KAUNO TECHNOLOGIJOS UNIVERSITETAS

FACULTY OF INFORMATICS

T120B166 Development of Computer Games and Interactive Applications

Party Table Game - Saboteur

IFF-6/11 Nerijus Dulkė
Date: 2019.05.27

KAUNAS
2019

Contents

Game description

1. **3D or 2D?** 3D
2. **What type is your game?** Board sgame
3. **What genre is your game?** Party game
4. **Platforms:** PC
5. **Scenario description:** In this game action is set in a mine. Players take on the role of dwarves, some are miners, some are saboteurs, but no one knows who is on their side. For miners goal is to get to the gold until the cards run out. For saboteurs the goal is to stop miners from reaching gold. After three rounds the player with the most gold wins.

Laboratory work #1

List of tasks

1. Create a map in at least 10x10 scenario elements.
2. Add at least 50 environmental elements.
3. Add lighting sources (at least 10 units, with different settings).
4. Complement the map with sound (at least 10 different sounds and at least 5 pieces of background music in different locations)
5. Use at least 20 different materials

Solution

Task #1. Create a map in at least 10x10 scenario elements

Since this is a table game, the “map” is a 15x15 table surface:

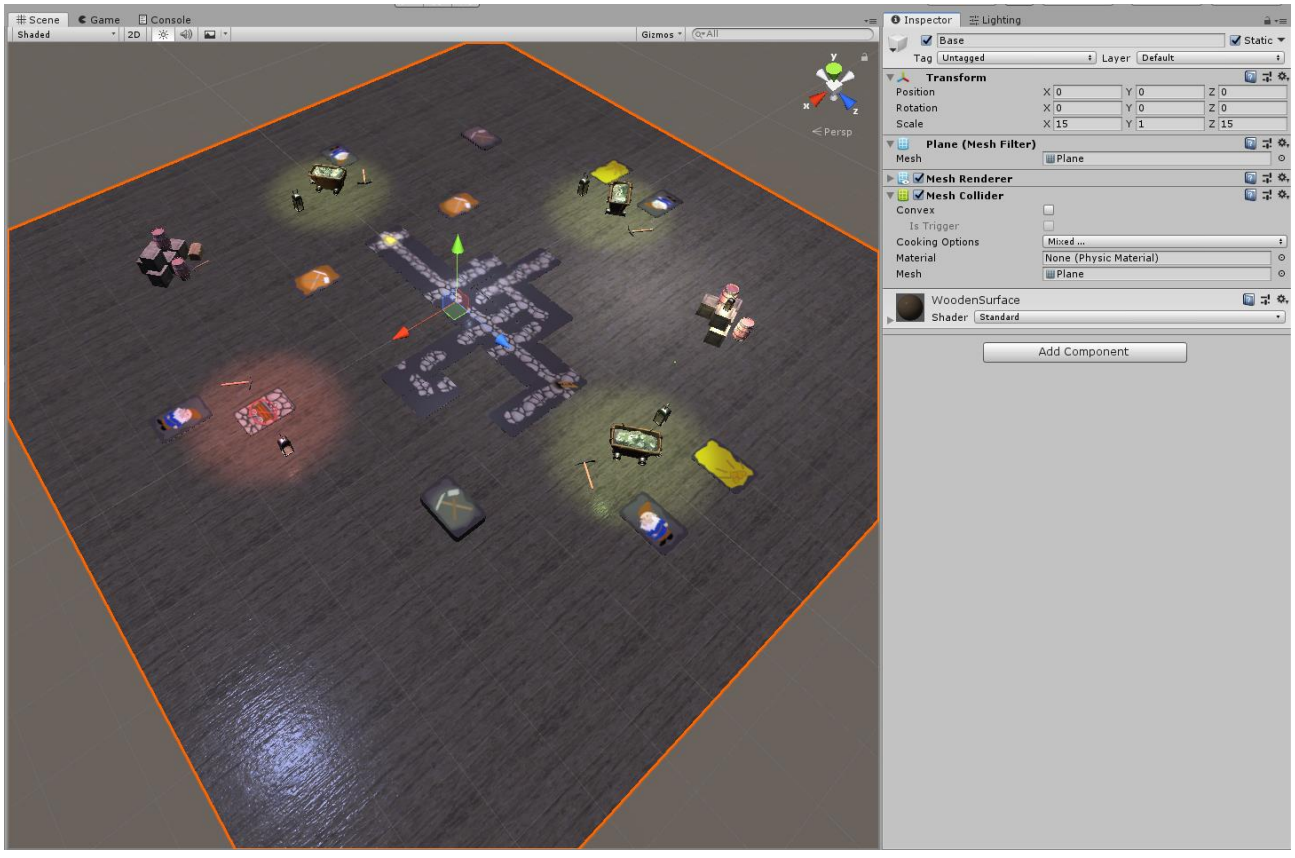


Figure 1 Table surface

Task #2. Add at least 50 environmental elements

Environment consists of different types of cards, player items and some decorative assets:

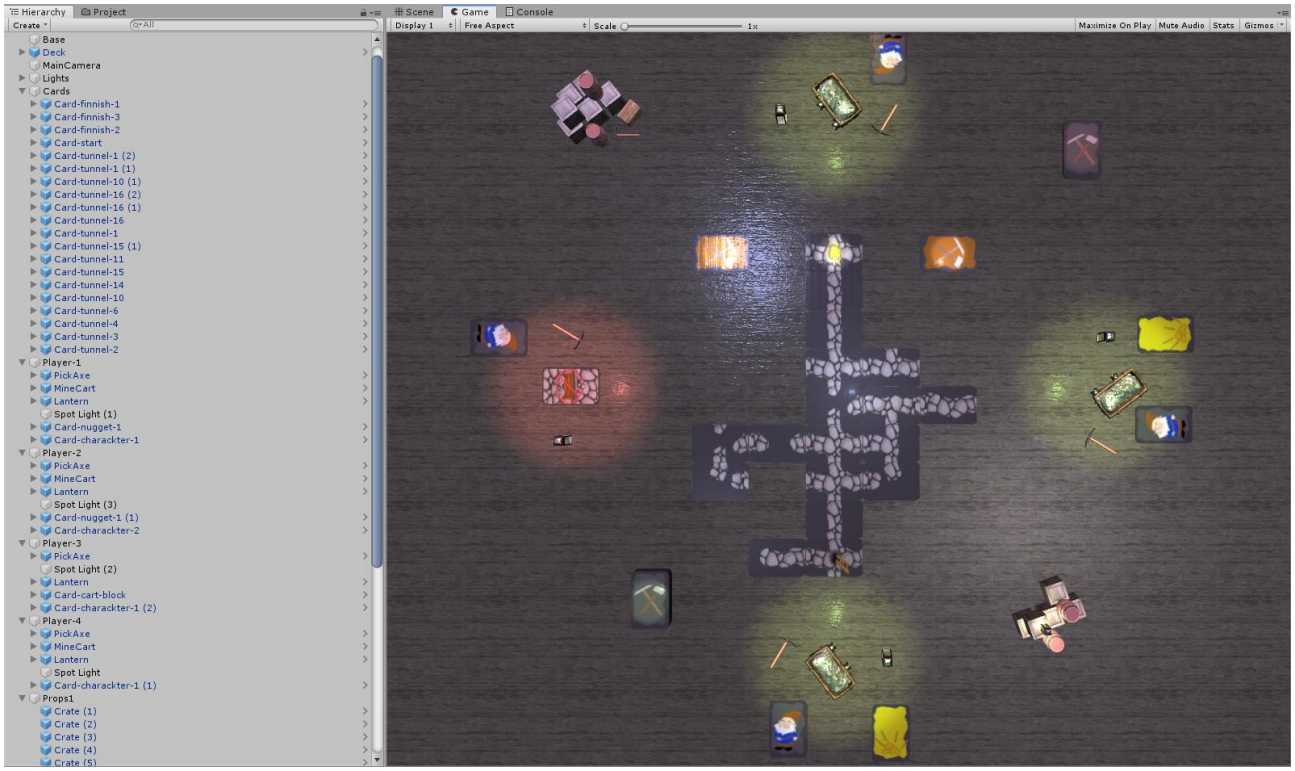


Figure 2 Enviornmental elements

Task #3. Add lighting sources

Different types of light sources were added. Directional and point lights for overall table lightning, point lights for finish cards, player items etc. Some examples are:

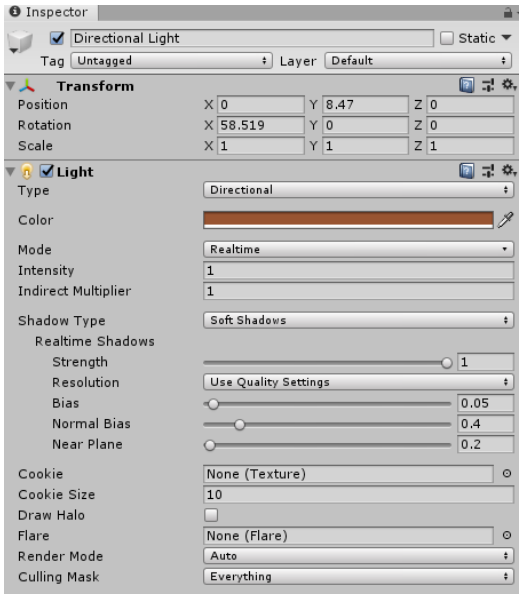


Figure 3 Light source example #1

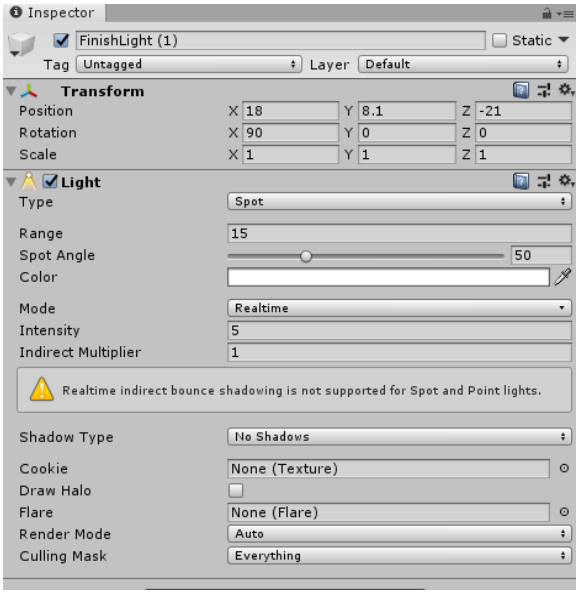


Figure 4 Light source example #2

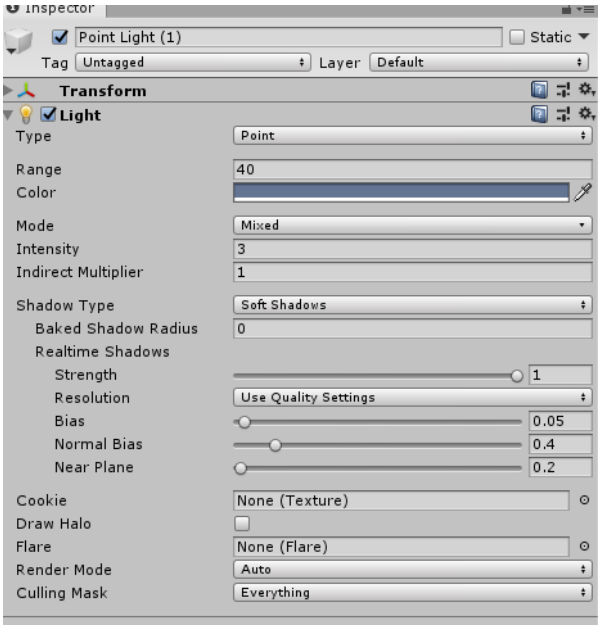


Figure 5 Light source example #3

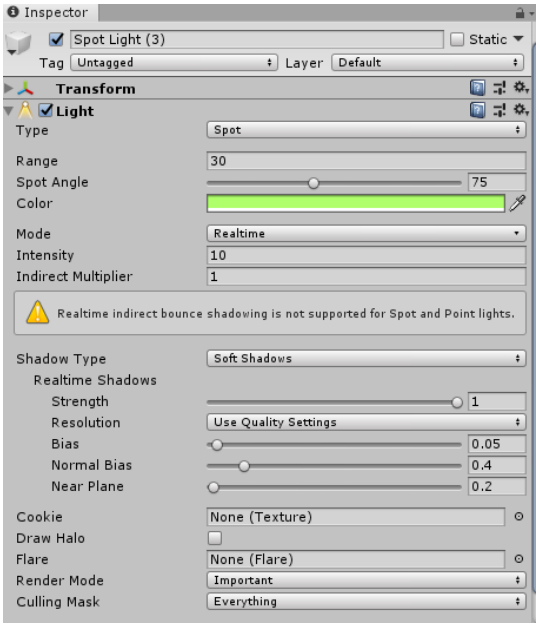


Figure 6 Light source example #4

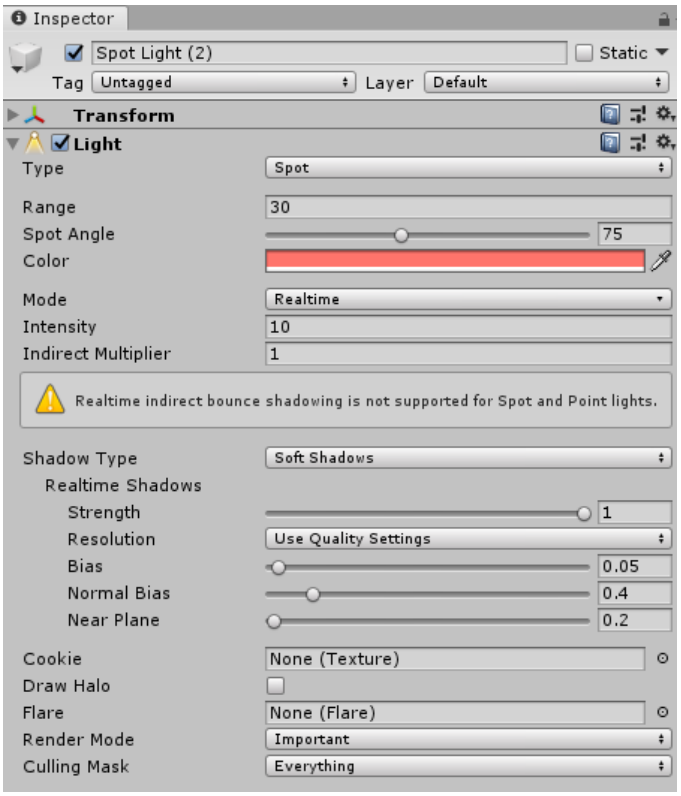
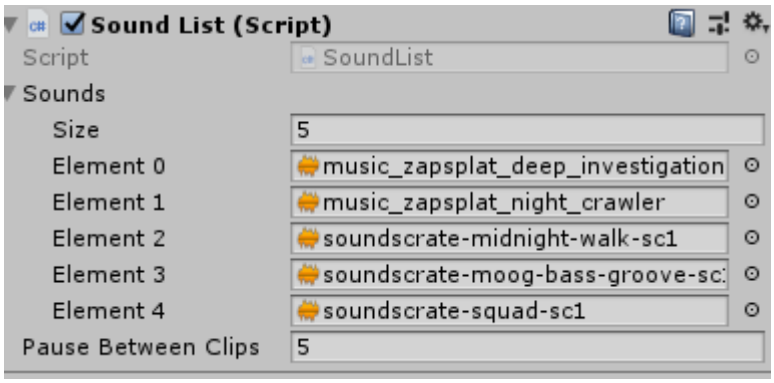


Figure 7 Light source example #5

Task #4. Complement the map with sound

Since this is a board game and there will not be any movement on the map, background music will be played using playlist:



Some sounds are also added (water dripping, wind blowing). Also some sound effects (e.g. lantern breaking) will be added later in the game, when some logic will be implemented.

Table 1 SoundList fragment

```
public class SoundList : MonoBehaviour
{
    public AudioClip[] sounds;
    public float pauseBetweenClips = 2f;
    AudioSource audioSource;
    int index = 0;
    float timer = 0f;
    bool timerRunning = false;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    void Update()
    {
        if (!audioSource.isPlaying && !timerRunning)
        {
            audioSource.Stop();
            timerRunning = true;
            timer = 0f;
        }

        if (timerRunning)
        {
            timer += Time.deltaTime;
        }

        if (timer >= pauseBetweenClips)
        {
            audioSource.clip = sounds[index++];
            audioSource.Play();

            timerRunning = false;
            timer = 0f;
            if (index >= sounds.Length)
            {
```

```
                index = 0;
            }
        }
    }
}
```

Start	Finds AudioSource component
Update	Waits until AudioSource component is not playing and then plays the next audio track from the list

Task #5. Use at least 20 different materials

Different materials for all card textures were used, and some environment materials (wooden surface, metal, etc.):

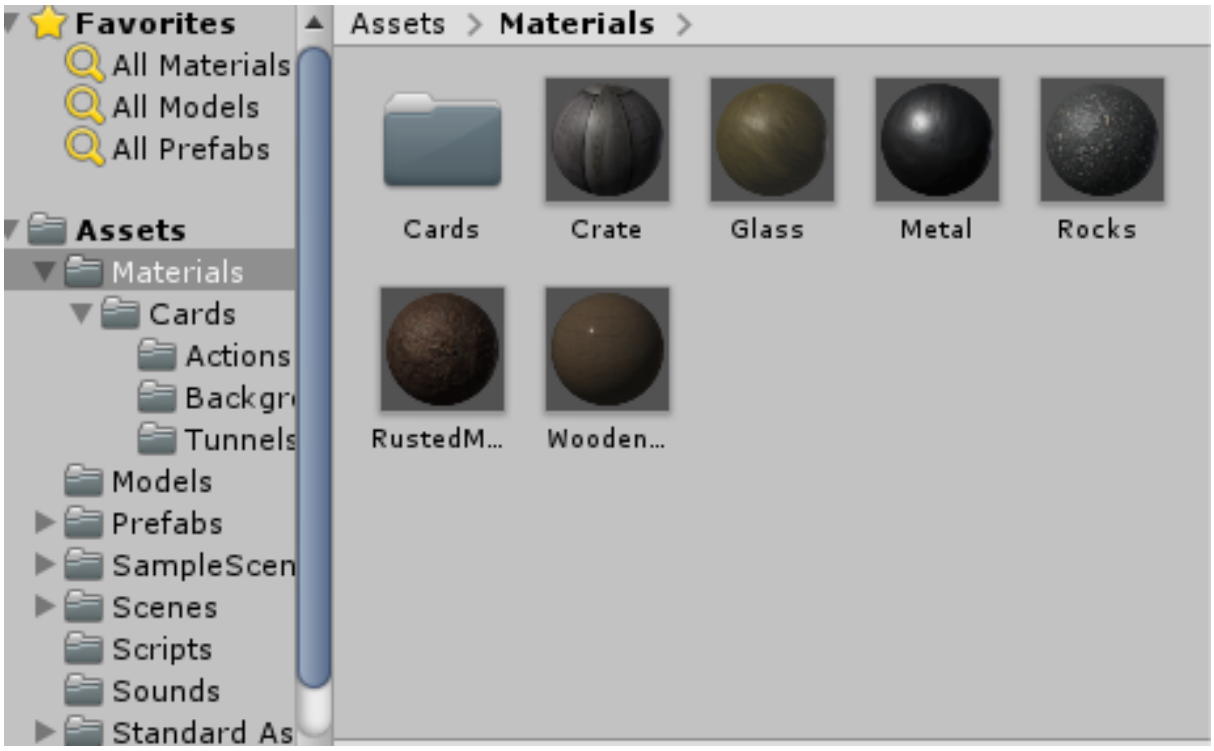


Figure 8 Material preview

Laboratory work #2

List of tasks

1. Create a terrain and color it using 5 or more materials.
2. Create 5 trees of your own construction and seed them using terrain tools.
3. Import a unique model and assign it to a third person character controller.
4. Add and animate 5 objects to your game map.
5. Create 5 unique particle effects.
6. Add a custom skybox.
7. Create 5 different physics materials and apply then in your project.
8. Assign an optimum collider based on the shape of the object for all 3D objects imported in the first laboratory work.
9. Add and animate 5 objects using physics, force and triggers to your game map.
10. Set static flag to all immobile objects and measure batching performance.
11. Try deferred vs forward rendering and measure performance for both.
12. Bake a lightmap for your scene and measure performance.
13. Optimize all textures depending on their parameters and measure graphical memory load.
14. Try hard vs soft shadows and measure performance.

Solution

Task #1 and #2. Create a terrain, color it, seed trees of your own construction

Figure 9 Colored terrain with seeded trees

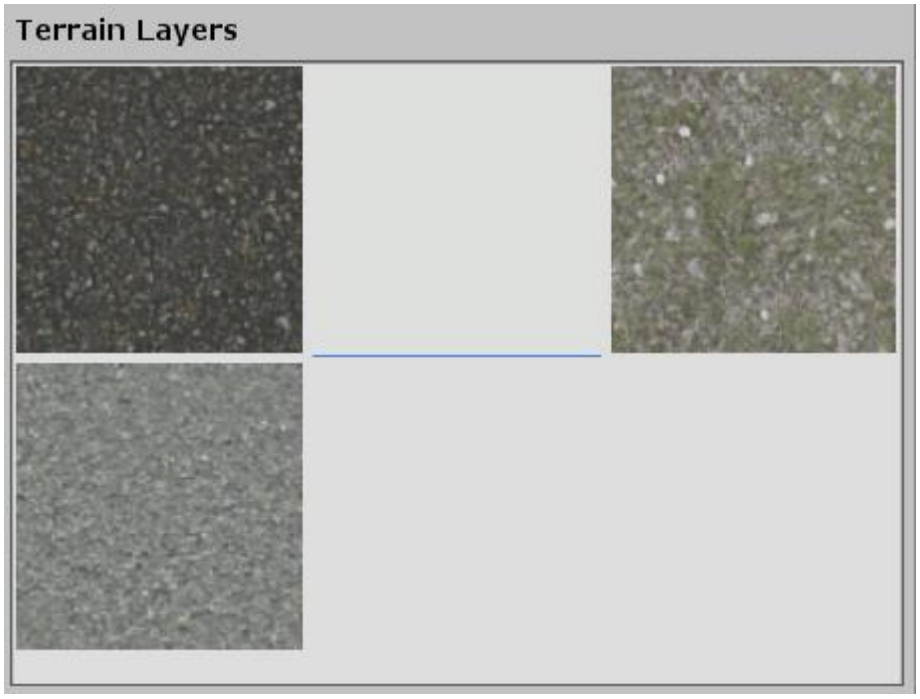


Figure 10 Different terrain layers

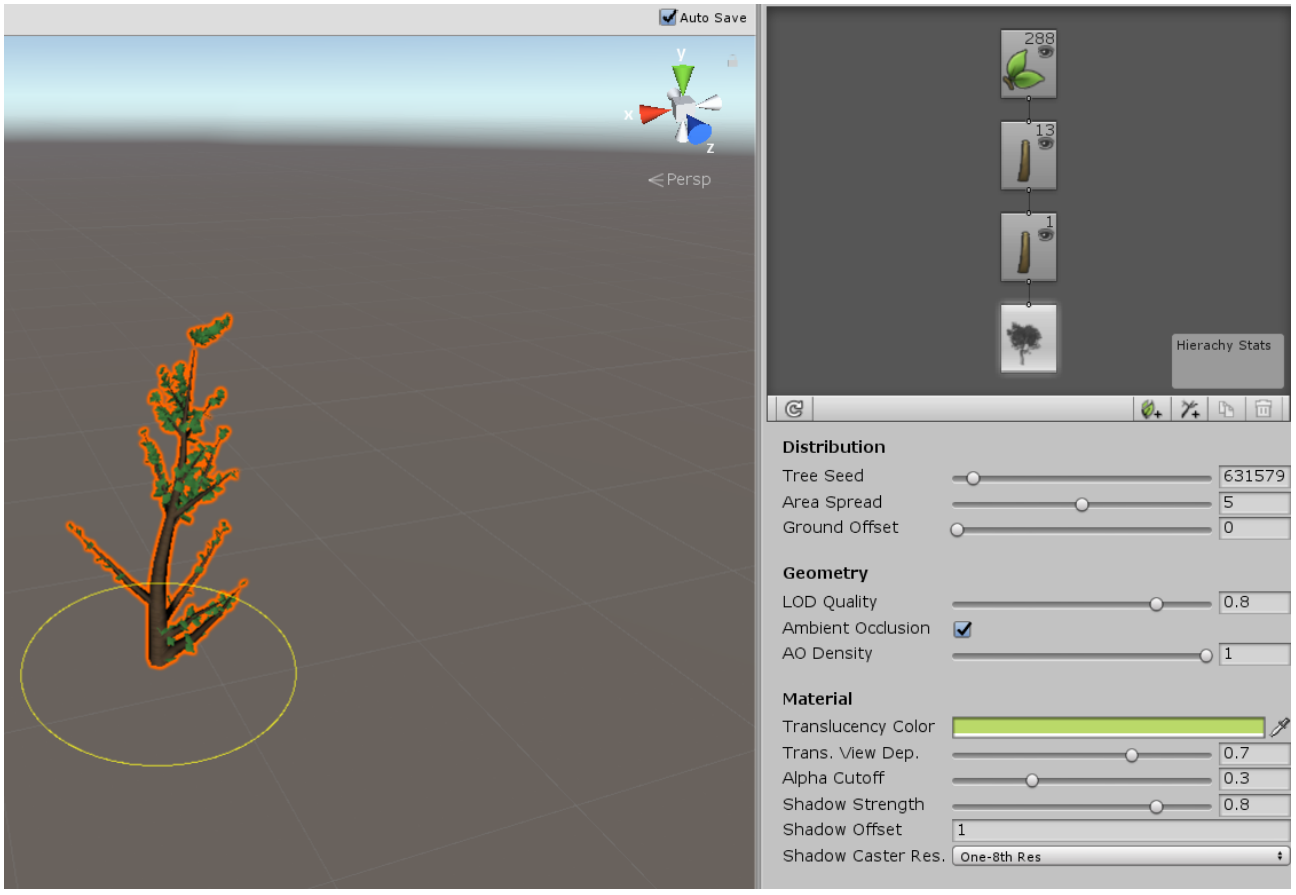


Figure 11 Tree #1

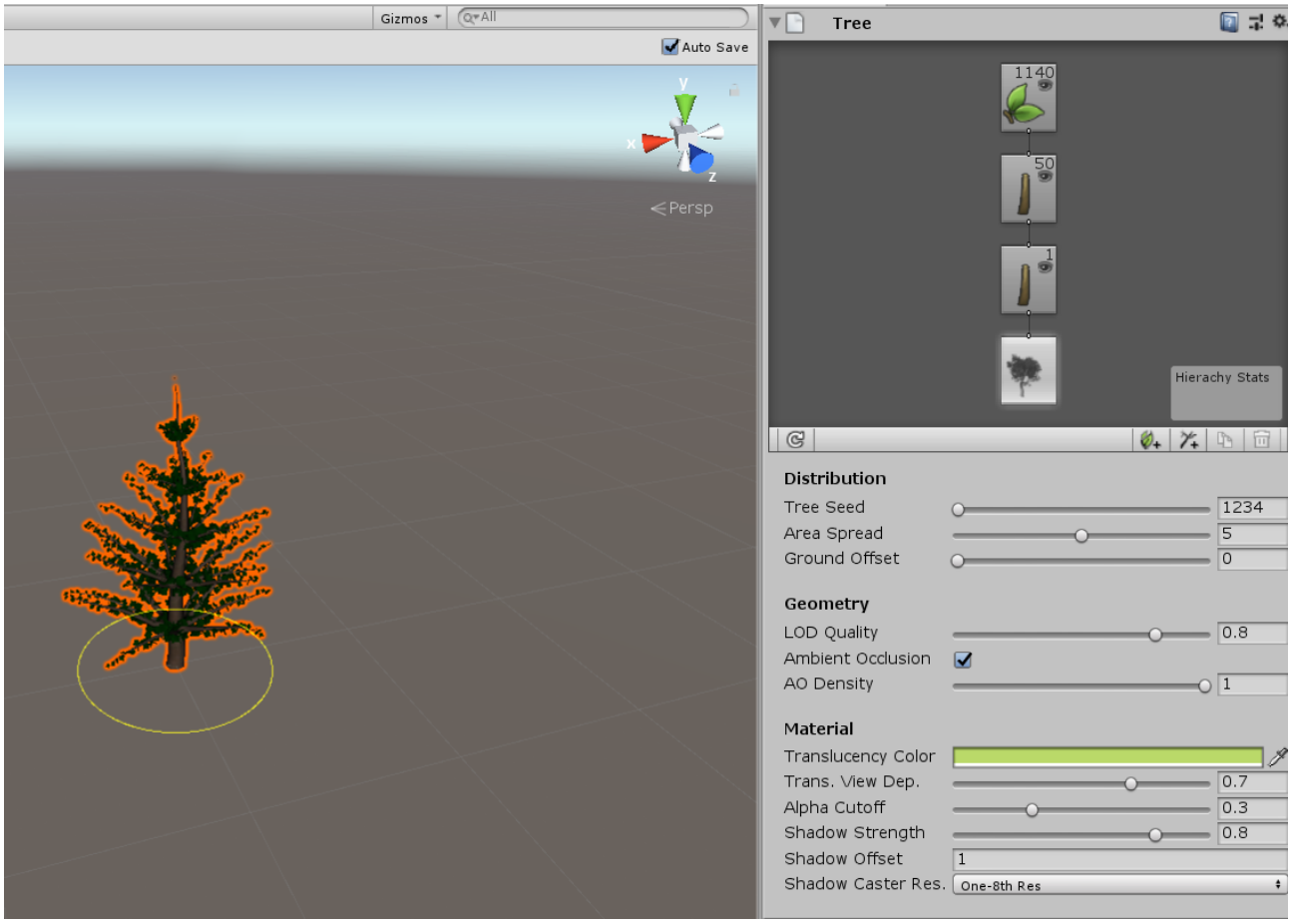


Figure 12 Tree #2

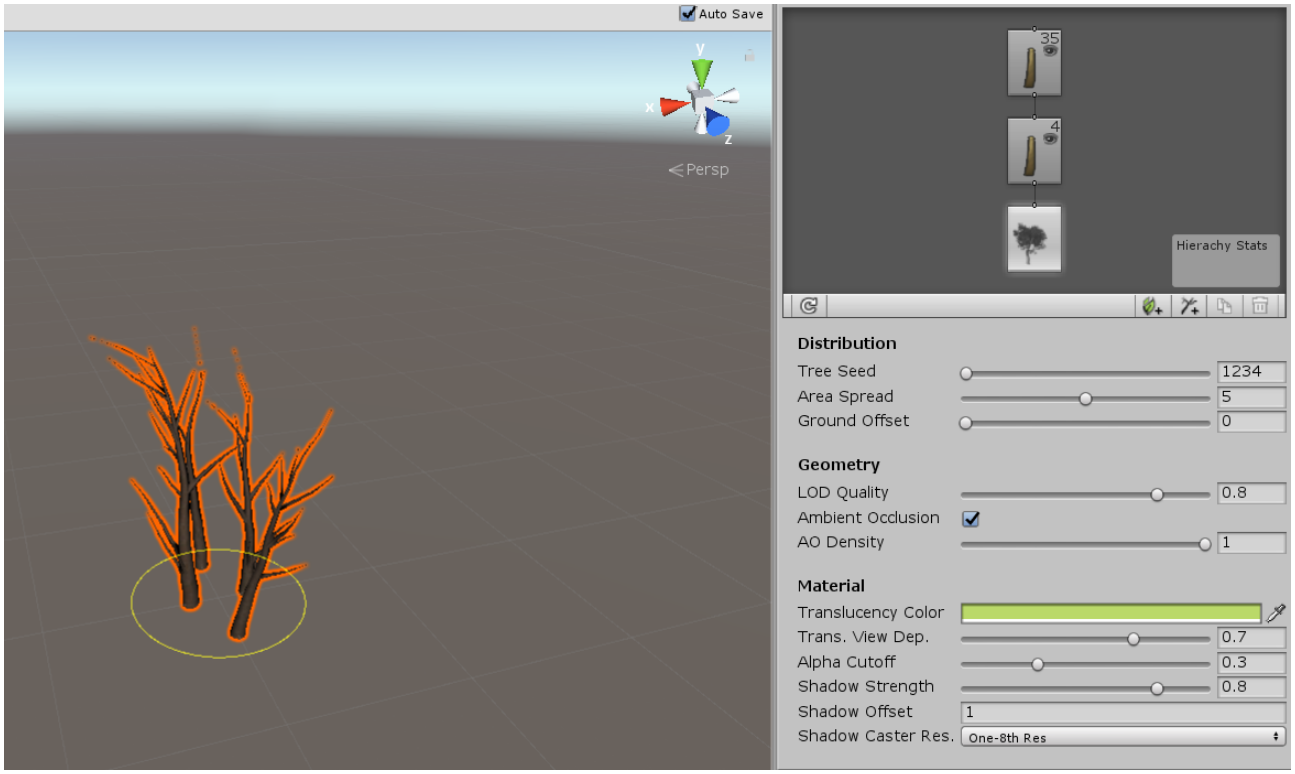


Figure 13 Tree #3

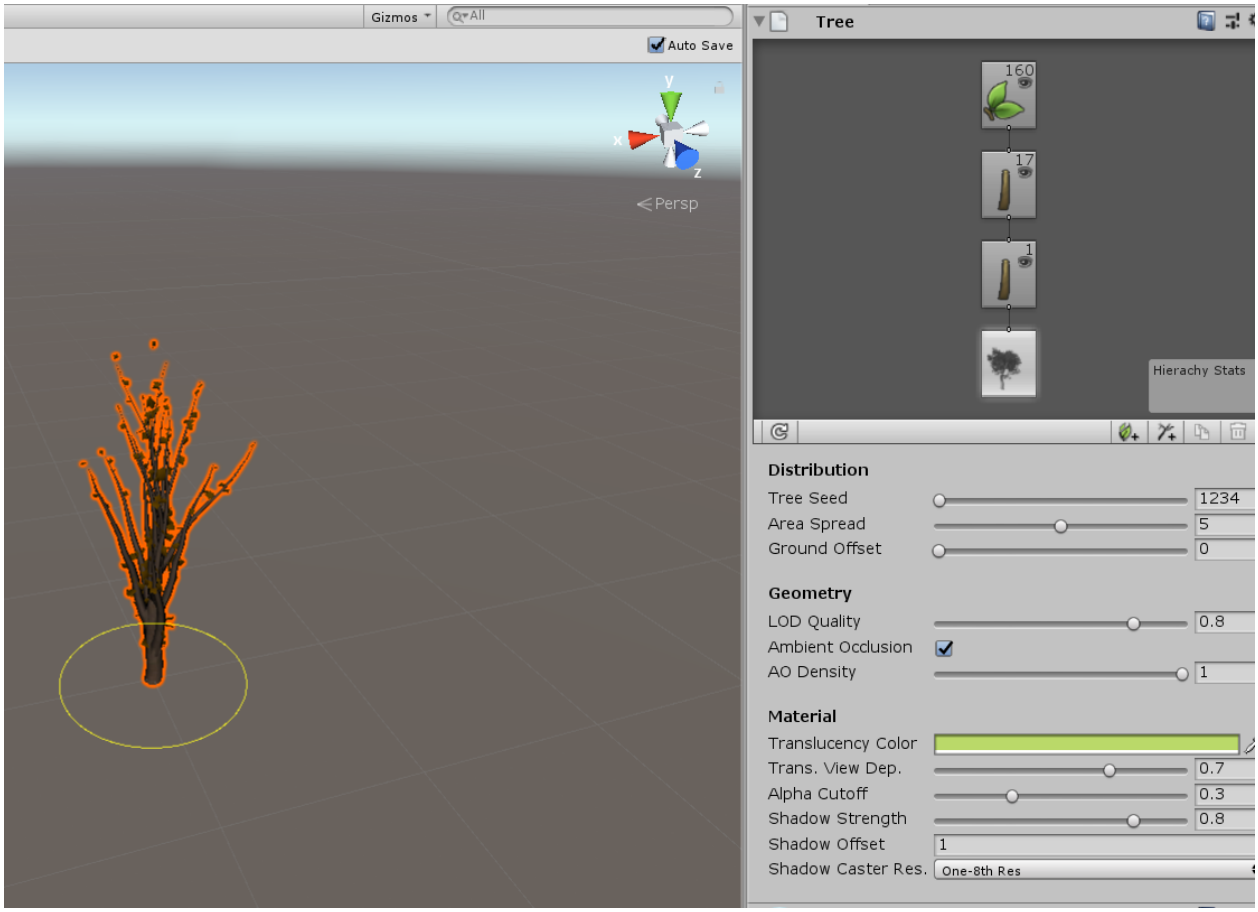


Figure 14 Tree #4

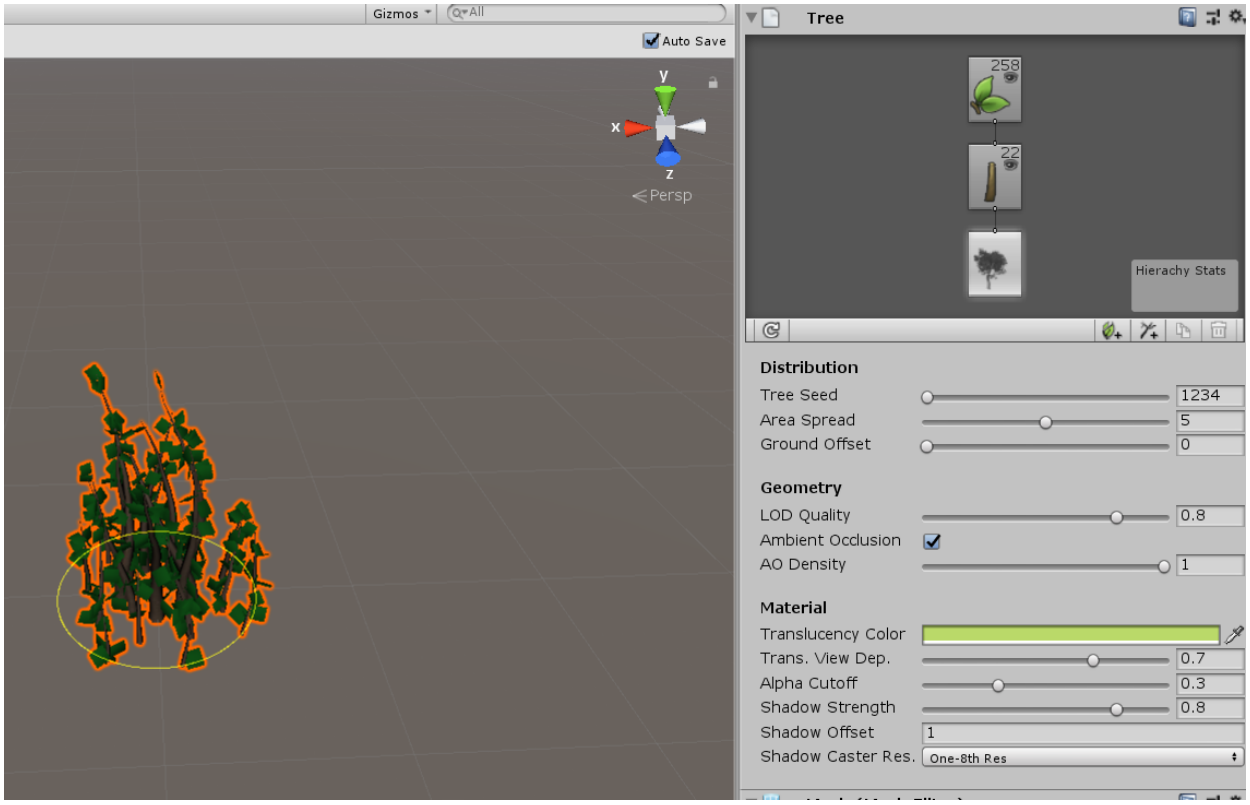


Figure 15 Tree #5

Task #3. Import a unique model and assign it to a third person character controller

Model taken from here.

Figure 16 Model with third person character controller

Task #4. Add and animate 5 objects to your game map

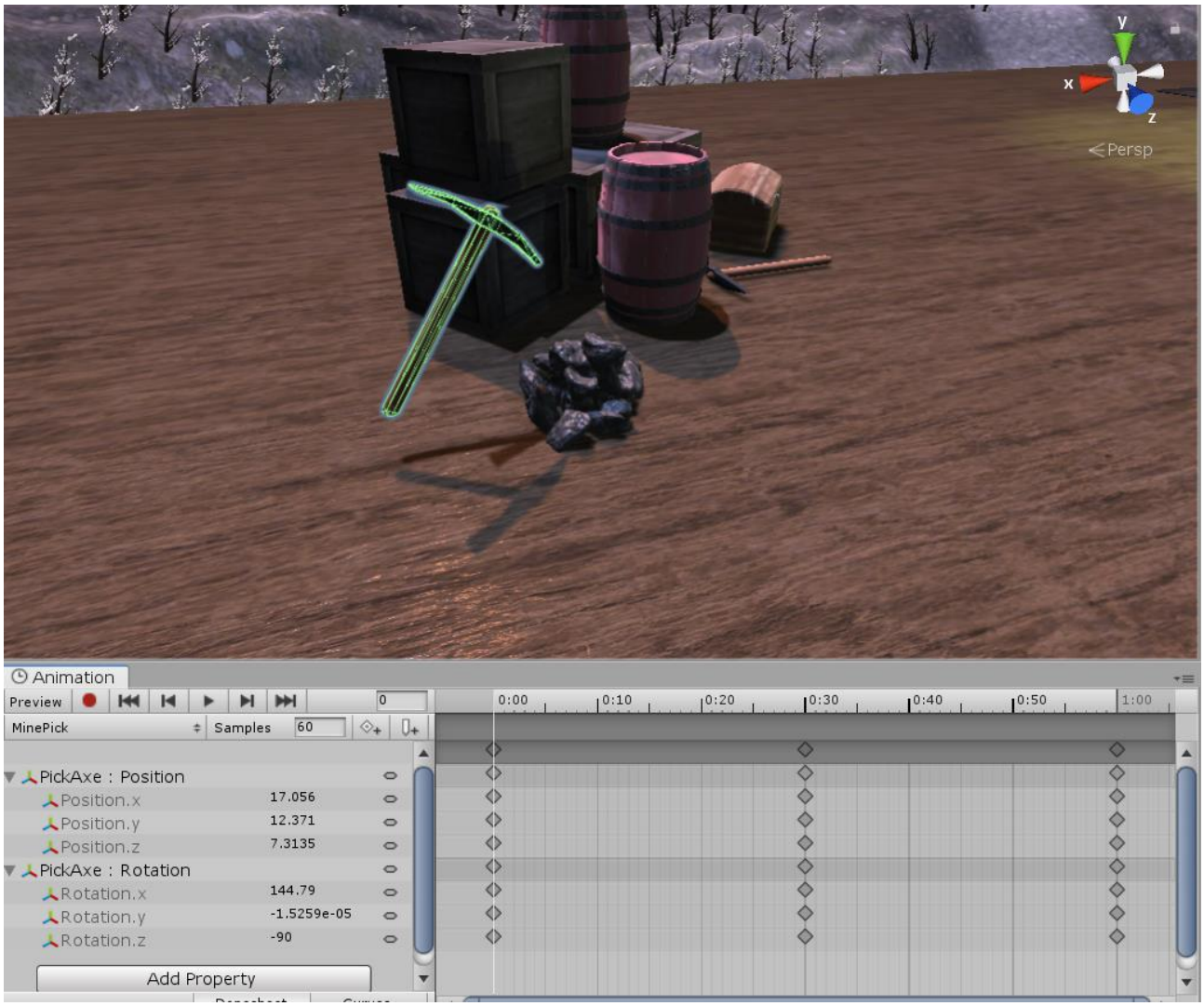


Figure 17 Animated pickaxe

Task #5. Create 5 unique particle effects

5 different particle systems added to game:

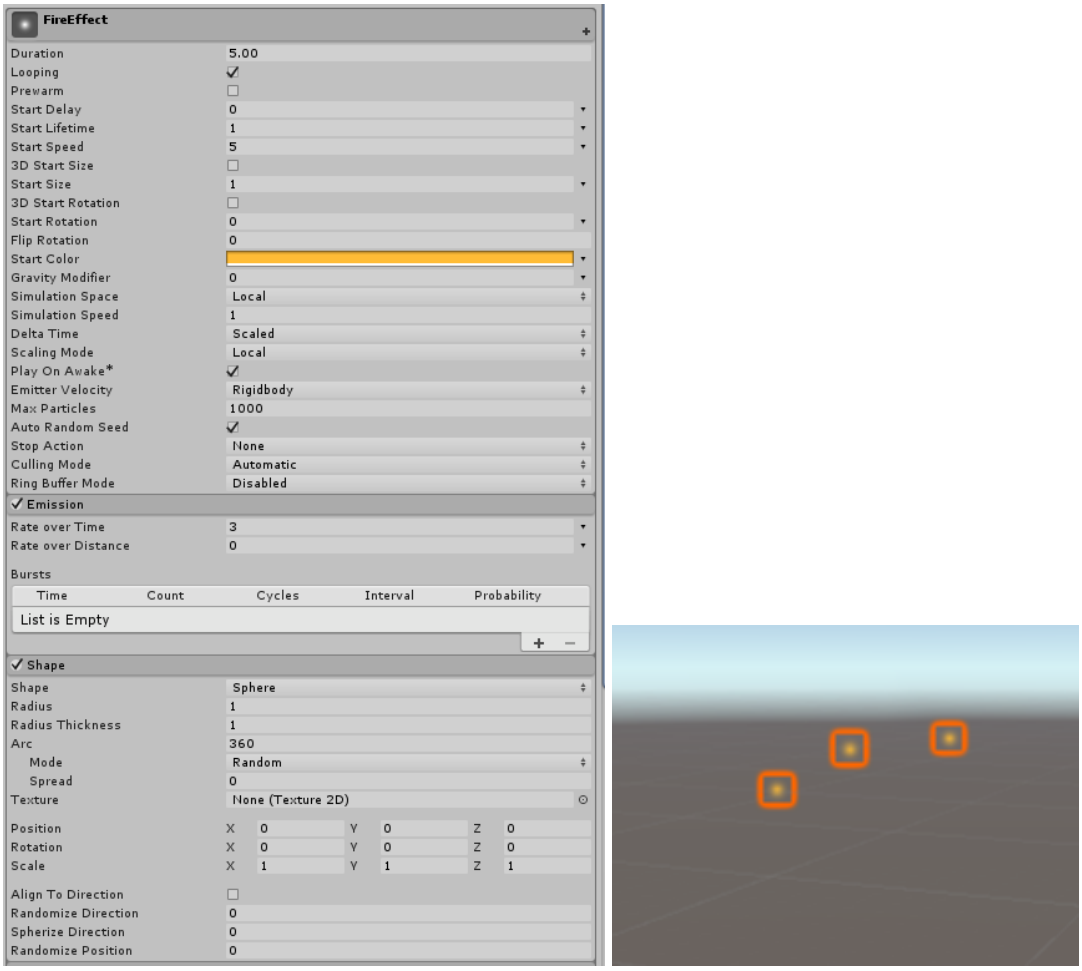


Figure 18 Fire effect particle system

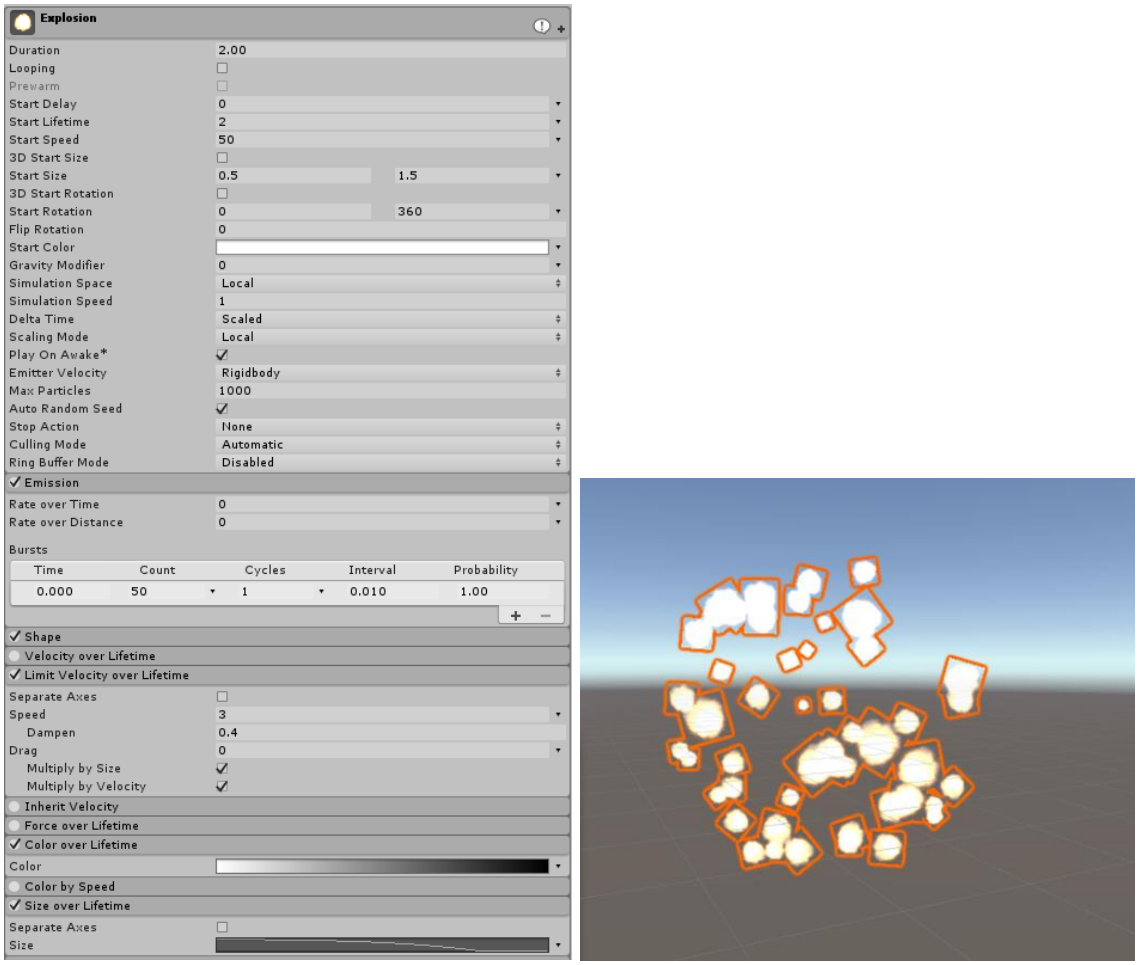


Figure 19 Explosion particle system

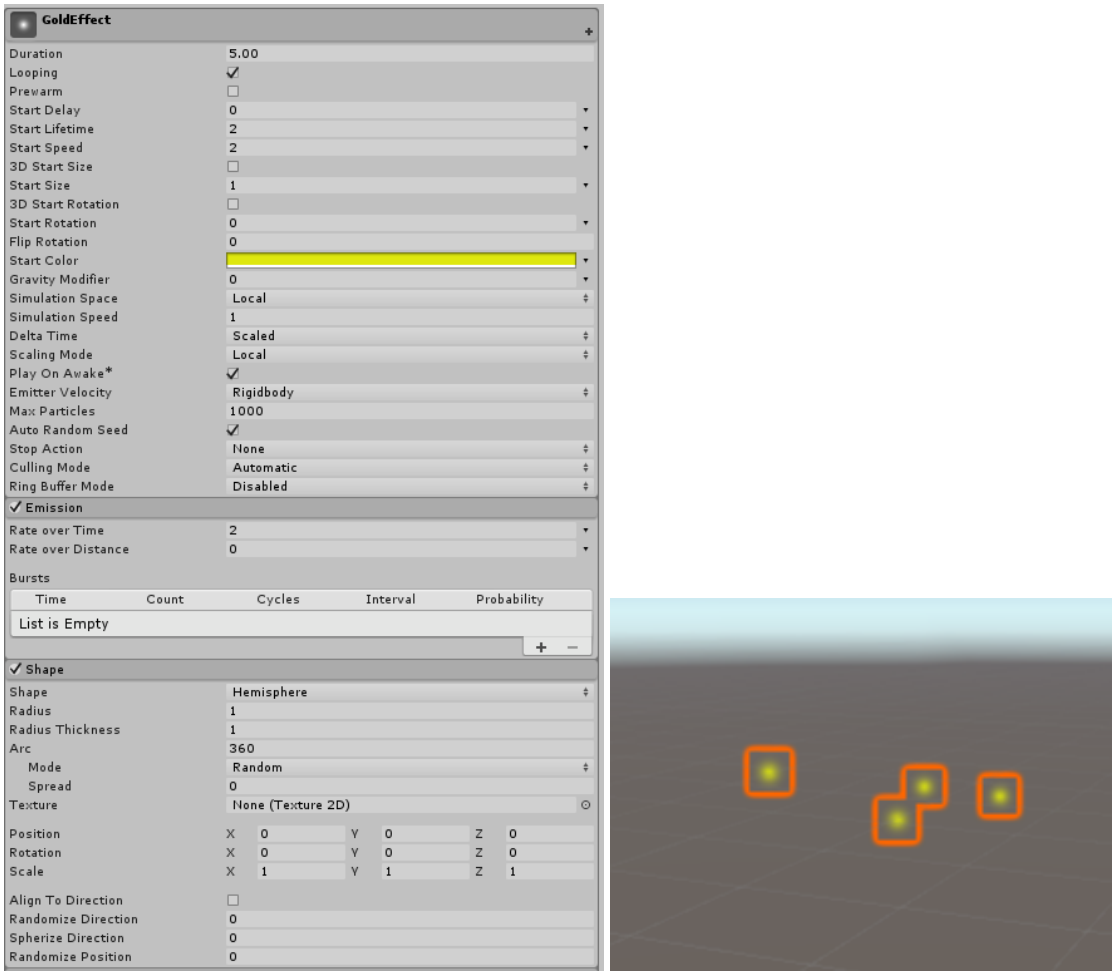


Figure 20 Gold effect particle system

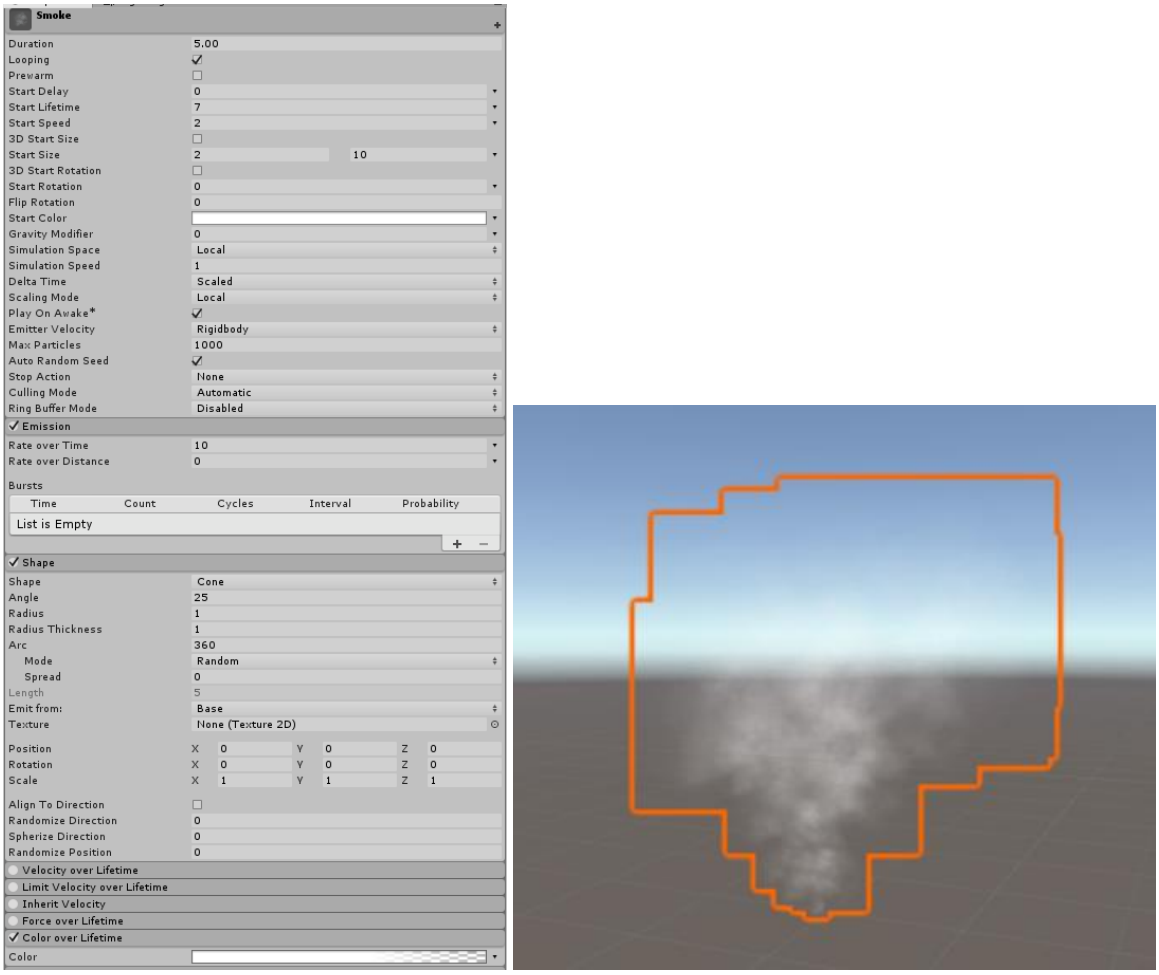


Figure 21 Smoke particle system

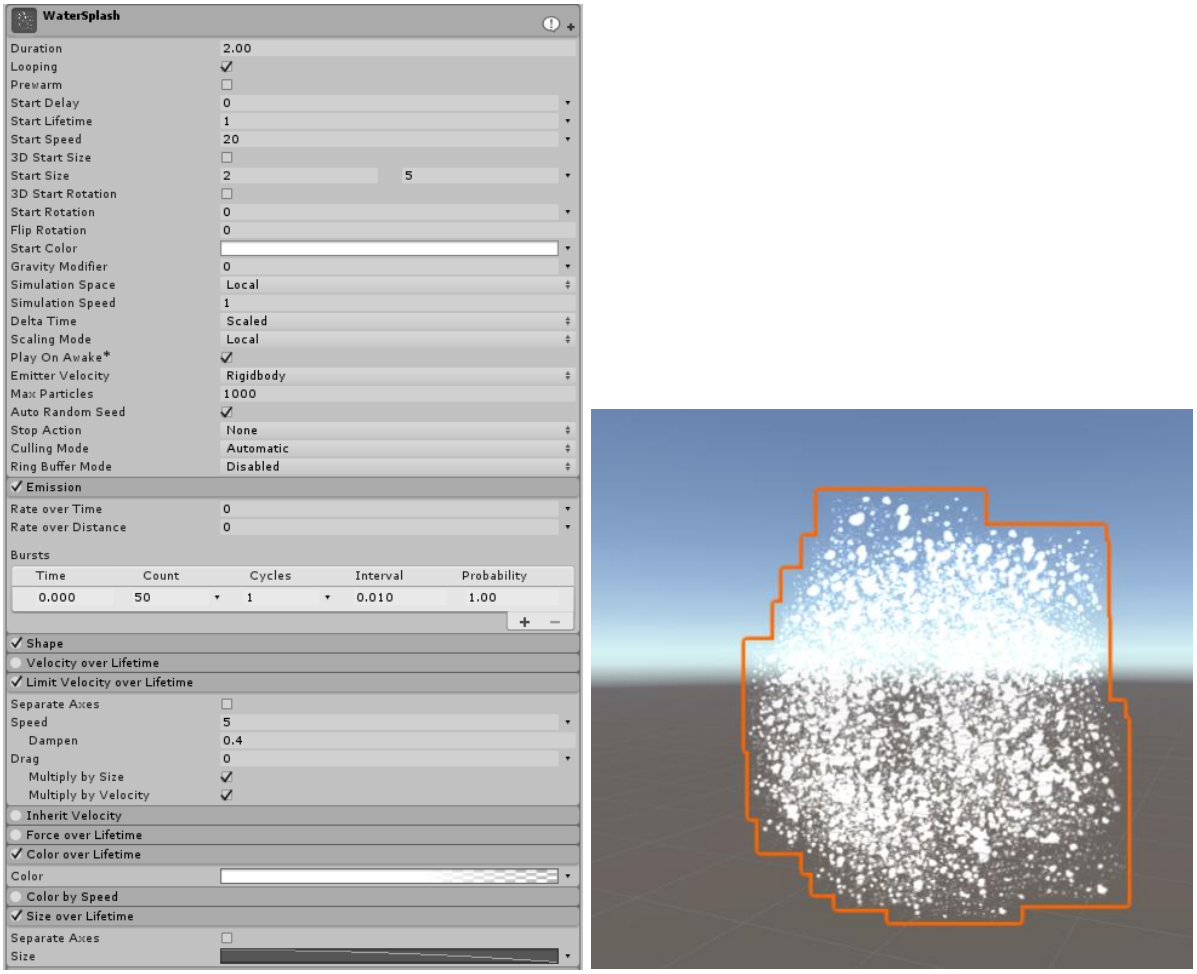
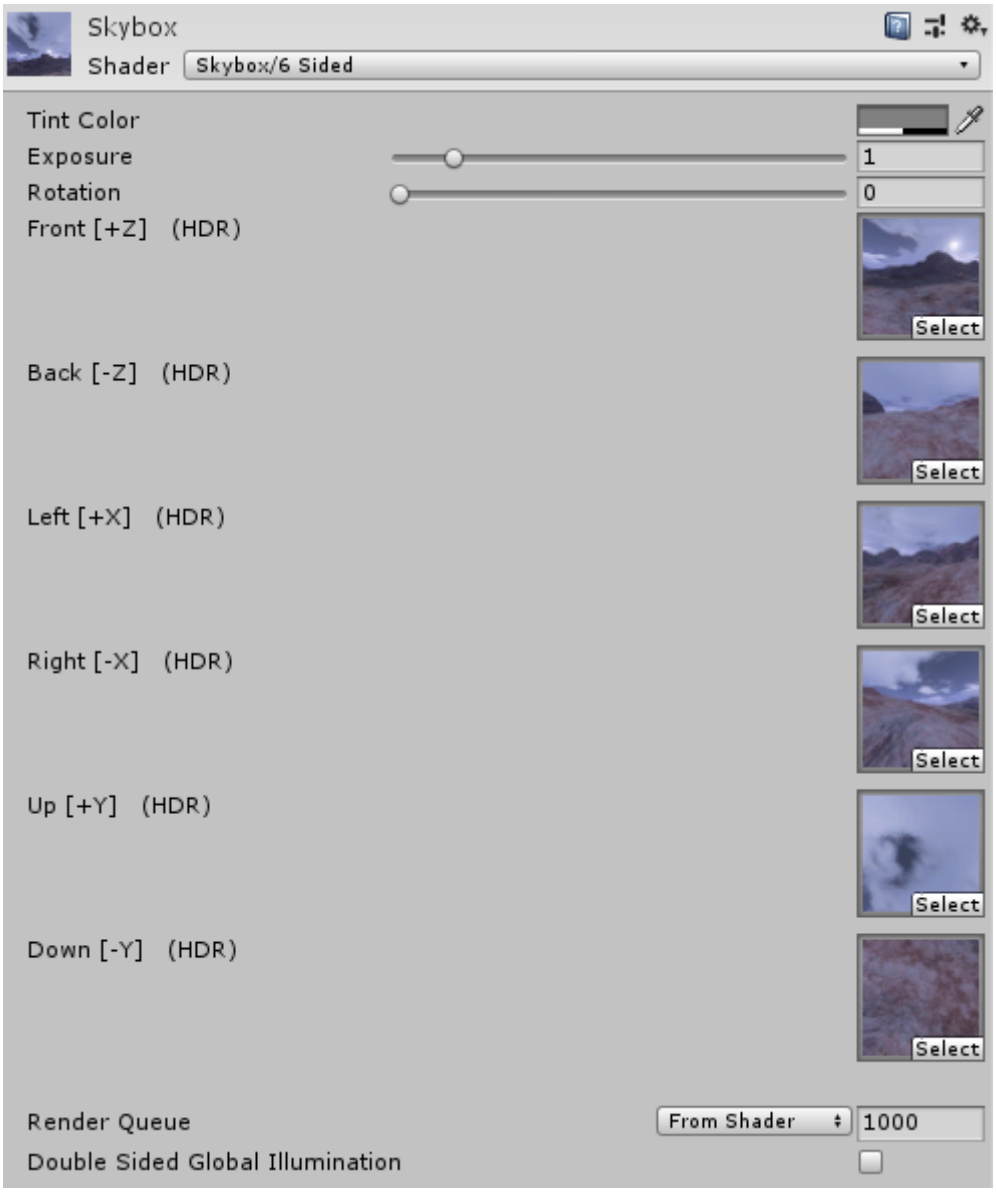


Figure 22 Water splash particle system

Task #6. Add a custom skybox

Skybox textures taken from here and used in a skybox type material.



Task #7. Create 5 different physics materials

Added 4 different physics materials:

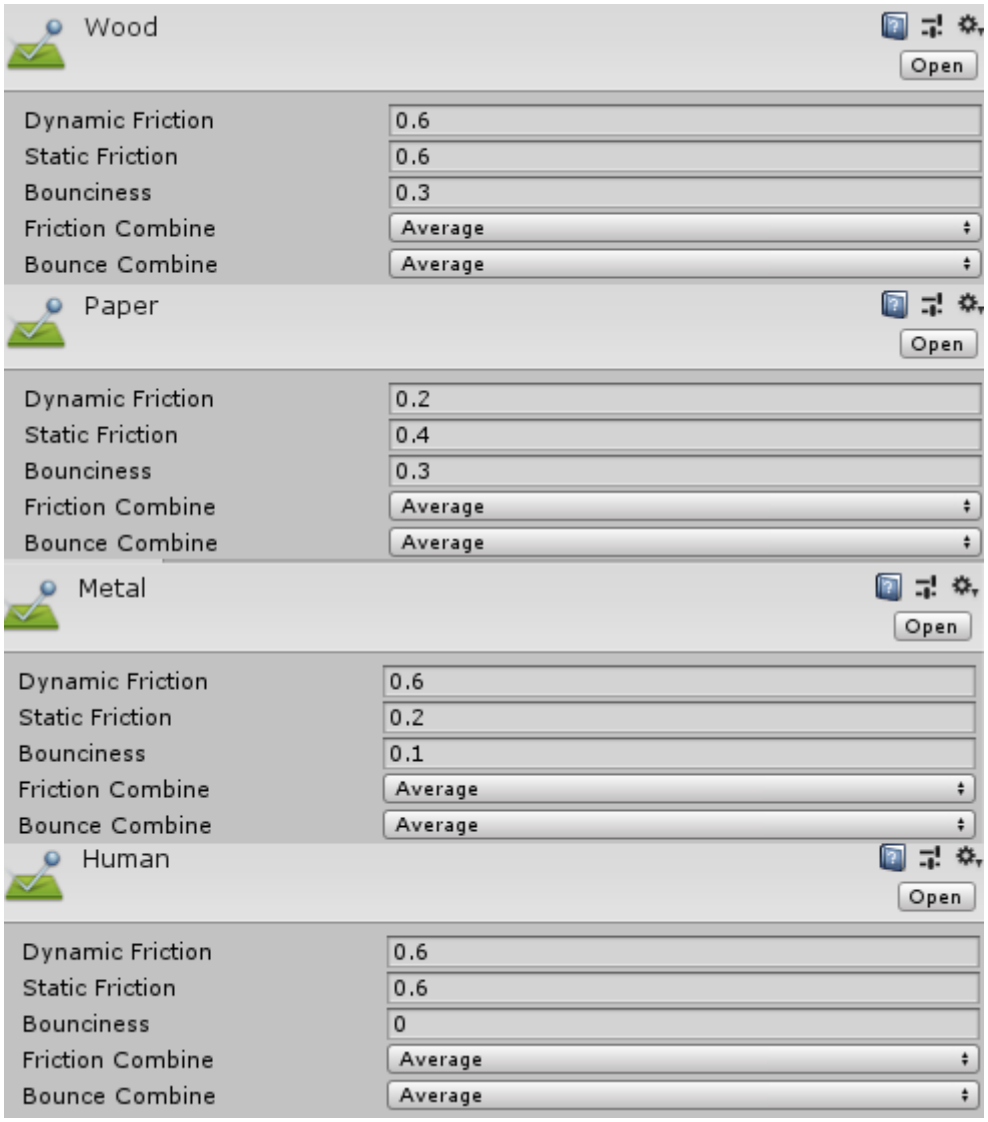


Figure 23 Physics materials

Task #8. Assign colliders for all 3D objects imported in the first laboratory work

Assigned different types of colliders for all props.

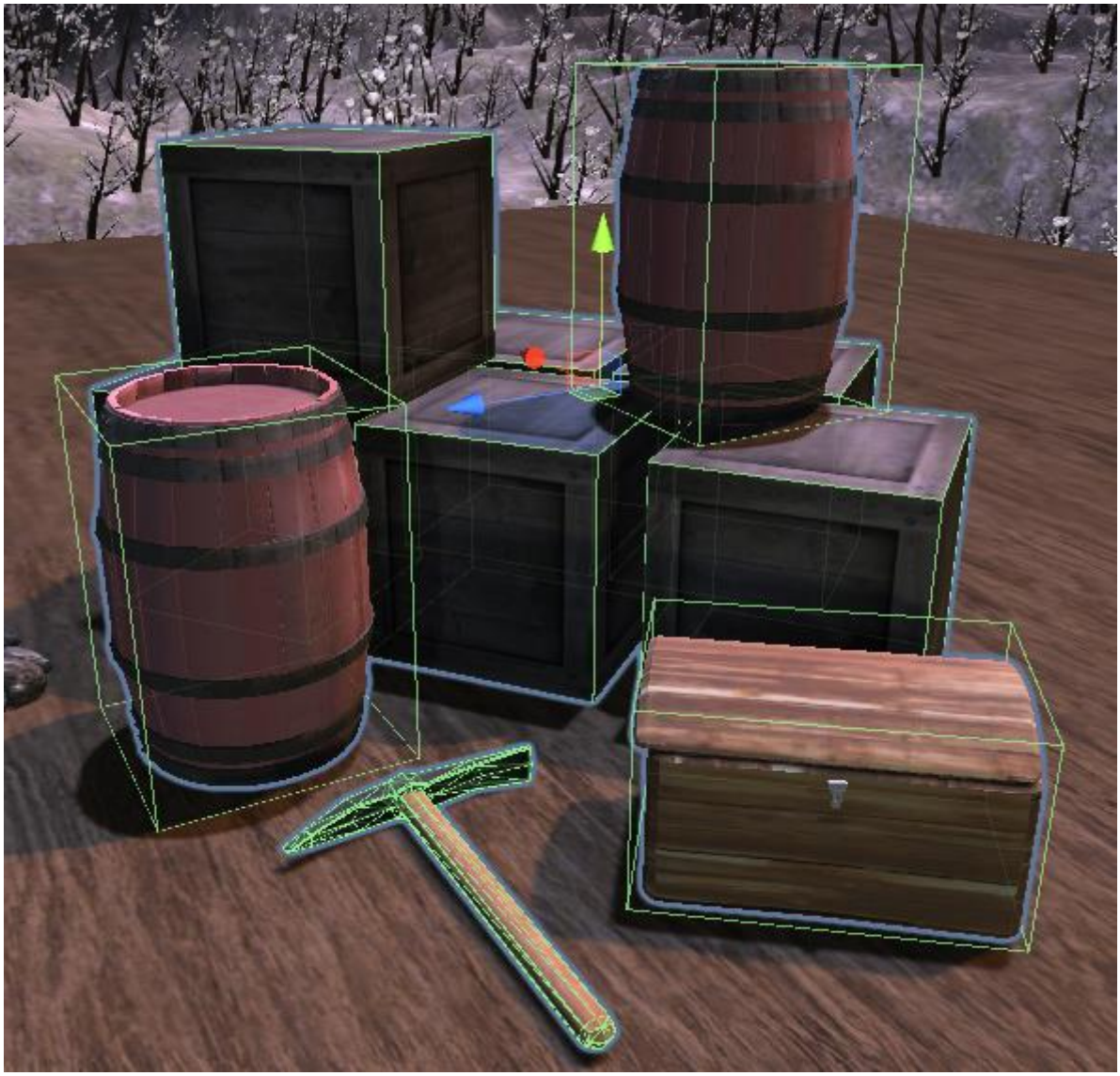


Figure 24 Colliders

Task #9. Add and animate 5 objects using physics, force and triggers to your game map

Chest is animated using triggers. On trigger enter chest opens and on trigger exit chest close. Mining pickaxe plays and loops animation when game is started.

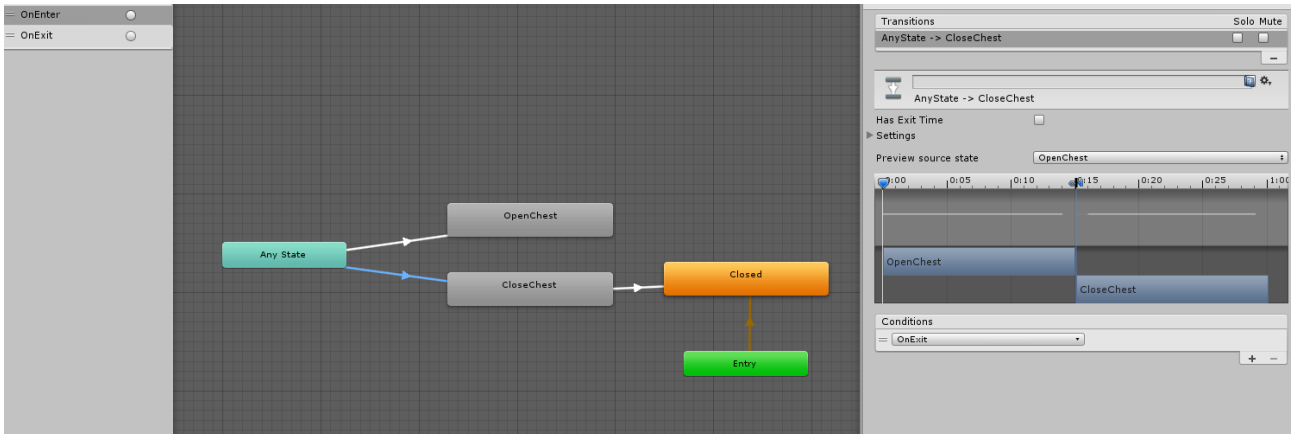


Figure 25 Chest animation controller

Table 2 Trigger script

```
public class Trigger : MonoBehaviour
{
    Animator animator;

    void Start()
    {
        animator = GetComponent<Animator>();
    }

    void OnTriggerEnter(Collider collider)
    {
        animator.SetTrigger("OnEnter");
    }

    void OnTriggerExit(Collider collider)
    {
        animator.SetTrigger("OnExit");
    }
}
```

Task #10. Set static flag to all immobile objects and measure batching performance

Batches with static flags: 1315

Batches without static flags: 1345

Task #11. Try deferred vs forward rendering and measure performance for both

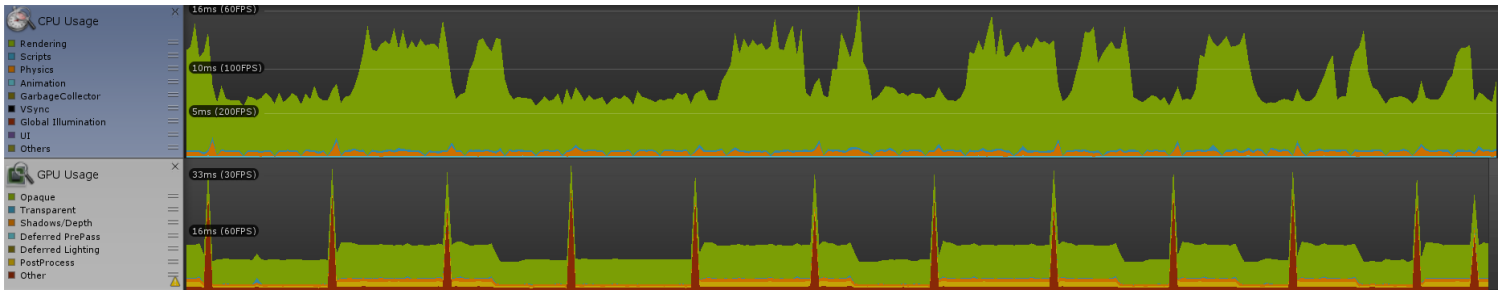


Figure 26 Forward rendering performance

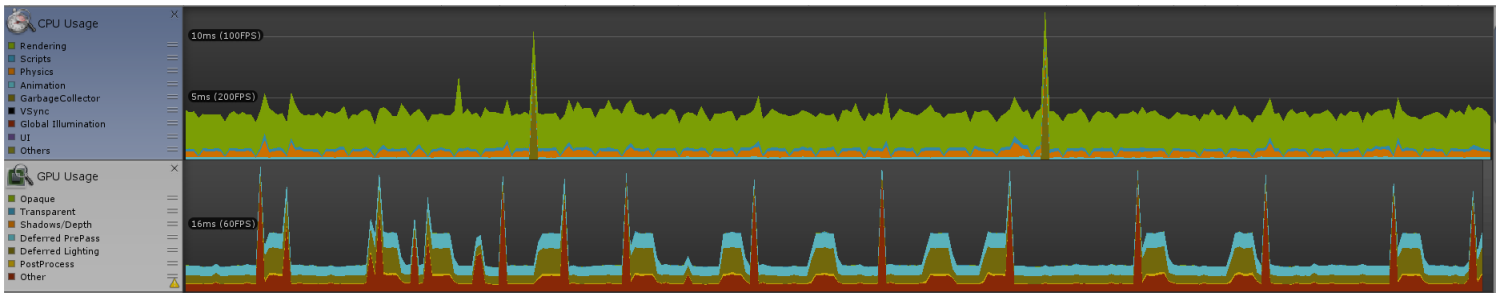


Figure 27 Deferred rendering performance

Task #12. Bake a lightmap for your scene and measure performance

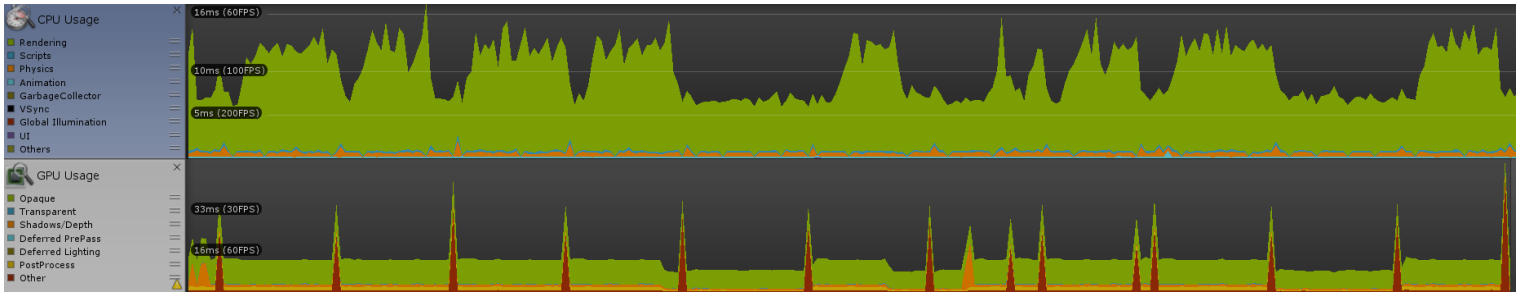


Figure 28 Default performance

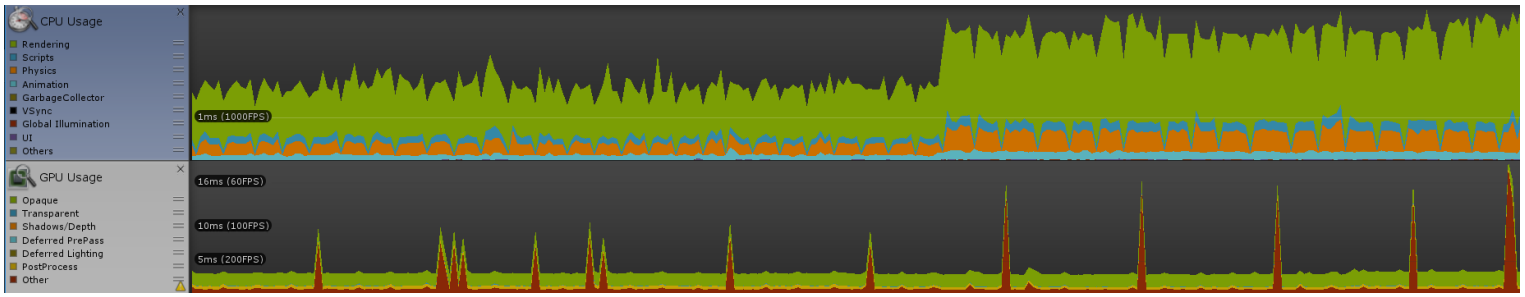


Figure 29 Performance with baked lightmap

As seen in the first picture (Figure 20), majority of CPU time is used on rendering. With baked lightmap rendering time decreased.

Task #13. Optimize all textures and measure graphical memory load

```
Used Total: 0.81 GB   Unity: 0.63 GB   Mono: 10.5 MB   GfxDriver: 61.3 MB   FMOD: 108.8 MB   Video: 0 B   Profiler: 2.6 MB
Reserved Total: 1.11 GB   Unity: 0.92 GB   Mono: 11.1 MB   GfxDriver: 61.3 MB   FMOD: 108.8 MB   Video: 0 B   Profiler: 16.0 MB
Total System Memory Usage: 2.33 GB

Textures: 492 / 152.6 MB
Meshes: 139 / 11.2 MB
Materials: 119 / 249.0 KB
AnimationClips: 27 / 4.3 MB
AudioClips: 7 / 107.5 MB
Assets: 2226
GameObjects in Scene: 325
Total Objects in Scene: 1470
Total Object Count: 3696
GC Allocations per Frame: 310 / 19.0 KB
```

Figure 30 Default memory stats

```
Used Total: 0.81 GB   Unity: 0.63 GB   Mono: 9.8 MB   GfxDriver: 60.1 MB   FMOD: 108.8 MB   Video: 0 B   Profiler: 2.6 MB
Reserved Total: 1.11 GB   Unity: 0.92 GB   Mono: 11.1 MB   GfxDriver: 60.1 MB   FMOD: 108.8 MB   Video: 0 B   Profiler: 16.0 MB
Total System Memory Usage: 2.33 GB

Textures: 530 / 127.1 MB
Meshes: 139 / 13.8 MB
Materials: 120 / 249.0 KB
AnimationClips: 27 / 4.3 MB
AudioClips: 7 / 107.5 MB
Assets: 2232
GameObjects in Scene: 325
Total Objects in Scene: 1496
Total Object Count: 3728
GC Allocations per Frame: 315 / 19.0 KB
```

Figure 31 Memory stats with optimized textures

Comparing these two images we can see that memory load for textures decreased by around 30MB.

Task #14. Try hard vs soft shadows and measure performance

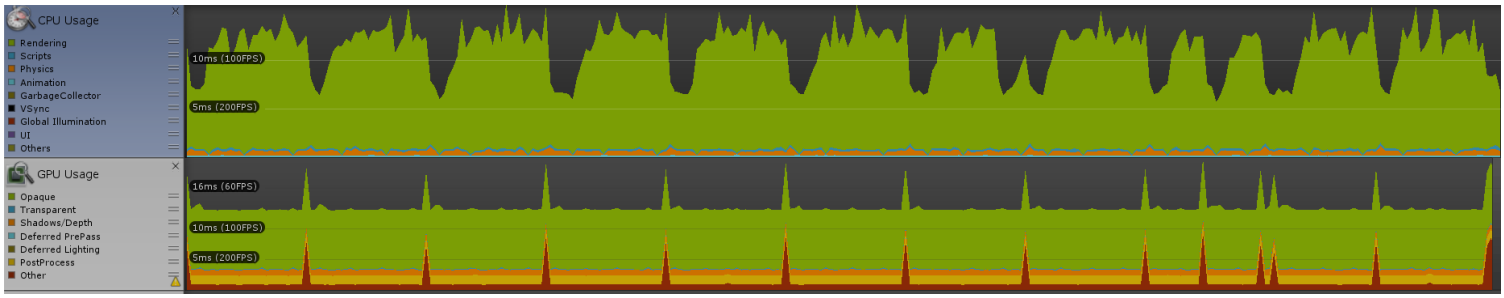


Figure 32 Performance with hard shadows

Default performance (Figure 20) was measured with soft shadows so we can use that for comparison. From these two images we can see that rendering hard shadows uses a lot more CPU time.

Defense task

Adding a ragdoll:

Figure 33 Ragdoll

Laboratory work #3

List of tasks

1. Add menu system
2. Options
3. Have GUI elements
4. Attack mechanics
5. AI implementation
6. Health/Power ups
7. 5 spawn points
8. Scoring system
9. Game over condition
10. High scores

Solution

Task #1. Add menu system

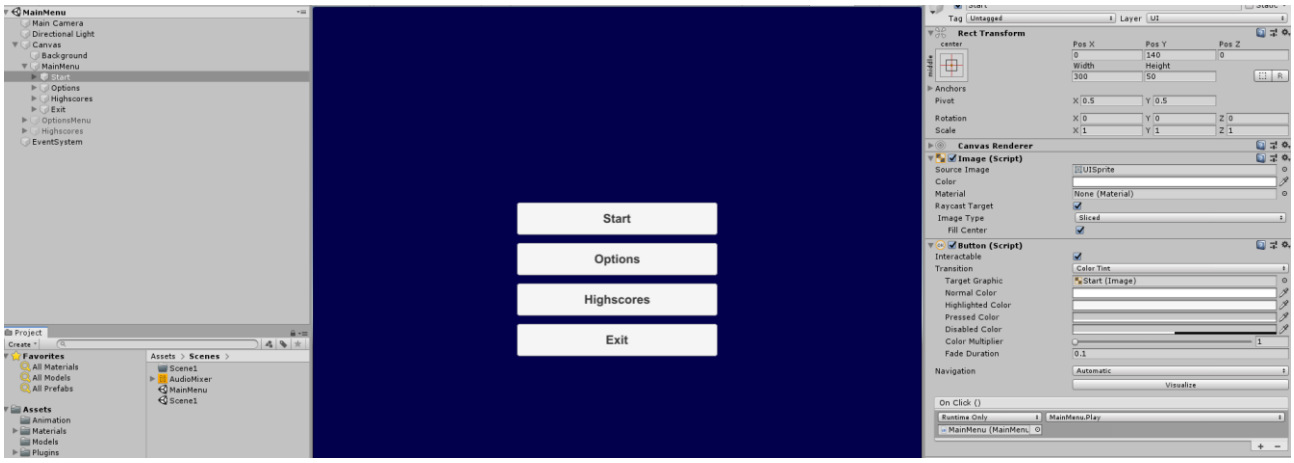


Figure 34 Main menu

Added menu system:

- Start – move to main scene and start the game
- Options – open options menu
- Highscores – show highscores
- Exit – quit game

Task #2. Options

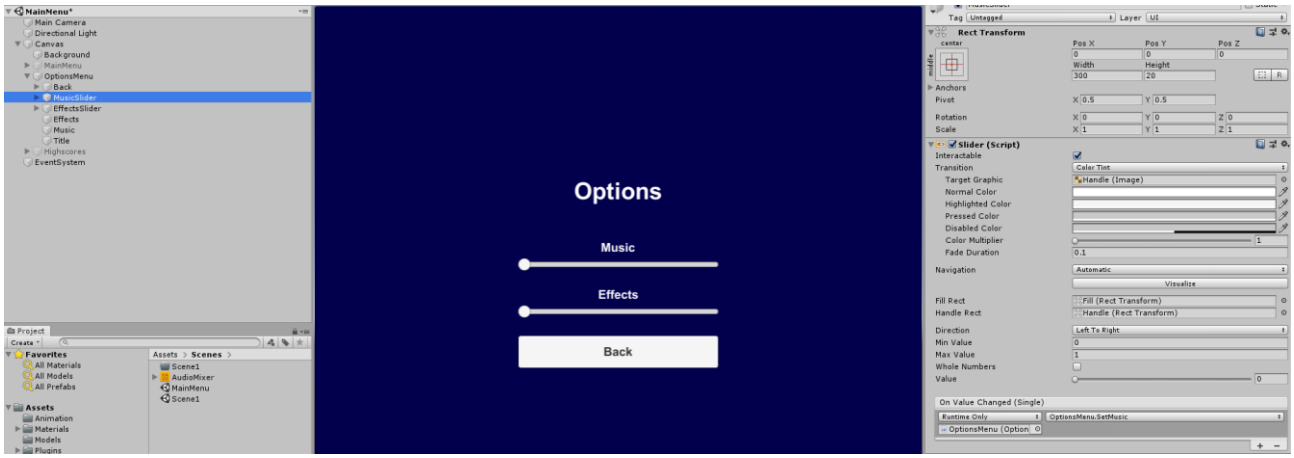


Figure 35 Options menu

Options menu contains settings for music and effects volume.

Task #3. GUI elements

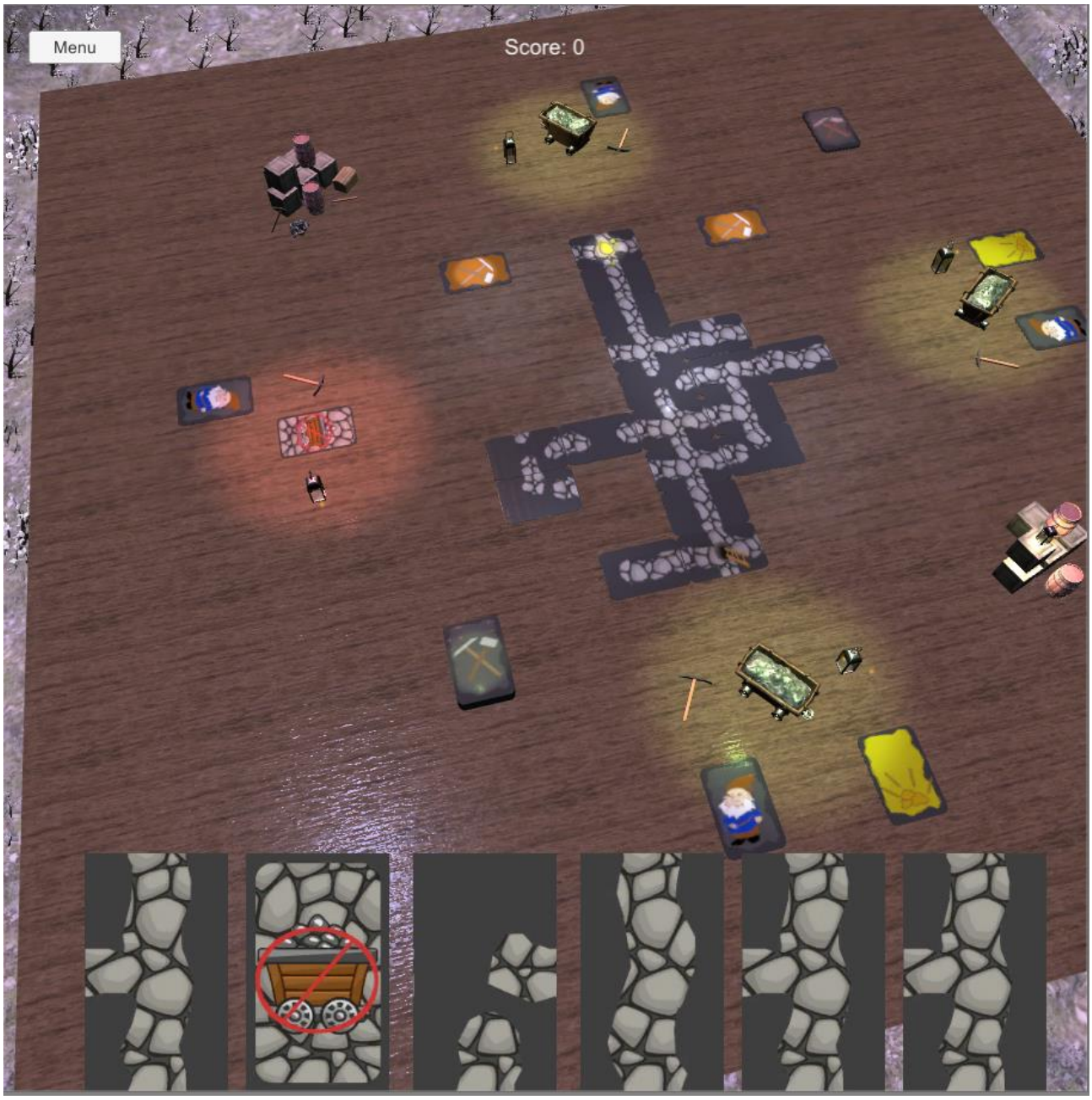


Figure 36 GUI

There are 3 main GUI elements:

- Menu button (top left)
- Current score (top center)
- Current cards in player's hand (bottom)

Task #4. Not implemented

Task #5. Not implemented

Task #6. Not implemented

Task #7. Spawn points

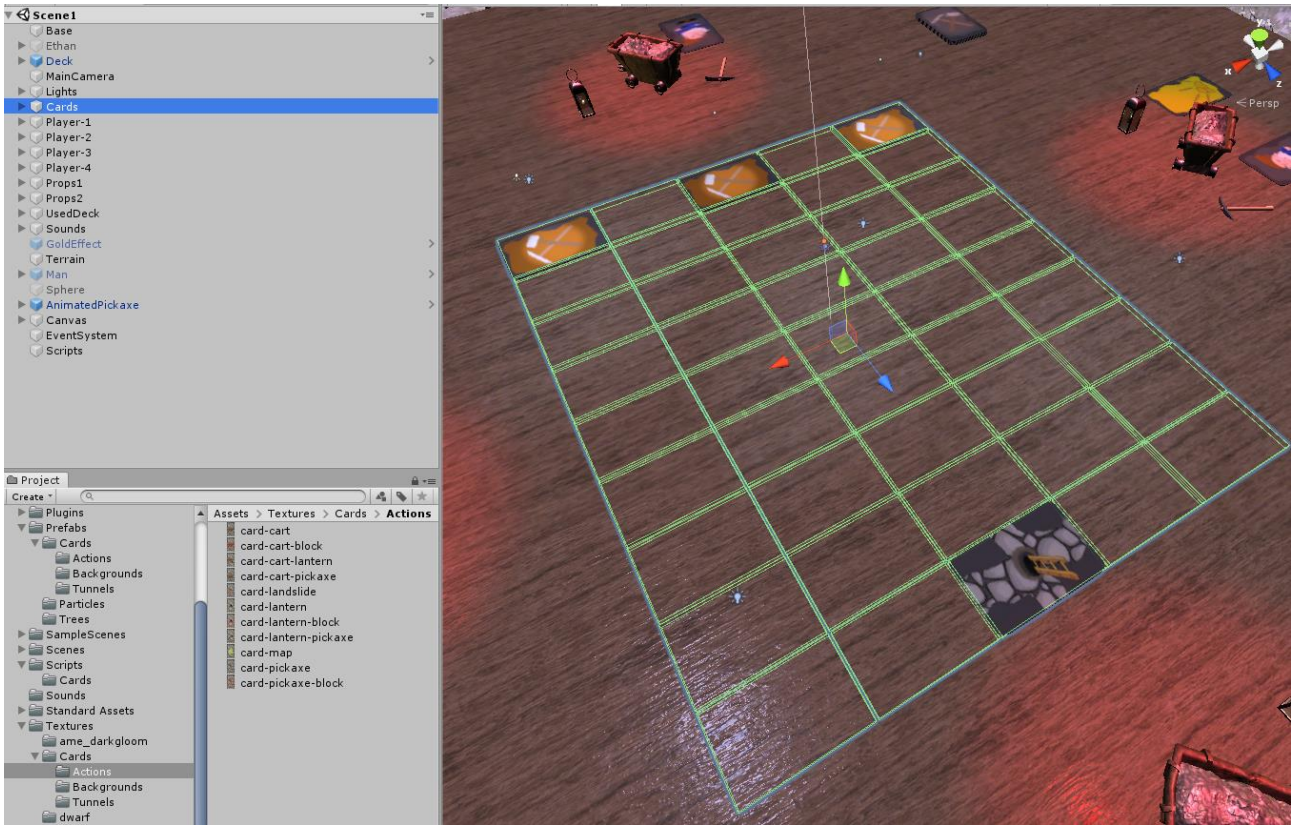


Figure 37 Grid of invisible objects set as spawnpoints

Spawn points for cards are invisible objects across the grid. Cards are instantiated from prefabs.

```
var rotation = Quaternion.Euler( x: -90,  y: 90,  z: 180);
cardObject = Instantiate(newCard.Prefab, transform.position, rotation);
card = newCard;
```

Task #8. Scoring system

In this game score is gold. After every match winner gets 1-5 gold. The players with highest score after 3 matches wins.

Figure 38 Score display

```
2 usages
private void UpdateScore()
{
    ScoreObject.text = $"Score: {score}";
}

1 usage
public void GameOver(int gold)
{
    score += gold;
    UpdateScore();
    GameOverScreen.SetActive(true);

    var currentHighscore = PlayerPrefs.GetFloat( key: "highscore", defaultValue: 0);
    if (score > currentHighscore)
    {
        PlayerPrefs.SetFloat("highscore", score);
    }
}
```

Task #9. Game over condition

Game is over when dwarfs reaches gold or runs out of cards. Random number of gold (1-5) is given for the winners.

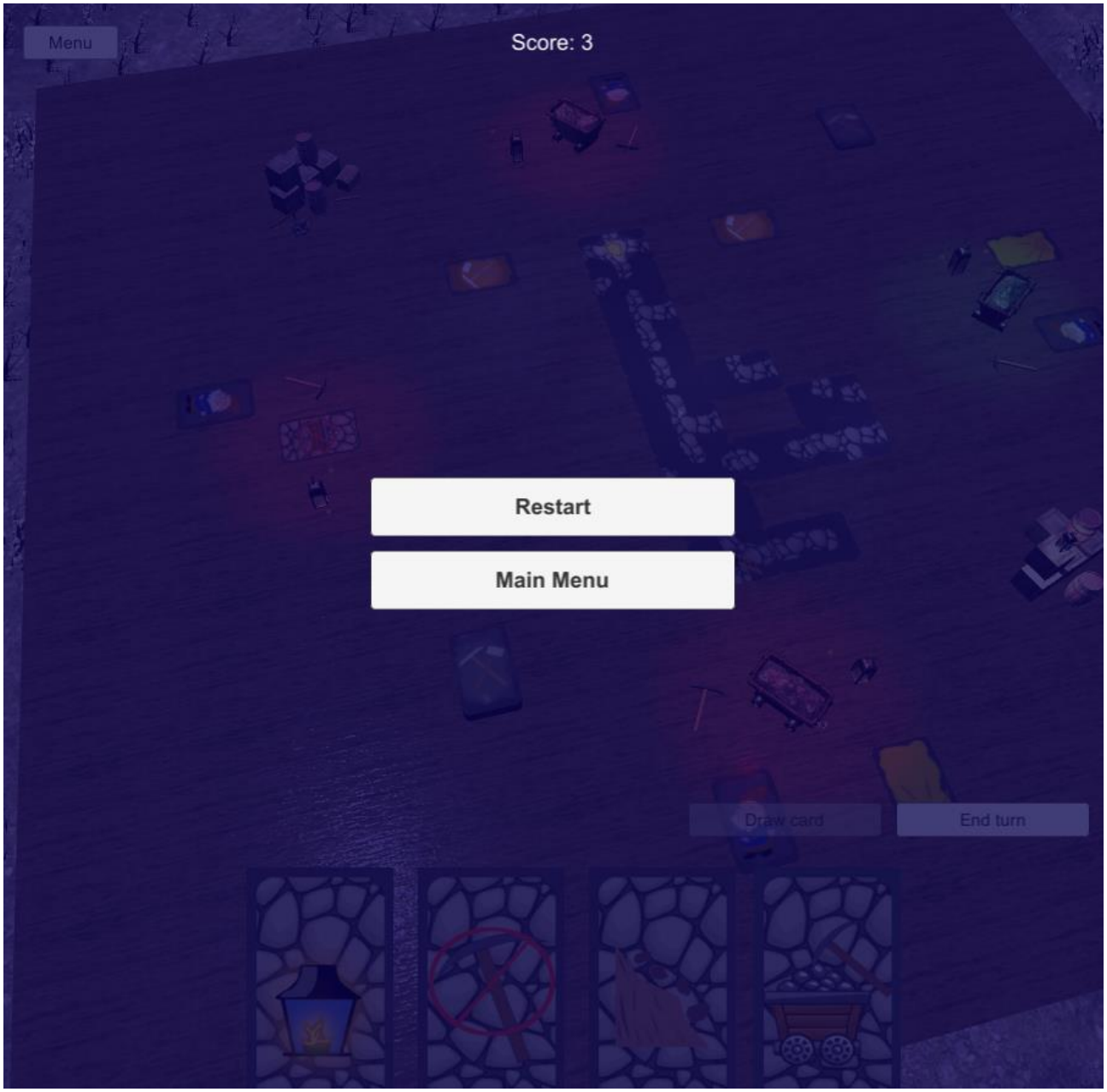


Figure 39 Game over screen

When player gets to the finnish card, it is fliped and if it has gold it is added to the score and game is over.

```
1 usage
private void Flip()
{
    if (!IsFinnish)
    {
        return;
    }

    cardObject.transform.Rotate(new Vector3( x: -180,  y: 0,  z: 0));
    if (gold > 0)
    {
        main.GameOver(gold);
    }
}
```

Task #10. Highscores

Highscore is saved in player prefs, and updated after every game.

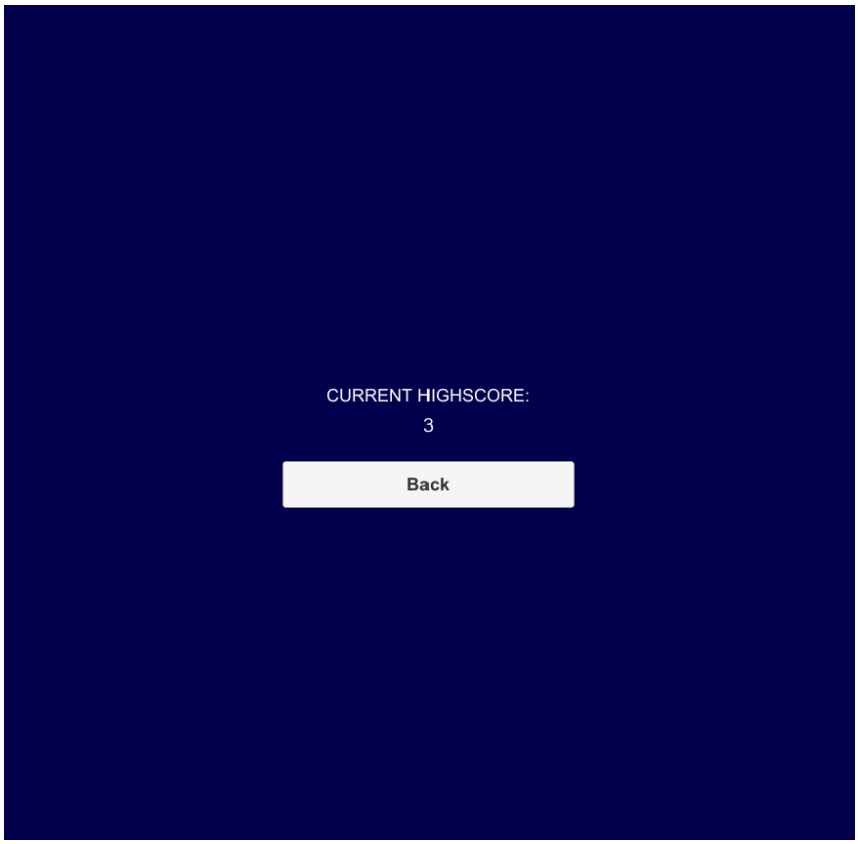


Figure 40 Highscore view

```
var currentHighscore = PlayerPrefs.GetFloat("highscore", 0);
if (score > currentHighscore)
{
    PlayerPrefs.SetFloat("highscore", score);
}
```

Defense task

Update code so every player has a card with a straight tunnel.

Added extra card with UP and DOWN connections for every player on start.

```
var cc = AllCards.FirstOrDefault(x =>
    x.connections.Contains(Card.Directions.Up) && x.connections.Contains(Card.Directions.Down));
for (var i = 0; i < Players.Count; i++)
{
    Players[i].Id = i;
    Players[i].DropHand();
    for (var j = 0; j < 5; j++)
    {
        Players[i].AddCard(DrawCardFromDeck());
    }
    Players[i].AddCard(cc);
}
```

Main functionality

Main functionality of this game is placing card and validating if they can be placed. I created scriptable object for each cards with various parameters (e.g. isAction, connections..). Parameter connections shows in witch direction a card has tunnel ending. Player can only connect tunnel ending if there is a tunnel ending on the neighbor card.

There's a code snippet that validates if a card can be placed:

```
private bool CanTunnelBePlaced(Card tunnel)
{
    if (Left != null && Left.card != null)
    {
        if (tunnel.connections.Contains(Card.Directions.Left) &&
            !Left.card.connections.Contains(Card.Directions.Right))
        {
            return false;
        }

        if (!tunnel.connections.Contains(Card.Directions.Left) &&
            Left.card.connections.Contains(Card.Directions.Right))
        {
            return false;
        }
    }

    if (Right != null && Right.card != null)
    {
        if (tunnel.connections.Contains(Card.Directions.Right) &&
            !Right.card.connections.Contains(Card.Directions.Left))
        {
            return false;
        }

        if (!tunnel.connections.Contains(Card.Directions.Right) &&
            Right.card.connections.Contains(Card.Directions.Left))
        {
            return false;
        }
    }
```

```
    if (Up != null && Up.card != null)
    {
        if (tunnel.connections.Contains(Card.Directions.Up) &&
            !Up.card.connections.Contains(Card.Directions.Down))
        {
            return false;
        }

        if (!tunnel.connections.Contains(Card.Directions.Up) &&
            Up.card.connections.Contains(Card.Directions.Down))
        {
            return false;
        }
    }

    if (Down != null && Down.card != null)
    {
        if (tunnel.connections.Contains(Card.Directions.Down) &&
            !Down.card.connections.Contains(Card.Directions.Up))
        {
            return false;
        }

        if (!tunnel.connections.Contains(Card.Directions.Down) &&
            Down.card.connections.Contains(Card.Directions.Up))
        {
            return false;
        }
    }

    return true;
}
```

There is a grid for cards to be placed. Each grid cell contains a placeholder object that has reference to card (if it's placed in that spot) and neighbor placeholders.

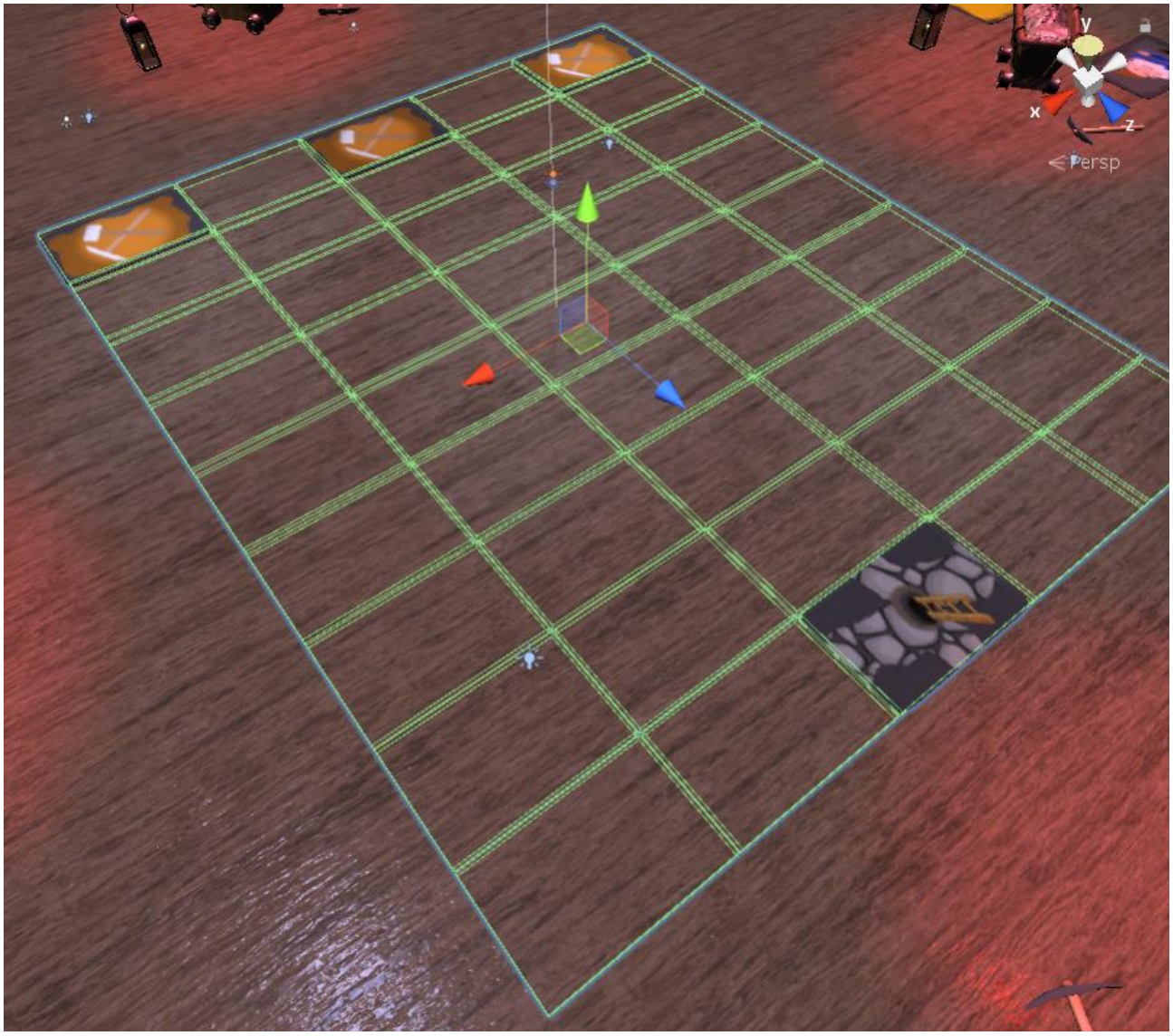


Figure 41 Card placegolder grid

User's manual

How to play

When game is open, user can:

- Press "Start" button to start a new game
- Press "Options" button to show options
- Press "Highscores" button to view current highscore

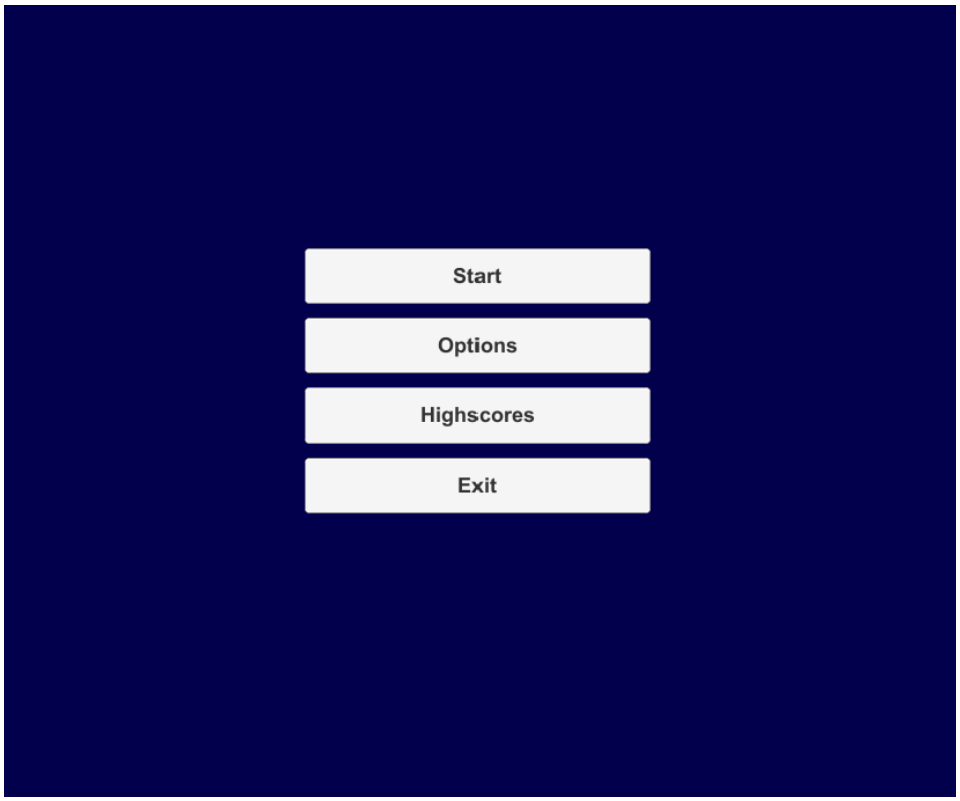


Figure 42 Main menu

When game is started user is given a hand of cards. User can select any card and place it if it's a tunnel card or use it on a player if it's an action card. A green mark is visible if a card can be used.

Figure 43 Placing tunnel card

Figure 44 Placing action card

User can also press „Draw card“ button to draw a card (if he has less than 6). Or end his turn by pressing „End turn“ button.

User can see his character car at the top left, and any blocks applied to him above his hand cards.

Figure 45 Character and blocks UI

Game rules

In this game action is set in a mine. Players take on the role of dwarves, some are miners, some are saboteurs, but no one knows who is on their side. For miners goal is to get to the gold until the cards run out. For saboteurs the goal is to stop miners from reaching gold. After three rounds the player with the most gold wins. Players need to place tunnel cards to get to the gold. Players can also apply blocks to other players, blocked players can not place any tunnels, but can use action cards to block other players or unblock themselves.

Controls

The game is controlled using mouse (left-button only). Player clicks on a card to select/deselect it and then clicks on a place to set the tunnel or a player to apply action card.

Literature list

Resources used from:

- https://www.textures.com/download/pbr0139/133174
- https://www.textures.com/download/3dscans0079/128150
- https://www.freelancer.com/contest/Concept-Art-SKETCH-Gnome-390351-byentry-8768095
- https://www.freepik.com/free-vector/cartoon-stone-texture_976364.htm
- https://www.vectorstock.com/royalty-free-vector/mine-cart-with-coal-vector-1747485
- https://all-free-download.com/free-vector/download/treasure-map-clip-art_23028.html
- https://www.kisspng.com/png-cartoon-ladder-illustration-cartoon-wooden-ladder-497343/
- https://www.turbosquid.com/3d-models/pickaxe-obj-free/912027
- https://sketchfab.com/3d-models/mine-cart-2beeab5f44704421bc10cd310db96860
- https://www.textures.com/download/3dscans0110/130564
- https://www.textures.com/download/pbr0031/133067
- https://sketchfab.com/3d-models/lantern-final-aead6bb3bab344eaab541c5ac52c657c
- https://sketchfab.com/3d-models/40-rocks-a6c5ab5b438f473cb1cafdf4099e1657
- https://free3d.com/3d-model/crate-86737.html
- https://free3d.com/3d-model/barrel-7685.html
- https://free3d.com/3d-model/wooden-chest-44006.html
- sounds from https://www.productioncrate.com and https://www.zapsplat.com
- https://www.textures.com/download/pbr0172/133207
- https://www.textures.com/download/asphaltcloseups0064/12319
- http://www.custommapmakers.org/skyboxes.php
- https://sketchfab.com/3d-models/elysiumvr-male-character-rigged-template-4d0f03c6734749218042546586667l0d

Annex

Main.cs

```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.UI;

public class Main : MonoBehaviour
{
    private const int maxHandSize = 6;

    public Text ScoreObject;
    public GameObject GameOverScreen;
    public MainGUIControl GuiControl;
    public List<Card> AllCards;
    public Button DrawCardButton;
    public CardPlaceholder Start;

    public Player CurrentPlayer => Players[playerTurn];
    public Card SelectedCard => selectedCard != -1 ? CurrentPlayer.Hand[selectedCard] : null;
    public bool IsTunnelCardSelected => SelectedCard != null && !SelectedCard.IsAction;

    private List<Card> deck = new List<Card>();
    private int score;

    public List<Player> Players;
    private int playerTurn = -1;
    private int selectedCard = -1;
    private bool madeMove;

    private static Main _instance;
    public static Main Instance => _instance;
    private readonly System.Random random = new System.Random();

    private Color enemyColor = Color.red;
    private Color playerColor = Color.green;

    public Card DrawCardFromDeck()
    {
        var index = random.Next(0, deck.Count - 1);
        var card = deck[index];
        deck.Remove(card);

        if (deck.Count == 0)
        {
            DrawCardButton.interactable = false;
        }

        return card;
    }

    private void Awake()
    {
        if (_instance != null && _instance != this)
        {
            Destroy(gameObject);
        } else {
            _instance = this;
        }

        Reset();
```

```
        UpdateScore();
    }

    private void Reset()
    {
        score = 0;
        playerTurn = 0;
        deck = new List<Card>();
        deck.AddRange(AllCards);
        deck.AddRange(AllCards);

        for (var i = 0; i < Players.Count; i++)
        {
            Players[i].Id = i;
            Players[i].DropHand();
            for (var j = 0; j < 5; j++)
            {
                Players[i].AddCard(DrawCardFromDeck());
            }
        }

        NextTurn();
    }

    public void DrawCard()
    {
        if (CurrentPlayer.Hand.Count < maxHandSize)
        {
            DrawCardButton.interactable = false;
            madeMove = true;
            CurrentPlayer.AddCard(DrawCardFromDeck());
            GuiControl.UpdateCards(CurrentPlayer.Hand, CurrentPlayer.Blocks,
CurrentPlayer.IsSaboteur);
        }
    }

    public void NextTurn()
    {
        madeMove = false;
        DrawCardButton.interactable = deck.Any();
        UpdateSpotLight(enemyColor);
        playerTurn++;
        if (playerTurn >= Players.Count)
        {
            playerTurn = 0;
        }
        UpdateSpotLight(playerColor);
        GuiControl.UpdateCards(CurrentPlayer.Hand, CurrentPlayer.Blocks,
CurrentPlayer.IsSaboteur);
    }

    private void UpdateSpotLight(Color color)
    {
        CurrentPlayer.Light.color = color;
    }

    private void UpdateScore()
    {
        ScoreObject.text = $"Score: {score}";
    }

    public void GameOver(int gold)
    {
        score += gold;
        UpdateScore();
        GameOverScreen.SetActive(true);
```

```
        var currentHighscore = PlayerPrefs.GetFloat("highscore", 0);
        if (score > currentHighscore)
        {
            PlayerPrefs.SetFloat("highscore", score);
        }
    }

    public bool SelectCard(int i)
    {
        if (i == -1 && selectedCard != -1)
        {
            selectedCard = i;
            return true;
        }

        if (madeMove || CurrentPlayer.Blocks.Any() && !CurrentPlayer.Hand[i].IsAction)
        {
            return false;
        }

        selectedCard = i;
        return true;
    }

    public bool CanInteractWithCard(Card card, bool isFinnish, bool canPlaceTunnel)
    {
        return card == null && SelectedCard != null && !SelectedCard.IsAction && !isFinnish &&
canPlaceTunnel
            || card != null && SelectedCard != null &&
SelectedCard.actionTypes.Contains(Card.ActionType.Landslide)
            || card == null && SelectedCard != null &&
SelectedCard.actionTypes.Contains(Card.ActionType.Map) && isFinnish
            || SelectedCard != null && SelectedCard.actionTypes.Contains(Card.ActionType.Map) &&
isFinnish;
    }

    public void CardUsed()
    {
        madeMove = true;
        DrawCardButton.interactable = false;
        CurrentPlayer.DiscardCard(selectedCard);
        selectedCard = -1;
        GuiControl.UpdateCards(CurrentPlayer.Hand, CurrentPlayer.Blocks,
CurrentPlayer.IsSaboteur);
    }

    public bool CanInteractWithPlayer(int id)
    {
        if (SelectedCard == null || SelectedCard.IsBlock && id == playerTurn)
        {
            return false;
        }

        return SelectedCard.IsBlock
&& !Players[id].Blocks.Contains(SelectedCard.actionTypes.First())
               || SelectedCard.IsAction && Players[id].Blocks.Any(x =>
SelectedCard.actionTypes.Contains(x));
    }

    public bool PathToStartExists(CardPlaceholder node, List<CardPlaceholder> visited = null)
    {
        if (!node.card.IsConnected)
        {
            return false;
        }
```

```
        visited = visited ?? new List<CardPlaceholder>();
        visited.Add(node);

        var neighbours = node.GetNeighbours();

        if (neighbours.Any(x => x.IsStart))
        {
            return true;
        }

        neighbours = neighbours.Where(x => !visited.Contains(x)).ToList();

        return neighbours.Any(x => PathToStartExists(x, visited));
    }
}
```

MainGUIControl.cs

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

// ReSharper disable once InconsistentNaming
public class MainGUIControl : MonoBehaviour
{
    public List<RawImage> HandCards;
    public RawImage BlockCart;
    public RawImage BlockLantern;
    public RawImage BlockPickaxe;
    public GameObject Character1;
    public GameObject Character2;

    private int selectedCard = -1;
    private const int defaultPos = 50;

    public void CardClicked(int i)
    {
        if (Main.Instance.SelectCard(i != selectedCard ? i : -1))
        {
            UpdateSelection(i);
        }
    }

    private void UpdateSelection(int i)
    {
        if (selectedCard != -1)
        {
            UpdateYPos(selectedCard, defaultPos);
        }

        if (i == selectedCard)
        {
            selectedCard = -1;
            return;
        }

        if (i != -1)
        {
            UpdateYPos(i, defaultPos + 20);
        }

        selectedCard = i;
    }
```

```
    private void UpdateYPos(int index, int y)
    {
        var pos = HandCards[index].GetComponent<RectTransform>().localPosition;
        HandCards[index].GetComponent<RectTransform>().localPosition = new Vector3(pos.x, y,
pos.z);
    }

    public void UpdateCards(List<Card> cards, List<Card.ActionType> Blocks, bool isSaboteur)
    {
        for (int i = 0; i < HandCards.Count; i++)
        {
            SetTexture(i, i < cards.Count ? cards[i].Texture : null);
            UpdateYPos(i, defaultPos);
            selectedCard = -1;
        }

        BlockCart.gameObject.SetActive(false);
        BlockPickaxe.gameObject.SetActive(false);
        BlockLantern.gameObject.SetActive(false);
        Blocks.ForEach(x =>
        {
            if (x == Card.ActionType.Cart)
            {
                BlockCart.gameObject.SetActive(true);
            }

            if (x == Card.ActionType.Pickaxe)
            {
                BlockPickaxe.gameObject.SetActive(true);
            }

            if (x == Card.ActionType.Lantern)
            {
                BlockLantern.gameObject.SetActive(true);
            }
        });

        if (isSaboteur)
        {
            Character1.SetActive(false);
            Character2.SetActive(true);
        }
        else
        {
            Character1.SetActive(true);
            Character2.SetActive(false);
        }
    }

    private void SetTexture(int handIndex, Texture texture)
    {
        SetTexture(HandCards[handIndex], texture);
    }

    private void SetTexture(RawImage image, Texture texture)
    {
        if (!image.IsActive())
        {
            image.gameObject.SetActive(true);
        }
        if (texture == null)
        {
            image.gameObject.SetActive(false);
        }
        image.texture = texture;
```

```
    }
}
```

Card.cs

```
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(fileName = "Card", menuName = "Card", order = 1)]
public class Card : ScriptableObject
{
    public Texture Texture;
    public GameObject Prefab;
    public List<Directions> connections = new List<Directions>(4);
    public bool IsAction = false;
    public bool IsBlock = false;
    public bool IsConnected = false;
    public List<ActionType> actionTypes = new List<ActionType>();

    public enum Directions
    {
        Left,
        Right,
        Up,
        Down
    }

    public enum ActionType
    {
        Pickaxe,
        Cart,
        Lantern,
        Landslide,
        Map
    }
}
```

CardPlaceholder.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class CardPlaceholder : MonoBehaviour
{
    public bool IsFinnish;
    public bool IsStart;
    public int gold;
    public GameObject cardObject;
    private Material material;
    public Card card;
    private Main main;
    private bool clicked;

    private CardPlaceholder Left { get; set; }
    private CardPlaceholder Right { get; set; }
    private CardPlaceholder Up { get; set; }
    private CardPlaceholder Down { get; set; }

    void Start()
    {
        material = GetComponent<MeshRenderer>().material;
```

```
        main = Main.Instance;
        FindNeighbours();
    }

    private bool hasNeighbours => Left != null && Left.card != null ||
                                  Right != null && Right.card != null ||
                                  Up != null && Up.card != null ||
                                  Down != null && Down.card != null;
    private bool ValidateTunnelIfSelected => card == null && main.IsTunnelCardSelected &&
hasNeighbours && CanTunnelBePlaced(main.SelectedCard);

    private CardPlaceholder NeighbourFinnish
    {
        get
        {
            if (Left != null && Left.IsFinnish) return Left;
            if (Right != null && Right.IsFinnish) return Right;
            if (Up != null && Up.IsFinnish) return Up;
            if (Down != null && Down.IsFinnish) return Down;
            return null;
        }
    }

    private void OnMouseEnter()
    {
        if (IsStart)
        {
            return;
        }
        if (main.CanInteractWithCard(card, IsFinnish, ValidateTunnelIfSelected))
        {
            material.color = Color.green;
        }
    }

    private void OnMouseExit()
    {
        material.color = Color.clear;
        clicked = false;
    }

    private void OnMouseOver()
    {
        if (IsStart)
        {
            return;
        }
        if (!clicked && Input.GetMouseButton(0) && main.CanInteractWithCard(card, IsFinnish,
ValidateTunnelIfSelected))
        {
            DoAction(main.SelectedCard);
            material.color = Color.clear;

            clicked = true;
        }
    }

    private void DoAction(Card newCard)
    {
        if (newCard == null)
        {
            return;
        }

        if (!newCard.IsAction && card == null && CanTunnelBePlaced(newCard))
        {
```

```
            var rotation = Quaternion.Euler(-90, 90, 180);
            cardObject = Instantiate(newCard.Prefab, transform.position, rotation);
            card = newCard;

            if (NeighbourFinnish != null)
            {
                NeighbourFinnish.Flip();
            }
        }
        else if (newCard.actionTypes.Contains(Card.ActionType.Map) && IsFinnish)
        {
            cardObject.transform.Rotate(new Vector3(-180, 0, 0));
        }
        else if (newCard.actionTypes.Contains(Card.ActionType.Landslide) && !IsFinnish)
        {
            Destroy(cardObject);
        }
        else
        {
            return;
        }

        main.CardUsed();
    }

    private void FindNeighbours()
    {
        var allCards = GameObject.FindGameObjectsWithTag("CardPlaceholder").ToList();
        var pos = transform.position;
        allCards.ForEach(x =>
        {
            var xPos = x.transform.position;
            if (Math.Abs(pos.x - xPos.x) < 0.5f && Math.Abs(xPos.z - (pos.z + 6)) < 0.5f)
            {
                Down = x.GetComponent<CardPlaceholder>();
            }
            else if (Math.Abs(pos.x - xPos.x) < 0.5f && Math.Abs(xPos.z - (pos.z - 6)) < 0.5f)
            {
                Up = x.GetComponent<CardPlaceholder>();
            }
            else if (Math.Abs(pos.z - xPos.z) < 0.5f && Math.Abs(xPos.x - (pos.x + 9)) < 0.5f)
            {
                Left = x.GetComponent<CardPlaceholder>();
            }
            else if (Math.Abs(pos.z - xPos.z) < 0.5f && Math.Abs(xPos.x - (pos.x - 9)) < 0.5f)
            {
                Right = x.GetComponent<CardPlaceholder>();
            }
        });
    }

    private bool CanTunnelBePlaced(Card tunnel)
    {
        if (Left != null && Left.card != null)
        {
            if (tunnel.connections.Contains(Card.Directions.Left) &&
                !Left.card.connections.Contains(Card.Directions.Right))
            {
                return false;
            }

            if (!tunnel.connections.Contains(Card.Directions.Left) &&
                Left.card.connections.Contains(Card.Directions.Right))
            {
                return false;
            }
```

```
        }

        if (Right != null && Right.card != null)
        {
            if (tunnel.connections.Contains(Card.Directions.Right) &&
                !Right.card.connections.Contains(Card.Directions.Left))
            {
                return false;
            }

            if (!tunnel.connections.Contains(Card.Directions.Right) &&
                Right.card.connections.Contains(Card.Directions.Left))
            {
                return false;
            }
        }

        if (Up != null && Up.card != null)
        {
            if (tunnel.connections.Contains(Card.Directions.Up) &&
                !Up.card.connections.Contains(Card.Directions.Down))
            {
                return false;
            }

            if (!tunnel.connections.Contains(Card.Directions.Up) &&
                Up.card.connections.Contains(Card.Directions.Down))
            {
                return false;
            }
        }

        if (Down != null && Down.card != null)
        {
            if (tunnel.connections.Contains(Card.Directions.Down) &&
                !Down.card.connections.Contains(Card.Directions.Up))
            {
                return false;
            }

            if (!tunnel.connections.Contains(Card.Directions.Down) &&
                Down.card.connections.Contains(Card.Directions.Up))
            {
                return false;
            }
        }

        return true;
    }

    private void Flip()
    {
        if (!IsFinnish)
        {
            return;
        }

        cardObject.transform.Rotate(new Vector3(-180, 0, 0));
        if (gold > 0)
        {
            main.GameOver(gold);
        }
    }

    public List<CardPlaceholder> GetNeighbours()
    {
```

```
        var list = new List<CardPlaceholder>();
        if (Left != null && Left.card != null) list.Add(Left);
        if (Right != null && Right.card != null) list.Add(Right);
        if (Up != null && Up.card != null) list.Add(Up);
        if (Down != null && Down.card != null) list.Add(Down);
        return list;
    }
}
```

MainMenu.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenu : MonoBehaviour
{
    public Text highscoreText;

    public void Play()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void Exit()
    {
        Debug.Log("Exiting");
        Application.Quit();
    }

    public void ToMainMenu()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
    }

    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void HighScore()
    {
        var score = PlayerPrefs.GetFloat("highscore", 0f);
        highscoreText.text = score.ToString();
    }
}
```

OptionsMenu.cs

```
using UnityEngine;
using UnityEngine.Audio;

public class OptionsMenu : MonoBehaviour
{
    public AudioMixer audioMixer;

    public void SetEffects(float volume)
    {
        audioMixer.SetFloat("Effects", volume);
    }

    public void SetMusic(float volume)
    {
```

```
        audioMixer.SetFloat("Music", volume);
    }
}
```

Player.cs

```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class Player : MonoBehaviour
{
    public int Id { get; set; }
    public string Name;

    public bool IsSaboteur;

    public Light Light;
    public List<Card> Hand { get; set; } = new List<Card>();
    public List<Card.ActionType> Blocks { get; set; } = new List<Card.ActionType>();

    private Main main;
    private Material material;
    private bool clicked;

    private void Start()
    {
        main = Main.Instance;
        material = GetComponent<MeshRenderer>().material;
    }

    public void AddCard(Card card)
    {
        Hand.Add(card);
    }

    public void DiscardCard(int i)
    {
        Hand.RemoveAt(i);
    }

    public void DropHand()
    {
        Hand = new List<Card>();
    }

    private void OnMouseEnter()
    {
        if (main.CanInteractWithPlayer(Id))
        {
            var color = Color.green;
            color.a = 0.3f;
            material.color = color;
        }
    }

    private void OnMouseExit()
    {
        material.color = Color.clear;
        clicked = false;
    }

    private void OnMouseOver()
    {
        if (!clicked && Input.GetMouseButton(0) && main.CanInteractWithPlayer(Id))
```

```
        {
            DoAction(main.SelectedCard);
            material.color = Color.clear;

            clicked = true;
        }
    }

    private void DoAction(Card card)
    {
        if (card.IsBlock)
        {
            card.actionTypes.ForEach(x =>
            {
                if (!Blocks.Contains(x))
                {
                    Blocks.Add(x);
                }
            });
        }
        else
        {
            Blocks.Remove(card.actionTypes.First());
        }
        main.CardUsed();
    }
}
```

SoundList.cs

```
using UnityEngine;

public class SoundList : MonoBehaviour
{
    public AudioClip[] sounds;
    public float pauseBetweenClips = 2f;
    AudioSource audioSource;
    int index = 0;
    float timer = 0f;
    bool timerRunning = false;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    void Update()
    {
        if (!audioSource.isPlaying && !timerRunning)
        {
            audioSource.Stop();
            timerRunning = true;
            timer = 0f;
        }

        if (timerRunning)
        {
            timer += Time.deltaTime;
        }

        if (timer >= pauseBetweenClips)
        {
            audioSource.clip = sounds[index++];
            audioSource.Play();
```

```
            timerRunning = false;
            timer = 0f;
            if (index >= sounds.Length)
            {
                index = 0;
            }
        }
    }
}
```

SoundLoop.cs

```
using System.Collections;
using UnityEngine;

public class SoundLoop : MonoBehaviour
{
    public AudioClip[] sounds;
    public float timeBetweenSounds = 10.0f;
    AudioSource audioSource;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
        StartCoroutine(Loop());
    }

    IEnumerator Loop() {
        while (true) {
            audioSource.clip = sounds[Random.Range(0, sounds.Length)];
            audioSource.Play();
            yield return new WaitForSeconds(timeBetweenSounds);
        }
    }
}
```