Survive – documentation

1. About the game

Survive is a FPS game with pretty simple aim. Player must try to survive as long as possible. To do so, it’s necessary to kill encountered enemies, which become stronger and stronger (e.g. have more life) as game goes on. Player’s character stays the same through out the game.

1. Technologies

The only used language is C#. Unity has been chosen as a game engine.

1. Art

Main game scene was built with the usage of (BitGem, 2020). Character models were taken from (mixamo, 2023). Some particle effects come from Unity Asset Store. Sound effects were found at (game, 2023).

1. Project structure

The root directory of the project consists of following sub-directories (the most important one are listed):

1. *Art* – contains models, sound effects, UI elements, main scene elements
2. *Particles* – contains particle effects used in the game
3. *Prefabs* – contains prefabs for player, enemies
4. *Scenes* – contains scenes used in the game
5. *Scripts* – contains all C# scripts used in the game
6. How to play

Use WSAD to move the character in the world. Press Tab to switch between first person and third person view. Press R to reload the weapon. Press/hold left mouse button to shot enemies.

1. Code

*Scripts* directory contains all the code used to create the game. Some scripts are grouped into a separate directory (e.g. for audio, pooling system, spawning system). The name of the each script suggest it’s usage in the game, for example:

* *CameraController.cs* contains all the things related to control of camera (movement, angle, currently used camera)
* *PlayerMovement.cs* contains things related to player movement (reaction to input, changing coordinates of model)
* *UIPlayerHealth.cs* controls the display of player’s current health

1. Software patterns

Many scripts are pretty simple and it’s not worth to describe them in great detail. Instead it is suggested that developer explores each of them and understand how they work. However, it’s worth to mention some patterns/describe some systems which are more sophisticated:

1. *Pooling system* (contained in *Pooling* directory) – the implementation of object-pool pattern (pooling, 2023). In a nutshell, it’s purpose is to make game object reusable. For example, when enemy dies, the game object is not destroyed (consuming operation). Instead, it’s disabled and returns to the pool (a container where reusable game objects wait for their next usage). System is built with the usage of two scripts : *Pool.cs* and *PooledMonoBehaviour.cs*. *Pool.cs* is the generic implementation of mentioned container. It’s responsibility is to manage a queue of game objects which can be reused (grow it when needed, add objects to the queue, remove objects from queue). *PooledMonoBehaviour.cs* is a generic class from which any game object that should be reusable in the game (e.g. particles, sound effects, enemies, shots) should inherit. It contains methods which allow to fetch particular game object from the pool and set it’s coordinates in the world. It also contains methods that allow game object to return to pool or return after certain amount of time has elapsed (for example dead enemy should not disappear immediately from the game scene).
2. *Scriptable audio system* (contained in Audio directory) – the aim of the system is to allow a game designer easily work with audio files. For example, when enemy is hit, it would be pretty boring if sound effect would be exactly the same each time. Instead, there should be some variation – one of many sound effects should be chosen, pitch/volume should vary each time. To achieve this scriptable objects (scriptable, 2023) are used. Furthermore, to make it easier to manage audio files, some custom Unity GUI elements are created with the usage of scripts contained in Editor directory (e.g. *RangedFloatDrawer.cs* creates a GUI for drawer which allows to select an allowed range of values for pitch and volume).
3. Spawning system (contained in *Spawner* directory) – system which enables to spawn particular game objects in chosen places in the game scene. Any class which inherits from *Spawnable* class (*Spawnable.cs* file) can be spawned. It’s worth to mention that *Spawnable* class inherits from *PooledMonoBehaviour*. As a result, each spawned game object is taken from the pool instead of being created from scratch. This improves the performance of the game. Spawner class (*Spawner.cs*) contains editable fields which describe the spawner (spawning rate, initial delay before first spawning, time to wait between each spawning, total number of objects to spawn, number of objects to spawn in each spawning).

It's worth to mention scripts which have the same prefix/suffix are related to each other. For example *Weapon.cs*, *WeaponAmmo.cs*, *WeaponAnimation.cs* are all related to weapons. Similarly, all scripts with UI prefix are related to UI elements (ammunition text, current health image, current count of killed enemies).

# Bibliography

BitGem. (2020, 08 17). *assetStore.Unity*. Retrieved from assetstore.unity.com: https://assetstore.unity.com/packages/3d/environments/historic/western-town-pixel-style-set-47563

game. (2023, 05 20). *opengameart*. Retrieved from opengameart.org: https://opengameart.org/

mixamo. (2023, 20 05). *mixamo*. Retrieved from mixamo.com: https://www.mixamo.com/#/

pooling. (2023, 05 20). *gameprogrammingpatterns*. Retrieved from gameprogrammingpatterns.com: https://gameprogrammingpatterns.com/object-pool.html

scriptable. (2023, 05 20). *https://docs.unity3d.com/Manual/class-ScriptableObject.html*. Retrieved from https://docs.unity3d.com/Manual/class-ScriptableObject.html: https://docs.unity3d.com/Manual/class-ScriptableObject.html