IS51016A: Audio Visual Computing

Coursework 2 Report

Andre Matheus Fedalto (33374468)

Arthur Floriani Martins (33374471)

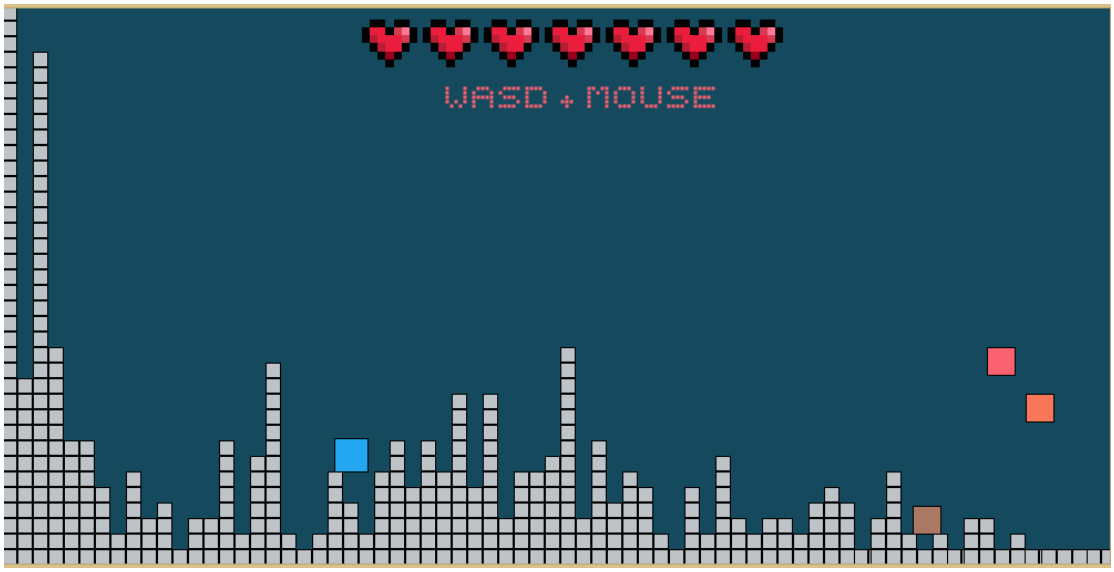
Executive Summary

FFT Shooter is a survival game created using Processing. It is a square world and you are a square that must survive to other squares shooting pixels trying to erode you. Every time you get shot you'll lose some of your pixels, and when you run out of lives you'll explode into a bunch of shapeless pixels. The movement is not random, but actually controlled by data parsed from the Fourier transform of the background music.

Features and User Interactions

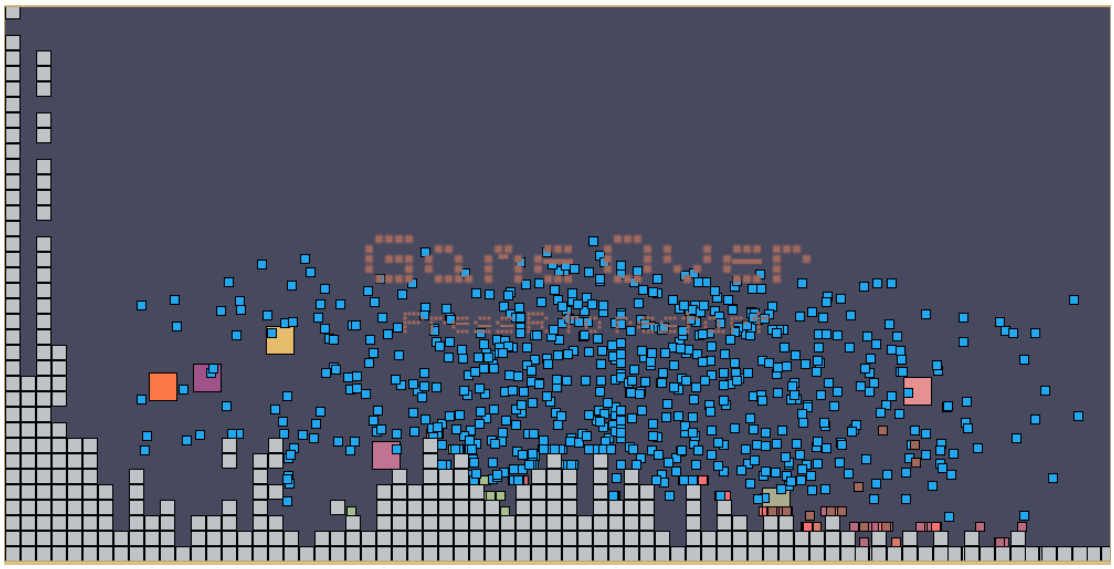
We wanted to build something fun and that we could use the concepts we learned in class, so a game was the way to go. With this game we were able to use concepts of bitmap representation, vector representation and maths, transformations, trigonometry, control, physics and, to go that extra mile, audio data.

The extra mile provided the interaction that runs in the background. Since we are interested in audio data extraction and processing, we extracted data on-the-fly from the soundtrack using FFT. This data is then processed and used to generate the terrain and also to control the enemies spawning time (always synced with the beat). If you change the soundtrack file you can expect the overall behaviour of the game to change accordingly.



There is also the basic gaming interaction using keyboard and mouse. You can use the arrow keys to move your square and the mouse to aim and shoot. You can also use spacebar to shoot, in this case it will always shoot towards the mouse position.

We are also into physics simulation, so working with vectors to simulate real world behaviour was a pleasure. We used the basics (position, velocity and acceleration) to implement the physics, so the game ended up with gravity, collisions, physics controlling the projectiles you shoot and also a particle system controlling the pixels that are eroded (which is really visually appealing).



Development

The code was created based on object-orientated programming, therefore we have specific classes to handle specific behaviours. There is a main class called *VisibleObject* which handles the objects that are visible in the screen. Since each type of object (terrain, enemy, player) have a specific behaviour, those classes inherits from *VisibleObject*. We end up with classes like *DestructibleTile*, responsible for handling the destructible terrain; *Enemy*, responsible for all the actions of the enemies; *Player*, responsible for your character and also handling the input interactions; *Projectile*, which is a class shared among *Player* and *Enemy*, since both shoot the same type of projectile but with different characteristics; *Particles*, which handles all the pixel eroding when the player, or the enemy, gets shot.

The *fftAudio* class is responsible for operating the background interaction between the soundtrack and the game behaviour. It loads and process the soundtrack beforehand, so when the game starts we already have data from the audio to use. So when the song is played (by AudioClass, mentioned later) the data is only synced. This class loads and chops the file in blocks which are then sent through and FFT and stored in a matrix. More detailed information can be found in the class source code, which is extensively commented. Also for audio processing we have the *AudioClass*, responsible for handling the sounds played during the gameplay, such as sound effects and background music [1]. Both *fftAudio* and *AudioClass* use Minim [2], both for fft and playback features.
The heart of the game is inside the *ObjectManager* which handles every action in the game, including the collision between objects, instantiation of every particle, spawning new enemies and every main action in the game. This *ObjectManager* class is controlled by a *Game* class, which orchestrate how the game loops will work. This *Game* class also integrate the fft with the game mechanics, generating the terrain based on the fft and also calling the methods inside *ObjectManager* to work with the music. An example of our basic physics simulation can be seen bellow. The code bellow is inside the enemy mechanics, which decides what to do on every loop. All the source is on our GitHub [3]

```
  //If it has been at least 1500ms from last jump and enemy is touching the floor
   if(millis() - lastJump > 1500 && grounded)
    {
      //Get the time of the current jump + add a random value to make all enemies jump
on different times
      lastJump = millis() + random (-300,300);
      //Checks if target(Player) is on left or right of the enemy
      //Change the velocity to UP and to the LEFT or RIGHT, so it will jump towards
the enemy
      if(target.x < position.x)
        velocity = new PVector(random(-3, -6), -15.0f);
      else
        velocity = new PVector(random(3,6), -15.0f);

       //If the game is over, enemy jumps only UP
       if(gameOver){
         velocity = new PVector(0, random(-14,-20.0f));
         lastJump = millis() + random(-500,0);

       }
       //Sets grounded to false as enemy has jumped
       grounded = false;
    }

    //Add the gravity (which is a constant)
    velocity.add(gravity);
```

References

(This section lists any external libraries, websites, documentaiton or other resources you have used.)

[1] Glass Lux – I'm a Machine. Downloaded from FreeMusicArchieve.com (http://freemusicarchive.org/music/Glass_Lux/)

[2] Minim library (http://code.compartmental.net/tools/minim/)

[3] Project github: https://github.com/Artfloriani/FinalProjectAV

[4] Font used on the project - http://www.dafont.com/pixelate.font