



University of BRISTOL

Development Team Report Parallax TD

Team Venom

Andrei Ilisei

Milan Zolota

Nikolay Nikolov

Patrick Johnston

Shayan Chaudhary

Stephen Livermore-Tozer

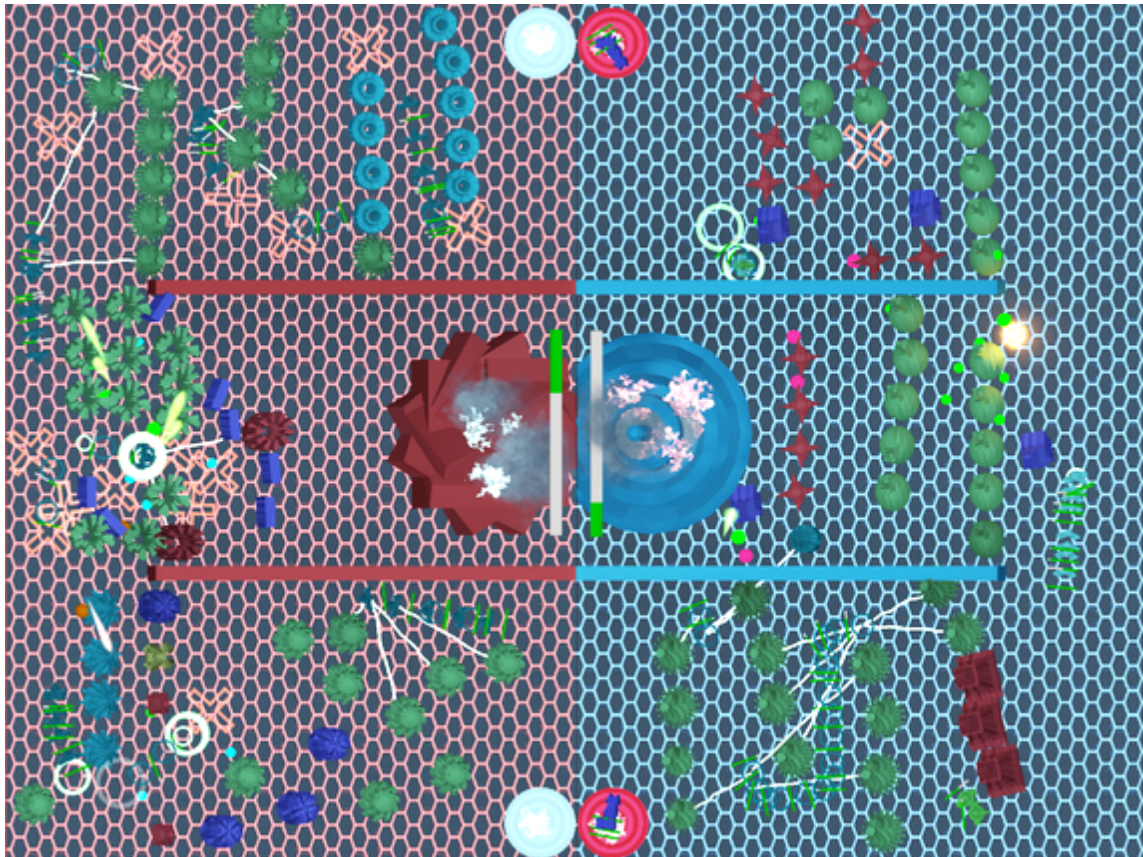


Table of Contents

[Development Team Report](#)

[Parallax TD](#)

[Team Venom](#)

[Table of Contents](#)

[Assessment — Relative weightings](#)

[Abstract](#)

[The Team Process](#)

[Project Planning](#)

[Individual Contributions](#)

[Patrick Johnston](#)

[Nikolay Nikolov](#)

[3D Modeling](#)

[Shading](#)

[Other graphics tasks](#)

[Sound](#)

[Gameplay design](#)

[Miscellaneous](#)

[Shayan Chaudhary](#)

[Application and user interface development](#)

[Appearance and Graphics for the user interface on Android application](#)

[Handling the network requests and app-game synchronization](#)

[Andrei Ilisei](#)

[Project Manager Responsibilities](#)

[Core Functionality of the game](#)

[Content/Asset management](#)

[Particle Systems and other graphics](#)

[Navigation System](#)

[Stephen Livermore-Tozer](#)

[Game Logic](#)

[Blueprint Interface](#)

[UI Design](#)

[Game Design](#)

[Artificial Intelligence](#)

[Milan Zolota](#)

[Turret tile detection](#)

[Networking](#)

[Fake clients](#)

[Software](#)

[Running the game](#)

[Extending the functionality](#)

[Software maintenance](#)

[Technical Content](#)

[Vision client](#)

[Navigation System and Pathfinding](#)

[Networking](#)

[Android Application](#)




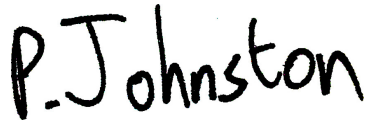


[Aesthetic Style](#)

[Gameplay](#)

[Server Code](#)

[Top Ten Contributions](#)

Assessment — Relative weightings

Name	Weight	Signature
Andrei Ilisei	1.0	
Milan Zolota	1.0	
Nikolay Nikolov	1.0	
Patrick Johnston	1.0	
Shayan Chaudhary	1.0	
Stephen Livermore-Tozer	1.0	

Abstract

Parallax TD is a competitive Tower Defence game. The Tower Defence genre is based on the mechanic of building towers to fight enemies called “creeps”; typically they are single player games where AI-controlled creeps enter a maze and must be prevented from reaching the exit — when a set number of creeps reach the exit, the player loses. Parallax TD is a “Tower Wars” type game, in which players compete directly against each other; each player must simultaneously build towers to defend themselves and summon creeps to attack the opponent team. It is by default a three-versus-three game, in which players on the same team share an arena — divided into zones owned by individual players — and hitpoints.

Another difference between Parallax TD and traditional Tower Defence games is that there is not a pre-existing maze for the creeps to traverse. Instead, players may place towers along the path of the creeps, creating barriers that may be placed to form their own maze. This ties into one of the most unique features of Parallax TD: towers are represented by physical tiles, placed onto a projection of the game world, which in turn provides a physical connection between the real world and virtual world. The projection surface can be anything, provided it has a white background; we use a slightly raised platform on the floor, as it provides the most comfortable and immersive experience for users.

Each player is given a set of tiles at the start of the game, which may be placed anywhere within that player’s zone; this is detected by a camera connected to our Parallax Vision software and a tower is built on the tile. This results in a unique experience, where players can creatively build their own maze out of our Parallax Tower-Tiles.

To the players, the main interface to the game — beyond building the towers — is the Android app, which functions as a controller. From this app, players may select a username, a level of experience and join games. When playing a game the app provides a UI, with real-time feedback on the status of the player and their team. It also accepts user input, allowing players to spawn various types of creeps to attack their enemies. The app is also used by players in the construction of towers; players select a type of tower using the app, and towers they build subsequently will be of that type.

The nature of the gameplay is fast, competitive, and tactical. Players must focus on defending against their enemies while simultaneously working together to bring them down. Some of the key strategies in the game are building complex mazes to stall creeps as long as possible, coordinating attacks with teammates to cause the most damage, and combining special abilities of the towers to maximise both offensive and defensive potential.

The Team Process

Prior to this project, most of us had already met each other and had worked together on several assignments; most significantly, in the Software Product Engineering unit of the second year. Being familiar with each other personally and professionally was one of the main reasons we decided to form a team. We were already comfortable with each other's work ethic and we knew each other's strengths and weaknesses. This led to a pleasant work environment where communication was always encouraged and everyone's opinion was respected, no matter how heated the discussions were. Due to our familiarity with each other, splitting the tasks did not represent an issue.

We divided the labour amongst ourselves efficiently and effectively, taking into account the previous experience of each member and their personal preference. Most of the team dedicated themselves to a particular area of our project's development, taking responsibility for areas such as design, maintenance, and integration. The remaining members took on more of a 'generalist' role, handling a larger number of smaller features and assisting during debugging.

At the start of the year we put a lot of focus into large, comprehensive tasks that consisted mainly in experimenting and learning the relevant technologies that we were going to use. We divided the project into three areas, within which we assigned two members: the game engine, the computer vision client, and Android controller. Once everyone started feeling confident of their required skills, our work focus shifted from the learning process to producing constructive code, mostly integrating the small pieces of functionality developed in the previously described learning process.

The development process in the first semester contained some roadblocks that slowed our output. Two of our members had a higher work-load in the first term, which reflected in their lack of time to work on the project. A team member was unable to run Unreal Engine on his computer; naturally, he undertook the artistic side of the game, creating the Maya models and other creative tasks.

In the second semester, we all became more devout to the project as our other workload decreased. Each member of the team is more productive when under pressure, which increased dramatically the rate of progress developing our project as the Games Day approached. By the end of February the core mechanics of the game were developed and we had a functional game.

The following two months were devoted to adding content to the game and making it as fun to play as possible. At this point in time all of us had specific roles within the project: Stephen extended the functionality of the game and implemented special abilities for the creeps and towers in C++, Nikolay worked on improving the models and general aesthetic aspects of the game, Shayan was in charge of the Android app, Milan continued improving the tile detection system and worked on the networking, Andrei worked on integration and content management while improving the visual of the game with Particle Effects and

Patrick worked on a large number of smaller tasks such as altering the projection matrix, dealing with the lack of support for synchronised audio for the layered structure of the sound, navigation system etc.

In terms of working as a team, we found that working in the same room was extremely productive as we could get instant feedback from each other on the features that we have just implemented. Also if one of us was stuck on fixing a bug, we were able to just move seats and try to help him.

With our private Github repository, we made good use of git to alleviate the need for making periodic backups and allowed us to work on almost any part of the project without need to notify any of the group. It also was very useful for keeping track of tasks accomplished bugs that had to be fixed and features that need to be added. Furthermore, the commit statistics from Github were particularly helpful for assessing the amount of work that each team member has put in and make sure that everybody does their fair share of work.

One of the main lessons learned from this project was that early and often integration is harder than it sounds when it comes to large scale projects. Our project had a lot of dependencies (towers needed a moving target (creep), towers and creeps need a team (for the reward system) and an AI controller, the whole game needed interaction with the tile detection system and input from players through an Android device etc.). Taking these into account (and the lack of availability of room 1.11), we decided to build the components one at a time making sure that they are fully stable before integrating them with the rest of the game. After getting familiar with the technologies involved in the project, we decided upon the class structure of our project and how they are going to interact with each other. Having this initial structure, we managed to isolate the features and be able to develop them without needing any prerequisite class. Though this strategy slowed down the integration process and delayed the initial stable release, it proved to be time-saving as we avoided implementing temporary/hacky solutions just for the sake of integration.

Project Planning

Although we had decided to stick with the paradigm of integrating early and often, we found ourselves unable to do so due to circumstances unforeseeable to us and outside of our control. Because we had frequent team meetings, however, we were able to adapt and make alternate plans to overcome any obstacles that came our way.

One frequent point of difficulty during the project came from the hardware. From the basic requirements, the hardware that would be needed to play the game included a projector, a projection surface, a camera, a number of Android devices, and a large number of physical tiles. Furthermore, we required a space with enough surface area on the ground to play the game, and a mounted projector capable of projecting on it. This provided immediate difficulties, as there were few areas that could satisfy these requirements. Initially, we intended to mount a projector onto the ceiling and have it project vertically downward onto the surface. However, after a significant amount of time spent pursuing this option, we were told before the Christmas vacation that this idea is infeasible. We were provided at the beginning of the year with a pico-projector (as a temporary solution) but we soon realised that the tile detection system needs further adjustment and calibration when using a full-sized projector (it ended up being a reliable and nearly perfect tile detection system, invariant to light or other factors).

Following this, we attempted instead to combine a ceiling-mounted projector with a mirror, allowing a horizontally mounted projector to project onto the ground. After building a prototype from cardboard and reflecting paper (which was not very successful), Richard Grafton helped us by building a mirror system which turned out to be a fitting solution of our problem. Doing this forced us to use a specific projector in the MVB, Room 1.11, meaning that all subsequent integrated testing could only be performed there. This went on to become a significant problem.

Room 1.11 is a reasonably large room, popularly used for lectures and events. This meant that for a significant amount of development time, the room was booked and unavailable for use. Furthermore, as some people wished to reserve the room ahead of time for access on short notice, the room was often block-booked for entire weeks. This was extremely detrimental to our ability to work on the game, especially as we neared completion and switched our focus from the individual components to the completed game. To overcome this problem, we had two main solutions. Firstly, we developed a set of “fake clients”, designed to emulate the Android app and vision device, giving us the limited ability to develop for and test the main game server. Secondly, we visited the MVB frequently to check whether the room was in use. Unfortunately this made it infeasible to plan meetings and work sessions in advance, resulting in a haphazard schedule and a strained development process.

At the start of the project, we were unsure exactly what hardware setup would be most effective. We spent some time initially trying to determine what projector would be the best for our game based on the size of the projection area and the resolution at which our

game had suitable graphical fidelity. This was difficult to determine, as we had not yet decided on an art style for the game or found a place where the game could be played. Eventually, however, we were settled in Room 1.11 and the difficulty of choice was removed from us.

We also spent some time trying to find a suitable camera. Our main requirements were that it have a high enough resolution to accurately detect the position of the tiles, and that it be able to pick out the tiles under the right lighting. We searched for cameras that would fit these requirements well, but in the end we settled with using a camera that we had readily available for development purposes. This camera was a Logitech C270 HD webcam, and it performed reasonably well for our purposes.

Creating the tiles was a task we deferred for a long time, as we could not be sure of their ideal size until we had finished testing the projector and camera, and they were not needed for testing purposes; during development, printed squares on paper were sufficient. The ultimate advantage of the tiles was in improving the user experience by making the towers feel like an actual solid structure. They also ensured that the paper did not become folded or creased in a way that interfered with their detection by the vision device. In order for the tiles to be recognised by the vision device, it was necessary to mark them in some way. We considered painting or engraving the tiles, but decided that it would be unnecessarily difficult, and less effective than covering the top of the tiles with the paper squares we had used previously. As the remaining requirements for the tiles were essentially only that they be solid and flat enough to not obstruct the projector, we chose to make them by laser-cutting squares of 4mm MDF as soon as we verified their optimal size.

Advancing the software to the point where the various components could be reasonably integrated took a long time. This was because the game could not correctly function in any meaningful sense until the vision device was fully operational, which was a major task in itself. Testing the Android app was also challenging, as we had access to a limited number of devices throughout development. In addition, both the Android and vision device needed to communicate using our networking system, which was also non-trivial to implement. Although we tested the networking between devices frequently, the various components that we developed could not combine to form a complete game until relatively late in the development process. This impeded our ability to test the game, especially in terms of user experience. To facilitate testing and development in these conditions, we created our “fake client” programs to emulate each of the components where necessary, and as a result integration proved virtually seamless when the time came.

Individual Contributions

Patrick Johnston

With the wealth of knowledge I have acquired from my many years of fixing errors in my compeers' and my code, I have gained the title of Lead Debugger. I have spent the majority of my game development time working with the other members for short amounts of time, hypothesising possible problems and coming up with inventive solutions. As a result, I have been involved in all aspects of the project and have a broad understanding of the project.

An example that took a particularly large amount of effort regarded the display of the arenas, which is implemented as a split screen of two separate arenas (one for each team), each of which has a dedicated camera to display it from different angles. Combining the output of the cameras left thick borders around the display, the development engine only supported scaling that maintains the aspect ratio. It seemed fitting to stretch the outputted display of everything in the appropriate axis, but strangely, there was no readily available method of doing so in the API. The implemented modification involved overriding a family of functions in the development engine that handle the viewport in order to change the view matrix and update the associated view and projection matrices; a change that required hours of research to discover a relevant overridable function, required my newly acquired knowledge from the graphics unit and object oriented programming.

I had written a basic system for managing the playing and manipulation of the background level music in advance of receiving the final music tracks from the composers. This was done before receiving the music, as the union of our development team with the composers was late in the development lifetime, and would leave us with limited time to implement any remaining changes and perform any tests. The system was implemented with the embedded scripting language of the development engine. Importing the music into the system seemed successful at first, but during the testing it was discovered that the separate layers of the music would fall out of sync. I did some experimentation and much research to discover that the engine had no functionality for keeping different audio cues in sync or even for tracking the position of the currently playing audio cue. The necessary change then was to adapt and partially rewrite the system to use an external API. This learning process included: researching for an appropriate API with an acceptable license, learning to use the API from scratch, translating the code to C++ for the chosen API, linking the library into the project via the complex build system; all of which had to be accomplished in the space of a day in order to perform a reasonable amount of testing.

Nikolay Nikolov

3D Modeling

One of my main responsibilities was creating the 3D models used for creeps, towers and bases. This was done in Autodesk Maya over the course of two months. Due to the high amount of content in the game (30 towers, 24 creeps and 2 bases), I had to ensure that all models look good and follow the general theme without being the same.

Shading

After creating the models, I was responsible for shading them. Unlike the modelling part which was done in Maya in which I already had some experience, this part was done all in the Unreal Engine which I had never used before. This part was mostly done in about a week, but I kept constantly improving on it throughout the rest of the development.

Other graphics tasks

I was also assigned a number of smaller graphics-related tasks. For example, I designed the lobby menu and came up with the theme which is used in some parts of the project (e.g. the poster, the slides and the loading screen of the mobile application)

Sound

In terms of the sound aspect of the game, I carefully considered what music would fit the theme best and explained to the composers with whom we worked what genres I thought would be appropriate and compiled a playlist of songs which illustrated my expectations. After several meetings with them, they sent us stems (i.e. individual audio tracks) of the song that they wrote which I then arranged and mixed down in order to ensure that the final result fits the game and the physical set up of the room. The research part was done over the course of a week and the sounds were mixed down in two days.

Gameplay design

In addition to the artistic parts of the game, I was also heavily involved in the gameplay design. After proposing the six-class structure to my teammates, I went on to curate a list of special abilities for towers and creeps which I then assigned to the classes according to their types and skill levels. Although the actual design of this stage was done in less than a week, it required a lot of research before hand which consisted in playing Tower Defence (or similar) games which was rather time consuming.

Miscellaneous

Although I was responsible for a number of tasks, I did not do solely individual work. I attended all team meetings and participated in most discussions about the different aspects of the game such as the game play and the general structure. In addition, I helped with testing by both running the game individually on my machine to determine the technical correctness of the features and by playing the game with my teammates to ensure that it is enjoyable and balanced.

Shayan Chaudhary

Due to my past experience in Android development, it was decided early on that I will be solely responsible for developing the android application which would serve the purpose of controlling the game.

Application and user interface development

My primary task was to design and implement the Android application. For this, I was required to design an easy-to-use navigational flow and interface. The user interface and navigational flow of the application finalised by the team was complex and the stock Android building blocks were not sufficient to build the GUI that was required, so this required me to do some extensive research into this topic. Furthermore more research was needed to maximise the experience for all devices and users. This added an extra layer of complexity as it meant that the application needs to adapt to various screen sizes, resolutions and Android operating system versions.

During the lifespan of this project, I have implemented and proposed a few user interfaces, some of which were discarded as the idea sounded good in practice but once it was implemented it did not feel right or we came up with a better way of doing it. This led to a fair amount of the work being discarded but it was not a full waste as the best features were brought forward and the design kept on evolving to the last minute. All the underlying mechanics were implemented in Java using Android's development tools – Android Studio.

Appearance and Graphics for the user interface on Android application

I also took upon myself to design the majority of the graphics for the user interface. This way, it would be easier for me to change content to exactly match the GUI elements. Alongside actually creating the designs, I wrote various classes ranging from creating custom buttons and dialogue boxes, to adding sound effects, animation and providing haptic feedback where needed to improve user experience of the application. I mainly used Gimp, Inkscape and Photoshop to create the graphics from scratch as I had picked up the relevant skills during the Web Technologies unit.

Handling the network requests and app-game synchronization

Alongside the user interface, I also spent a huge amount of time working on handling network requests. Not only is the app receiving messages, it will be also sending the appropriate user response/input back to the server. Therefore I needed to make sure that the app stays in sync with what is happening in the game and on the server. Furthermore, I needed to check that I am handling all the issues that could occur during gameplay for example, the app being minimised or manually closed, which changes the state of the app and loses the connection to the server. So on restarting the app, I should be able to reconnect to the existing game, load the appropriate content and continue where I left off. I needed to cover all of these cases which actually turned out to be a huge time consuming task.

Andrei Ilisei

Project Manager Responsibilities

After we formed our team I was elected (by my colleagues) as Project Manager. Having this role, I had to constantly keep track of our progress throughout the project, checking which tasks have been accomplished, assigning new tasks and setting the priority of the features that still had to be implemented. Furthermore, I researched potential options for the setup of the game and discussed their feasibility with Richard Grafton. After reaching an agreement I acquired most of the props for the game (tiles, duvet cover as projection screen etc.)

Core Functionality of the game

At the beginning of the project most of the features that we wanted to implement needed a some core functionality that they could rely on. In order to be able to start developing them as soon as possible I worked with Nikolay and Stephen on building the base mechanics of the game (defining the main classes of the game and their interaction).

Content/Asset management

One of my main responsibilities was content management. I imported and integrated all the Maya models and tested them to make sure that they worked as expected when combined with the already existent back-end functionality; this ended up being more time consuming than I expected as we kept changing the characteristics and also because the models came with a lot of (hidden) preset options that did not fit with the mechanics of our game. On the other hand this gave me the opportunity to experiment with different features and have a broader understanding of the Engine.

Particle Systems and other graphics

I was in charge of making most of the Particle Systems and graphics of the game (fire, smoke, electric bolts, spawning portals, explosions, gun firing flame etc.) and embedding them within the assets already existent. I have also developed the complementary Blueprint scripts that trigger them at the right moment in the game in order to match them with the actual gameplay events (fire spreading on a base when it loses lives, portals from which towers rise, waves of damage from tower projectiles). The hardest part was finding the right balance between performance and visual quality, while tweaking them to look sharp on the projector.

Navigation System

I worked with Patrick on developing a system that identified if a newly placed tower would obstruct the path or not. The solution that we found was quite simple and easy to implement but it was the result of many failed attempts.

Stephen Livermore-Tozer

Most of my experience and ability in software development is focused on developing, expanding, and maintaining large codebases. To make the best use of this, I assumed the role of Lead Programmer for this project.

Game Logic

My primary task was to design and implement the game logic. All of the underlying mechanics of the game were written in C++, as compared to Blueprints it provided greater speed, better matched my own experience, and was better suited to create a large codebase with several layers of abstraction.

The overall span of the code included handling the in-game logic, the menus and game state transitions, and the interface between the game and the netcode. The code was written very tightly coupled with the Unreal API and makes use of many of the offered features.

Blueprint Interface

To enable Blueprint content generation I developed an extensive Blueprints interface that provided access to relevant variables and events. This gave our content developers the ability to program complex behaviours with access to data and low level functionalities within the game. I also worked with the content developers to provide assistance in implementing the required behaviours, and wrote much of the Blueprint code.

UI Design

I worked heavily on designing the user interface for the game server, using Unreal Motion Graphics to create interface widgets. This included creating interactive elements, such as buttons, and HUDs featuring and displaying data, updated in real-time.

Game Design

I worked as one of the lead game designers, helping to plan gameplay mechanics and design content. This role took greater focus earlier in development, when the overall had yet to be fleshed out. After the game was mostly designed and being implemented, I continued to design various in-depth game mechanics, including the basis of special abilities. Later on I contributed to game balancing and fine-tuning the content to ensure a high quality of overall game experience.

Artificial Intelligence

I designed and implemented behaviour trees using the editor provided by the Unreal Engine. These behaviour trees define the actions taken independently by creeps and towers, and are created using a graphical editor. They were designed to be easy to understand and expandable, allowing new developers to easily create their own new behaviours and combine them with the existing ones.

Milan Zolota

Turret tile detection

Using physical objects to interact with the game is the most distinguished feature of our game. Because of that, in the beginning of the game project, it was required to create a prototype detection program that to determine whether it was possible to detect the tiles perfectly (i.e. no false positives and negatives). This prototype was created using OpenCV, a camera and a pico projector which was used to project various images on top of black squares to simulate what it would be like with a real projector and tiles. This prototype was good enough to determine that such detection is doable and would be reliable enough.

The final version of the detection program is very reliable and works perfectly(i.e. no false positives and negatives) in various lighting conditions. It also includes a perspective transform so that only the projected screen is used for detection instead of the whole frames recorded by the camera. The program is also able to find the projected screen in the frames and therefore the perspective transform is calibrated automatically before every round of the game.

The first prototype was finished within a couple of weeks after we were given a pico projector and a camera to prove that such detection was doable. After that, it was improved and adjusted to the changes in the game throughout the whole development of this game.

Networking

After the detection prototype was finished and the Android app and the game itself became usable, communication between these was needed. The networking was done using the standard Socket library for the Android app, and using the Winsock library for the game, the detection program and the fake clients. The networking is very stable and allows players to reconnect their Android app in case the connection to the game is broken or their phone restarts, etc.

Networking had to be developed throughout the whole development of the game, especially when new features that required communication between the game and the Android application were added to the game.

Fake clients

Because the game requires six players to be connected with the Android client and the camera mounted and the detection running, testing the game would be really difficult. Because of this, fake Windows console clients were made that could be used to simulate the vision detection program and Android clients and the communication between them which made the development process and testing much easier.

This task was finished in a couple of days, but the clients had to be modified every time there was a change in the networking protocol.

Software

Running the game

After the necessary setup of the various components of the hardware (projection system, camera, local area network), the software user simply needs to launch the game, click the 'Start game' button, launch the computer vision client and he will be taken to the lobby. From here, the six players launch the app on their Android phones, choose their class, and when ready, touch the 'Ready' button, at which point, the game commences.

Extending the functionality

We established the scope of the project very early on; namely that the primary game engine and computer vision would be run on Windows and the controllers would run on Android. For this reason, cross compatibility was neither aimed for, not achieved. It stands that extending the functionality of the project to become compatible on Linux or home consoles will require the translation of certain sections in the core of the code. Extending the functionality of the controller to be supported on iOS or Windows phone will also take a considerable effort.

Software maintenance

Due to the object oriented design of the product, different feature in the code have been abstracted which makes it possible to freely rewrite their internal logic without causing inconsistencies in the rest of the code. Most importantly, the C++ codebase exposes a wide range of events and variables to Blueprints. This means that game content can be created or modified through the Unreal Editor without having to modify or recompile the code. The content exposed to Blueprints have been carefully encapsulated to ensure that changes made do not interfere with the fundamental game logic.

Technical Content

Vision client

Using physical objects to play a digital game was one of the main distinguishing features of the game from the beginning. Because of that, a way to detect physical objects placed on top of the projection and in-game use of their positions relative to the projection was needed. This detection needed to be extremely reliable so that it would not result in using any false positives or negatives. It also needed to be very precise so that the in-game towers would be built at the same position and therefore projected exactly on top of the physical tiles.

Using Kinect or a similar camera to detect tiles using depth was out of question because the tiles would have to be too tall and produce a shadow on the projection. Some experimenting was done with a tweaked infrared camera (PS-Eye with lenses from a Logitech C270 HD) but no significant improvements over a regular camera were achieved. In the final set-up, a regular webcam (the same Logitech C270) and OpenCV were used.

This approach is interesting and novel because instead of using a device that provides us with data that can be used straightaway, we designed our own system that extracts the required information from a video from a camera. It was also challenging because it is not an easy task to create a perfectly accurate system.

To even begin trying to detect turret tiles and their positions relative to the projection, detecting the projection in the video recorded by the camera was necessary. This was done by:

1. Converting the newest frame to a HSV matrix.
2. Using a HSV filter to filter out specific HSV ranges that should not occur in the projection.
3. Finding the biggest contour in the result from 2.
4. Trying to approximate a polygon around the contour.
5. If there are 4 corners in the approximated polygon, they are the corners of the projection in the frame from the camera.

During this stage, we only project white colour to make the second step easier.

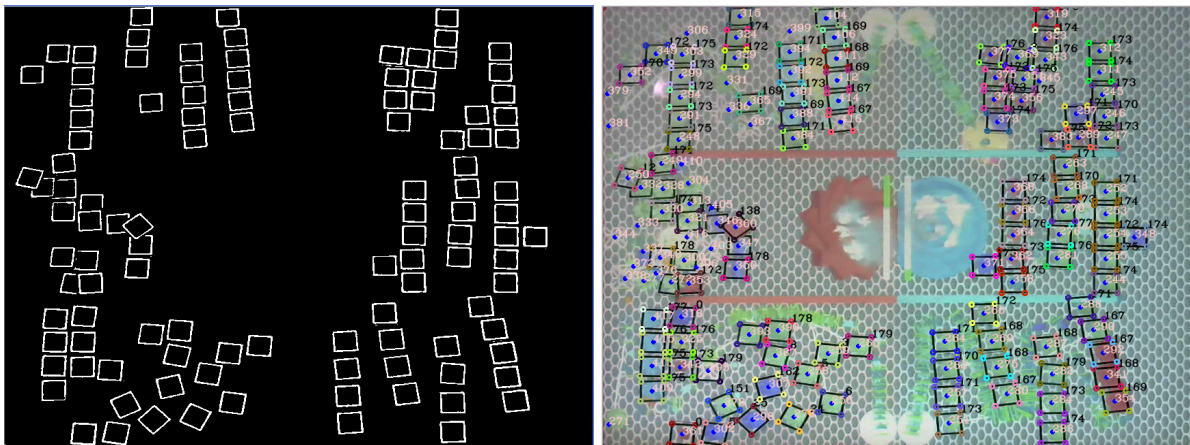
After the projection is found in the video, we can proceed to detecting turret tiles and their position relative to the projection, within the projection area in the frames. To detect the tiles, they needed to have some specific features so that they could be identified easily and these features needed to be visible even with the game projected on top of the tiles. With all this considered, it was decided that black squares printed on white pieces of paper that are stuck to the tiles would be suitable. The main reason for this decision is that the back part of the paper will reflect significantly less light than the white part and therefore the projection will not interfere with this feature. On top of that, there aren't any black squares in the game either.

The detection of the black squares stuck on top of the turret tiles was done by:

1. Fixing the perspective of the projection area so that it's rectangular and the ratio of the sides matches the ratio of the actual projection's dimensions.
2. Using a HSV filter to filter out bright areas from the area with a fixed perspective.
3. Finding contours larger than a specified size.
4. Approximating a polygon for every contour found.
5. If there are 4 corners in the polygon, they are, most likely, the corners of a tile in the projection screen.

Having found the corners of every black rectangle in the projection, calculating the centres of the tile and their rotation is trivial. These positions and rotations are then stored and updated(tracked) on every frame. If a centre of a tile did not move for a specified amount of time, it was considered as a turret that was placed and would not move again and its position relative to the projection area and rotation were sent to the game for processing.

This process with some additional processing, e.g. to remove false positives, resulted in a very reliable and precise detection system under various lighting conditions.



Navigation System and Pathfinding

A simple and lightweight system that detected if a new placed tower completely obstructed the path of the creeps was needed and none of the built in functionality of the engine offered that without actually obstructing the path for a few frames.

This was implemented by having a simplified copy of each arena under the actual arenas and using collision boxes that simulated the towers.

Whenever a new tower was placed on the floor, the game follows these steps:

- 1) Spawn collision box on the mirrored arena
- 2) Wait for the Navigation Mesh to be rebuilt (0.1 - 0.4 seconds)
- 3) Use a simplified creep to check if it has a path to the destination.
- 4.1) New tower did not obstruct the path:
 - 1) Destroy collision box
 - 2) Spawn actual tower (with a collision box high enough to pass through both real, and simplified arenas in order to reduce the total number of objects in the map).

4.2) New tower obstructed a path:

- 1) Destroy collision box
- 2) Spawn a placeholder that shows that the current tile obstructs the path.

The actual challenge in this task was the fact that the pathfinding algorithms from the engine are not perfectly precise and we had to experiment with a lot of parameters in order to be sure that a creep will always have a path to its target.

Networking

Because the game is run on a tower computer, every player plays the game using their own Android device and the vision detection is run on a different computer for performance and flexibility reasons, there is a lot of communication between these devices and this communication needs to be efficient and stable.

The networking was done using the standard Socket library for the Android app, and using the Winsock library for the game, the detection program and the fake testing clients.

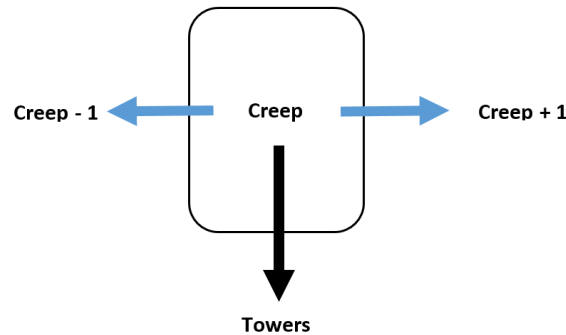
To make the communication stable, our own communication protocol was implemented. This protocol defined how the messages between the devices had to be structured, what the handshake had to be, etc. The protocol also allows players to reconnect their Android app to the game in case the connection is broken or their phone is restarted, etc.

The networking is also always non-blocking, or run on an entirely separate thread so that the main thread isn't slowed down or affected in a bad way if there are communication issues between two devices.

Since it had to be lightweight, each message is only five bytes long. The first byte represents the ID of the command being sent or received and the other 4 bytes being contain various parameters corresponding to that command.

Android Application

The graphical user interfaces we designed on paper was too complex to build using the stock components provided by Android software development kit. Our initial design required us to implement a 4-way swiping window, where swiping top/down would switch between towers and creeps, and swiping left and right would switch between different creeps/towers. This is illustrated in the picture below.



From stock Android components we had a viewpager which only was only able to swipe horizontally. It did not support top/down swiping as its intended use was to swipe between pages. To solve this problem, we used fragments and implemented our own simple gesture recognition which detects left, right, top and down gestures on the Android screen and display the correct fragment view. Alongside this, we created transition animations for swiping to give it an appropriate look and feel, instead of the default non animated instant change.

However, when we fully implemented and tested the application, it was decided by the majority of team that there is too much swiping going on and it does not feel right. So the new evolved graphical user interface required only left and right swiping to separate tower and creep fragment views, where for each fragment we needed to dynamically generate a grid list of thumbnails depicting x numbers of creep and towers for the chosen class and within each thumbnail show appropriate details, images and statistics about the creep and tower. The generation of these stacked views was all done programmatically of course, as the number of creeps and towers varied for different classes. This made the application very flexible, so adding new content and making changes to current towers and creep all of the sudden become a very simple task.

In addition to designing the user interface, we had to make sure the application is supporting on multiple devices of different screen and resolution sizes. The aim was to make sure the GUI was responsive and adapts to the variety of devices android could operate on. The user interface of the in game activity is show below.



Aesthetic Style

During the early stages of planning, several possibilities for the art style of the game were considered, including fantasy, medieval and military themed approaches. In the end, a polished futuristic sci-fi style was chosen in order to fit the low light conditions needed to ensure that the projection is indeed visible. In addition, this choice complemented the physical approach of the project.

However, the projector which was provided for us turned out to be problematic. After testing the initial concept designs with the system, it was discovered that the quality of the picture was poor, both in terms of resolution and colours. In order to compensate for this unexpected problem, the overall theme of the game was changed to an 80s-90s-inspired arcade sci-fi style. This way the low quality was used to add to the aesthetics of the game rather than reduce its appeal. Several methods were used to achieve this style.

To begin with, the 3D models for the creeps and towers were designed using mostly straight lines and perfect circles and a low level of detail in order to emulate the 8-bit graphics of arcade games from the 20th century. Despite the retro approach to the graphics, all content was still made 3D in order to give the game a slightly modern feel which resulted in a well balanced mixture between old and new in terms of aesthetics.

Moreover, working in three dimensions opened possibilities to use Unreal Engine's powerful capabilities such as particle effects. They were mainly used to enhance the dynamics of the game and highlight specific events (such as spawning creeps, building towers, towers shooting, bases being attacked or explosion waves generated by area of effect projectiles). Most of them were made from scratch, but for the ones that needed a higher level of detail and a more realistic look (fire and smoke), content available in the engine was adapted and integrated in the game.

In addition to modeling the characters in the game in a 2D-like fashion, the content was shaded using basic procedural materials in order to keep an almost single-colour look while still being able to show a certain level of depth to the user. This ensured that when viewed from above, the game gives a feel that the player is physically above the battlefield which contributes significantly to one of our game's main aims, i.e. being able to interact with the game in real life rather than simply using a mouse and a keyboard.

The main game world is not the only visually represented part of Parallax TD. The graphics interface of the mobile app was designed to fit the overall art style. However, it has a slightly more sophisticated and modern approach to it. This is due to the fact that the Android devices used as secondary controllers have a higher, or at least similar, resolution than the projector despite the large difference in screen size which means that going for low level graphics would have been a waste of hardware. This led to a compromise between keeping to the general game art style and appropriately using the phones' high quality screens. This was achieved by designing a clean and simple sci fi based interface which looks polished while still maintaining a vintage feel.

In order to further solidify the atmosphere created by the visuals in the game, a main theme was tailored to the game. This was achieved in two stages. Firstly, a playlist with inspirational songs was made. This included tracks of different electronic genres from the 90's such as techno, electro, IDM and ambient. This helped the composers understand the atmosphere intended for the game, i.e. a rugged, off-beat, mildly intense feel.

After receiving stems (i.e. individual files for different layers) for the song written by the composers, they were arranged in a way which was then programmed to match the game's progress. In addition, some post processing such as EQing and effects was applied to the tracks in Ableton Live in order to achieve the desired final sound. This was a particularly important part because a pair of near field monitors is used in the game which requires careful adjustments to the sounds in order to achieve an appropriate balance between the individual elements in the theme. Although mixdowns are not actively considered by the users while playing the game, the alterations to the score had a significant impact on the final atmosphere and therefore increased the extent to which the music helps the players feel connected to the game.

This is a screenshot of the arrangement in Ableton:



As the Unreal Engine's built in audio functionality proved to be unsuitable for synchronised playback of multiple audio tracks, this feature was implemented in irrKlang which is a C++ library, which provides the necessary functions.

Gameplay

Although the technology used in this project is essential to the overall gaming experience, it was not the only aspect of the game considered to be of high importance. The gameplay was also designed in a way which makes use of the image processing component and ensures that the users enjoy themselves which is ultimately the goal of any game.

In terms of the foundations of the gameplay, i.e. the main objectives and player abilities, a Tower Wars-like approach was chosen due to the increased dynamics of the mode. There was a rather lengthy discussion about Tower Defence games and the more experienced players in our team all agreed that classic TD games can sometimes feel too static and unengaging. In addition, these games tend to last for over forty minutes while this group project requires the play time to be around five minutes. Therefore, a more dynamic and fast-paced style was achieved by adding the ability to attack the opponent rather than simply defend.

The 3 vs 3 multiplayer setup of the game is also rather important. As mentioned in the previous paragraph, adding a competitive side to the game made it more interesting and dynamic which is augmented even further by the physicality of putting down the towers. Adding teams was done because at its core the game is based on strategy and introducing the possibility for team work further increased the game's ability to be enjoyable even after playing a few rounds. In addition, letting the players team up encourages them to engage with each other which moves their focus to the game rather than the phones.

Another important aspect of the gameplay is the content. While some of the team members had played a lot of Tower Defence games, some had never played anything similar. This gave us an insight how players at different levels feel about such games. The conclusion was that there is a rather steep learning curve which led us to the design of some basic and easy to play with creeps and towers. However, the more experienced members of our team found playing the game with that content too easy and simple. Therefore, a class system was implemented - each player is given a choice between six player classes where each class has its own creeps and towers. Some classes are aimed at people with no Tower Defence experience, while others are more suitable for more skilled players. This ensures that the game is enjoyable for everyone. In addition, different classes have a different balance between towers' and creeps' strength which increases the game's replay value.

Ensuring that the classes are suitable for their respective skill sets was done by adding different types of towers and creeps to different classes. For example, the beginner classes have only two types of tower attacks - classic and splash (i.e. towers which do damage around its target). Due to the lack of special synergy between the towers, these classes do not require any advanced tactics and are suitable for beginners. The other classes, however, have a more complicated structure and require some more thought to reach their full potential. Although this means more thought is involved, these classes are

still basic enough so that someone who has never played Parallax TD before can come up with the main strategies needed to perform well after simply looking at their class.

Since the game's enjoyability is dependent on being able to fairly compete against the opposing team, balancing all the classes was essential which was achieved by simply playing the game numerous times and observing when certain creeps and towers are over- or underpowered. This was particularly helpful because it also allowed us to search for bugs and determine how fun the game is.

As mentioned in the abstract, Parallax TD differs from most Tower Defence games because it lets the players build their own mazes using the towers rather than prebuilding obstacles and restricting the players' freedom. Although this does slightly complicate the game, it also makes it significantly more enjoyable because it lets the players profit from the physical interactions and it opens up more possibilities for creative placement rather than arranging the towers in a dull and procedural manner.

Server Code

The game was developed in the Unreal Engine, which introduced a large set of advantages and drawbacks. To make the best use of UE4 the codebase was tied tightly to the Unreal API, adopting the engine's style and coding practices. With this in mind we created our custom classes, structs, and enums using the unreal macro specifiers, allowing them to interact with the higher level features of UE4. Furthermore, all of our classes were descendants of the UObject and AActor classes, making them fully-fledged Unreal objects with access to all the features of the engine.

The design of our game was highly modular, with different regions of code being mostly isolated and accessed externally through an interface. The advantage of this is that each region is easy to keep internally consistent, and may have its internal logic completely rewritten without needing to alter external code. This was an important part of the early development process, as code frequently needed adjusting as the requirements for the game grew.

To handle the overall flow of the game application, classes based off of the Unreal actor class AGameMode were used. This class is responsible for starting and stopping games and giving a high level view of the game's logic. We used these classes to create logic for the main menu, game lobby, and pre-game setup, as well as the game itself. These classes are ultimately responsible for everything that happens in the game, as they set up and manage the other components of the game - as well as handling their own transition logic.

Each of these game modes is associated with an Unreal level that defines an external interface. In the case of the main menu, pre-game setup and lobby game modes, a specific level is used for each. These levels include UIs created using the Unreal Motion Graphics editor, which provide buttons and display data.

The in-game game mode is not associated with a specific level, but instead requires the level to include a set of actors. These actors mark specific points and areas on the map, including the team and player zones, and the spawn points of creeps. These actors are detected by the game mode, and used in the game logic.

Communications with external devices are handled by two classes, each descended from `AInfo`, an actor class that has no physical representation in the game world. One of these classes handles all interaction between the server and the Android devices, `APlayerConnectionHandler`; the other relays communication between the server and the vision device, `AVisionDeviceReceiver`. These classes provide a complete abstraction of their respective external devices to the server, providing a consistent and simple interface to the game logic.

Part of the job of the vision device receiver was to provide the server with meaningful tower coordinates when towers are detected. Doing this required conversion between the real-space tower coordinates and the in-game tower coordinates. This was accomplished by using data read from the game level, as explained above. A series of transformations, using the real-space team coordinates and the in-game team and player coordinates, are able to perfectly transform coordinates between real-space and game-space.

Most of the actual game logic is handled by a set of actors representing independently acting agents within the game world. In addition the class `ABattlefield` is a manager class which tracks the towers and creeps within a single team's play area, as well as that team's stats. These battlefields are managed by the game mode, linking the specific game logic to the high level control class.

Each agent acts primarily through their Tick function. Tick functions are a fundamental part of Unreal's actors, and are called every frame that the game runs. This makes them useful for handling continuous actions, which include motion and event triggering. In the case of some actors which are dependent on others, such as debuffs - which attach to creeps and repeatedly activate some effect - the default tick function is not used; instead a custom tick function is called by the actor on which they depend, to better synchronize their logic.

The creep and tower classes, `ABasicCreep` and `ABasicTower`, are abstract `UClasses`, meaning that they cannot be spawned directly in the world themselves. Instead, they are meant to be inherited from by a Blueprint subclass; effectively, they provide an interface to the game logic for content developers. This allows content developers to use the Blueprint graphical interface to create towers and creeps with their own models, animations, and other components. In addition they provide an extensive set of `UProperties` and Blueprint implementable events; this exposes an interface to the Blueprint class that allows developers to define new, complex behaviours and redefine certain existing behaviours without interfering with the underlying game logic.

Top Ten Contributions

- **Computer vision (tower detection)**
Tower tile detection with automatic calibration of detection of the projected arena.
- **Multiple platform networking**
Local area networking between the development engine, the computer vision system, and multiple Android clients.
- **Procedurally arranged music**
Multiple layers of audio are manipulated with respect to the health of each team's base.
- **Performance optimisations**
Optimised game logic that supports hundreds of creeps, towers and projectiles acting independently in parallel.
- **AI, navigation and pathfinding**
Automatic regeneration of navigation data when a tower is spawned.
- **Android app**
A small, lightweight Android app that needs no configuration.
- **Open to all varieties of players**
Beginner or expert, young or old; all are a viable audience for our game.
- **Reduced testing opportunities**
As our game has a specific hardware setup, its integrated version can only be played in room 1.11 which had a limited availability. This significantly impacted our ability to test the game.
- **Gameplay**
The game provides an enjoyable and team-oriented environment with a sufficient amount of content to maintain its quality over time.
- **3D modeling**
56 distinct three dimensional models (30 towers, 24 creeps and two bases) were created for the in-game characters.