

Pixel Hero Android game:

Module Code: CS2PJ20

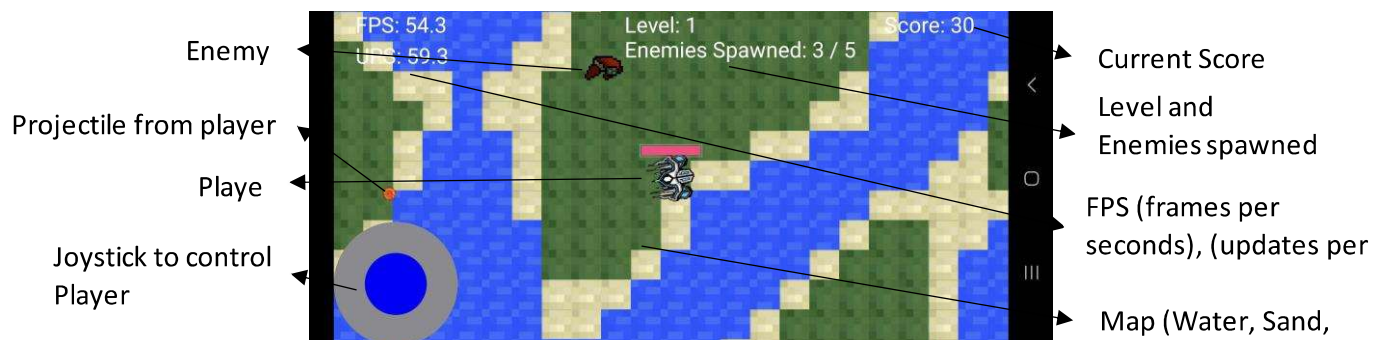
- Assignment report Title: Android Game
- Student Number: 30020810
- Date (when the work is completed): 23/03/2023
- Actual hrs spent on the assignment: 32
- Assignment evaluation (3 key points):
 - I learnt threading which will be very applicable to future projects
 - I developed my OOP design techniques further
 - I learnt how to build a java game in android studio that will help if I ever need to build an app or another game in the future

Abstract:

PixelHero is a survival 2d top-down shooter in a procedurally generated terrain where the water tiles slow the mech (player). The levels become progressively more challenging regarding spawn time, damage, and speed. When the player dies, their game overview stats are displayed on the screen.

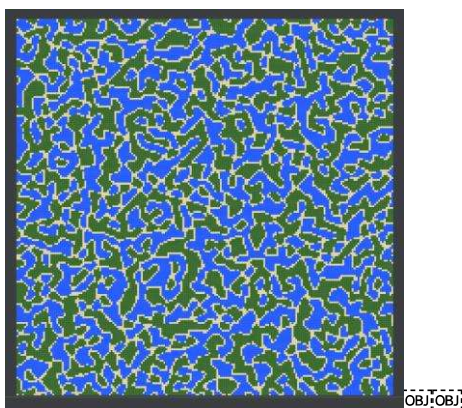
Section 1 - Introduction and Showcase:

This view is what the player sees when they start the game:



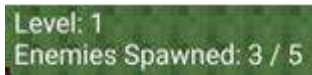
Map:

The map is procedurally generated before the player starts using OpenSimplex2 (to save computational power whilst playing) so that each instance of the game is a new map and, therefore, a new challenge as the terrain of the map directly affects the player's speed. Water tiles offer a 40% reduction in speed, and sand tiles provide a 20% reduction in speed. The player cannot move off the map's borders to avoid this mechanic.



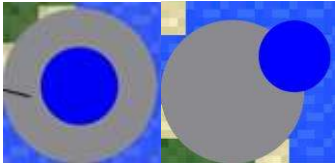
Levels:

The game has a levelling system such that every map is different, and every wave is a new level of difficulty in terms of speed, spawn rate, wave size and damage. The player must then survive these levels for as long as possible hence the survival aspect of the game. The level/enemy is shown to the player at the top of the screen.

A green rectangular UI element with white text. The first line reads "Level: 1" and the second line reads "Enemies Spawnd: 3 / 5".

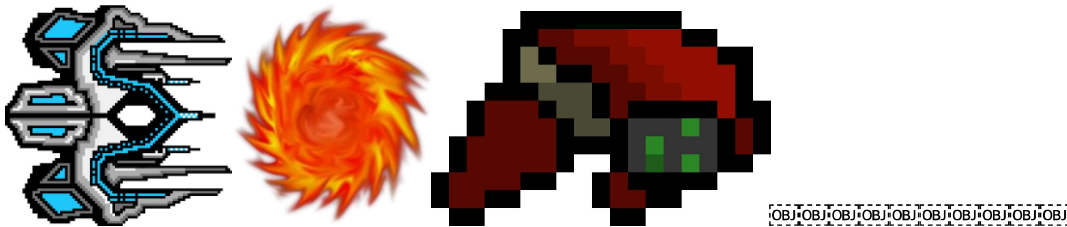
Controls:

A joystick is used that controls the rotation/speed of the player depending on the x, and y position of the blue dot, which recentres if the player lets go



Characters:

The player is a mech (left) that shoots a fireball (middle) at the enemy (right)



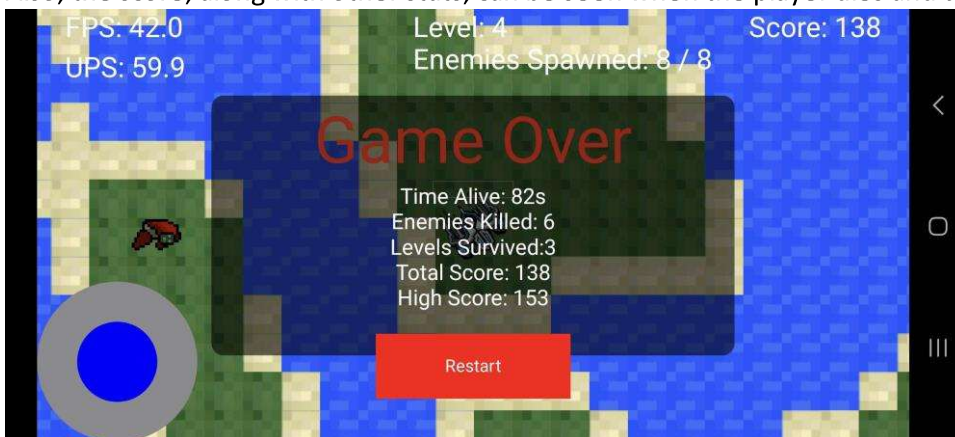
Score (points):

This is shown to the player during the game, where one point is gained for every second survived, as well as 10 from each enemy killed.

It can be seen in the top right of the screen:



Also, the score, along with other stats, can be seen when the player dies and the game ends:

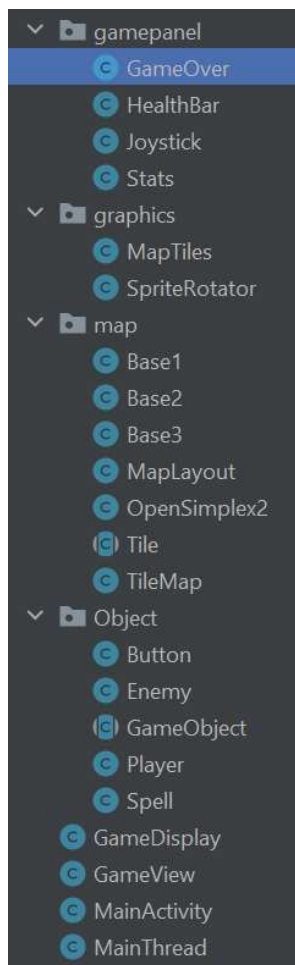
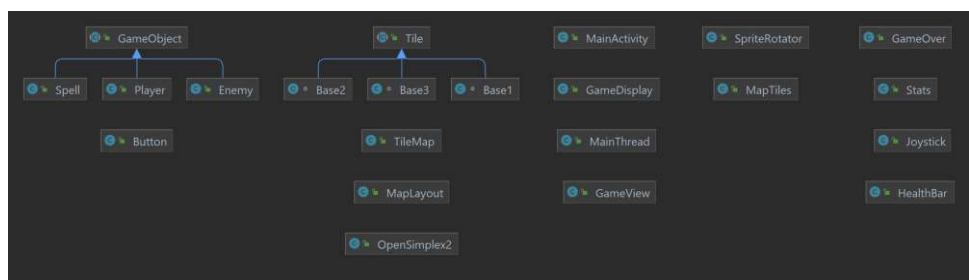


Section 2 – OOP Design:

As you can see below, I used 2 superclasses, Tile and GameObject. Tile manages the tile on the map, containing the three bases corresponding to Grass, Water, and Sand. GameObject includes all the moving entities in the game: Spell, Player, and Enemy. I used folder structure to arrange each of the classes into related folders.

This OOP design made it very easy to customize and add to my game. For example, suppose I wanted to add another tile. In that case, I could just extend the tile class to include it, or If I wanted another game object, like a friendly character or another player, I could extend GameObject. GameObject also made it very easy to handle interactions between the different entities as it had methods like `getDistance` and `isColliding`.

All classes were finished, but if I wanted to develop this game further, I would turn the enemy into another superclass so I could add multiple types of enemies.



Section 3 - Memory Usage and Speed improvements:

I put significant thought into this aspect of the game as I would like the user experience to be as seamless as possible.

Memory Usage:

Two objects are spawned into the game the enemy and the spells, so I added a condition that if they are out of bounds (where 1f = 1 screen width/height from the player), they are removed, so excess memory is not used:

Enemy Memory Optimization:

```

enemies.removeIf(enemy -> {
    enemy.update(player);
    boolean isCollidingWithPlayer = enemy.isColliding(enemy, player);

    if (isCollidingWithPlayer) {
        player.takeDamage(damage);
    }

    boolean isOutOfBounds = enemy.isOutOfBounds(gameDisplay, factor: 10f);

    return isCollidingWithPlayer || isOutOfBounds;
});

```

Spell Memory Optimisation:

```

spells.removeIf(spell -> {
    spell.update();
    if (spell.isOutOfBounds(gameDisplay, factor: 1.5f)) {
        return true;
    }
});

```

Speed Improvements:

The most computationally tricky task in my game is map generation, so before the game starts, I pre-generate the map at a lower 8-bit colour which resulted in significant performance uplift:

```

private void initialiseTilemap() {
    int[][] layout = mapLayout.getLayout();
    tilemap = new Tile[MapLayout.NUMBER_OF_ROW_TILES][MapLayout.NUMBER_OF_COLUMN_TILES];
    for (int iRow = 0; iRow < MapLayout.NUMBER_OF_ROW_TILES; iRow++) {
        for (int iColumn = 0; iColumn < MapLayout.NUMBER_OF_COLUMN_TILES; iColumn++) {
            tilemap[iRow][iColumn] = Tile.getTile(layout[iRow][iColumn], mapTiles, getRectByIndex(iRow, iColumn));
        }
    }
    // Create a bitmap to draw the map on in 8 bit color
    Bitmap.Config config = Bitmap.Config.ARGB_8888;
    mapBitmaps = Bitmap.createBitmap(
        width: MapLayout.NUMBER_OF_COLUMN_TILES * MapLayout.TILE_WIDTH_PIXELS,
        height: MapLayout.NUMBER_OF_ROW_TILES * MapLayout.TILE_HEIGHT_PIXELS,
        config
    );
    // Create a canvas to draw the map on
    Canvas mapCanvas = new Canvas(mapBitmaps);

    for (int iRow = 0; iRow < MapLayout.NUMBER_OF_ROW_TILES; iRow++) {
        for (int iColumn = 0; iColumn < MapLayout.NUMBER_OF_COLUMN_TILES; iColumn++) {
            tilemap[iRow][iColumn].draw(mapCanvas);
        }
    }
}

```

Here the tilemap is pre-rendered in 8 bit color

Section 4 - Improvement/Extension:

Improvement 1:

My game being a top-down shooter meant I had to do some extensive research on how to centre the player on the screen whilst everything else moved in the background around it.

This was the source I used to implement it that explained the vector calculations necessary:

[\(80\) Ep. 13 - Center player in the middle of the screen | Android Studio 2D Game Development - YouTube](#)

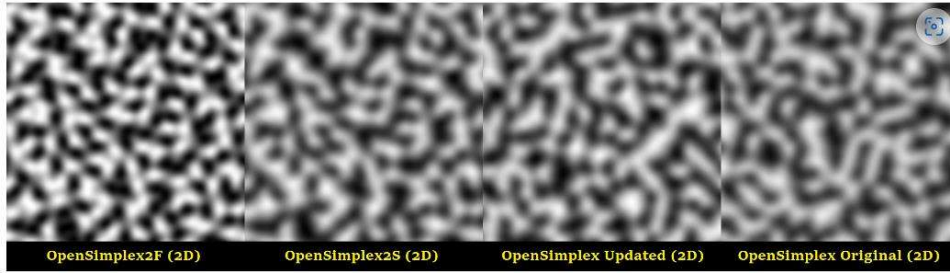
Improvement 2:

Another innovation I used was the map generation class that uses this technology:

<https://github.com/KdotJPG/OpenSimplex2>

That creates the 2d noise to procedurally generate my map:

2D Noise



Section 5 - Conclusion:

I am pleased with the results of my android game. It taught me a lot about thread management, object-orientated programming and the mechanics needed to create a game. If I had more time, I believe I would expand on the enemy class to add more enemies with varying levels of health and speed. The spell class could do with some similar expansion to add more spell types.

GitLab with APK and ZIP:

[Members · James Grayshon / JavaSimpleGame · GitLab \(reading.ac.UK\)](#)