Linnéuniversitetet
Kalmar Växjö

Technical Details

1DV437 Introduction to Game Development

Autor: Krasen Anatoliev Parvanov
Email: kp222pt@student.lnu.se

Game Summary

The game's name is Soldiers VS Robots and it is a 2D game created with Unity 2019.4 engine.
The player can adjust the sound settings, resolution, choose a full-screen mode, and view the top 10 of the highest scores. In the game, the player controls a soldier in fulfilling one of the two different missions. The player steers the soldier in the mission scenario where the player needs to find and pick-up a crystal guarded by robots as fast as possible and within the game time limit. The robots are spread around the game world in different positions and try to stop the player by shooting lasers which can kill the soldier. The soldier can shoot bullets at the robots and destroy them for shooting the robot the player gets points. The crystal is protected by the “boss” robot. The game is over when the crystal is retrieved, time is over or when the player is killed by the robots.
A complete list of features can be seen in Game Description Assignment 1 (the prototype is no longer an accurate representation of the game world). Additional features were added allowing the player to pause and resume the game while playing as well as in-game zoom in/out camera features.

Game Design and Implementation

Architecture

The game was designed using a Layered architectural approach while still taking into account the reusability of game objects and breaking each layer into a modulus and objects/entities. The user interfaces view controller manages the camera and updates the UI elements while a grid was used to design the game world and game entity within the game world(L1). To increase reusability and maintenance prefabs game objects are used for the game entities with assigned tags to be able to identify different groups of objects and their functionalities are controlled and updated by scripts(L2). Scriptable objects, data file manager, game, and player state manager scripts were used in the lower levels(L3).

Persistent storage was used with unity application persistentDataPath to make it work on a multi-platform. The data is saved in JSON format since it is not sensitive information(L4).

Design Patterns and Data Structures

The game was designed mostly by the Ad-hoc approach by taking into account the most common object-oriented design paradigms and patterns such as inheritance and singleton. The enemy objects are designed using inheritance which increases reusability and allows for easily introducing more enemy types in the future. For the game and player, the singleton pattern and scriptable objects were used in order to maintain consistency throughout the gameplay. The game state manages the mission time and difficulty selected by the player and the player state manages the name, health, points, and additional player information. That will allow for more easily adding extra functionality to the scriptable objects and states for both the player and game in the future. In addition to that, the singleton pattern was used for the menu audio in order to achieve consistency when changing the menu senses.
The game uses a simple List of players' structures to store the player objects. Since the game is not that advanced and the only data that is saved and quired is the players it is not needed to use a more advanced data structure.

Collusion and Geometry

The collusion between the objects in the game world is handled by using Unity 2D colliders. By using Unity in-build physics rigid body, collider, triggers, and tag(for identifying types colliding objects in the game world) the game can handle the movement, rotation, and collusion of different objects in the game. Basic movement is done by using Unity physics move position on the rigid body, and rotation and direction of objects(laser, bullets) by changing velocity and rotation. The player soldier has two 2D colliders, a box 2D collider that handles the collusion with the laser, and a circle 2D collider that acts as a trigger for the game world items and allows the player to pick up items once they are in close proximity. Similarly, the enemy box 2D collider handles the enemy collusion with the soldier bullets. The laser and bullets have their one ellipse and circle 2D colliders which manage the action and animation and sound played depending on the object they collide with. And the items(cristal, weapon, and health) have their circle and polygon 2D colliders to manage interaction with them once they are close to the player. In addition to that in the game, the world was added a Tilemap 2D collider as an additional movement constraint for the player as well as helping the player to follow the path/road of the mission and not "wander" around in the game world.

Textures

There were no special textures, lighting, and/or shader created specifically for the game. A custom slide material was created and used to manage the bouncing and interaction/collusion between the player rigidbody and the rest of the game world. The sprites used for designing the game world were from external sources. However, some of the sprites were changed using Paint so that they can be adapted for the gameplay. That included resizing, cutting, turning(mirror and rotating) the sprite specialty for the

enemy and player characters. The rotation and mirror were needed in order to have the necessary sprites for animation. The complete list of the assets used can be seen at the end of this document.

Animation

The main animation effects were done by using the animator and the sprite that were available. Most of the animations are simple, for example, the enemy death effects and hit effects are treated as game objects in the script and the animations that are played on collusion. The pick-up items in the game world are simply animated so that the player can spot them more easily, their animation is a one-state animation that loops all the time. The enemy movement animation is done by using a simple state machine with boolean transition parameters and X, Y values for a simple 2D directional animation for the different animations/states. The player animation is achieved by a state machine that is a bit more advanced than the enemy, that is it has more states like shooting, dead, idle, and movement that is mainly because the soldier sprite set included the necessary sprites to achieve that effect. Nevertheless, the logical transition between the different states is done in a similar manner as the enemy by using boolean parameters and 2D simple directional.

Discussion and Reflection

Going over the final game design and delivered features and the initially planned ones, it can be said that there are no major deviations. Some features were added(such as camera zoom in/out and pausing of the game) but that did not affect the codebase in any significant way. Another logical addition is the enemy boundary restriction, where the enemy robots can move, chase and follow the player only within their corresponding section of the map, and when the player is out of range the enemy is teleported back to their starting position. That didn't affect the code significantly and was done for gameplay purposes. Since Unity had very detailed documentation and a community forum, it was not difficult to resolve the basic coding tasks. Advance coding refactoring was not considered since this is the first time for me using C#, but having a Java background helped me to implement basic encapsulation, inheritance paradigms, and singleton patterns. Most issues regarding the implementation of the game were design related. Since I do not have any design background it was difficult to find sprites that I can use and adjust them to the pixel ratio in a proper way. This can be seen in the game where the quality of the characters and animation is sometimes inconsistent. That was also the reason to decrease the in-game world building significantly(the game does not have houses, cars, and more detailed elements). Another issue, in the beginning, was the basic logic and correlation between the Unity engine and the scripts but with help of unity documentation, forum, and practical lectures that were cleared out.
If the project is to be continued in the future a minor restructuring of the game may be needed in regards to audio control. I would like to introduce an audio controller for corresponding entities that will help with managing the sound effect especially if they are more characters introduced in the future. In addition to that, expanding the game world and improving the in-game design quality will be something that I would consider. Adding additional missions will also be quite easy since the soldier, all the enemies and items exist as prefabs and can be reused.

Appendix

Complete Asset List

Original asset list:
Heroe - https://opengameart.org/content/2d-soldier-guy-character
Crystals - https://opengameart.org/content/basic-gems-icon-set
Map Tiles Sprites - https://opengameart.org/content/lpc-tile-atlas2 - https://opengameart.org/content/lpc-tile-atlas
Enemy - https://opengameart.org/content/platformer-sprites
Items - https://opengameart.org/content/various-inventory-24-pixel-icon-set - https://opengameart.org/content/ascension-2-galaxy-icons
Epic Adventure Orchestral Background Music (Free Sample) by Marma - https://assetstore.unity.com/packages/audio/music/orchestral/epic-adventure-orchestral-background-music-free-sample-23837

Newly added:
Explosion effects - https://opengameart.org/content/explosion
Free sound effect pack by Olivier Girardot - https://assetstore.unity.com/packages/audio/sound-fx/free-sound-effects-pack-155776
FREE Casual Game SFX Pack by Dustyroom - https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116

Removed:
Bullet - https://opengameart.org/content/bullet-collection-1-m484