Software Project CA1 – Casual Game

Jake Black – N00193326

Overview

My casual game is a survival-horror platformer game called ‘College Escape’. The aim is to progress along a single floor of a college building fighting monsters to reach the end of the level and escape.

Design

The design phase of my project took significant time, as I decided to design most of the assets and music myself using Adobe Photoshop and Reaper. I had never designed a sprite sheet before, so found a sprite sheet with similar movements to those I’d like to implement for my own character and drew over it in Photoshop. I used the sprite sheet from the original Prince of Persia as my template. Using the mouse to draw, I gradually became a bit more confident and even drew a few poses for the character myself without needing the template. I drew the player’s smoke projectile and the enemy’s slime projectile using Aseprite.

Based on my game design document, I originally intended to have 3 characters, and had created a concept for the fighter character, as well as the cleric. However, after realising how long it had taken me to finish the cleric’s sprite sheet, I decided to stick with only one.

I then used Aseprite to design my monsters. I’m not brilliant at drawing, so decided to make gelatinous, blob-like enemies, with two variants. I used Sprite Illuminator’s bevel and height tools to add the illusion of depth to the enemies.

For the game’s background, I took a small snippet from a screenshot of another survival-horror game and used Photoshop’s sharpen tool for a rougher look. I then used the clone stamp tool to paste it along the entire 1560 pixel width of my background. I again used Sprite Illuminator’s bevel and height tools to add depth to the image, and finally generated a normal map using smart-page.net’s ‘smartnormal’ tool.

For the first enemy and the background, I had access to Sprite Illuminator’s free trial, so I was able to use their normal map packing tool. This allowed me to use Phaser’s Light2D pipeline to turn the background and enemy body into ‘reflective’ surfaces which the light would bounce off. Unfortunately, when the trial ran out, I couldn’t do the same for my second enemy design. I would have had the same problem later anyway, after I couldn’t get my ‘multiatlas’ sprite sheet animation to work from within my Enemy class.

I drew the doors myself in Adobe Photoshop. These are just a small spritesheet with only two frames, one closed and one open. For the final exit door, I was able to use Sprite Illuminator’s ‘metallic’ depth style to make the bar and exit sign reflective, having also generated a normal map for this door. The printers I made simply by shrinking down an image of an actual printer until it became pixelated. The desk is a slightly edited version of a pixel art desk posted on pixelartmaker.com

I used BMFont to generate a bitmap version of my font ‘Hanging Tree’, which I downloaded from dafont.com.

Finally, I drew the death and win screens in Adobe Photoshop. I used brightness and saturation masks, the burn tool, selective colour, and the light rendering tool to create the death map. I found the title music and enemy roar/pain sounds on freesound.org. For the player's death sound, Sean (whom the character is based on, at least visually) provided a voice recording of himself, to which some reverb has been added.

I created the title music myself in Reaper, using my keyboard and the 'reaSynth' VST plugin. I've applied some gain and reverb effects and used the midi editor to make the loop just right.

Originally I had planned for player attacks to play little role in the game's mechanics, but decided the idea of just hiding from the enemies would be boring in a platform game. I added enemy long-range attacks, the ability to crouch to avoid them, player 'smoke' projectiles which can be fired at the enemy, a player melee attack, and health systems for the player and enemies.

Implementation & Testing

Player

The player is a simple thirty-two frame spritesheet animation, with right and left movement, a melee attack which deals 0.5 hit points of damage, and a long range 'smoke' projectile attack which deals 1 hit point of damage. The player can also crouch and 'hide' in a doorframe. I haven't fully implemented the hiding mechanic in the sense that the player can press the up arrow key when in front of a door and their opacity will be reduced, but this does not affect whether or not they're seen by the enemy.

Bullets

The smoke projectiles or bullets are implemented using a Bullet class. This is an extended sprite which has speed, scale, and direction properties, and fire and update methods. I decided to only allow attacking to the right to push the player towards the end goal and increase the difficulty.

```
  createBullets() {
    //We create a group of bullets, each an instance of our Bullet class
    this.bullets = this.physics.add.group({
        classType: Bullet,
        maxSize: this.ammo,
        runChildUpdate: true
    });

    this.bullet; //This allows us to access each SINGLE bullet
    this.lastFired = 0;
  }
```

Within the GameScene, this function is called in the create() to create an array of bullets, limited to the size of my 'ammo' variable, which I've initialized to 8 in the create() function. The lastFired variable is instantiated to 0, then in the update, whenever a bullet is fired, lastFired is set to the time passed in the update + 500. This is then checked each time the player tries to fire, to ensure the time passed is greater than lastFired, which prevents bullet spamming. In this way, the player may only fire once every 500 milliseconds. The bullet direction is always set to R.

When the bullet is fired, we instantiate our bullet using:

```
this.bullet = this.bullets.get();
```

The bullet's update function is run, and it moves forwards by multiplying the main update's delta variable with its own set speed, and adding this to its x position. When the bullet's update counter exceeds 50, the bullet is destroyed, and the counter is reset. This ensures the bullet won't linger on the screen when the player shoots and moves forward, and the bullet won't run offscreen and hit an enemy the player can't even see.

Pickups

Pickups and bullets are associated in some ways. I've shrunk down a photo of a packet of cigarettes to make the 'cig' sprite. I then create a group of these in this function:

```
  createAmmo(){
    this.cigs = this.physics.add.group({
      key: "cig",
      repeat: Phaser.Math.Between(2,10),
      score: 5,
      setXY: {
        x: this.randomPos(),
        y: this.scaleH/1.52,
        stepX: Phaser.Math.Between(500,2000),
        stepY: 0,
      },
      pipeline: 'Light2D',
    });
  }
```

The pickups are given a random quantity, a random starting position, and a random stepX to make for a more unpredictable game experience.

Whenever a bullet is fired, the ammo drops by one. When this runs down to 0, the player can no longer fire. However, if a player intersects an ammo pickup while their ammo is less than the max value of eight, their bullets will be refilled back to maximum, and the pickup will disappear. I've done this using this function:

```
for (let i = 0; i < numCigs; i++) {
/*You have a max of 8 bullets. Ammo can't be collected unless you have less th
an 8. Just refills to max.*/
  if(!this.keySpace.isDown && cigs[i]){
  if (
      Phaser.Geom.Intersects.RectangleToRectangle(
        this.player.getBounds(),
        cigs[i].getBounds()
      )
    ){
    if(!this.keySpace.isDown){
    /*Return ammo when you pick up cigs*/

          if (this.bulletCount>0) {
            cigs[i].setVisible(false);
            cigs[i].setActive(false)
            cigs[i].destroy();
            if(this.ammo<8){
              this.ammo+=this.bulletCount;
              // this.ammoPrint = this.ammo-1;
              this.bulletCount = 0;
            }
          }

      }
    }
```

The EnemyBullets are implemented in the same way using their own class, but are fired repeatedly and automatically once the player is within a certain range of the enemy. The enemy's bullets will always move left. If the player is crouching, the bullets will collide with the player without causing any damage, then play a 'pop' sound when the player is hit. Otherwise, they will deal 0.5 damage.

Collisions

Originally, my collision detection was very inefficient. I was using individual collision functions between each enemy and the player, and each enemy and the bullet instance. I then looped through each enemy and applied these functions to them.

```
  collideBulletsPlayer(){
    for(var i=1; i<this.enemies.length+1; i++){
      this.physics.add.overlap(
        this.bullets,
        this['enemy' + i],
        this['collBulletEnemy' + i],
        null,
        this
      );
    }

    for(var i=1; i<this.enemies.length+1; i++){
    this.physics.add.collider(
      this.player,
      this['enemy' + i],
      this['collPlayerEnemy' + i],
      null,
      this
    );
    }

    this.handlePlayerEnemyCollider = this.physics.add.collider
      (this.enemies,this.player
      );
  }
```

This function loops through the players, and applies the collBulletEnemy and collPlayerEnemy functions, with a number suffixed, to each of the enemies.

These two functions check for collisions and increase player or enemy damage, and deactivate bullets.

In the case of collPlayerEnemy, the hitCountIncrease function is called, which increases the player's hit count up to a certain point, then the hit count is handled by the changePlayerTint function running in the update. The player will turn darker shades of red as they take damage. The player will eventually be killed if their hit count increases above 20. A similar function, changeTint, is applied to the enemies.

Whenever an enemy dies, they are pushed to the 'enemiesDead' array, which is read in as the player's score. This is then passed into the game over scene within the GameOver function.

Instead of this, I've now created the enemies in a group/array like so:

```
  createEnemies(){
    this.enemiesGroup = this.physics.add.group();
     //Instantiate as an empty array, then push enemies as they're created.
    this.enemiesArray = [];

    for(var i =1; i<4;i++){
      this['enemy'+i] = new Enemy(this,this.randomPos(),this.scaleH/1.7,'creature','walk');
      this.enemiesGroup.add(this['enemy'+i],true);
      this.enemiesArray.push(this['enemy'+i]);
    }

    for(var j = 4; j<7;j++){
      this['enemy'+j] = new Enemy(this,this.randomPos(),this.scaleH/1.7,'newCreature','walking');
      this.enemiesGroup.add(this['enemy'+j],true);
      this.enemiesArray.push(this['enemy'+j]);
    }

    this.physics.add.collider(this.enemiesGroup,this.platform);
  }
```

I iterate over each child of the enemiesArray using a for…in loop like so:

```
    for(const enemy of this.enemiesArray){
      this.physics.add.collider(this.bullets,enemy,function(){
        this.bullet.setActive(false);
        this.bullet.setVisible(false);
        this.bullet.destroy();
        enemy.hitCount++;
        enemy.checkHealth(this)
        this.monsterHurt.play();
        //No callback function.
      },function(){
      },this);
```

Within one function, I create three colliders like this one to handle collisions between the bullets and the enemy, the player and the enemy's bullets, then the player and the enemy.

This change greatly simplified my code and allowed me to remove almost 200 lines.

Lights

Some of my assets have been packed with normal maps included, which is like a negative height map used to create the illusion of height and reflective qualities. I have applied this to a few objects, but in the background, for example, I create the light effect by calling:

```
   this.lights.enable().setAmbientColor(0x333333);
```

I then use this:

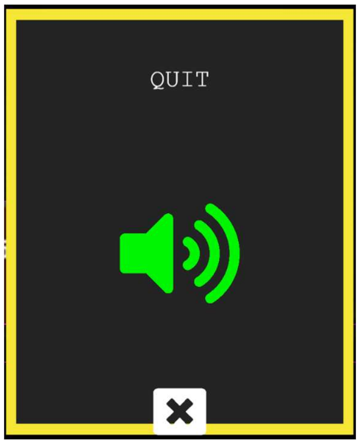
```
    this.lights.enable().setAmbientColor(0x555555);
    //  Track the pointer
    this.input.on('pointermove', function (pointer) {
        light.x = pointer.x;
        light.y = pointer.y;
    });
  }
```

Which creates a light instance and allows the light to follow the player's pointer.

This light can then be added to other objects by applying the 'Light2D' pipeline to them. I did once have this working on my enemy having created a multiatlas spritesheet but found it too difficult to apply after adding the Enemy class.

Pause menu

The pause menu can be enabled in-game by pressing the tab key. This brings up a menu which allows the player to quit the game and return to the title screen. The PauseMenu is a class, an instance of which is created when tab is pressed. When the 'close' icon is pressed, the instance is destroyed. This feature only partially works. If the player returns to the title screen via this menu, of the title screen's options will lead into the GameScene, and the music will overlap once the GameScene is started. The game also can be muted from the pause menu, which works as intended.



Score

I thought a ‘high score’ would not make sense in my game’s case, as there are only six enemies. Instead I’ve implemented a ‘fastest death’ localStorage variable which is accessible from the ‘Records’ page. This variable records the player’s time in-game and is reset if the player dies faster than they ever have before.

Tweening (Removed Feature)

My enemies did have a tweening animation, which caused them to move back and forth from their random position until coming within range of the player, but I had to remove this after having difficulty applying it to enemies once the Enemy class had been added. They did work in this way:

```
      createEnemyTween(scene,x){
        this.tween = scene.tweens.add({
          targets: this,
          x: { from: x, to: x+300 },
          ease: 'Linear',
          duration: 2000,
          repeat: -1,
          yoyo: true,
          flipX:true,
          paused:false
        });
        if (this.tween.isPlaying() && this.seesPlayer)
        {
          this.tween.pause();
            console.log('tween paused')
          }
        else
        {
          this.tween.resume();
            console.log('tween resumed')
        }
      }
```

How I Achieved the Requirements

Animate single characters with spritesheets

I have animated three game characters based on spritesheets.

Produce assets

I have produced most of my assets myself in Photoshop, Aseprite, and Reaper.

Include sound with ability to mute in options

I have included title music, main music and enemy death/pain sounds, which can be muted using the speaker icon in the options menu.

Running total score, also visible in gameOver

A running total of enemies killed is visible in the gameScene, which is carried over into the gameOver scene if the player dies.

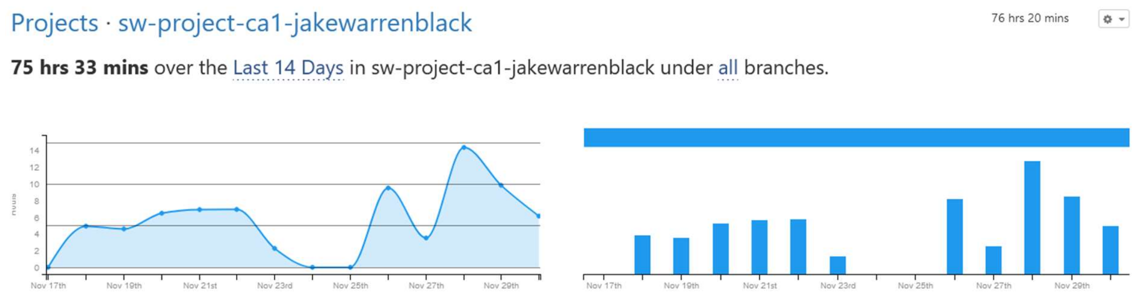
High score stored and retrieved as local storage object

I thought the idea of a high score was unsuitable for my style of game, so instead have implemented a ‘fastest death’ record which is stored as a localStorage object and retrieved in the ‘Records’ scene. This is the same in principle.

Additional features & creativity

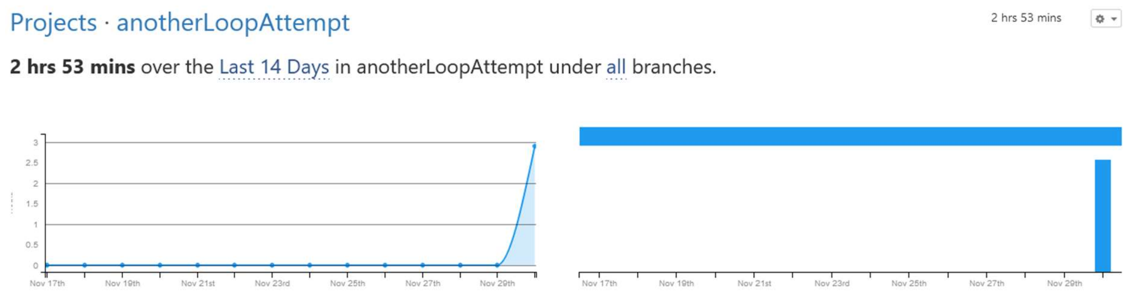
I have produced many of my own assets and included additional features such as the lighting effects and in-game pause menu.

Project/Time Management


Projects · sw-project-ca1-jakewarrenblack

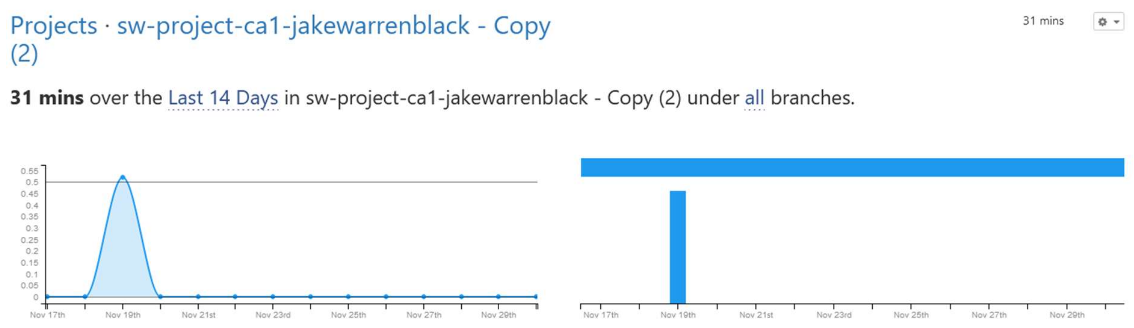
76 hrs 20 mins

75 hrs 33 mins over the Last 14 Days in sw-project-ca1-jakewarrenblack under all branches.


Projects · anotherLoopAttempt

2 hrs 53 mins

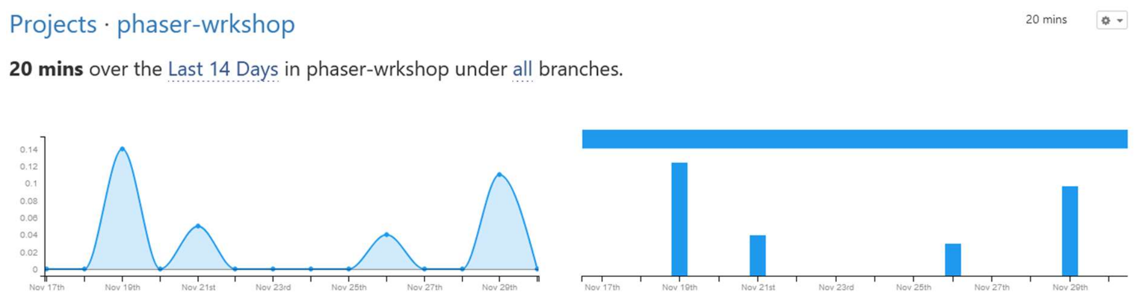
2 hrs 53 mins over the Last 14 Days in anotherLoopAttempt under all branches.



Projects · sw-project-ca1-jakewarrenblack - Copy (2)

31 mins

31 mins over the Last 14 Days in sw-project-ca1-jakewarrenblack - Copy (2) under all branches.



Projects · phaser-wrkshop

20 mins

20 mins over the Last 14 Days in phaser-wrkshop under all branches.

In the WakaTime Visual Studio Code extension, I have logged 80 hours working on this CA over the last two weeks. I would estimate I spent the same amount of time on it in the two weeks prior.

This equates to 10 out of each 14 days working a full 8 hours on the game. I would say most of this time was spent either trying to add the Light2D pipeline to my animated spritesheet or trying to simplify my collision detections by converting them to a loop. To me, even if only in the case of my first problem, the time it took to find a solution mostly by trial and error or by consulting years-old posts on the Phaser forums is indicative of poor documentation for Phaser 3.

To any future developers who would hypothetically work on my project, I would suggest avoiding the Phaser documentation itself, which can feel very ‘arcane’, and instead using Github user rexrainbow’s notes on Phaser 3.