CS4303: Video Games
The Game - Group Report

Lecturers: Dr Joan Espasa Arxer, Prof Ian Miguel
Matriculation Numbers: 190019931, 190022658 (Group 23) 09 May 2023

Overview

The goal of the final practical for the Video Games module, “The Game'', is to demonstrate an ability and understanding for making video games (possibly using techniques discussed in lectures) by developing a video game of our own design using the Processing graphics library (written in Java).

Requirements

The practical problem specification outlines the following requirements and deliverables for the produced game:

- Source code to run the game
    - Written in Processing
- A Group Report
    - Documenting the design, implementation and evaluation of the game
- Individual Reports
    - Per group member, discussing the individual contributions towards completing this practical
- A “Player’s Guide” for playing the game
- A commented video demonstration of the game in action

Note: Both the player’s guide and the comments for the video demonstration can be found within this report (under the Design section - Player’s Guide, Example).

Design

The success of any video game is often dependent on two key factors: the mechanics and the graphics. Game mechanics are the rules and systems that govern how the game functions, while graphics are the visual elements that define how the game looks.

In the following sections, we will delve deeper into the design and implementation of the interesting game mechanics and graphics of EFIR, exploring how these two elements work together to create an engaging and immersive gaming experience.

Context

“EFIR” (pronounced as “ever”) is a Survival based, First Person Shooter (FPS) game.

Figure 1: Title Screen

The title visually comprises a play on the word “FIRE” (rearranged letters) and phonetically sounds similar to the word “EVER”. This ties to the premise of the game, which is to continuously fuel a flame until its energy radiates across and lights up the entire map. The

inspiration for the game dates back to the control of fire by early humans. Being able to control a large enough flame not only provides us with warmth but also safety, to survive.

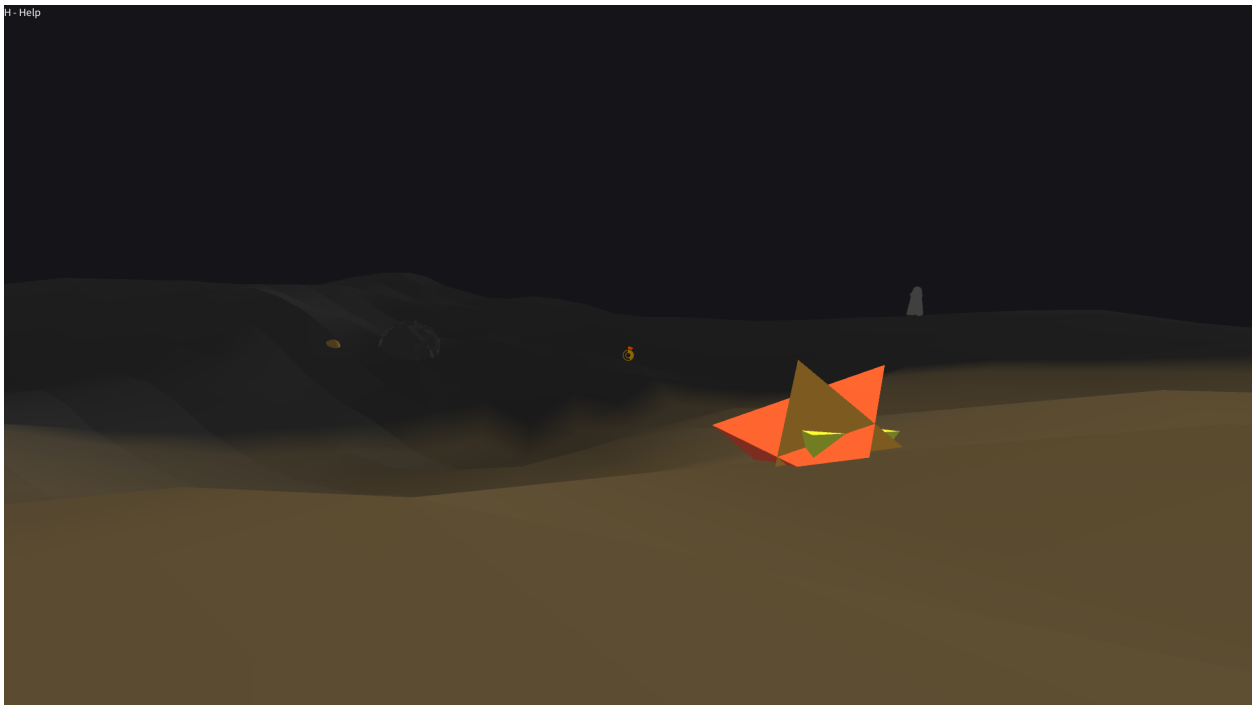

Figure 2: First Person Gameplay

In EFIR, you, the player, plays as a hero spawned in a dark, unknown land next to a burning, but depleting fire. This fire radiates energy in the form of light and heat of a certain radius surrounding it. This energy both heals the player (provides warmth) and hurts any enemies standing within its radius (provides safety). It is because of this, that enemies standing outside of the radius are afraid of the fire and will attempt to avoid it. These enemies, randomly spawned in the darkness, however are not afraid of the player. As such, the player is only safe within the fire's radius and if the player steps outside of it, they are subject to both the cold (which hurts the player) and any roaming enemies which may spot the player (and attempt to hurt the player). In order for the player to survive and win, the flame must be fueled, by shooting it, to the point at which its energy radiates across and lights the entire map. Fuel can be acquired from enemies upon their death from either the player shooting them or the flame burning them. Note that enemies do not drop fuel from their own attacks, even if it kills them, as in this instance, enemies use the fuel to execute their attacks. In EFIR, the concept of fuel and health is the same, picking up fuel to fuel the flame heals the player's HP so that it can be used on the flame or to survive. The player

loses the game if either they or the flame dies. Note that the flame becomes more difficult to fuel as it increases in size (decays faster) to make the game more difficult as the player progresses.

Figure 3: Map (Third person) Gameplay

Figure 4: Map (Third person) Gameplay - zoomed

The game is played from both first and third (top down) person perspectives.

The first person perspective adds to the game's immersion as the player sees and plays through the eyes of their character. In fact, in EFIR, we prioritised minimising any visual clutter on the screen to further add to the immersion. The interface is very simple and intuitive, consisting of a single crosshair which gives details about the player (HP, blocking status) and the fire (HP, direction). (See Figure 8)

The third person perspective allows the player to make more strategic choices taking into account the topology of the map and location of enemies/resources. This is especially important in EFIR as the map is procedurally generated with high/low ground and entities/obstacles such as the fire, enemies, bushes/shrubs, fuel etc.

Figure 5: Player blocking

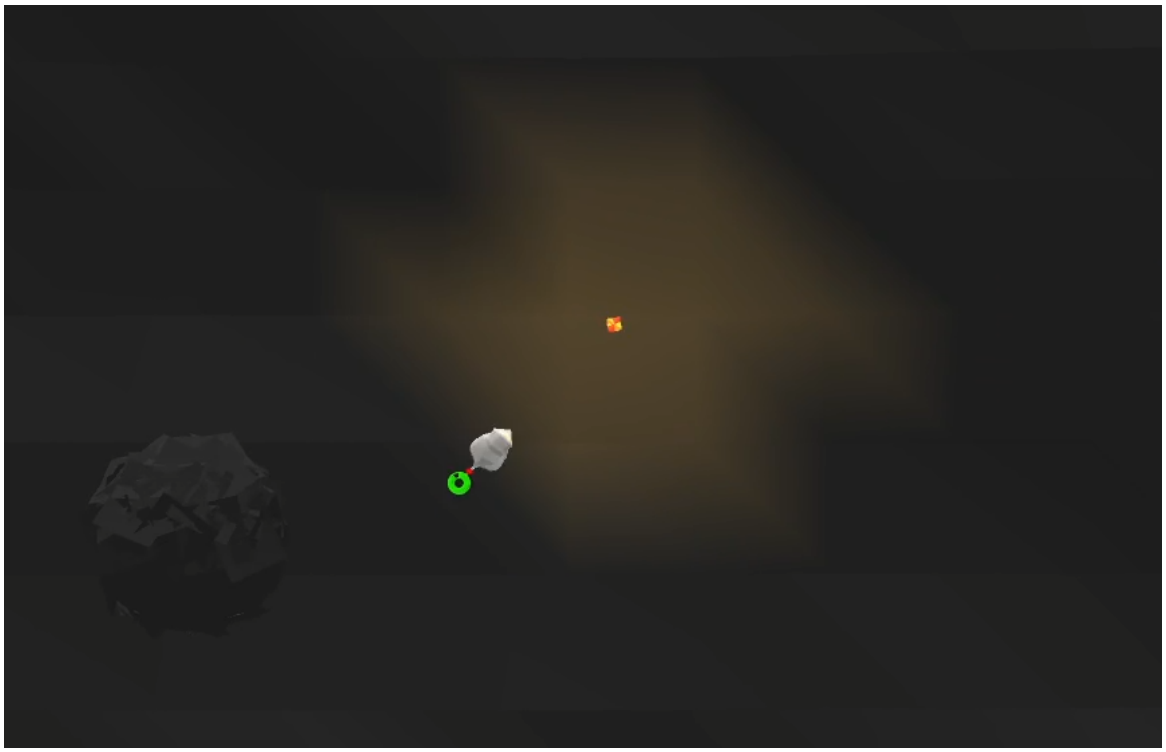

Figure 6: Player dashing

Figure 7: Player shooting

Figure 8: Help

In addition to being able to walk, look around, and shoot (as with other FPS games), in EFIR, the player can also block and dash. Dashing gives the player a momentary burst of speed in whichever direction they are moving in at the cost of some of the player's fuel/hp and blocking reduces incoming damage (from enemies and the dark) by 50 percent but at the cost of the player's speed. Shooting is also an important aspect of the game to discuss, in EFIR, the player shoots a projectile from their position in the direction they are facing. The projectile, bullet, is subject to gravity. This is done on purpose in order for the player to be able to shoot the fire anywhere on the map, even when the fire is not within line of sight of the player. In terms of what the player can shoot, the ground/outside map will absorb the bullet, the fire will be fueled by it, enemies will be hurt (and knocked back) by it, and bushes/shrubs will be damaged by it. Note: Bullets have no effect on fuel dropped though fuel does disappear after a while.

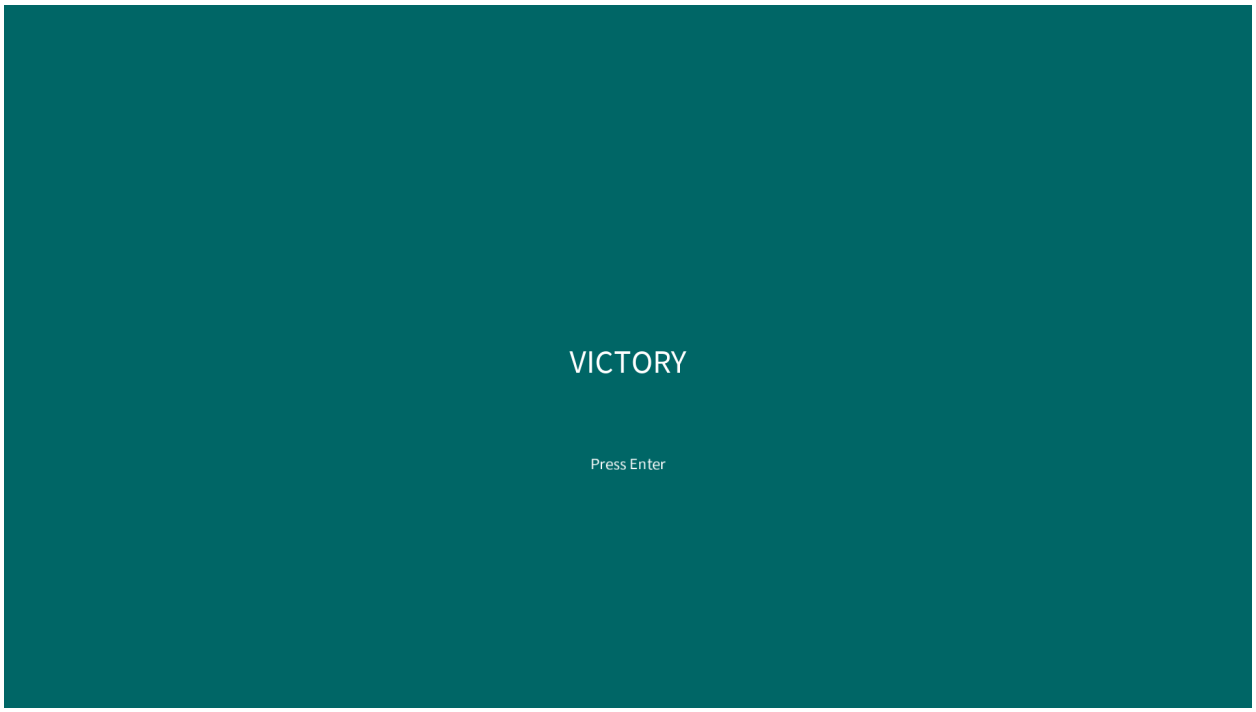

Figure 9: Victory screen

Figure 10: Defeat screen

Eventually, the game is either won or lost by the player and they will be presented with either a victory or defeat screen where they can decide to go back to the main title screen and play another game.

Player's Guide

The game EFIR is a survival/defence RPG where the goal is to survive. To survive, you need to kill enemies and collect fuel for the flame in the centre. If the flame runs out, you will die. And you win if you manage to expand the flame's territory to cover the whole area.

Rules

- Survive
- Kill enemies for fuel
- Fuel heals player
- Enemy hurts player
- Block reduces speed and damage taken
- Fire fuel at fire to increase its energy

Lose by

- Player runs out of health
- Fire runs out of energy

Win by

- Maximising fire's energy

Interface

- Fire's direction is shown on the crosshair
- Player's hp is shown on the crosshair

Controls

- w - forward

- a - left
- s - backward
- d - right
- SPACE - dash
- left mouse button - shoot
- right mouse button - block
- TAB - swap view
- SCROLL - zoom

Example Gameplay

Some exemplary gameplay of the game can be found under Videos/Gameplay_old.mp4. This gameplay is slightly outdated but shows the main game states and interactions between entities. Since the video, the game has been updated in the following ways:

- Updated help menu (see Figure 8)
- Inclusion of bushes as obstacles (see Figures 2 and 3)
    - The player, bullets and enemies cannot go through these
    - The player can shoot it to damage and eventually destroy them
    - They are spawned randomly around the map on map generation
- Knockback of enemies (experimental)
    - We weren't quite sure whether including knockback of enemies would make the game too difficult, however the option is there, and it is currently included in the latest version of the game.
- Updated win/lose screens (see Figures 9 and 10)
    - Following the minimalistic style of our game, "You Win" has been changed to "Victory" and "You Lose" to "Defeat".
- Upgraded bullet-ground collision detection (see Implementation, Mechanics)
    - Uses the bilinear interpolation method for finding the height of the ground at a particular point.

Due to time constraints, we were unable to provide an updated video showing all these interactions; however, we have included an additional video of the updated game which

covers most of the possible interactions and states (Gameplay_new.mp4). We decided to include both of these to show the general gist of the game though the old version more accurately reflects a normal run of the game and the new version more accurately reflects the depicted game but attempts to showcase the interactions within the game rather than play the game casually. Both of the videos however demonstrate and follow the description of the game as described in the previous section.

Implementation

This section will attempt to discuss the implementation of the game in terms of the game mechanics and game graphics as discussed in the design section. To highlight the particularly interesting and noteworthy aspects of the game's implementation (involving notable technical challenge(s)), a top down approach will be taken in order to better convey how everything works together without going too much in detail of every single aspect of the game's implementation (the source code is available for this).

Mechanics

The main interesting game mechanics of the game's implementation include: Physics, Procedural Generation, AI, 3D, Collision Detection, and Lighting.

Physics

We made use of a force based particle physics engine with additional rotational attributes in addition to the linear equivalents (e.g) Position -> Orientation, Linear velocity -> Rotational velocity, Linear acceleration -> Rotational acceleration, Force -> Torque). Note that the rotational force provided in this engine is simulated (as we do not consider the point at which rotational force is applied, instead directly manipulate the orientation similar to the integration steps done for linear motion - Force->Acceleration->Velocity->Position) and hence, we are not making use of a rigid body physics engine. This allows our player and other entities to look in different directions, especially important for an FPS game. The

use of a damping factor also helps to simulate air resistance and makes movement of characters feel more realistic (not starting/stopping instantly).

The use of forces also provides a realistic scene as it is modelled over the very equations that can apply to real life.

The use of particles for everything also allows extending the game to be easier since new entities can be trivially added through inheritance. This is exemplary in how the Player, Enemies, Fuel, Bullets, Bushes, Fire are all entities.

Procedural Generation

The map is procedurally generated and consists of: a 2d array of vertices representing the height of the terrain at different points; some bushes of random sizes placed within the map; some enemies periodically placed within the map; a fire and the player placed somewhere randomly on the map. Perlin noise was used in order to make the terrain seem natural as it generates “random” values which are related to the previous and next generated “random” values.

One interesting challenge was in trying to find the height of a point on the map between vertices, for this we employed the use of bilinear interpolation since the mesh is made of flat triangles. The use of bilinear interpolation also increases the performance of the application as there is no longer a need to create so many vertices as the entities can be placed on the map between vertices.

AI

Enemies make use of a simple decision tree to determine their behaviour at any given point. They are either in a roaming state, avoiding state or attacking state. This has been efficiently implemented through a simple if statement which describes what direction the enemy should be looking. As the enemy is constantly moving forward, this means we can map different behaviours based on how we choose where the enemy is to look.

If the player is outside of the fire radius and is within the range of the enemy, the enemy will look at the player and enter an attacking state. Otherwise if the enemy is next to the fire’s range, it will stare away from the fire and therefore enter an avoiding state. If neither of these conditions are true, the enemy simply enters a wandering state looking in a

“random” direction. Perlin noise is used to choose the direction so that the enemy smoothly transitions between moving in different directions rather than constantly changing directions every frame. Note that the enemies’ awareness to the fire’s range increases as the fire’s range increases, this adds to a sense of increasing difficulty in the game as the player progresses.

3D

One of the main challenges in building this game was to incorporate another dimension. We wanted to add another dimension to make the game more immersive, as we live in three dimensions. This was simplified once understanding the systems of spherical coordinates, cylindrical coordinates, cartesian coordinates and how to transition between them. Simply put, we can use the angles introduced with the addition of the (2) new possible rotational axes in the system with vectors to move around the 3d space. Processing also supports 3d capabilities with P3D and this means we can visualise this movement in the 3d space with standard graphics practices such as translation, scaling, rotation, etc of different objects (such as the camera). Processing also allows us to create new graphics contexts which means we can layer a 2d interface on top of the 3d interface as we have done in the game.

One simplification we made was that every entity on the land is represented as a sphere internally, this made collision detection simple as we are only required to check that spheres are a certain distance from one another using the PVector dist() method. Additionally, collision detection with the ground and bullets is simplified as we do not check every point on the ground each frame but rather, only the height of the point of ground the bullet is directly above/below. With bilinear interpolation, not only is this quick but also accurate.

Lighting

In order to show the fire’s energy, one might assume that we changed the colours of the vertices in the land depending on its distance from the fire. This is extremely inefficient, especially with many vertices and so we used trigonometry and the lighting instead. - A Spotlight with a certain cone angle is placed above the fire, a distance determined by the reach of the fire (TOA). This is much more efficient as there is no need to check every vertex

every frame. Additionally, this creates a more engaging and realistic scene, in fact, the use of spotlights allows us to specify how much light energy should be focused on the centre of the cone of the light (making light fall off naturally is possible). However, since we wanted to make it easy for the player to distinguish the parts of the terrain that are dark and the parts of the terrain that are light, we decided to make the spotlight very biassed towards the centre.

Graphics

Our design philosophy was to make the game both immersive and as minimalistic as possible. It can be hard to strike a balance between these two features as immersive experiences are usually realistic, and real life is not minimalist but rather complex. Throughout development we learnt that consistency is key to find a good balance between these desired qualities.

Colours

A minimal and simplistic colour scheme has been used to bring the player's attention to the most important aspects of the game and increase visibility in this way. Since the win condition consists of fueling the flame to its maximum energy, it's important to effectively convey how much energy the flame currently has and its current distribution across the map visually. The shades of black and white used for the environment and the colour that the fire radiates produces a contrast which allows the player to easily see the boundaries of the flame and therefore, their progression. Certain entities are highlighted, such as fuel and the flame itself with bright colours, this makes it easy to see where these entities are which is especially important as these entities expire. Enemies on the other hand, do not expire and therefore are not highlighted in any colour, the same goes for bushes and the terrain. Bullets are also a bright colour to show where the player is shooting, and once they hit an enemy, enemies also take on a slight colour to show that they've been hit (along with the knockback they experience). The crosshair is obviously important and therefore bright to assist with the player's aim and also to convey information such as the player's and fire's health and direction effectively. Note that the player model themselves does not take on a bright colour despite being able to expire, this is because we felt no need to convey this

information with other indicators such as the healthbar and the screen turning red when the player's health reaches below 25 percent. Additionally, giving colour to the player model may make the player feel more separate from the in game character.

Models

Simple models have been used to represent the different entities in a similar fashion.

The fire is made up of three rotating tetrahedrons of different "fire" associated colours. This is because the classical element, fire, is often associated with triangles and tetrahedrons - Plato.

The player, enemies and bushes are free models sourced from SketchFab. These models were specifically chosen as they do not rely on colour to convey their definition. - Gives freedom for us to associate them with their own colours (shades of black and white) without looking terrible.

The terrain is made up of many triangles to make a mesh using Processing's Triangle Strip. This gives an easy way to create terrain out of a 2d array of vertices. Additionally, each vertex can be associated with a colour, and since perlin noise was used to generate the heights of the vertices in the terrain, associating higher heights with brighter colours can give the illusion of a subtle light source, representing the night sky. This also gives extra contextual information to the player when in the third person view as they may identify which points of the map may be advantageous (high ground is lighter).

Note that we had also explored adding wood logs to support the flame, however this did not look right with the logs expanding as the flame expands and hence this was omitted from the final game (however the code is commented and may still be uncommented).

Interface

The interface was kept as minimalistic as possible whilst attempting to convey as much information as possible. We described this as an "efficient" interface. A highly efficient interface combined with intuitive indicators adds to the immersion.

An example of this would be the crosshair which serves as the player's health bar, the player's blocking status, the fire's health bar, the fire's location relative to the player and also as a crosshair for aiming.

The screens and menus were also kept minimalistic with a single colour background of the colour scheme used by the game and minimal, descriptive text. Elements within these screens are also consistent to add to the intuitiveness and learnability.

Evaluation

Testing

The game was developed as it was intended and has been playtested. Further fine tuning of values is possible but the game has been tested with different values and is winnable yet challenging. The gameplay has been tested and a combined version of games has been recorded as in example gameplay.

In order to test the performance of our application, we referred to an FPS (Frames Per Second) counter included within the game. For each of our machines, we were able to hit and maintain a steady, playable frame rate (60 FPS) as seen in the example videos provided.

Critical Appraisal

The game has a variety of elements. The strengths and weaknesses are highlighted below:

Strengths

- Enemy generation: The enemies are randomly generated with fixed intervals and a maximum number of them.
- Map generation: The map includes the obstacles and ground level which look very pleasing and add a new dynamic.
- Projectile physics: The bullets shot by the player have physics applied to them which also adds an interesting play style.

- Lighting: The darkness and light that indicate the territories are visually appealing and are very seamless.
- Aesthetics: The minimalistic design which includes the crosshair with its hp bar and direction feature, the help menu and the 2d map.
- Simple controls: The controls are very simple and intuitive.
- 2D map: This gives the player a birds-eye view of the entire map.
- 3D game: Adds to the immersion, we live in a three dimensional world, it's intuitive for us to be able to look up, down, left, right, forward, backwards.

Weaknesses

- No settings: This is not very important in the current state of the game where there are not many things that can be user controlled.
- No leaderboards and/or levels: This limits fun since there is no way to compete with other players and may leave the player with a lacking sense of progression outside of individual games.
- No background sounds: This is an optional detail that can be implemented but is not very important. It also requires the use of additional libraries.
- Not fully optimised: Although some optimisations have been made to the game (see Implementation), this game may be difficult to run for weaker machines.

Word Count: 3590