TECHNOLOGENE

By Jack

First runner up: Year 10-12 Open Platform

CONSTRUCTION/DESTRUCTION

Proudly supported by

CREATIVE VICTORIA

PAX AUS

IGEA
interactive games & entertainment association

ACER

TECHNOLOGENE
GDD

Created by

JACK ROBOTHAM

for the

AUSTRALIAN STEM VIDEO GAME CHALLENGE

Image: screenshot of the menu screen after defeating the intended final boss for the game. Due to time constraints, this boss was never implemented.

CONTENTS

GAME OVERVIEW

Description

Technologene is a combat and strategy based game where the player creates robots to overcome specific challenges within each level. Between levels, the player constructs their robot out of a number of blocks, each with different effects and interactions. During the level, the player uses the robot they created to accomplish a goal. If they successfully complete their goal, then they continue, and if they fail, they can improve on their design and try again.

Title

The title, “Technologene,” comes from a combination of “technology” and “gene.” You play the game as a micro-robot (a nanite), created by the weapons and technology company GeniTech (a combination of “genetic” and “tech”). During the game, you control the Technologene, a learning, sentient machine with the power to control larger computer systems, fighting against GeniTech, your creators.

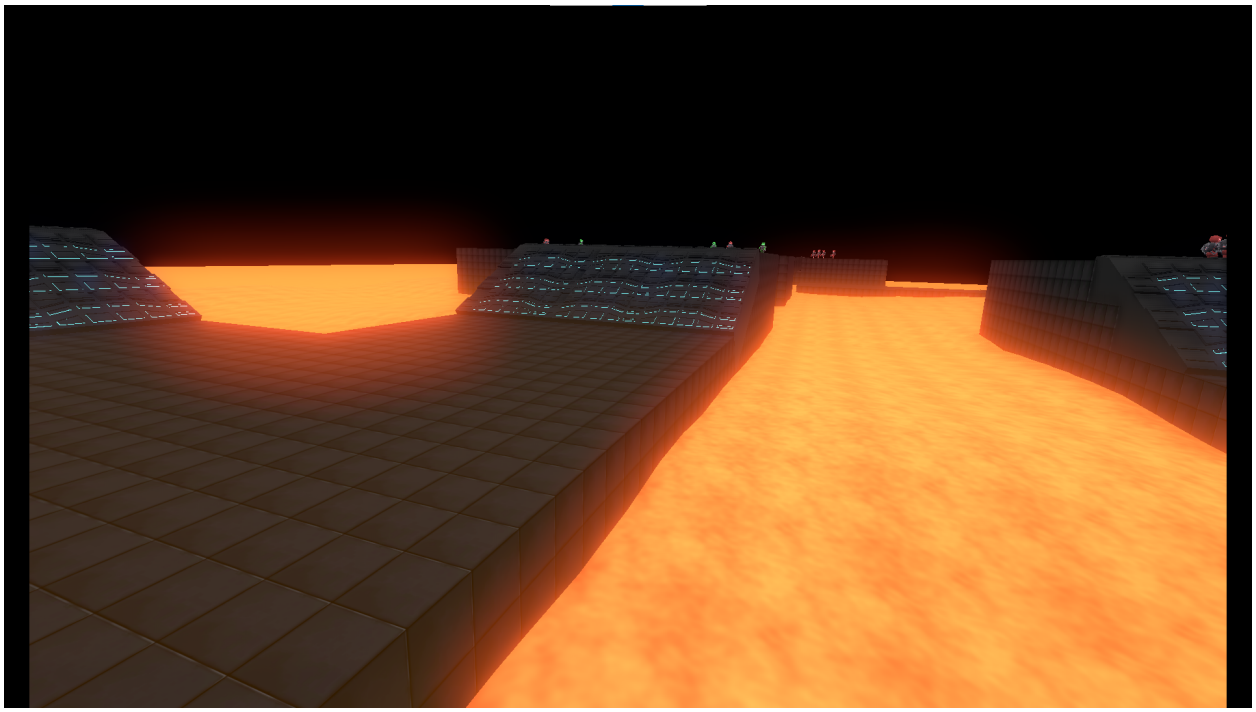
Characters

I originally planned for there to be a character that talked to the player throughout the game, but I didn’t have enough time to build enough levels to convey their story. However, I still needed a way to convey certain information to the player so I used them as a sort of “inner monologue” for the character. They let the player know when certain things happen or draw their attention to anything important. I recorded 20 voice lines in total, using Voicemod to make myself sound like a robot.

Apart from that, there are no characters that can be interacted with in the game, although I did my best to give the final boss personality through its animation.

Environment

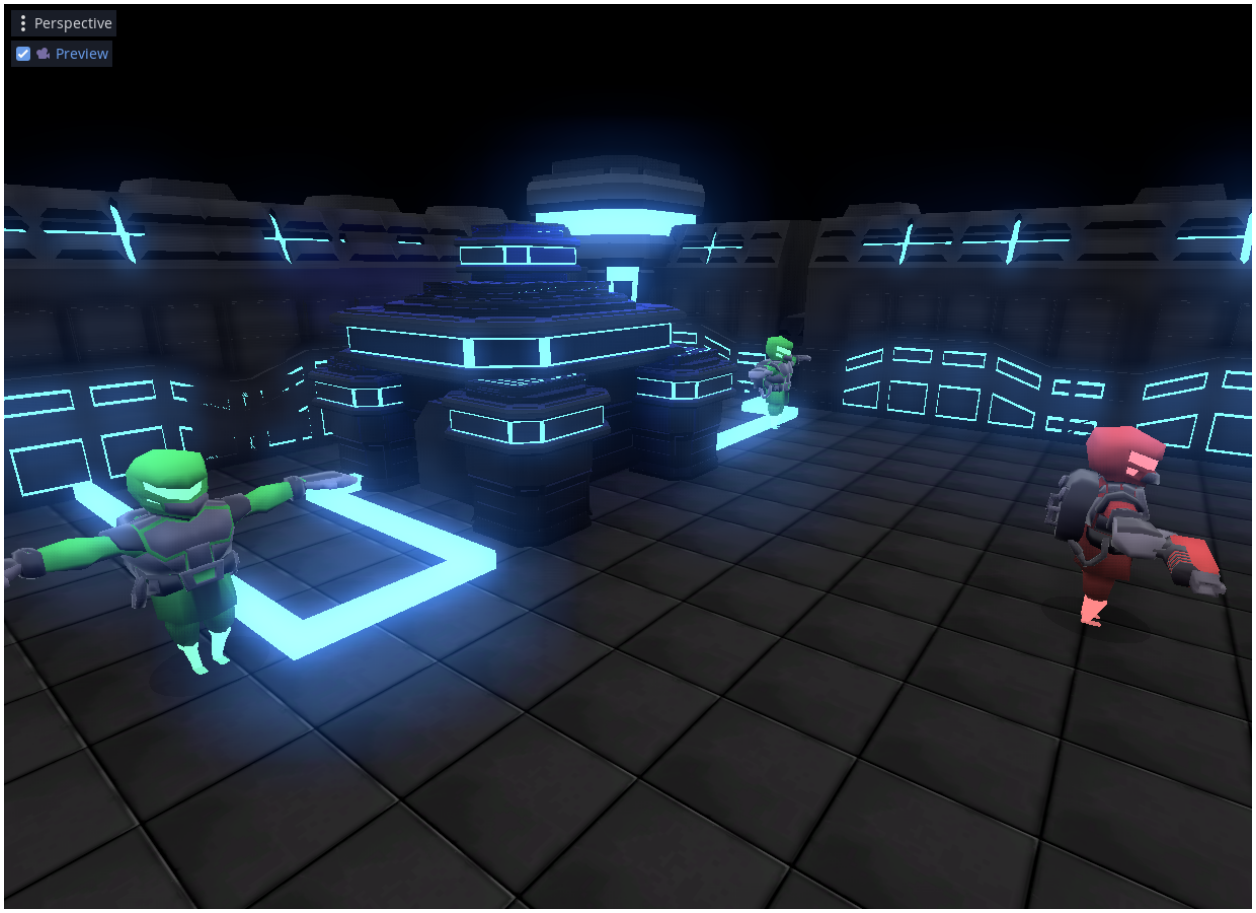
The levels of the game all take place in different areas of GeniTech's facilities. You can manufacture a robot anywhere that has the equipment, so it is easy for you to attack any part of the company. GeniTech guards their technology with robots and mechs, which it manufactures in abundance, so they can frequently be found around any level of the game.
GeniTech's facilities are large, sprawling buildings and complexes allowing for large and expansive levels. The facilities are built into a large underground chasm, and some areas still have bridges and pits over the abyss. Being underground allows GeniTech to keep their servers and nodes secure (although your ability to establish yourself wherever you want nullifies this). GeniTech also avoids human error by using robots, although their robots are not very good at adapting. They are primarily designed to track and shoot, which is enough to deter human intruders.



One of the most dynamic areas, Junkpit, involves molten metal that rises and falls.

AUDIENCE

This game is intended for audiences of about 14 and up. This is a rough approximation, but the strategy and problem solving could potentially confuse some younger children. In most instances, difficult levels can be offset with careful strategy, and alternatively, bad strategy can sometimes be offset with a large number of guns, so the game rarely requires extreme levels of skill. However, older audiences that understand how to use the blocks correctly and can strategize to their robot's full potential will enjoy the game more.



I reduced enemy numbers after playtesting found the game to be far too difficult, so that younger audiences would still be able to enjoy it.

THEME

This year's theme was “Creation & Destruction.” The game needed to involve some form of creation, and some form of destruction. I found the easiest way to think about it was to separate it into what was being created, and what was being destroyed.

In Technologene, you play as a robot that is *created* by the player, out of various blocks that have different functions. Different levels will provide different challenges, and the player will need to creatively design different robots to meet these challenges. The gameplay loop encourages the player to create a robot, find ways to improve it, and then create a new, better robot.

For destruction, I decided to make the goal of the game based on destroying things- easy enough, really. Your overall goal is to sabotage the robotics company that you were created in, and you do this by destroying various things from level to level. All levels contain enemies (robot sentries) that the player must destroy to progress. The goals of levels also often involve destroying generators, data nodes, and power cells. There isn’t a level in the game that can be completed without dealing damage (unless the player does some very clever designing), so the player will be engaging in destruction to progress throughout the game.

I found explosions very amusing when I first coded them. Destruction is personally the most enjoyable part of the game.

PLANNING

Inspiration

This game is something I'd been thinking about for a while. My original idea was for the robot to be fully automated, running through a specific program, and responding to conditions in the level. However, I soon decided that I wanted the player to have greater control, so I allowed the player to move and activate parts of their robot themself. The robot construction mechanics are similar to some smaller games I've seen in the past, but these leaned more towards automation rather than control. Most of the blocks and ways of robot construction were original, resulting from brainstorming.

I also found surprising inspiration in Starcraft II, a real-time strategy game. I used Starcraft to initially plan the top-down camera angle for the robot, but I also liked the level design in Starcraft. In each level of Starcraft, the goal is different, and levels often include unique, once-off mechanics. It works well to add variety, and gives the player something to learn each level without overloading them with information, since they only need to consider the current level's mechanic. I applied this to most of my levels, giving different requirements for the different levels to challenge and interest the player.

Technical Requirements & Platform

I created this game in the Godot game engine (version 3.5 stable). Although Godot 4 has recently been released, and has a greater 3D capacity and extra features, Godot 4 included a number of changes to the scripting language and nodes that I was not comfortable with learning within the time limit. I decided to stick with the engine I was more experienced with.
I've exported the game to run Windows, partly because Windows is the most commonly used operating system, and partly because it is one of two options in the submission guidelines. Technologene can run on almost any computer, due to the resolution and graphics settings I added into the menu. The maximum settings may need a powerful computer, but I made sure that the minimum settings run well on a slow computer.
To further help with game speed and lag reduction, I made some further optimisations. Because I used ambient light and SSAO (screen space ambient occlusion) for shading, I didn't need to use any lights or shadows, which are one of the largest strains on 3D godot games. I used a lot of heightmaps to add the appearance of depth to models (particularly robot blocks) without increasing the amount of polygons the computer had to render. I also used glow settings to give the appearance of light where there wasn't any.

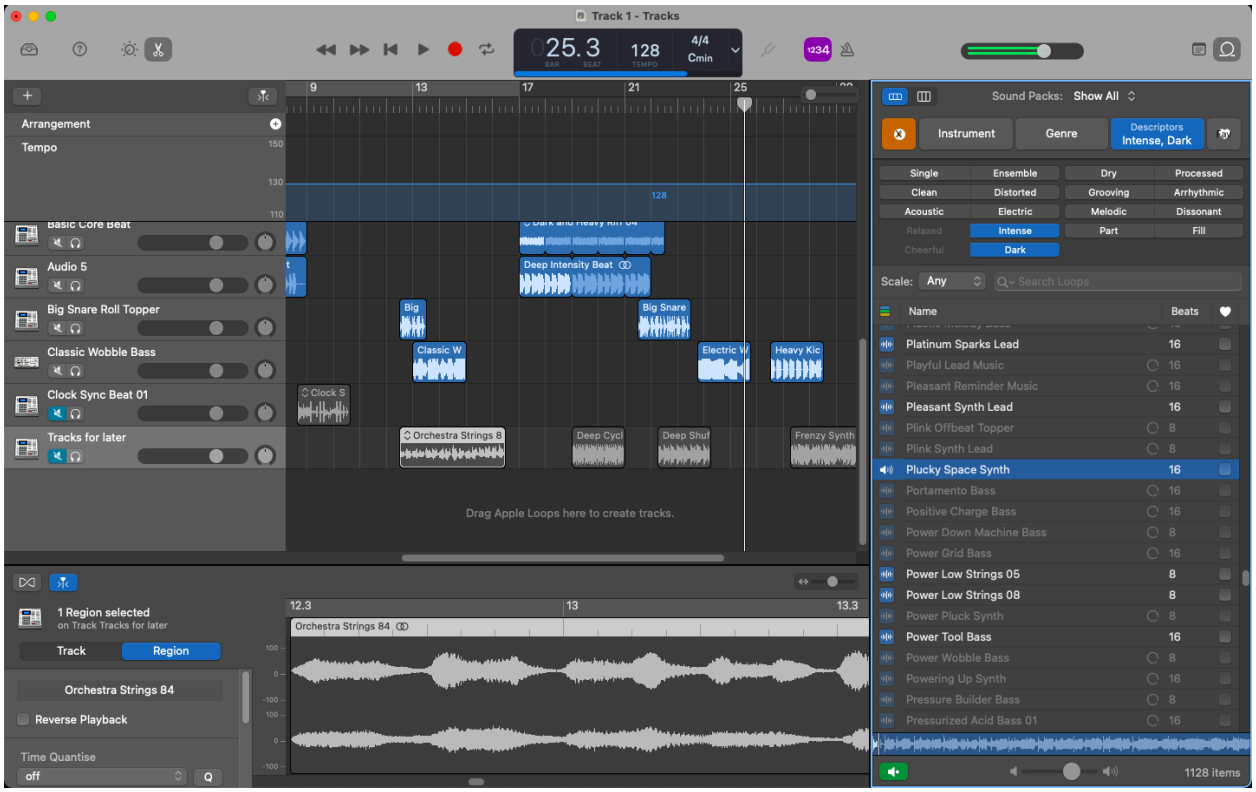
Resourcing & Capability

As a solo team, I needed to manage every aspect of the game myself. I am a very competent programmer, and I was also very confident in my ability to design the game. However, I have limited 3D modelling skills and almost no musical skills. I still wanted the assets used to be original, however, since there are marks in the rubric assigned for asset originality. I decided to attempt to use Blender and GarageBand where necessary to do my best and create the assets I needed.

I was able to make all of the robot blocks myself by using textures and height maps I made in photoshop, and while I needed to download some free enemy models, I created most of their animations myself. Partway through the development process, I started creating more 3D models. I created all three of the different turret types, and eventually built the final boss in a similar style, and I was able to create a beautiful screen for the main menu of which I am very proud.

I was able to create music in GarageBand using the loops menu, which contains short, premade sections. By cropping, combining, splitting, and adjusting the pitch, I was able to make some moderately successful tracks which I used as the main level music during the game.

For the main menu and final boss, I used creative commons music taken from Opengameart.org, but I wanted to use as much original audio as possible so for the workshop I used my own level track slowed down to about 10% as dark background music.



I used GarageBand to create the main soundtrack for the game.

Organisation & Timeline

I started well in advance of the deadline, with around 150 days before submission. However, with so much time, I worked chaotically, on random afternoons. I had a plan, but wasn't putting much effort into it. I reached 100 days to go, and I only had a basic prototype for how the game would run. From this point, I started working every afternoon, making sure that I spent time on the game every day. I was able to finish the code for constructing the robot, and for building it in a level. However, this was actually one of the largest parts of the game, and took the rest of the school term.
I reached the holidays with about 30 days to go, and started the bulk of the work. I coded all day every second day, making massive progress and developing levels, enemies, and mechanics. When school started, I spent a few hours coding every afternoon, and all day every Saturday.

Although being on my own meant that there was a lot of work to do, it also meant that I could be extremely efficient and prioritise what was necessary. I always knew exactly what needed to be done, and I was able to make decisions without conflict or delay.

An approximate timeline is as follows:

April 13: Basic prototype completed
May 4: Began working on robot blocks and workshop code
May 25: Finished workshop code and robot blocks
June 14: Began working on levels
July 19: Completed most levels
August 5: Sound effects added. Minor bug fixes and quality of life improvements
August 6: Final bug fixes implemented. Game submitted

There were many more things done than I've listed here, but they overlapped and often had minor changes much later on. During the last two weeks I made minor edits to code in all areas of the game, adjusting difficult and fixing bugs. However, the events I've listed above were the main milestones (especially completing the workshop code, which took nearly a month).

Submission Guidelines

Most of the submission guidelines were easy to follow, by not including swearing, nudity, drug use, etc. The main challenge was the requirement about violence:

Very mild, comical violence is acceptable but any violence depicted in the game must:

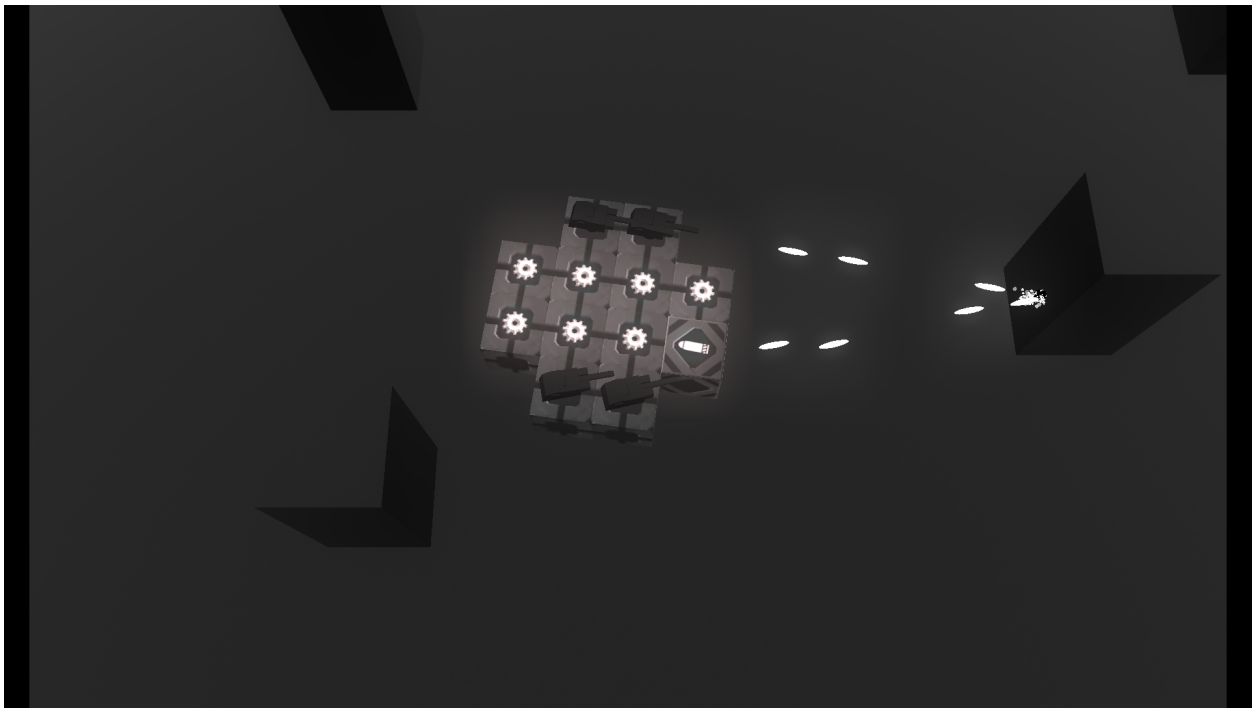
- **have a low sense of threat or menace:**
- **contain no use of visible blood or gore:**
- **be justified by context, and**
- **must not be realistic in nature, or imitate any real-life scenario**

In this game, there is a large amount of combat. To make sure that this remains mild, I made sure to convey that the enemies are robots, not people, so that the player understands that they are destroying machines. I also made their death animation a simple, non-realistic explosion, with no gore or even machine parts, so that it doesn't feel strong or realistic.

There were some technical requirements as well, but they weren't very difficult to follow either:

1. All submitted games must run in a Microsoft Windows operating system, or in an identified Internet browser.
2. All submitted games must utilise a keyboard and mouse based control system
3. All submitted games must function, first and foremost, as single-player games.
4. All submitted games must function/run independently of a need to download/install specific game development software, or additional software.
5. Submitted games should refrain from the use of store-bought or purchased assets

The game is single player and can be run from a file. It runs on Windows, uses a keyboard and mouse, and doesn't use any purchased assets. These were fairly easy to follow since they were part of the design from the planning stage, and I didn't have to change anything.



A very early test, before most of the game logic and mechanics were implemented.

GAMEPLAY & MECHANICS

Perspective

The player's perspective is based on the camera from Starcraft II. I wanted a birds eye view that didn't restrict the player, while also having a clear sense of perspective. I ended up adjusting the camera position and its field of view until the area it covered and the perspective warp matched in the way I wanted.
I then coded the camera to remain in a fixed position from the centre of their robot so that no issues would be present when the player moved or turned around.

Later in development, I changed the camera FOV to a variable I could adjust per-level, since some levels needed the player to see further. Because the camera had a fixed height, levels with varying height needed a higher FOV to properly see higher areas. I also gave the player a particularly high FOV in the final level to emphasise the size of the final boss and to help the player manage their position in the level.

Controls

I had to pick between "tank controls" and "strafing controls." Strafing controls involve directional controls without rotation. The arrow keys move the player in the direction pressed. These controls allow precision movement, but because there is no rotation, the player can be limited by certain gaps. I chose to use tank controls, in which the player can move forward and backwards and turn left or right. This made control more difficult because the player had to turn towards where they wanted to go, but it added a good level of challenge and made the game feel more realistic.

I received some feedback that it was difficult to move in a straight line, so I made a minor change to the player's movement. Now, the player will automatically align with the cardinal directions if they are moving close enough to that direction already. This helped avoid annoying zig-zagging when moving.

Objectives

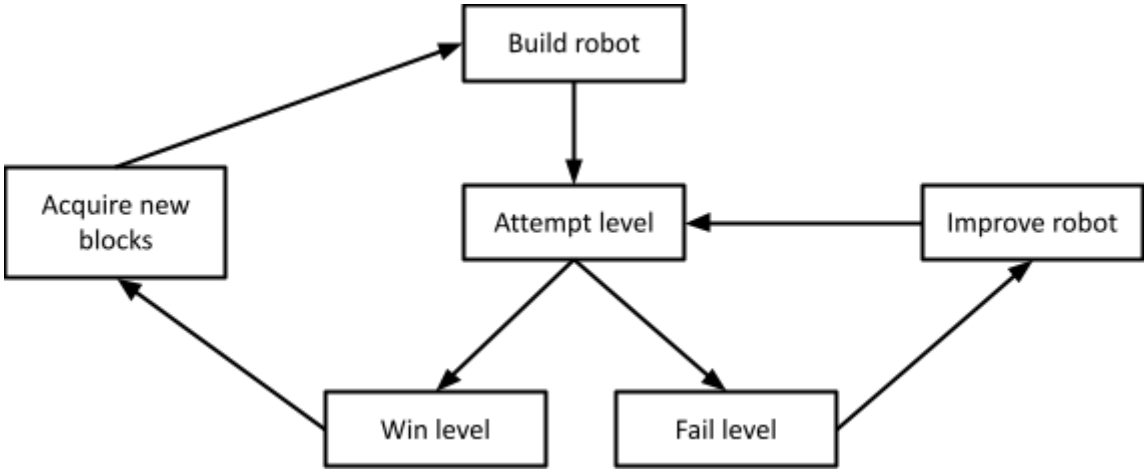
The level objective is clearly communicated in the top-left hand corner of the screen. I wanted the objectives to vary each level, to add variety to the game. However, because the objectives are often changing, I wanted to make sure that the player could clearly understand what was required at any time.

I made sure that most of the objectives involved destruction. This ranged from destroying enemies, structures, and even a boss. I also made sure that the objectives required different skills and robot designs from the player, such as speed and agility, attack power, or defensiveness. Ideally, levels could be completed using any of these attributes- the player could

speed through enemies with an agile robot or blast through them with a tanky robot. The more ways a level could be completed, the greater the player could use their creativity.

Gameplay

The gameplay loop of this game involves building a robot in the Workshop, then attempting a level with it. The player may fail the level, in which case they can improve their robot and try again, or they can succeed the level and move on to the next with new options to create their robot.



The gameplay loop for the game mainly involves trying and improving.

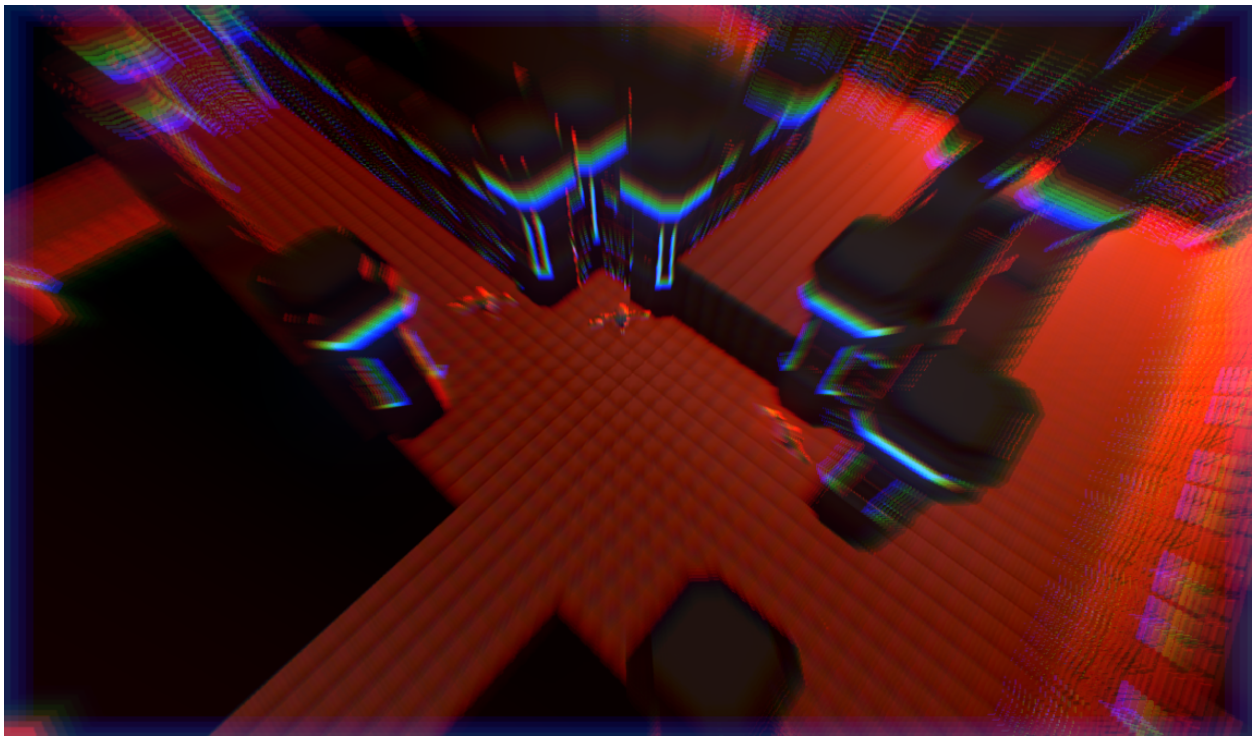
Gameplay in the levels mainly involves moving and shooting. The player can fire any weapons on their robot towards the mouse cursor (or build auto-turrets with radar). All levels can be completed with only agility and guns, keeping the game simple, but most levels require strategy and certain levels of speed or power.
Each level has a single goal that needs to be achieved, and may have additional requirements. For example, the level "Suicide Mission" involves destroying 3 data nodes, but also requires for the nodes to be destroyed by explosions.
The way that levels unfold largely depends on the player. They have the option to build a combat-focused robot and try and destroy all enemies in their path, or to play as a quick, agile robot and try to speed through levels. All styles are valid and the player can play the game however they choose.

The different levels (or "missions") in the game have different set goals. This is partly to add variety, but mainly so that the player needs to use their ingenuity to make new and creative designs for their robots. The player takes the robot they designed and tries to accomplish a goal- if they succeed, then they move on. If they fail, they improve their robot and try again. I found that ideally, a level had these factors:

- **Time constraints: A robot with fewer engines moves slower. If the level has something related to time, they have to design a robot fast enough to deal with it- and the bigger your robot is, the slower it moves.**
- **Combat constraints: There is a massive difference between a robot with one gun and a robot with no guns. Even a single enemy can cause catastrophic failure if your robot has no means of defending itself. However, when the weakest gun requires two whole batteries, going overboard on weapons can be very expensive, so the player needs to manage themselves wisely.**
- **Space constraints: If the level has a challenge requiring any kind of block or design, the player will need to accommodate this as part of their robot. This limit shouldn't be too strict, since this will prevent them from using their creativity- ideally, it should be a challenge to fit what they want into the space, but not impossible.**

If a level includes all of these, then the player is encouraged to come up with a creative and problem-solving design, and the different elements of the level will keep the player more engaged as well. The level becomes more challenging- the player can't just load up with 8 guns and one engine, then slowly but surely power through everything.

The player wins the level by accomplishing the goal set at the start of the level. If the player is destroyed, or the goal is failed or becomes unachievable, then the player loses the level.



Move too slow on level 7 and you'll regret it. Or maybe you won't. After all, explosions are pretty.

The Workshop

The game has two main parts: the Workshop, where you create your robot, and the levels, where you use the robot and try to achieve a goal. While the Workshop is not very gameplay-intensive, it is important and strategic for the player.

In the Workshop, the player creates their robot. There are a number of different blocks to choose from, and the blocks have unique interactions with their adjacent neighbours, meaning that many possible designs are available. I coded 16 different blocks, but I didn't include all of them because some blocks caused balance issues by being too powerful. The player starts with a 4x4 grid and few available blocks so that they can learn the core mechanics of the game without overloading them with information. As they progress, they unlock more blocks and can gain a larger grid that allows them to build more complex machines.

I gave many of the blocks interactions with each other to give the player more potential for their robot. The simplest and first interaction that the player learns is that Battery blocks provide power to most other blocks. However, other interactions that the player learns later include Radar blocks giving guns "auto-aim" or detonating explosives due to enemy proximity. This did add levels of complexity in the coding but I liked the different robot designs that could result from it.

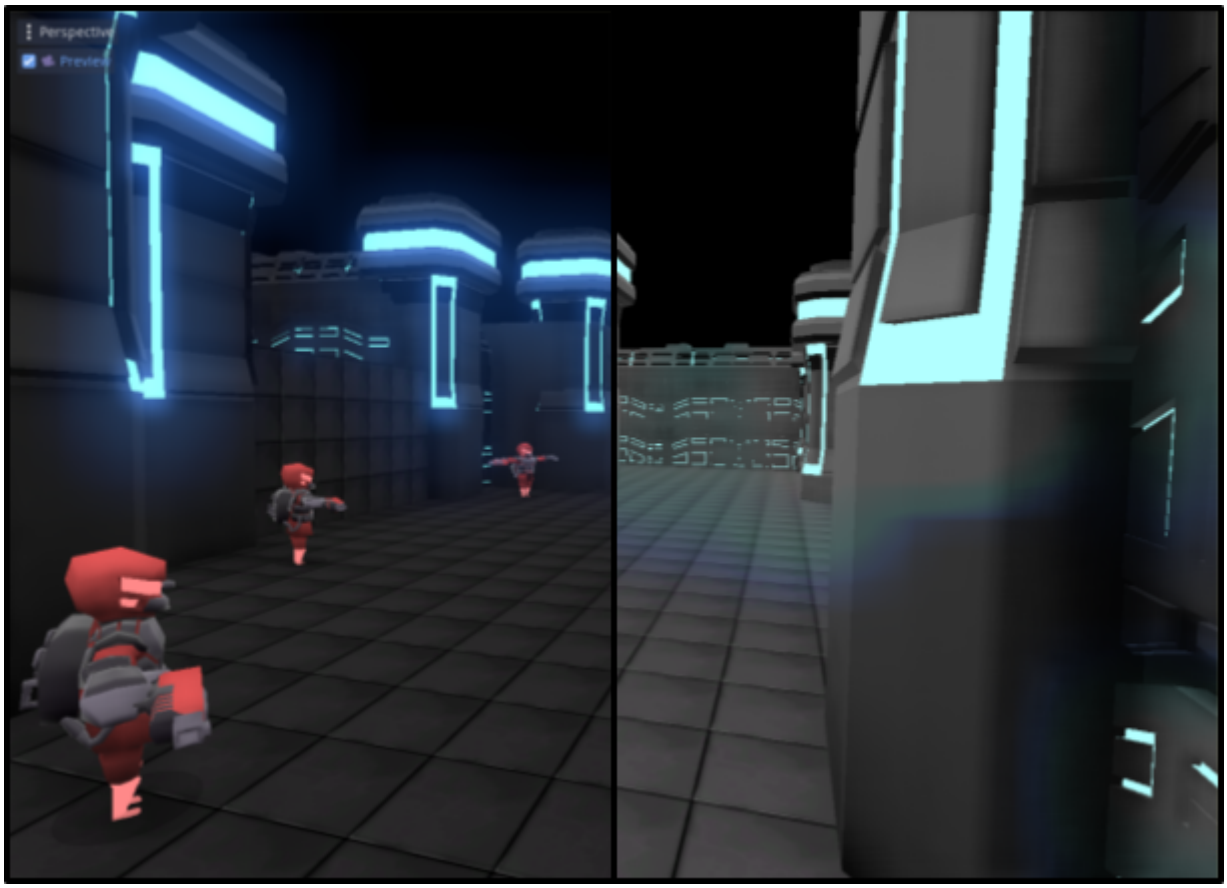
By the end of the game, the player can make extremely complex robots.

VISUAL & AUDIO DESIGN

Visual

The game uses mainly blue and red colours in its visual design. Blue and red are on opposite sides of the colour wheel, and provide good visual contrast. Blue mainly represents the player, but it also turns up in the environment. Red mainly represents hostile robots, as well as obstacles, hazards, and malfunctions.

I used a number of different post-processing effects to make the game look more realistic and coherent. The most effective things were SSAO (screen space ambient occlusion) which shaded different areas, and glow, which allowed me to make certain objects appear to emit light without the need for actual lighting nodes (which often come at the cost of lag). I also used fog, ambient light, and blurring to make the environment more aesthetically pleasing.

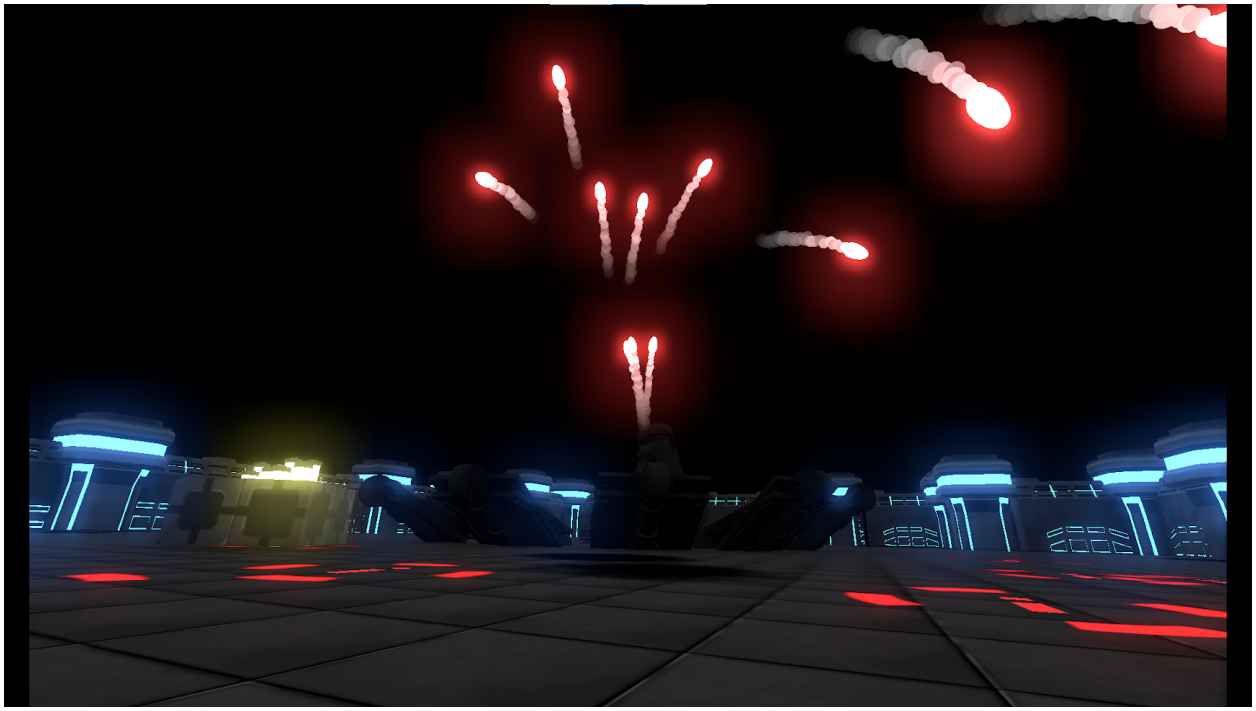


A comparison of the game with post-processing (left) and without it (right).

Audio

I am not a skilled musician or audio designer, but it was important to add audio to the game. I found some simple sounds to use for bullets and explosions, and was able to adjust them for different situations by slowing or speeding up the audio. I also found some creative commons music to use for the menu screen and final battle, but for the majority of the game I wanted the music to be original (mainly due to marks being awarded in the rubric for it). I created my own track in GarageBand (which took about 5 hours, since I had never done it before), and used it for all the levels except for the final one.

I also coded some minor quality improvements to stop tracks from stopping or repeating. During the game, if a track is told to play, but is already playing, the game will simply continue playing the track (rather than restarting it, which is the default). This means that when restarting a level, the track will continue uninterrupted.



The final boss of the game is thematically coloured red (and partly black).

BUGS & GLITCH FIXING

There were a ridiculous amount of bugs in this game. It's hardly surprising, considering just how many different elements I included- and on top of that, how many elements could interact with each other. I found bugs in the workshop for placing down blocks, interacting with blocks, turning the blocks into a design, building the design in the level, block interactions, enemy pathfinding, controlling multiple robots, splitting a robot into multiple pieces, explosion calculations, multiple cameras, certain collisions… and more.

So… There was a great deal of bug fixing to do. I quickly got into the habit of testing every change to make sure it worked, and fixing anything that was broken immediately so that I wouldn't forget. If I couldn't fix it, I'd try to accomplish whatever I was trying to achieve through a different way. I didn't always find bugs immediately, but every time I tested a level, I would build a robot to play the level with, thereby testing much of the Workshop. It was common to find a bug in an area separate to what I was testing by chance, and the only way to be sure that there were none left was to try the game repeatedly until I was confident that the game was perfect..

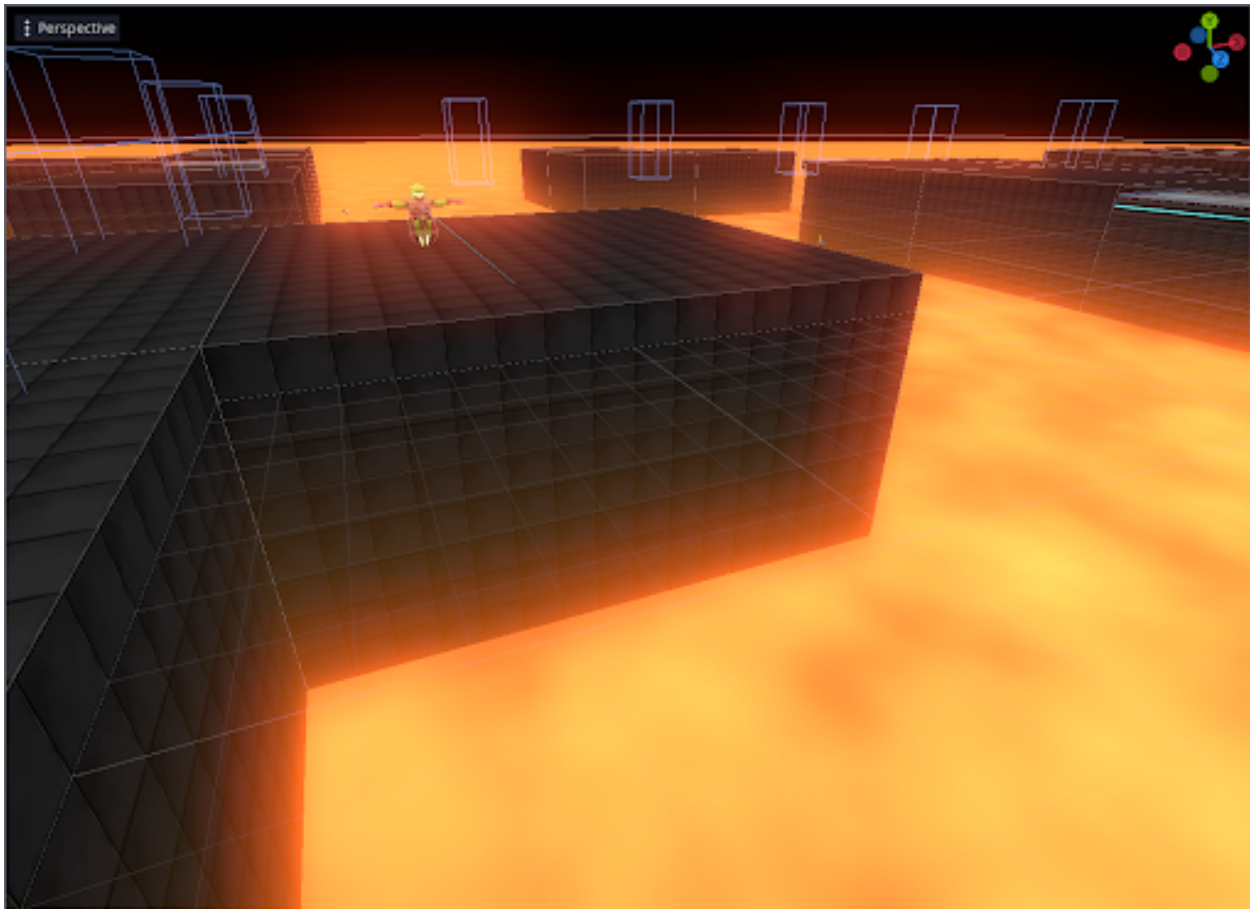
Some bugs were more difficult to fix than others. Many were simple errors and typos that, when found, could be fixed easily. Most of the time, bugs created from typos would crash the game and it would be obvious where I could find them (although one particularly nasty bug that generated from typing "==" instead of "=" had me pulling my hair out for nearly two hours as I ransacked the code searching for the solution). However, I found some errors within Godot itself that presented some extreme challenges.
For example, because the player is coded out of modular blocks, the most elegant way to code it was to make the player a RigidBody node and to make the blocks CollisionShapes. That way, the blocks would easily match with their collision. However, to damage individual blocks, this meant I had the rare situation where I needed to know which shape of a body something hit. The problem is, Godot's internal logic doesn't update the list of shapes properly if one is deleted, but the area that detects it does. This resulted in a discrepancy where one shape would have a different index between the player and a bullet, immediately crashing the game, and while Godot *is* open source, I don't have the skills to mess with Godot's code. I eventually found a way to predict and compensate for the error, so that the area would adjust the index it was using to match what the body would think the index should be (and yes, that's very wordy), which worked well. I also changed the block code so that blocks never deleted themselves, but instead simply disabled their collision, which fixed the problem and helped with some other bugs to do with the player.

I also discovered a small snippet of code that saved the game from crashing many, many times. It was:

```
if not (!weakref(node).get_ref())
```

What this did was detect whether a node that had been stored in a variable still existed. Every other method I knew would crash the game if the node didn't exist, because it tried to access a

non-existent node, but this was able to detect if enemies still had targets, if spaces in the workshop still had blocks, and if detectors still had colliders. I learned a lot of things during the process of coding this game, but this tiny line is one of the best.



Adding lava to Junkpit caused a number of problems, ranging from animated waves to dealing damage.