Blake Freer
June 19, 2020
ICS4U Final Project - Overview

My final submission is not a typical computer science project. I created a 2D Game Engine, structurally based off of Unity. There is no user interface to use the engine, so the game is created entirely through typed code. When you run the project, a sample game will run, one that I created using the game engine. My final project is not the game, but rather the game engine that I made to run it. Since there is not much to see on the surface, I would like to explain the functionality of some of the most important classes. All classes are documented as well.

- MainPackage
    - **Main.java**: Creates the JFrame for the application
    - **ContentScreen.java**: Extends JPanel, handles the game loop (start, update, draw), and takes raw input.
    - **GameData.java**: Holds most information about the game, such as screen dimensions, frame rate. It handles rendering and some smaller tasks.
    - **InputManager.java**: Takes raw input events from ContentScreen and converts them into useable data forms, such as “GetKeyDown” returning a boolean if a key is pressed, or “GetAxis2D”, which converts WASD keys into a Vector2 output, making it much easier to use input for character controls.
- Engine
    - GameMath
        - **Vector2.java**: One of the most used classes. It represents a 2-dimensional vector with x and y components, and provides many functions for doing math on vectors.
        - **Mathf.java**: Provides some additional mathematical functions, like Clamp, which modifies a value to ensure it stays within a range. Other functions are Sign, which returns the sign of a value, or SmoothDamp, which gradually makes a value approach another (code copied from Unity, but translated from C#).
    - Physics2D
        - **Physics.java**: Performs Raycasting, which uses the Liang-Barsky algorithm for line clipping. It sends out a ray and checks for colliders, and returns information about the collision, if there is one. This is used extensively in the player controller scripts.
- GameObjects
    - **GameObject.java**: A container for all objects in the game. It has properties for the object’s name, tag, position in the hierarchy, and many others. Provides methods for adding and getting components to the GameObject, finding objects in the hierarchy tree, etc.
    - **Scene.java**: A subclass of GameObject, acting as the root of the hierarchy and holds the colliders in the world.

        - **Camera.java**: Subclass of GameObject, represents the viewing point of the game, and converts world-space points into screen pixel positions, making it easy to move and resize the perspective of the game.
        - Components
            - **Transform.java**: Holds the position of a GameObject.
            - **Component.java**: An abstract class for adding behaviours to GameObjects.
            - **Colliders**: Represent geometry of an object for collisions.
            - **Renderers**: Gives a GameObject a visual component, for drawing circles, rectangles, or Sprites (SpriteRenderer is most used)
            - Under **src/Assets/CustomScripts**, there are several other components that provide specific functionality to objects in the scene, but are not general enough to be included in the game engine itself.
- Editor
    - **TileMapCreator.java**: Takes in a .tileset file to convert a tileset sprite sheet into several tile GameObjects that can be used to create a world. It then uses a .map file to use the tiles to create the world. You can see the .tileset and .map files under Assets/TilemapData

Assets: Assets is where images, animation data, world data, and custom scripts are stored. Anything specific to the game being made is stored here, while everything general for the game engine is stored elsewhere. Under Assets/Animations, there are several .anim files that store paths to images and frame durations, so entire animations can be stored together in a file, which an Animator can read and process.

I created all of the artwork and animations for the game, using Aseprite.

Unused: Several scripts that I made, but decided were not necessary for my project.

In total, there are over 3700 lines of code, 2400 for the engine, 600 that are unused, and the reset in assets.

Known Errors:
There are occasionally two errors that will occur:
NullPointerException (look below and you will see AddRenderJob) - I believe this happens when gameData tries to render the scene, but not all objects and their renderer components have been updated. If the error persists, lower the FPS in MainPackage/GameData.java

ConcurrentModificationException:
When GameObjects are destroyed (cannonballs or coins), they must be removed from the scene. This originally caused a ConcurrentModificationException, since the objects were removed from an ArrayList while iterating over the list. To fix this, objects to be destroyed are

stored in gameData.objectsToDestroy, and these are removed after the entire frame has been updated. This mostly fixed the error, but it still occurs rarely, and I do not know why.