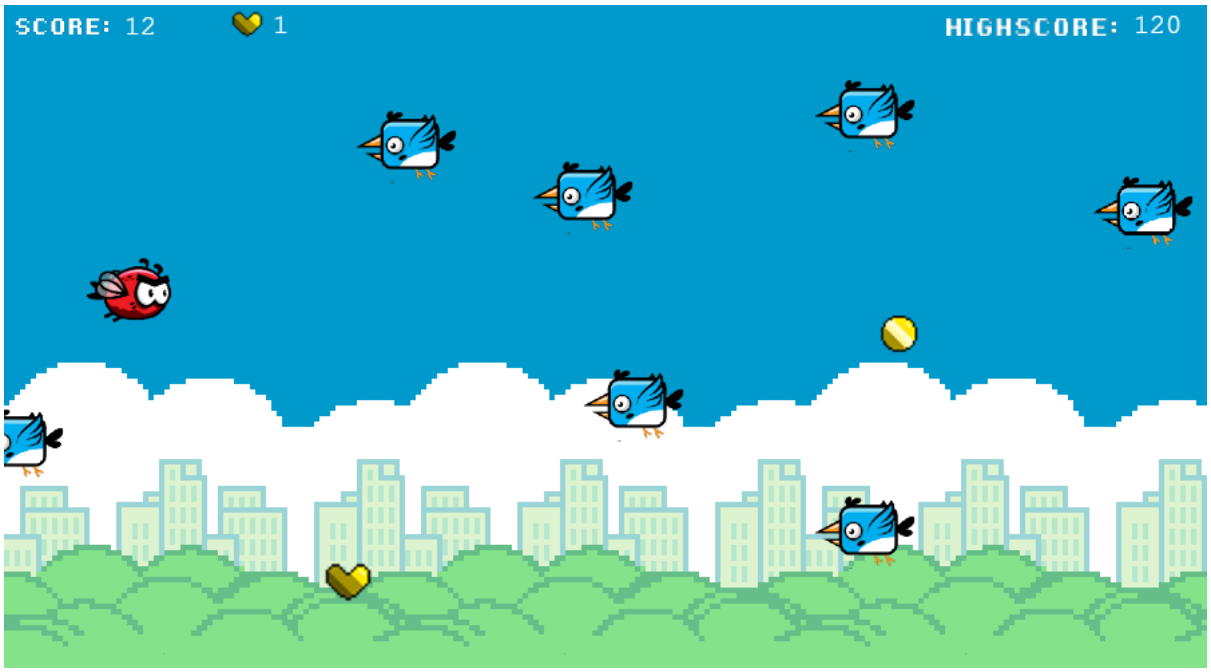
FlappyBug

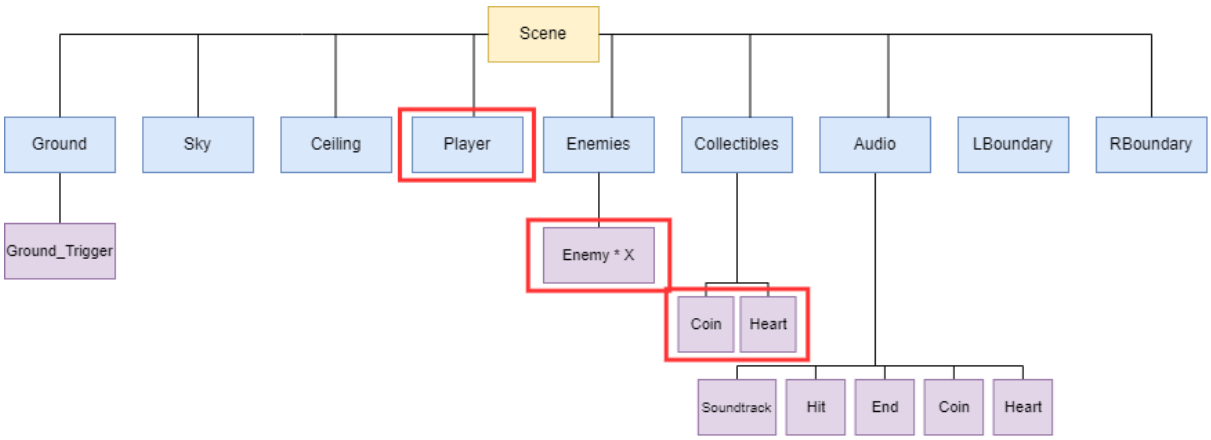
Nathaniel Keeler | 265080 | MIB 7 | SS22 | PRIMA



1. Units and Positions

The 0 position is at the center of the playing field. There weren't really any changes needed for the origin since that would have just added more work for repositioning when working in the visual editor. The 1 is the measurement in which the playing field is measured. The field has a dimension of 4x2. Therefore, you have an area of -1 to 1 on the Y-Axis, where enemies and items can spawn. Because most of the calculations were done concerning the X-Axis, you could work well with these Units. Player, Enemies, and Items are scaled down to fit the playing field. The Units and Positions could have been done better. In the beginning the playing field and the player were quickly added to get a running prototype, which is why later changes to this aspect would have brought a lot of work with it and thus weren't changed.

2. Hierarchy of game elements



Though it might not look like it, most of the game was created in Code (shown in red), except for the basic building blocks like the Ground and Sky, as well as physical barriers which were created in the visual editor. Enemies- and Collectibles are container nodes for when sub nodes are spawned by code and get added as a child. The Audio node holds multiple audio files which are then played whenever they are called upon. For the flying sound of the player, the audio was loaded by code. Ground and Sky nodes create the ground and sky in the scene. Since the Ground acts as a barrier for the player through its rigid body, there was another Child node added with the same position and slightly larger dimensions, which just acts as a physical collision trigger. The remaining nodes (Ceiling, LBoundary, RBoundary) are for creating physical barriers and for hiding objects that pass behind it.

3. Editor – What was done where

Player, Enemy, Coin and Heart are created in Code since they all have different properties, some of which are shared. Also, things like sprite animations are found to be similar and can quickly be applied to all these classes in code, while tweaking some parameters. Working in the code editor turned out to be a faster process as to working in the FUDGE visual editor.

4. Script components

There are four different Script components. LinearMovementScript, SineMovementScript, CoinMovementScript and HeartMovementScript are used for Enemies and Collectibles and define different movement patterns (as their name suggests), which can be randomly attached to different nodes. Apart from that, they also reposition enemies and collectibles which have passed the left border of the game field, to random positions to the right of the game field. This way there is always a fixed number of enemies and collectibles in the game, and they don't have to constantly be constructed and deconstructed.

5. Extends – Derivations from classes

The Player, Enemy, Coin and Heart classes extend FUDGE's ƒ.Node class while the script components LinearMovementScript, SineMovementScript, CoinMovementScript and HeartMovementScript extend the ƒ.ComponentScript class.

6. Sound

Sounds are used for background music, flying, hits against the ground or enemies, collectible pickups, and when the game is over. The flying sound is played while pressing the spacebar, while the other sounds are played on collision with the respective objects.

7. Visual User Interface

The VUI displays the current score, the number of lives and the high score and is controlled by the GameState class.

8. Event-System

The Event-System is used for collision handling. When a collision occurs between the player and objects that have their physical properties set to “is_trigger”, a physics event is called which handles how to react to the colliding objects appropriately. An example could be the collision with an enemy or with the ground, which would reduce the number of hearts the player has, while collecting a coin would add 25 points to the current score.

9. External data – Changing game parameters

The “data.json” file is used for changing different game parameters such as the game speed, the starting high score you compete against and how many enemies are going to be spawned. Once you surpass the high score from the json file, your high score gets saved in the local storage and gets displayed in the game from then on.

B. Physics

Almost all nodes have rigid body components. While some just have these for easier collision detection with the player and are not affected by the actual physical world (gravity etc.), the player uses physics for the controls. When the spacebar is held, a constant upward force is applied to the player, which creates a more realistic feeling for the flying-controls.

C. Animation

The Player, Enemy and Coin objects all have animations using sprite sheets.