SpheresTD Game Description

Submitted as part of the requirements for:
CE818 High-Level Games Development

Name: Martynas Petuska (1602657)

Supervisor: Dr Diego Perez

Date: 16 December 2017

Abstract. The extensive description of the game and its implemented features.

Table of Contents

1 Introduction

SpheresTD is a classic tower defence game revolving around the player's strategy of where and when build / upgrade a specific structure to defeat incoming enemy waves. The player income for the further construction is measured in gold which can be gained by defeating enemies or selling previously build turrets. Then the player can spend it on the new turret construction or upgrades. This document will discuss the game's features and their implementation.

2 Menus

The game starts with the Main Menu screen featuring many options. The "New Game" button transitions the menu to the game mode selection screen, while "Load Last Saved Game" button can be used to load last saved game if it does exist. If there is no saved game fount, the player will see the error pop-up asking to start a new game instead. Clicking the "Exit" button quits the game. Under the options tab, the player can set the game's SFX and background music volume using the sliders, or simply mute them by clicking the respective box.

When the "Start New Game" button is clicked the following screen asks the player to select the game mode from "Classic" or "Survival". If one selects "Classic" the game difficulty must be chosen from the following screen before starting the game.

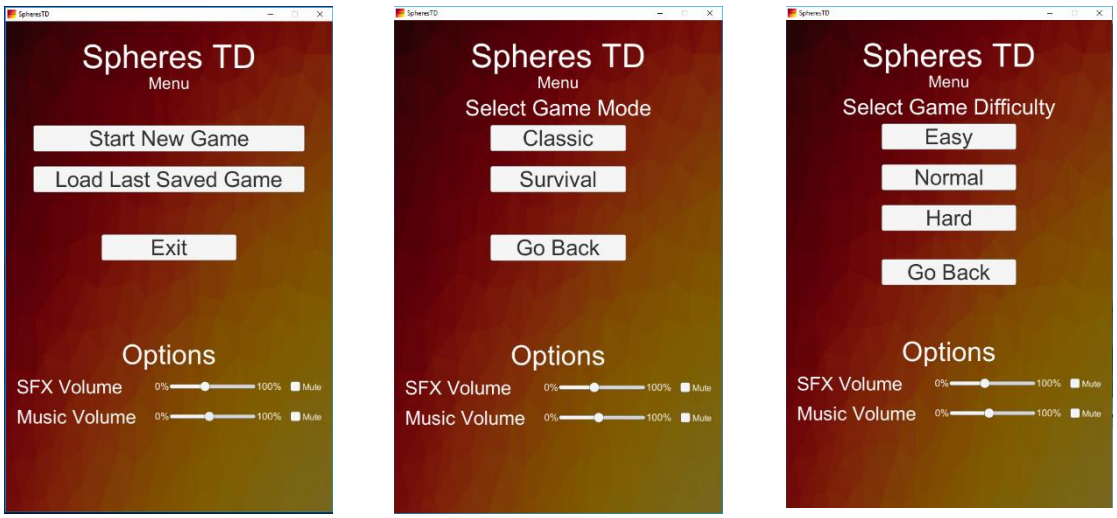

Figure 1Main Menu Screens

Entering the menu starts with the fade in animation and then once the game mode and, if needed, the difficulty is selected, or the saved game is loaded, the menu transitions into the respective level by fading out and then back in once the level is loaded utilising “Fader” script. In-Game the player can also access the pause menu by either clicking” Escape” key or the button at the top of the shop menu. The pause menu allows to change the sound settings, to go back to the main menu or save the current progress and quit the game.

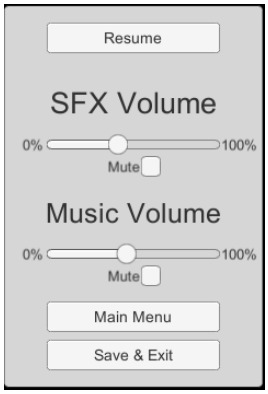

Figure 2Pause Menu

3 Game Modes

The game features two game modes: classic and survival. Even though the mechanics for them are the same, the gameplay elements and the goal itself differs. This section will explore those aspects of each of the game modes.

3.1 Classic

The player’s goal in the classic game mode is to survive and defeat all the incoming waves of each level in the game. Each level has 10 waves and successfully defeating them all transitions the game to the end-level screen where the player can either quit the game or move to the next level. The accumulated gold is not transferred between levels; however, the players Health is. Defeating the last wave of the last level wins the game and the player is transitioned to the victory screen where it can either go back to the Main Menu to start the game in a different game mode / difficulty or Quit the game.
This game mode also has three difficulty settings (Easy, Normal, Hard), each of which scales the enemies hitpoint amount appropriately to the selected difficulty.

3.2 Survival

Survival mode features a slightly different approach to the game. Here, the player is bound to eventually be defeated, however the challenge rises from the unique wave mechanics – each wave has more enemies than the one before as well as the enemies have marginally more health than the ones before. This tests the player with how many waves one can defeat before losing. When the player finally does lose all of his Health, the game transitions into the Game Over screen where the player is shown with the number of waves he’s defeated and then given the options to either Quit the game or go to the Main Menu.

4 Player Shop

To use the game's shop, the player must first select the desired node. Afterwards the menu is updated with the viable options for that node (build new turret, upgrade, sell) with their respective prices. If the player clicks on one of the buttons and he has enough gold, the structure is built and the menu changes to show the upgrade options for that structure. Every structure can be sold for 80% of its initial cost using the same shop menu.

4.1 Structures' Specifications

There are three basic structures, each having separate upgrade tree. The bellow showed specification windows are a tooltip pop-ups from the in-game shop menu. Each of them appear and begin following the mouse cursor each time the cursor is moved over the respective structure's button in the shop menu.

Turret

Basic long range turret with medium damage and fire rate. Upgrades into either Advanced Turret (even longer range, higher damage and fire rate) or Rocket Launcher (slow, long-range weapon firing explosion missiles which deal medium area damage).

Figure 3Turret build tree

Gunner

Basic short range weapon with high fire rate and medium damage. Upgrades to either Advanced Gunner (low-medium range, higher damage and superior fire rate) or Blaster (relatively quick, short range weapon firing explosion missiles which deal high area damage).

Figure 4Gunner build tree

Freezer

Utility weapon to slow down enemies. Does no damage, but instead emits a laser beam slowing the enemy. Upgrades to Insta Killer (slow, short range weapon machine with insane damage).

Figure 5

5 Saving and Loading

Saving and loading of the game is done mostly via the “GameController” script’s functions and coroutines. The data is stored as a PlayerData class object holding all the required information for loading the game utilising only the most basic data types.

5.1 Saving

All the saving within the game is done utilising GameController.Save() method or other methods referencing to it. The method works in a following way:
Firstly, all the game nodes are found and their respective structure type is stored in the string[] nodeInfo variable (nodeInfo[i] = null if the i'th node has no structure build on it). Afterwards, the PlayerData playerData variable is initialised. This variable is the filled with the game's current level and wave indexes, as well as player's Health and gold values. Furthermore, the nodeInfo array is written into the playerData object too, at which stage the object is ready to be written to the file. To prevent manual data modification, the playerData object is serialised using BinaryFormatter bFormatter and written into the file at Application.persistentDataPath.

5.2 Loading

All the saving within the game is done utilising GameController.Load() method or other methods referencing to it. This method has an optional parameter for the desired level index, otherwise it'll attempt to load from the save file. The method works in a following way:
To begin with, it is important to note that the method utilises the coroutine Load-Level() which has two different forms: one where PlayerData data is passed as a parameter and another where int levelIndex is passed. Since the latter is just a simplified version of the former, only the PlayerData parameter case will be explained. First part of the coroutine stops the game utilising Time.timeScale = 0. Afterwards the FadeOut() animation is called and the coroutine stops until the animation is finished. From there, the difficultyMultiplier is set from the PlayerData object and the level is loaded respectively to the level index written in the file. The coroutine pauses until the level is fully loaded, when it sets the player's gold and health values written to the file to the respective in-game data slots. The level and wave indexes are also loaded in the same way. Finally, the method iterates through all the nodes within the loaded level and builds a turret of the type stored in the PlayerData.nodes[] array respectively. This finalises the loading procedures and the method then calls for the FadeIn() animation. Once it's done, the Time.timeScale is set back to 1.

Appendix

Figure 6"GameController" script

```
using UnityEngine;
using System.Collections;
using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour {
    public static GameController controll;
    public string fileName = "Save_1";
    public GameObject loadErrorTooltip;

    public float sfxVolume = 1f;
    public float musicVolume = 1f;
    public AudioSource backgroundMusic;
    public bool muteSFX = false;
    public bool muteMusic = false;

    public float diffMultiplier = 1.2f;
    public int level = 0;
    public int hp = 10;
    public bool inGame = false;

       void Awake()
    {
        if (controll == null)
        {
            controll = this;
            DontDestroyOnLoad(transform.gameObject);
            backgroundMusic = transform.GetComponent<AudioSource>();
            Screen.SetResolution(600, 900, false);
        }
        else Destroy(transform.gameObject);
    }

    public void Save()
    {
        GameObject nodes = GameObject.Find("Nodes");
        int nodeCount = nodes.transform.childCount;
        string[] nodeInfo = new string[nodeCount];
        for (int i = 0; i < nodeCount; i++)
        {
            GameObject currentTurret =
nodes.transform.GetChild(i).GetComponent<NodeController>().turret;
            if (currentTurret) nodeInfo[i] = currentTur-
ret.GetComponent<TurretController>().type;
        }

        PlayerData playerData = new PlayerData();
        playerData.health = StatMaster.controll.hp;
        playerData.gold = StatMaster.controll.gold;
        playerData.level = StatMaster.controll.level;
        int wave = StatMaster.controll.wave - 1;
        if (wave < 0) wave = 0;
        playerData.wave = wave;
        playerData.nodes = nodeInfo;
```

```
        playerData.difficultyMultiplier = diffMultiplier;

        BinaryFormatter bFormatter = new BinaryFormatter();
        FileStream file = File.Create(Application.persistentDataPath + "/" +
fileName);
        Debug.Log("File saved at: " + Application.persistentDataPath);

        bFormatter.Serialize(file, playerData);
        file.Close();
    }

    public void Load(int levelIndex = 0)
    {
        if (File.Exists(Application.persistentDataPath + "/" + fileName) &&
levelIndex == 0)
        {
            BinaryFormatter bFormatter = new BinaryFormatter();
            FileStream file = File.Open(Application.persistentDataPath + "/" +
fileName, FileMode.Open);
            PlayerData playerData = (PlayerData)bFormatter.Deserialize(file);
            file.Close();

            StartCoroutine(LoadLevel(playerData));
        }
        else if (levelIndex == 0)
        {
            PlayerData tempData = null;
            StartCoroutine(LoadLevel(tempData));
        }
        else StartCoroutine(LoadLevel(levelIndex));
    }


    IEnumerator LoadLevel(PlayerData data = null)
    {
        if (data == null)
        {
            Debug.Log("No save file found!");
            GameObject tooltip = (GameObject)Instantiate(loadErrorTooltip, con-
troll.transform);
            Destroy(tooltip, 2f);
            yield break;
        }

        Time.timeScale = 0;
        Fader.controll.FadeOut(0.1f);
        while (!Fader.controll.stable) yield return null;
        controll.diffMultiplier = data.difficultyMultiplier;
        SceneManager.LoadScene(data.level + 1, LoadSceneMode.Single);
        while (Application.isLoadingLevel) yield return null;
        StatMaster stats = StatMaster.controll;
        stats.hp = data.health;     stats.hpValue.text = stats.hp.ToString();
        stats.gold = data.gold;     stats.goldValue.text =
stats.gold.ToString();
        stats.level = data.level;   if (data.level == 0) stats.levelValue.text
= "SURVIVAL"; else stats.levelValue.text = stats.level.ToString();
        stats.wave = data.wave;     stats.waveValue.text =
stats.wave.ToString();

        for (int i = 0; i < GameObject.Find("Nodes").transform.childCount; i++)
        {
            GameObject node = GameOb-
```

```
ject.Find("Nodes").transform.GetChild(i).gameObject;
            node.GetComponent<NodeController>().BuildByType(data.nodes[i]);
        }
        Fader.controll.FadeIn(0.1f);
        while (!Fader.controll.stable) yield return null;
        Time.timeScale = 1;
        controll.inGame = true;
    }

    IEnumerator LoadLevel(int levelIndex = 1)
    {
        Time.timeScale = 0;
        Fader.controll.FadeOut(0.1f);
        while (!Fader.controll.stable) yield return null;
        SceneManager.LoadScene(levelIndex, LoadSceneMode.Single);
        while (Application.isLoadingLevel) yield return null;
        Debug.Log("Level " + (levelIndex - 1) + " loaded!");
        Fader.controll.FadeIn(0.1f);
        while (!Fader.controll.stable) yield return null;
        Time.timeScale = 1;
        controll.inGame = true;
    }

    public IEnumerator Starter()
    {
        Fader.controll.FadeOut(0.1f);
        while (!Fader.controll.stable) yield return null;
        SceneManager.LoadScene(1);
        while (Application.isLoadingLevel) yield return null;

        Fader.controll.FadeIn(0.1f);
    }
}

[Serializable]
public class PlayerData
{
    public int health, gold, level, wave;
    public string[] nodes; //Stores the turret indexes for each node.
    public float difficultyMultiplier;
}
```

References

Brackeys, 2016. *Initial Game Design Inspiration.* [Online]
Available at: https://www.youtube.com/user/Brackeys

Fields, T. & Cotton, B., 2014. *Mobile & Social Game Design: Monetization Methods and Mechanics.* Second Edition ed. Boca Raton: CRC Press.

HTML Color Codes, 2016. *Icon, Main Splash Screens Background.* [Online]
Available at: http://htmlcolorcodes.com/assets/images/html-color-codes-color-tutorials-hero-00e10b1f.jpg

juskiddink, 2010. *Explosion Sound.* [Online]
Available at: https://www.freesound.org/people/juskiddink/sounds/108640/

Play On Loop, 2016. *Background Music.* [Online]
Available at: http://www.playonloop.com/2016-music-loops/mind-in-motion/

Prodigious Creations, 2015. *Decrepit Dungeon LITE.* [Online]
Available at: https://www.assetstore.unity3d.com/en/#!/content/33936

Quadrante Studio, 2014. *QS Materials - Stone vol.1.* [Online]
Available at: https://www.assetstore.unity3d.com/en/#!/content/26441

Robinhood76, 2014. *Launcher Fire Sound.* [Online]
Available at: https://www.freesound.org/people/Robinhood76/sounds/250154/

urupin, 2013. *Turret Fire Sound.* [Online]
Available at: https://www.freesound.org/people/urupin/sounds/192104/

Vertex Studio, 2013. *Turrets Pack.* [Online]
Available at: https://www.assetstore.unity3d.com/en/#!/content/9872