score is less than any of the high scores or the number of high scores in the database is less than 5, this player's score is saved in database.

```
public bool SaveScore(int score, string playerName)
{
    int hasSaved = 0;
    if (dbConnection != null)
    {
        dbConnection.Open();
        using (IDbCommand command = dbConnection.CreateCommand())
        {
            try
            {
                command.CommandText =
                    "insert into CuboidHighScore ( ID, Name, Score, Date) "
                    + " Values" +
                    "( NULL, \"" + playerName + "\" , " + score + " , \"" + DateTime.Now + "\" )";
                //ID is set null, but will be incremented in db

                hasSaved = command.ExecuteNonQuery(); //returns 1 if command has executed
            }
            catch (Exception e)
            {
                Debug.Log(e);
            }
        }
        dbConnection.Close();
    }

    return (hasSaved == 1); //returns true if command is executed
}
```

Also, if the number of high scores after saving is more than 5, the last one is deleted from database to keep a light database.


```
public bool DeleteFromDb(int id)
{
    int hasDeleted = 0;
    if (dbConnection != null)
    {
        dbConnection.Open();
        using (IDbCommand command = dbConnection.CreateCommand())
        {
            try
            {
                command.CommandText =
                    "DELETE FROM CuboidHighScore" +
                    " WHERE id = " + id + " ; ";

                hasDeleted = command.ExecuteNonQuery(); //returns 1 if command has executed
            }
            catch (Exception e)
            {
                Debug.Log(e);
            }
        }
        dbConnection.Close();
    }

    return (hasDeleted == 1); //returns true if command is executed
}
```

To get all scores from database ordered by scores in ascending order (top first):

```
public List<Score> ReadingFromDatabase()
{
    List<Score> scores = new List<Score>();
    if (dbConnection != null)
    {
        dbConnection.Open();

        using (IDbCommand dbCmd = dbConnection.CreateCommand())
        {
            string sqlQuery = "SELECT * FROM CuboidHighScore ORDER BY Score; ";
            dbCmd.CommandText = sqlQuery;

            using (IDataReader reader = dbCmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    Score score = new Score(
                        Convert.ToInt32(reader["Id"]),
                        Convert.ToString(reader["Name"]),
                        Convert.ToInt32(reader["Score"]),
                        DateTime.Now
                    );

                    scores.Add(score);
                }
            }
        }
        dbConnection.Close();
    }
    return scores;
}
```

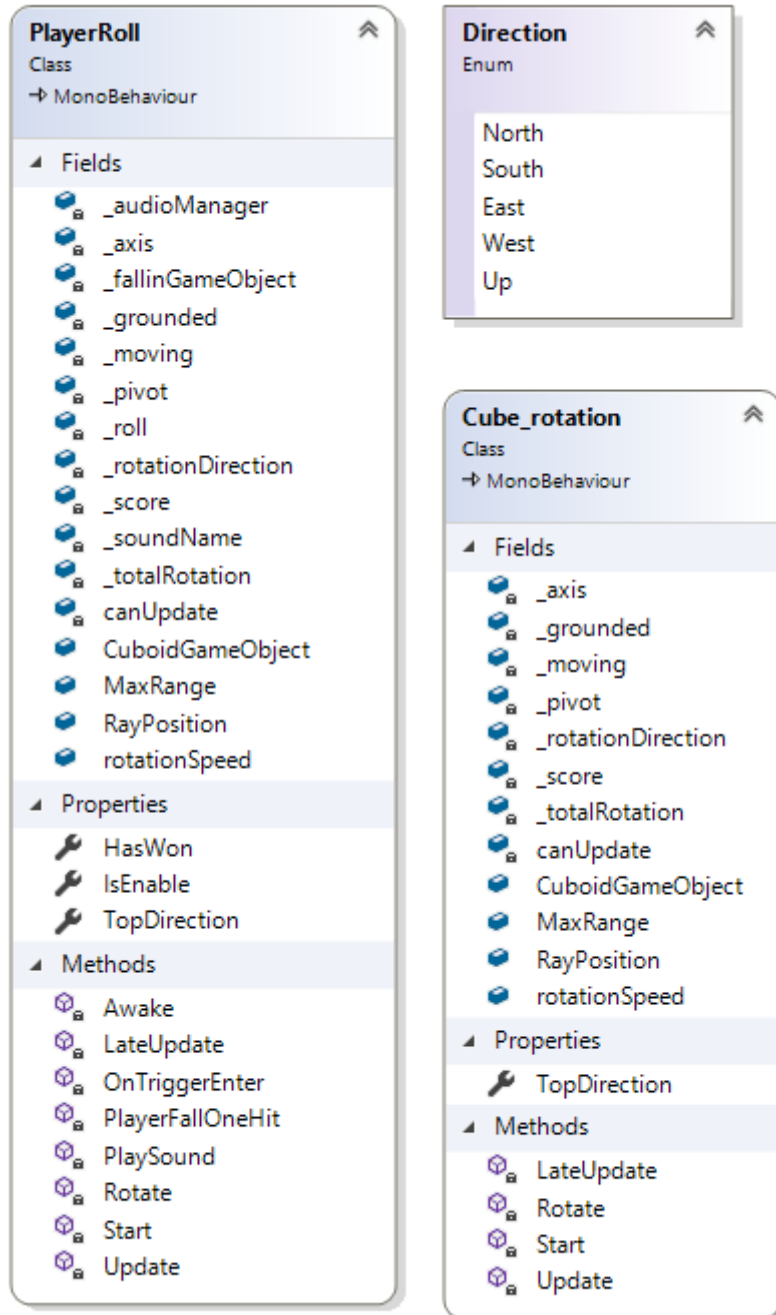
Then, the scores are displayed on the screen:

```
private void DisplayScores(List<Score> scores)
{
    inputField.gameObject.SetActive(false);
    OkButton.gameObject.SetActive(false);
    _highscores[0].gameObject.SetActive(true); //title

    for (int i = 0; i < scores.Count-1; i++)
    {
        _highscores[i+1].gameObject.SetActive(true);
        _highscores[i + 1].text = (i+1) + ". " + scores[i].PlayerName + ": " +
scores[i].ScoreValue.ToString();
    }
}
```



UML



Score
Class

- Fields
 - _date
- Properties
 - Id
 - PlayerName
 - ScoreValue
- Methods
 - Score (+ 1 overload)

SaveData
Class

- Properties
 - CurrentLevel
 - PlayerX
 - PlayerXRotation
 - PlayerY
 - PlayerYRotation
 - PlayerZ
 - PlayerZRotation
 - Score
 - TopDirection

ScoreDB
Class
→ MonoBehaviour


- Fields
 - _connectionString
 - _score
 - dbConnection
 - scoreDb
- Methods
 - Awake
 - DeleteFromDb
 - GetScore
 - InitializeDb
 - OnGUI
 - ReadingFromDatabase
 - SaveScore
 - SetScore
 - Start
 - UpdateScore

Sound
Class

- Fields
 - Clip
 - Name
 - Pitch
 - Source
 - Volume


EndScreenController
Class
→ MonoBehaviour

- Fields
 - _highscores
 - _name
 - HighScore1
 - HighScore2
 - HighScore3
 - HighScore4
 - HighScore5
 - HighScoreTitle
 - inputField
 - OkButton
- Properties
 - GameScore
- Methods
 - DisplayScores
 - OkBtn
 - Quit
 - SaveScore
 - Start
 - TextChange




StartScreenCont... 


Class
→ MonoBehaviour

Fields

-  ContinueButton




Methods

-  NewGameButton
-  Quit
-  Start



Switch 


Class
→ MonoBehaviour

Fields

-  anim
-  bridges
-  IsOn





Methods

-  OnTriggerEnter
-  Start




Timer 


Class
→ MonoBehaviour

Fields

-  _currentTime
-  _displayedTime
-  _isTimerOn
-  _totalTime




Methods

-  FixedUpdate
-  OnGUI
-  Start



StrongSwitch 

Class
→ MonoBehaviour

Fields

-  anim
-  bridges
-  IsOn

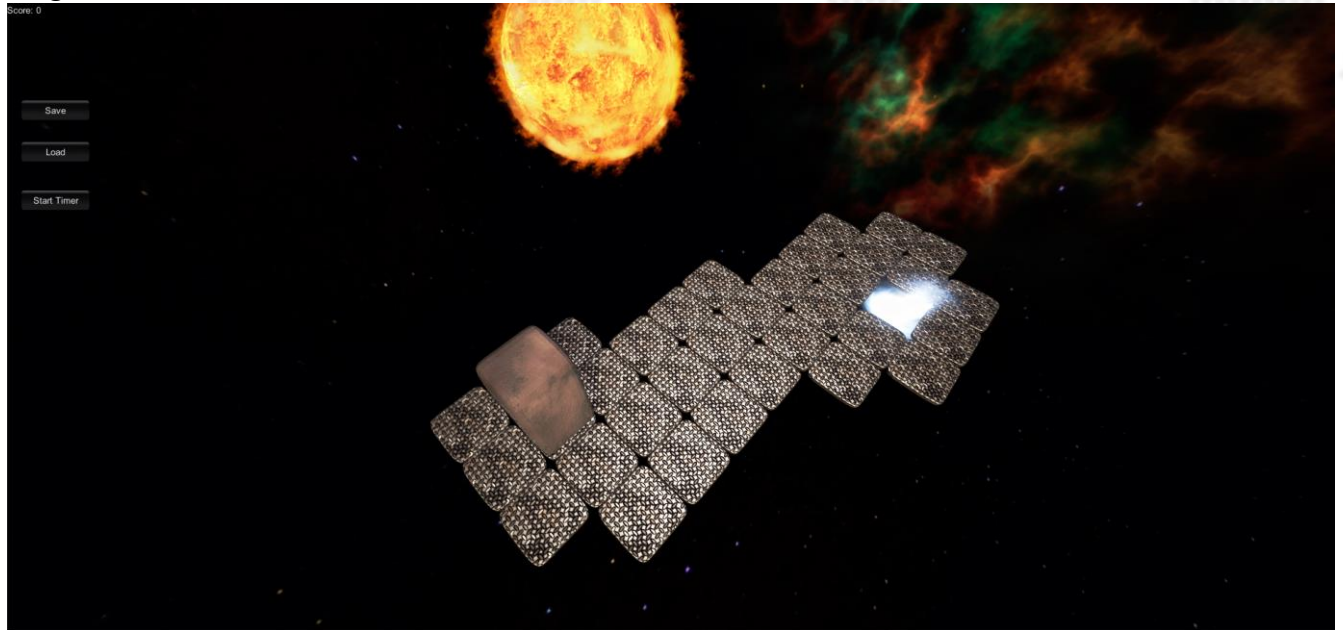
Methods

-  OnTriggerEnter
-  Start

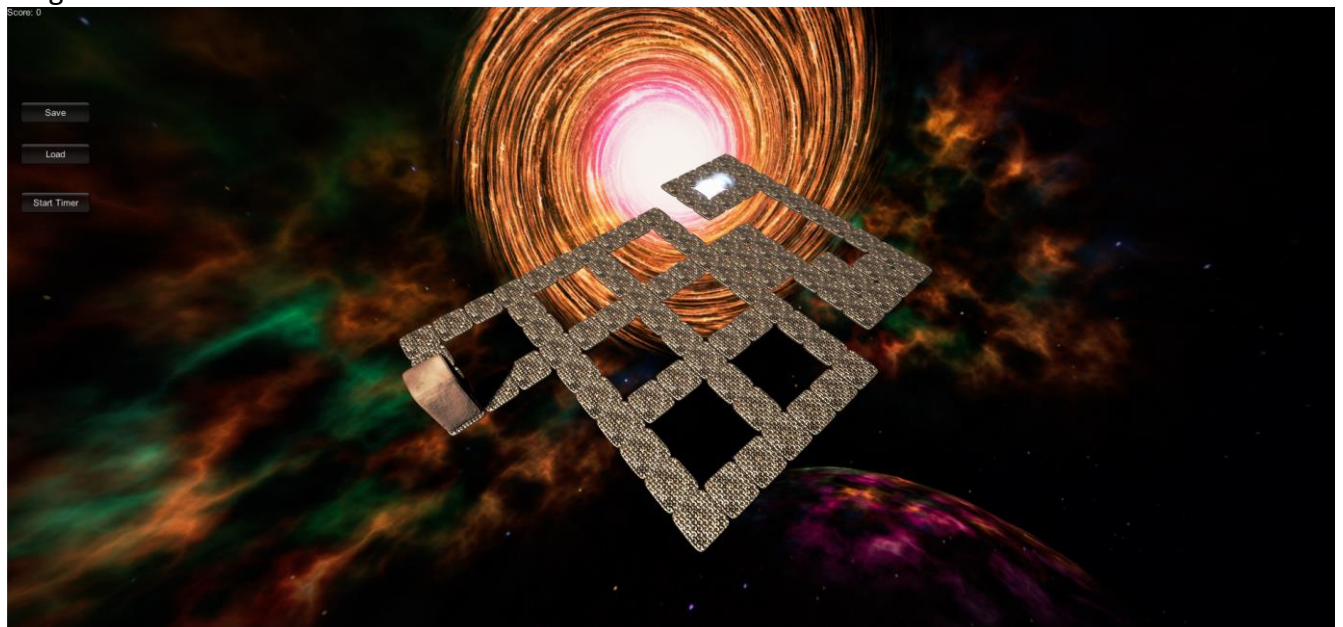
The stages are as follows:

Bricks only:

Stage 1

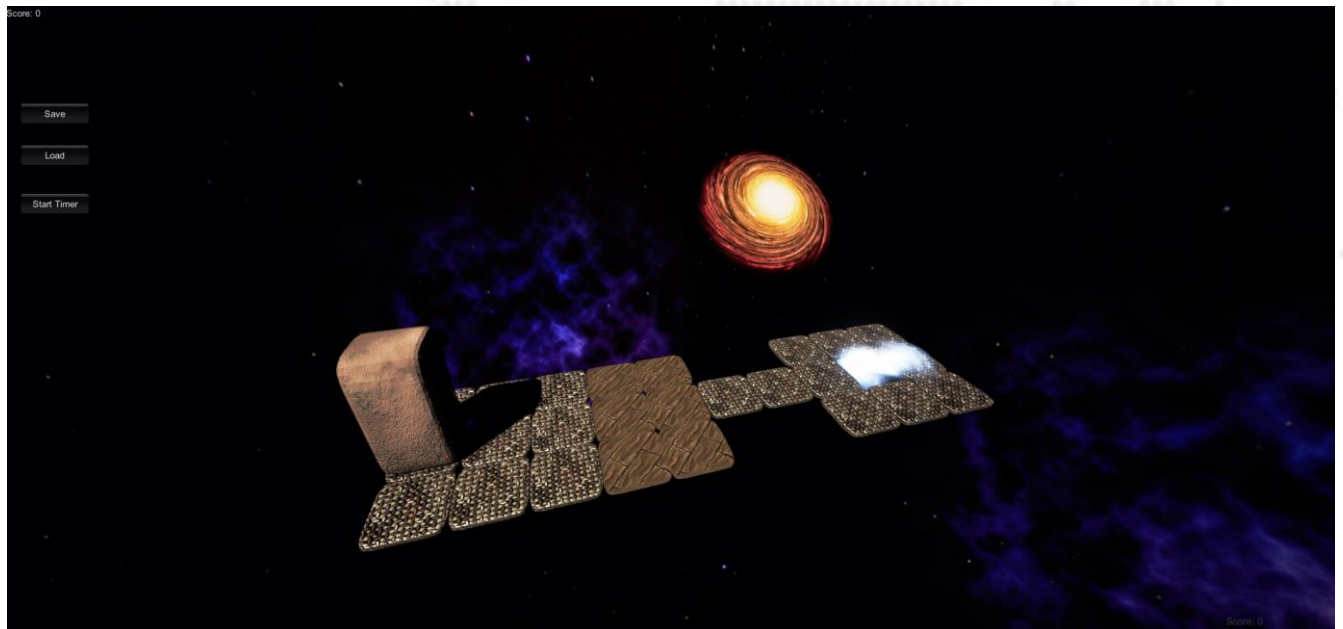


Stage 2:

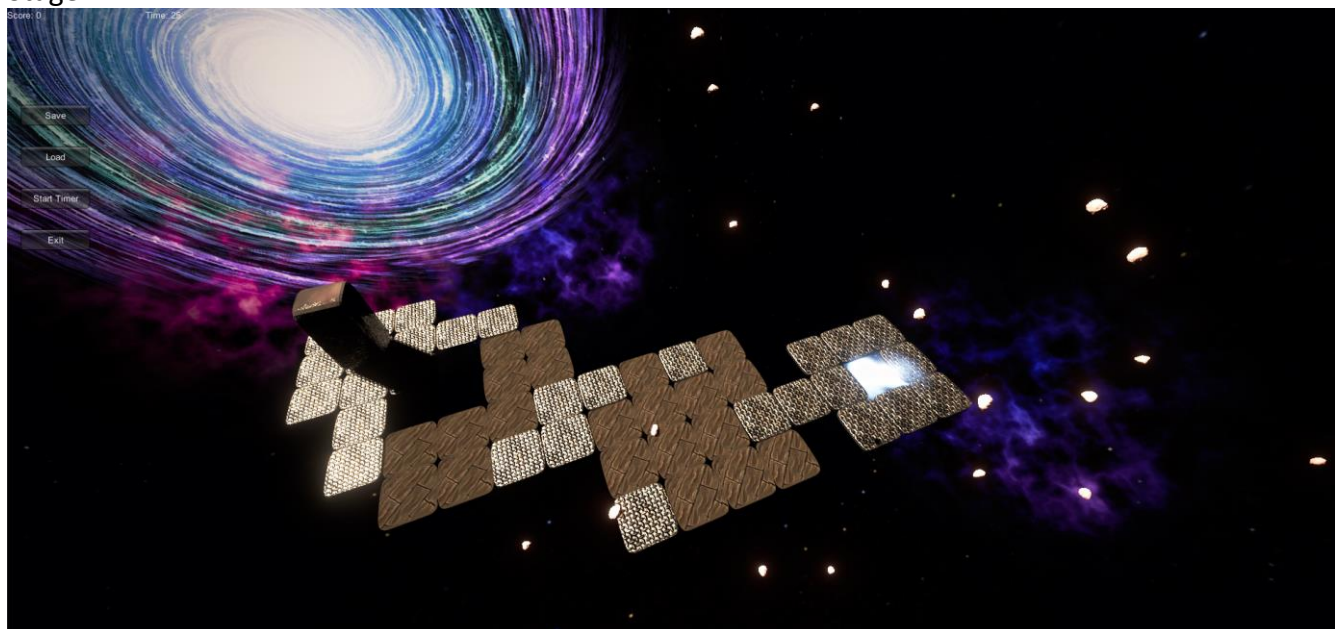


Bricks and Wood:

Stage 3:

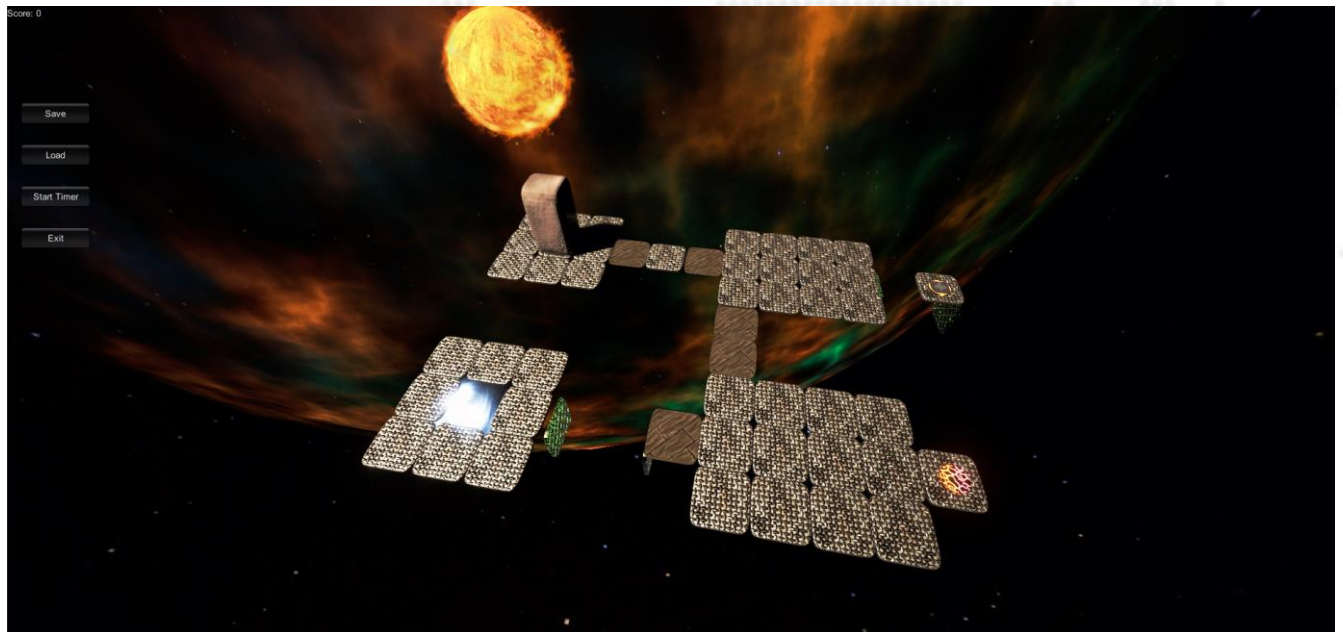


Stage 4:

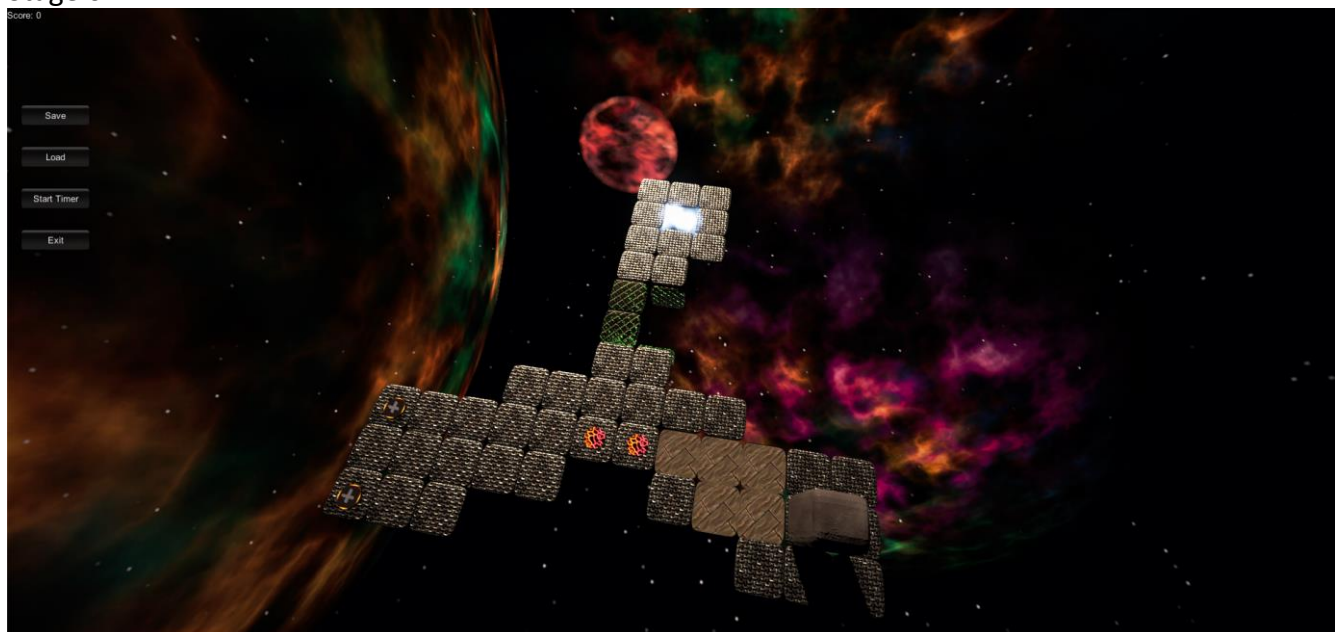


Wood Squares, Normal and Strong Switches:

Stage 5:

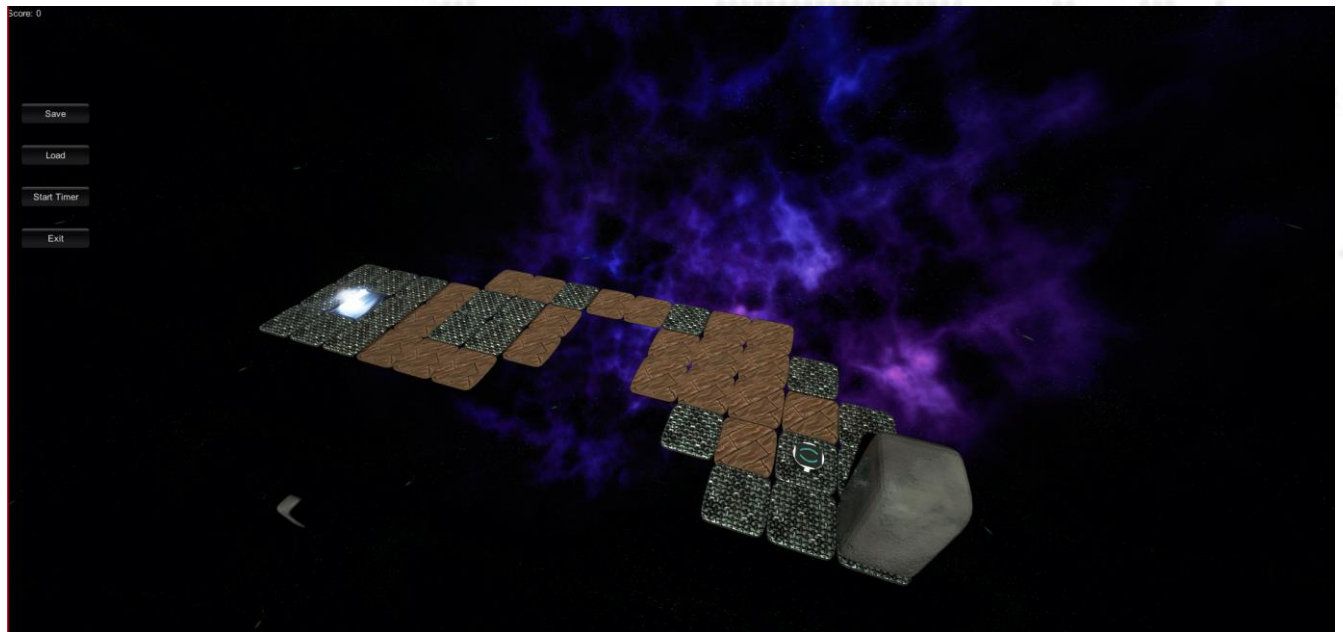


Stage 6:



Wood Squares and Teleportation:

Stage 7:

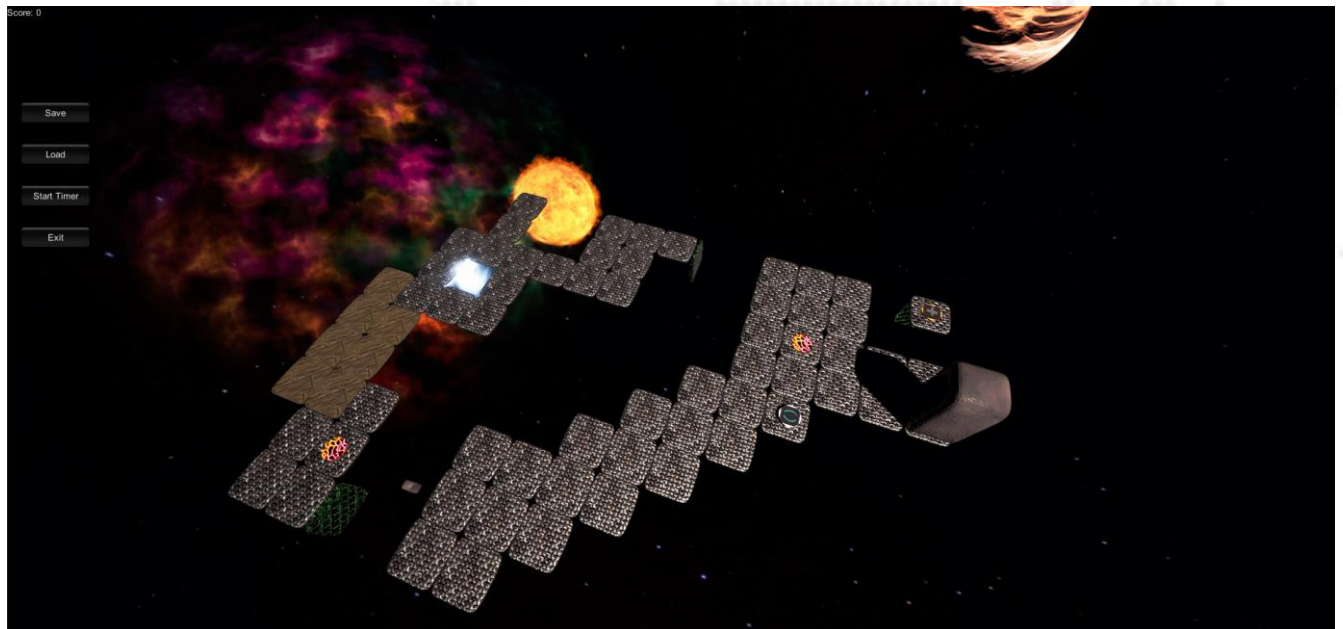


Stage 8:



Everything:

Stage 9:



Stage 10:



5 REFERENCES

<https://www.sqlite.org/index.html>

<https://forum.unity.com/threads/rotating-cuboid-script.163603/>