Game Report

姓名	谢东	院系	计算机学院	学号	L180301003
姓名	王晓敏	院系	计算机学院	学号	L180300104

Introduction

As a result of our work, we would like to present you our simple game modification with new textures, game objects.

Unity Engine

Unity is a cross-platform computer game development environment. Unity allows you to create applications that run on different operating systems, including personal computers, game consoles, mobile devices, web applications, and more.Unity was released in 2005 and has been continuously evolving since that time.

The main advantages of Unity are the presence of a visual development environment, cross-platform support and a modular component system. The disadvantages include the appearance of difficulties when working with multicomponent circuits and difficulties when connecting external libraries.

Thousands of games, applications, and simulations are written in Unity, spanning many platforms and genres. At the same time, Unity is used by both large developers and independent studios.

Possibilities

The Unity editor has a simple Drag & Drop interface that is easy to customize, consisting of various windows, so you can debug the game right in the editor. The engine supports two scripting languages: C #, JavaScript. Physics calculations are performed by the PhysX physics engine from NVIDIA.

A project in Unity is divided into scenes (levels) - separate files containing their game worlds with their own set of objects, scenarios, and settings. Scenes can contain both, in fact, objects (models), and empty game objects - objects that do not have a model ("dummy"). Objects, in turn, contain sets of components with which scripts interact. Also, objects have a name (in Unity, two or more objects with the same names are allowed), there may be a tag (label) and a layer on which it should be displayed. So, any object on the scene must have a Transform component - it stores the coordinates of the location, rotation and size of the object along all three axes. Objects with visible geometry also have a Mesh Renderer component by default, which makes the object model visible.

Collisions can be applied to objects (in Unity, the so-called colliders), of which there are several types.

Unity also supports solid and cloth physics, as well as Ragdoll physics. The editor has an object inheritance system; child objects will repeat all changes to the position, rotation and scale of the parent object. Scripts in the editor are attached to objects as separate components.

When importing a texture into Unity, you can generate an alpha-channel, map -levels, normal-map, light-map, a reflection map, but you cannot attach a texture directly to the model - a material will be created to which a shader will be assigned, and then the material will be attached to the model. The Unity Editor supports writing and editing shaders. The Unity editor has a component for creating animations, but animations can also be created in advance in the 3D editor and imported with the model, and then split into files.

Unity 3D supports the Level of Detail system (abbreviated LOD), the essence of which is that at a distance from the player, highly detailed models are replaced with less detailed ones, and vice versa, as well as the Occlusion culling system, the essence of which is that objects, geometry and collisions that do not fall into the field of view of the camera are not visualized, which reduces the load on the central processor and allows you to optimize the project. When a project is compiled, an executable (.exe) game file (for Windows) is created, and the game data (including all game levels and dynamic link libraries) is created in a separate folder.

The engine supports many popular formats. Models, sounds, textures, materials, scripts can be packed in the .unityassets format and transferred to other developers, or laid out for free access. The same format is used in the internal Unity Asset Store, where developers can share various elements that they need when creating games for free and for money. To use the Unity Asset Store, you must have a Unity developer account. Unity has all the components you need to create multiplayer. You can also use a version control method that suits the user. For example, Tortoise SVN or Source Gear.

Unity includes the Unity Asset Server, a Unity-based collaborative development toolkit that adds version control and a range of other server solutions.

Game

We have made several modifications to a basic game, that we would like to present to you:

- Endless game generation
- Random obstacle spawner
- Coin spawner and collector
- Score system
- Level hardness design
- New textures and game objects

- Singleton handlers
- Music
- Double jump

Level generator.
We have started with endless game generation. We have discussed two different types of endless game generation. The first variant was to generate pseudo-infinite number of levels by hand. We had to manually create many prefabs, which will represent the new-level. Each prefab would have a ground and number of obstacles, all added manually.

This variant wasn't good enough by multiple reasons. First, we would have to somehow show levels and keep the track of how many levels the user had passed. Second, generating level prefabs manually will take a lot of time and data storage.

Our second variant was to create a more traditional endless-run level. The player will run forever on one level. In order to create and endless-run, we have created a simple level-section prefab, which contained only ground. We have tracked the EndPoint of each level section and written a C# script, that would Instantiate a new section for an endless number of times, while the player is alive. We have added the empty game object to the game scene, that would handle the section generation.

Random obstacle spawner.
Random spawn obstacle was a bit tricky. Our first obstacle spawner was partly manual. We have created multiple level sections from the level generator and manually placed obstacles to the level sections. Then we have updated level spawner, so it could randomly select the next level section with placed obstacles. This was a working prototype, but it wasn't good. As we have only created five different level sections, the random obstacle spawner wasn't exactly random. It followed five different level patterns. So we have updated it.

We have decided, that it would be optimal to spawn a new obstacle every 1 to 5 seconds. We have decided that for an easy game, obstacle should spawn every 3 seconds. To the level generating script, we have added a function, that would spawn an obstacle. But where is the randomness? We have implemented eight different prefabs of different size and texture of different obstacles. The script now randomly pics an obstacle and places it ahead of the user.

Coin spawner and collector.
We have decided, that we want to add a new game-play feature of collecting coins. Coins have to spawn randomly every 1 or 2 seconds ahead of the player. But the coins should be randomly placed on the level, the height above the ground should be random.

We have added a new script, that would randomly pick a height and spawn a coin every 1.25 of a second. We have created a new empty game object and assigned a script. Coins were spawning, but how to collect them?

Every coin game object was converted into a collision able object and we have added the collision check with the player. When the play gets into a collision with the coin, the algorithm triggers a function, that removes the coin and adds a record, that the coin was collected.

Score system.
Again, we have discussed two variants of score tracking. First was to track the time during the game. The seconds, was to track the score with coins collected. We have decided to track the score with coins.
In order to make the score available from any part of the game, we have created a singleton, which will keep the data about the latest score and the record of the highest ever score. We have faced and issue here. The singleton class has to be assigned to a game object. We have placed an empty game object on a menu scene. If the game was started from the other scene, then the singleton object won't be initialized and score system will fail.

Every time, the player collects the coin, the player controller triggers and updates a score in the singleton. As the game finishes, the score is represented for the player.

Level hardness.
As a part of the new game-play, we have decided to implement three different levels of hardness. We have achieved this with changing the time between two obstacles spawn. The harder level is, less the time between obstacles.

We have added buttons to the main menu, and new functions for these buttons. The buttons record the game mode and trigger the obstacle generator.

When the game is over, retry button starts the level with the same game mode.

New textures.
We have added different types of textures and prefabs to the game. Most of them are connected with the game itself: obstacles, coins, etc.

Music.
We have added music and an icon to switch between silent mode and music play mode.

Double jump.

We have added a double jump for better gameplay.

The difficulties we have encountered.

When switching between scenes, the music was switched off.

We encountered difficulties in adding animations. The script didn't work as it should, and the player flew at the top instead of running.

Contribution

王晓敏 was in response of graphics work of textures, prefabs, music and level design.

谢东was in response of level generation and object spawning.