Team: Bear Witness

Game: Geometa

Multiplayer Top-down Shooter with Shape Mechanics and Procedurally Generated Arenas

University of BRISTOL

Team Members

Demyan Hibbard : 2230128
Daniel Lovell : 2262181
Farid Bin Mohd Nizam : 2210975
Jacque De Boehmler : 2267754
Josif Trenchovski : 2305723

Signed Declarations

Demyan Hibbard : DEMYAN HIBBARD

Daniel Lovell : DANIEL LOVELL

Farid Bin Mohd Nizam : FARID BIN MOHD NIZAM

Jacque De Boehmler : JACQUE DE BOEHMLER

Josif Trenchovski : JOSIF TRENCHOVSKI

Top Five Contributions

1. We created a multiplayer game with a feature that allows players to **choose their own map** based on a real-world location and generate it in real time.

2. We **utilised Photon Fusion 2** to handle all the networking for the multiplayer gameplay and added our own enhancement code for visual optimisation by adding local visualisation for certain animations.

3. We have **Buildify Blender implemented** in our source code to generate the map in real time and give the map a mix of 2D and 3D feels, where all the data of the map is taken from an open-source map(www.openstreetmap.org). The data taken is used to create a base 2D map and the 3D buildings are created using the generated map.

4. We have **a scoring system and a convexity check for the shape detection**, which gives the game more dynamics and various outcomes from the shapes created between the players. The scoring system will have an impact on the damage dealt and the convexity only allows a convex shape to be made.

5. We developed the game with over 85 merged pull requests, 850 commits and more than 150 closed tracked issues that are hosted on GitHub with continuous integration(CI) implemented on it. These are the results from the over 30 meetings throughout 12 weeks of making the game.

Video link: https://youtu.be/mw1UzViBsj4

Nine Aspects

Team Process

- Weekly Agile sprints using a Kanban board for tracking tasks and progress. [GitHub and Kanban Board]
- Regular Meeting on Thursday to plan the week and a Monday check up. [Meetings]
- Testing the game together to discover bugs and work together. [Meetings]

Technical Understanding

- Researched the difference between different networking libraries to see which one fits our needs the best: Photon Fusion 2, Mirror Networking, Fish-Net, Netcode for Entities and Netcode for GameObjects. [Individual Contributions]
- Researched Buildify in Blender and compared against generating buildings from scratch. [3D Map Generation]
- Researched OpenStreetMap Overpass API to find out how to make the API requests and parse the response. [Software, Tools, and Development]

Flagship Technology Delivered

- Real-World Map to 3D buildings: Automatically generate 3D buildings from a specified location's map data. Buildings are created with tile sets, laid out according to real coordinate data. [Demo and Video]
- Real-World Map to 2D map: Automatically generate a 2D map in our game's art style from a specified location's map data. This includes roads, paths, stairs, grass (such as fields), water (such as lakes) and fences/walls, giving an in-game representation of the main map features of the real-world area selected. [Demo and Video and Technical Content]
- Multiplayer: Synchronisation of movement and game state for multiple players, allowing for 12 players to compete in fast-paced action in real time. [Demo and Technical Content]

Implementation & Software

- Created detailed pipeline from the host selecting an area to it appearing in game in a mix of 2D and 3D graphics with accurate building and terrain info. Using Leaflet, OpenStreetMaps, Blender and Unity. [Abstract and 2D Map Generation and 3D Map Generation]
- Design of all game assets in GIMP, Aseprite and Unity. Including sprites, animations and map tiles. [Asset Creation]
- The game itself is designed in Unity using C#. [Game Engine]

Tools, Development & Testing

- All the source code is hosted on a GitHub repository for better version control throughout the project, and continuous integration is implemented on the repository. [GitHub and Kanban Board]
- Documented every change made to ensure team members' understanding of the code. [GitHub and Kanban Board]
- Every issue made is assigned to a team member, who works on a separate branch to avoid clashing, and the pull request made requires multiple reviewers as protection. [Software, Tools, and Development]

- Use Unity Editor as the main game editor to develop the game and as the main testing platform for the changes made. [Software, Tools, and Development]
- Photon fusion, Audacity and Buildify are other software used to add more functionality to the game, such as networking and map generation. [Software, Tools, and Development]
- The testing methodology used is mainly integration testing on functions added and user testing on a stable version. [Integration Testing, User Testing & Testing Session]

Game Playability

- Movement and controls are immersive and intuitive [User Testing]
- Balanced gamemode. (inexperienced players can play, but there is a competitive element as well) [User Testing]

Look & Feel

- Created our own sprites for our characters and objects. [Asset Creation]
- Implemented sounds for a better realistic feel. [Art and Sound]
- Added in-game music to create an immersive experience. [Art and Sound]
- Used a mix of 2D and 3D graphics with a top-down view to keep to the 90s arcade style. [Video and Technical Content]

Uniqueness and Innovation

- The Player(host) can pick anywhere in the world as the in-game map, which is then generated in real-time. [Map Generation]
- The game has a concept of 'Team Brawler' and 'Capture the Flag' but with a unique geometry element integrated into it to unlock abilities and create more strategic gameplay. [Game Concept]

Report & Documentation

- Every change made is well documented throughout the development period by explaining the changes made on the pull request. [GitHub and Kanban Board]
- Spent a good amount of time on the report to create a high-quality academic report.
- The report is split into sections and assigned to a relevant team member according to their strong suits.

Abstract

Overview
Geometa is a top-down ability shooter with a strong emphasis on teamwork and strategic positioning. Players take on the role of knights and wizards thrown into chaotic modern-day arenas, battling in procedurally generated environments based on real-world locations chosen by the players themselves.

The core gameplay revolves around activating powerful abilities through team formations. Abilities are tied to geometric shapes and the precision of your team's formation determines the ability's effectiveness.

The primary game mode is a capture the flag scenario, players earn points through both enemy eliminations and successful flag captures, represented in-game as magical crystals. The team with the highest score at the end of the match claims victory.

Story
During a long-forgotten medieval war, a mysterious and malevolent witch - motivated by nothing more than boredom - cast a powerful spell that disrupted the fabric of time. In an instant, knights and wizards from opposing sides were ripped from their era and transported into the modern world.

Now trapped in the present day, these warriors are forced to continue their struggle in unfamiliar environments. Their only hope of returning home lies in collecting magical crystals from their enemies and bringing them back to their base which houses a magical power. Once enough crystals are secured, the energy released activates a portal capable of sending them back to their rightful time.

Caught between two eras, the knights and wizards must adapt to new surroundings, work as a team and fight for their chance to return to the past.

Tech Overview
The multiplayer functionality in Geometa is powered by a custom-modified version of the Photon Fusion 2 networking library, utilising a host-and-clients architecture.

The game's map is a hybrid of 2D and 3D elements, created in parallel and seamlessly integrated. Arena selection is handled by the lobby host through Leaflet, which interfaces with OpenStreetMap to gather geographic data for the chosen location. During runtime, this data is processed to generate the environment dynamically.

Building geometry is constructed using Blender with the Buildify add-on, running in headless mode for automated asset generation. Simultaneously, OpenStreetMap data is analysed to identify features such as roads, grass, and bodies of water. These features are rendered in Unity using Sprite Shapes, allowing efficient tiling and polygon mapping for 2D elements. The generated 3D buildings are exported as a GLB file and imported into Unity, completing the arena.

The entire map generation process takes between 10 seconds and 1 minute, depending on the size and complexity of the selected area.

Gameplay Loop

Once all players have launched the game, one player designates themselves as the host and creates a lobby, which is identified by a unique name. Other players join this lobby, select their teams, choose their characters (knights or wizards) and set their display names. During this setup phase, the host selects the arena location and defines team spawn points.

Once all players have loaded into the arena, the host initiates the match. Players spawn at their designated team locations and can choose how to approach the battlefield - whether by exploring side lanes and alleyways for tactical advantages or charging directly toward the opposing team.

Gameplay typically unfolds with knights, equipped for close combat, rushing toward the enemy, while wizards maintain distance, using their ranged abilities to support from the backlines. This often leads to chaotic engagements where knights dash between opponents, wizards launch fireballs across the arena and spawn ogres to add another layer of challenge.

Effective teamwork and positioning are crucial. One common winning strategy involves knights breaking through enemy lines to disrupt and stun opponents, creating openings for their teammates to deal significant damage and activate abilities through precise formations.

Teams earn 2 points for each elimination and 10 points for capturing the flag. The match continues for 8 minutes, after which the game concludes. The team with the highest score is declared the winner and a post-match leaderboard displays player statistics along with an MVP (Most Valuable Player) for each team.

Player Controls and Ability Interaction

Players select between two classes, knights and wizards, each offering distinct attack ranges and abilities. While their combat roles differ, both classes share the same method for activating abilities, which are tied to team formations. The game continuously tracks the geometric shapes formed by nearby teammates. At any time, a player can hold either ‘Q’ or ‘E’ to preview the triangle or square their team is forming. Releasing the key triggers the corresponding ability, with its effectiveness scaling based on how precise or regular the shape is. The more perfect the formation, the stronger the resulting ability. Formations can include any mix of knights and wizards, though only the player who activates the shape gains the ability.

For knights, activating a triangle formation triggers an area-of-effect damage ability, with the damage scaling according to the accuracy of the triangle. If a knight activates a square formation, nearby enemies are stunned, with both the range and duration of the stun increasing as the shape becomes more precise. Wizards, on the other hand, gain different abilities. When a wizard activates a triangle formation, they are granted a fireball for five seconds, which can be cast using right-click. The fireball’s size and damage also scale with the precision of the formation. Activating a square formation as a wizard summons an ogre that seeks out and attacks the nearest enemy. The ogre’s damage, and lifespan are all determined by the quality of the square. All of the ability power scaling has visual feedback with colours.

Beyond these abilities, basic controls remain consistent between classes. Players use ‘WASD’ to move, left-click to perform basic attacks and shift to dash, knights having a longer dash distance and a shorter cooldown compared to wizards. Flags are picked up simply by walking over them and can be dropped by pressing ‘C’.

The Team Process and Project Planning

Development Process Overview

Our overall development process was conducted through three main processes, these being: group meetings, GitHub and the Kanban board, and Testing sessions.

Meetings

We made use of the scheduled meeting times by meeting 1 or 2 hours before them as a group. These meetings occurred biweekly and served to accomplish a number of important tasks, namely:

1. **Discuss what everyone had done since the last meeting**. This meant giving an explanation of what issue(s) had been tackled, how the implementation works (a basic description of the code or the game design feature such as a new sprite) and if it doesn't work, group members could then try and help out by giving ideas which usually resolved whatever bug or problem the person was facing. This worked well as it meant everyone knew what each other was doing and how they were doing it, so that any team member could theoretically switch between tasks if need be.

2. **Debate what new features could be added to the game** in order to make the game meet the criteria. For each idea put forward by a group member, we would take a few questions into consideration: how would the game profit by the inclusion of the feature, how much time would it take to implement the feature and where has the idea come from (e.g from a previous testing session or from another game). These questions would give us a sort of ratio as to the benefit (to the game) and cost (in time) and how likely it was to provide the benefit, whether it was based of user feedback or tried and tested in an established game and then a conclusion made, based on these factors, would put the feature either on the kanban board, to be assigned immediately or later on, or in the bin.

3. **Dish out GitHub issues**. For anyone who had completed their issues from the previous meeting, they would be assigned an issue from the kanban board so that their new issue would be: free (ie. no one is working on it already), relevant to what they have been working on already (ie. if you had added a feature to the user interface before, you are more likely to be assigned a user interface issue again) so that previous knowledge can be reused effectively and, lastly, the issue would be one that they want to do.

4. **Deal with any interruptions or miscellaneous issues**. This could be, for example: talking about the designs of the posters or stickers or, if someone was ill or otherwise indisposed and they were working on an important issue that needed to be completed before an upcoming testing session say, with their agreement the issue would be reassigned to someone else so that we could meet all deadlines, even with the loss of a group member for a week.

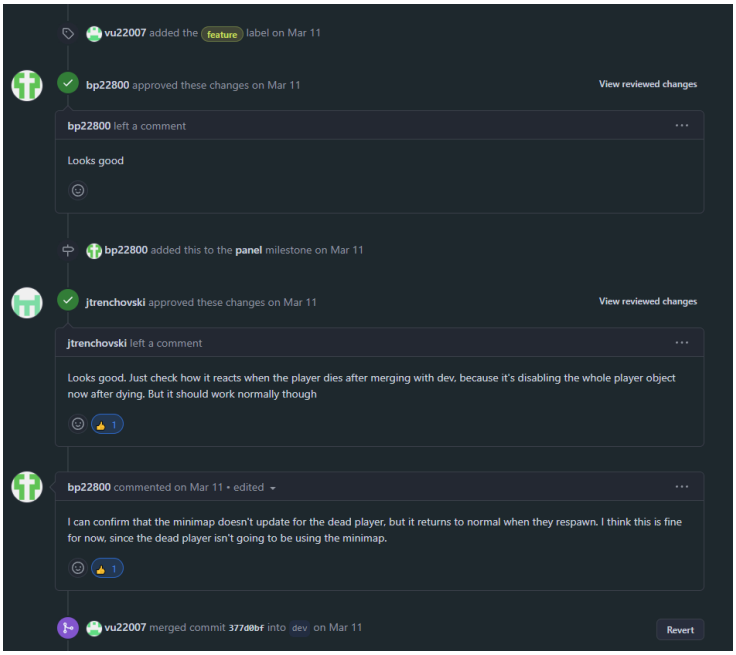
5. **Discern group members' feelings on the game's direction**. This was important as it meant every member was happy with how the game was progressing so that everyone would enjoy working on the game and more hours could be put in without any quiet resentment resulting in lack of quality.

Overall, our meetings were vital in maintaining clear communication, steady progress and a positive team dynamic throughout the project. By consistently reviewing work, debating new ideas, and reallocating tasks when necessary, we kept everyone aligned and engaged with the game's development. These meetings ensured that potential

problems were addressed early, that each team member's strengths were effectively utilized, and that morale stayed high even during busier periods. Without the structure and collaborative spirit fostered by these sessions, the project would not have progressed as smoothly or as successfully as it did.

GitHub and Kanban Board

The GitHub was critical for project management as it allowed the team to remotely view the current status of development by providing: an interface to see what everyone is doing via the kanban board and the branches view - where you can see made commits ahead of the development branch and effectively gauge progress on that issue, active pull requests which allowed the team to discuss changes made by a branch remotely, including problems with the provided solution some members might have such as: not understanding how the solution solves the issue, the spotting of a typo or badly written/inefficient code listed in the changes or simply a suggestion on how to better solve the issue. Here is an example of the sort of discussion that takes place in a pull request:



The Kanban Board served as our issue manager where each member could see what was currently being worked on, see what was queued up to be worked on next and see what was currently in review as a pull request. This system worked especially well as it meant it was virtually impossible for two members to be working on the same issue by accident or for two members to unknowingly make conflicting changes to the same area of the project. Each issue moved through a clear pipeline: from "To Do", to "In Progress", to "In Review" and finally "Done", once it had been tested and merged. This visual workflow made it easy to identify bottlenecks or areas where additional support was needed. It also encouraged accountability, as everyone's progress was visible to the team at all times. Furthermore, by linking pull requests directly to issues on the kanban board, we maintained strong traceability between problems identified during testing sessions or meetings and the code changes that resolved them. Overall, using GitHub and a structured Kanban Board was a major factor in keeping the project organised, on schedule and collaborative despite the challenges of remote communication.

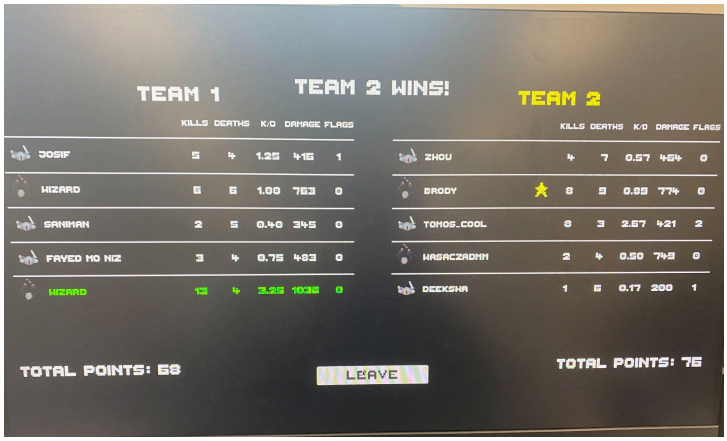
Testing sessions

We tried to host as many testing sessions as we could in the limited time that we had as we felt our game needed them especially, due to the balancing of the game being very important as you are against other players and if one character felt more powerful than another to play, for example, it could lead to frustration in users which we want to avoid. Because of this, we hosted a testing session every Wednesday in the room where our game would be demoed on Games Day in the weeks leading up to it; this meant we could get room-specific feedback, such as what to put on the TV screens, along with general user feedback. These testing sessions generally went as follows:

1. **Preparation.** An hour before testing users arrived we would meet and set up the room so that we could begin the testing session as soon as people came in and sat down. Along with this, we established what feedback we wanted out of the testing session and adjusted our user feedback survey accordingly.

2. **Testing.** Once enough people had arrived either through prior invitation or by being found nearby and asked to volunteer, the testing session would begin. This just entailed each person picking their team according to which table they were on, and their character based on personal preference (we found first-time players often picked the wizard as opposed to the knight) and then the game would start. Group members not playing with the testers would then walk around and observe the game in progress, taking note of any bugs or poor gameplay design choices that may surface.

3. **Conclusion.** After thanking participants, we asked them to fill out our survey which included a mix of qualitative and quantitative questions so that we could measure and compare results across testing sessions to see if we were making progress in certain areas such as user interfaces and also, gather peoples opinions on features that could be added to improve the gameplay experience. We would then collate this data and the feedback brought about by group members speaking to players of the game and compile it into new issues to go on the kanban board, using the same process as we would for new features brought up during meetings, as talked about earlier.



Picture of a testing session and a leaderboard from it

Overall, the testing sessions were invaluable in refining our game and ensuring a more balanced, enjoyable experience for players. Regularly hosting sessions in the actual demo room allowed us to fine-tune both the gameplay and the presentation elements, making adjustments based on real user feedback in an authentic setting. By combining observational notes with survey responses, we were able to identify and prioritize issues effectively, integrating this feedback into our development workflow through our kanban board. These sessions not only helped us catch and fix problems early but also allowed us to continuously improve the game's design, balance and overall user experience leading up to Games Day.

Looking back

What went well

- **Effective testing sessions:** Hosting regular testing sessions, especially in the actual games day room, provided valuable feedback. It helped balance the game and improve the overall user experience significantly
- **Good team communication:** Open discussions about game direction during meetings helped maintain team morale and ensured everyone felt included in the project

- **Responsiveness to feedback:** We incorporated survey and observational feedback into new Github issues, adopting an agile and iterative development style that ensured continuous development of our game
- **Flexibility and Adaptability:** When team members were unavailable, reassigning tasks quickly prevented delays and kept momentum strong

What we would change if we were to start again

- **Establish a clear and thorough form of the game early** in writing and perhaps draw some diagrams of certain aspects of the game, such as the UI. This would have been helpful as, for the initial few weeks, it seemed as though we all had different versions of the game envisioned. Making sure we were all on the same page would have meant we could get started faster.
- **Better initial task breakdown:** some issues may have been too broad at first and by breaking these issues into smaller, more manageable tasks would have made it clearer to the team how to solve issues while also increasing the divisibility of these large tasks
- **More documentation of ideas:** writing a short summary of what was talked about during a meeting may have provided a clearer representation of what was agreed on, decreasing the likelihood that members have conflicting ideas on what was decided upon and reducing confusion

Conclusion

Overall, the team worked very well together by employing these aspects of the team process effectively and we were able to steadily improve our game and address issues as they arose. Although there were areas we could improve on, the experience provided valuable lessons in technical development and team collaboration and we are proud of the work we produced and the way the team grew and adapted over the course of the project.

Individual Contributions

Demyan

Jan 20th:
- Created original game structure/design: rules, win conditions, abilities and their interactions

Feb 1st-4th:
- Created first iteration of the game's map
- Created 3 sprites for the 3 characters

Feb 5th:
- Created the map that was used for test days

Feb 10th:
- Added new sprites for characters, gun and bullet along with an animation system

Feb 11th:
- Created animations for all sprites
- Added colliders for players, buildings and arena walls

Feb 14th:
- Looked at and designed sprites for the 2D map

Feb 18th:
- Designed new class system and second iteration of abilities

March 3rd:
- Created sprite for the knight with idle, walking and attacking animations

March 4th:
- Created sprite for the wizard with idle, walking and attacking animations

March 11th:
- Redesign of the map
- Full design of in-game music system

March 19th:
- Designed new slashing effect for knights

March 20th:

- Created the first iteration of the summon with dragon sprite
- Research into building sprites for new graphics design

March 24th:

- Made the knight's sword larger and gave it a new animation with a larger attack radius
- Created Geometa logo and poster

March 26th:

- Reworked the melee attack; fixed attack rate with the damage radius matching the sprite

March 29th:

- Complete redesign of the summon with new features, new sprite and new animations

March 30th:

- Updated sprites to correctly align with the games lore
- Rework of the fireball

March 31st:

- Created video for game along with all games day info poster designs

April 3rd:

- Worked on summon changes and bugs
- Added a new points system and mvp functionality to the leaderboard

Daniel Lovell

Jan 20th - 22nd:

- Added initial character movement mechanics and implemented player respawning.

Jan 26th:

- Researched multiplayer libraries, looking at the differences between Photon Fusion 2, Mirror Networking, Fish-Net, Netcode for Entities and Netcode for GameObjects. I decided on using Photon Fusion 2 to handle the networking for our game, using Fusion's host mode, where one player acts as the host, and other players connect as clients to the host's computer, using Fusion's relay network to handle the connection.

Jan 27th:

- Improved the shooting mechanics with auto-fire, and made bullets deal damage to players on collision.

Feb 2nd:

- Completed initial setup of multiplayer code, allowing players to host and join using Fusion's prototype GUI, and having basic player movement synced. Also made bullets spawn across the network with their movements synced, and player health synced so players can damage and kill each other.

Feb 6th:

- Added networking for the current shape feature implementation, so that the shape objects are spawned across to all clients when a shape is placed by a player, so all players can see and interact with them.

Feb 9th:

- Made networked player movements smoother by enabling client-side prediction for rigidbodies.
- Optimised bullet networking by only syncing the initial bullet spawn, and simulate it locally on each client.
- Made bullet collisions utilise Fusion's lag-compensated hit detection.

Feb 13th:

- Fixed visual bug on clients where bullets get briefly stuck on impact with a player or wall before despawn.

Feb 15th:

- Created designated respawn areas for each team where the flags now spawn initially.
- Added a game timer to the UI that shows the time left before the game is over.

Feb 17th - 24th:

- Created the 2D map code that fetches from API and loads into game using preset bounding box in Bristol.

Feb 27th:

- Improved the popup message system to utilise RPCs, allowing the host to broadcast messages to all clients for events such as your team's or the enemy team's flag being picked up, or when a team wins.

Mar 5th:

- Made new shapes system work in multiplayer, so that shape previews are only seen locally, and activated shapes are seen by everyone, and damage caused by shapes is synced.

Mar 7th - 11th:

- Created main menu UI that provides a custom interface to Fusion's host and join functions.
- Created lobby UI that allows players that have joined the game to select their team and character (knight or wizard), and they can see their selection alongside other players on the screen. Once all players have clicked the "Ready" button, the host can start the game, which takes all players into the game with their selections applied.
- Made game go back to main menu if host closes the game, instead of freezing for the clients.

Mar 12th - 17th:

- Refactored the networking code, so that there is a separation of visual/sound effects from networked state.
- Fixed some other networking-related bugs as a result of the refactor.
- Fixed client projectile fire delay with local "dummy" bullet created on client when they fire it.
- Fixed flags from being able to be thrown through a collider and getting stuck.

Mar 18th - 21st:

- Added ability for players to enter a display name in the lobby, which is displayed above their head in game.
- Fixed shape previews from lingering when dead, and fixed uninitialised player UI bug.

Mar 22nd - 25th:

- Created leaderboard UI that appears when game is over and shows the winning team and the stats for each player collected over the course of the game, such as kills, deaths, damage dealt and points won.

Towards the end:

- Added final tile designs to Sprite Shapes for roads, paths, grass, water, etc. in 2D map.
- Added new 2D map features such as stairs and walls/fences, with gaps generated along walls/fences where there is a gate node in the map data. Also made map gen more robust by retrying the API call if it fails.
- Made the 2D map preload alongside 3D map in the lobby scene and fixed bugs with 2D map generation.

Farid Bin Mohd Nizam

Jan 20th:

- Added camera script for the player camera. The camera will follow the player throughout the game, and the view for the player can be adjusted in the Unity editor.

Jan 27th:

- Added a user interface(UI) for the player, which includes a health bar on top of the player and an ammo counter on the top-left of the player view.

Feb 6th-11th:

- Added flags for both teams and made it pickable when the player collides with its collider. The flag is also droppable by pressing ‘C’, and the flag also drops when the player dies.
- Added a winning condition; the team wins when both flags are near the respawn point.

Feb 17th-21st:

- Made a universal cooldown handler script that can be used to handle any cooldown UI for in-game abilities.
- Added a dash ability for the player, and it has its own icon to indicate the dash cooldown.
- Made the player auto-reload and added an icon for the cooldown indicator. The reload time also varies based on the player's current ammo.
- Added a death layer, which contains the respawn timer when the player dies.

Feb 27th - Mar 2nd:

- Created an Area of Effect(AoE) ability for the wizard character and added a cooldown UI for the ability.
- The player can use the ability by pressing ‘T’, and the fireball will appear at the player’s current cursor position.

Mar 5th:

- The ammo indicator is now at the left of the player’s health bar and only contains a number with a ring on it to indicate the number of ammo left.
- Since the game theme has changed, the cooldown icon has been changed to fire instead of a gun to fit the theme.

Mar 6th - 11th:

- Integrate the wizard's AoE ability into the triangle shape, where the wizard can only use the ability when a triangle has been made, and it has a fireball animation.
- Change the AoE cooldown icon usage to indicate the time left for the player to use the AoE after making the triangle.

Mar 12th-14th:

- Added a melee attack for the knight character and removed all the UI that is not related to the knight character, such as the ammo indicator and AoE icon.
- Change the input for the AoE ability from ‘T’ to ‘right click’ on the mouse to fit the gameplay experience.

Mar 17th:

- Added an indicator for the flag drop area, which is the player’s respawn point. The indicator includes an arrow on top of the player’s character and a circle around the respawn point to indicate the drop area.
- Added cooldown icons for the triangle, square, and the speed pick up on top of the player’s health bar.

Mar 20th - 29th:

- Change the game-winning conditions by creating a point system for the game based on user feedback.
- The player’s team will now get 10 points for bringing the enemy’s flag to their respawn point and 2 points for killing an enemy. Both teams' points are displayed on the top left.
- Change the camera settings to fit the current game look, and make some layering changes on the assets.
- Added a pop-up message for the game time warning to indicate that the game is almost end.

Towards the end:

- Did some bug fixing and added some simple UI toward the end.

Jacque

Jan 20th:

- Created git repository, setting up Unity with git and committing initial skeleton code including the script for the player and the game controller with some starting functions for the team to work off of

Jan 26th:

- Added firing a bullet from the player character with left click including functions for ammo and reloading
- Added the prefab factory static class which allowed any script to initialise a prefab in the resources folder since it was static

Feb 5th-11th:

- Added pickups for health and mana, including the sprite design
- Added a simple but effective hurt animation when damaged that fades the sprite to red and back, this animation stayed all the way through the design process
- Small change to make it so that the character faces in the direction of the cursor, also adjusted the UI so that it does not rotate with the player to avoid disorientation

Feb 14th - 28th:

- Added popups to appear on the player’s UI so that messages such as “reloading” or “enemy has your flag” can be broadcast to the player easily in varying speeds and colours
- Added a pause menu by pressing escape so that you can exit the game

Mar 4th:

- Added the damage indicators that appear over players when they are damaged to provide important feedback to users on a successful hit
- Added the mana bar to show how much mana a player has for shapes

Mar 11th:

- Added the minimap in the bottom left of the UI as it was highly requested through user feedback; this included the indicators for players with different colours representing the teams and icons for the flags

Mar 17th:

- Added backgrounds to UI elements to make them more visible, reworked AOE to damage in set intervals with 5 damage per tick rather than per frame with tiny amounts of damage to make it feel more impactful.

- Various balancing changes including: Limiting wizard movement by increasing cooldown on their dash, decreasing passive mana regen and increasing cost of shapes. All balancing changes were done as a result of our testing sessions.

Mar 19th-21st:

- Various powerup reworks including: A re-spriting of the health and mana pickups to fit the new style of the game (medkit -> chicken leg etc.), a new speed pickup that increases movement speed for some time, pickups respawn after 20 seconds, messages when picking them up, pickups appear on the minimap
- Added spawn protection for 4 seconds after respawning to prevent spawn-killing

Last few days:

- Knight square changed to stun enemies as the slow did not feel very impactful
- Various balancing changes and bug fixes including a rework for how speed is managed on a player and the AOE spell radius now scaling with shape regularity

Throughout:

- Arranging meetings, arranging testing sessions, collecting feedback and compiling them into issues for the kanban board, delegating issues based on what people have been working on so far, reviewing most pull requests giving my opinion for possible changes/bug fixes, making final decisions on various game design features, mediating opinions from team members and, of course, many bug fixes.

Josif

Jan 20th:

- Implemented first iteration of the shape-based abilities. The player could place a triangle, square and pentagon for which each vertex carried a circle collider that detects players. When the shape object detects a player at each vertice, a speed, damage or heal buffs is activated.
- Added a live preview: the shape preview follows the cursor and rotates towards the player. When the button is released the prefab of the actual shape is spawned.

Feb 4th:

- Implemented the second iteration of the abilities (of the knight). Added a live preview where the closest on-screen allies are connected by a line renderer (sorting the player's location around the shape's centroid to prevent crossing diagonals). Wrote a function that checks if the shape is convex and prevents concave activations.
- For the triangle activation a line collider linking the players is spawned and damage is done on all of the players detected. Based on user feedback, changed this to a polygon collider to apply damage on the whole area of the triangle .
- When the square ability is activated a circle collider is spawned at each vertice that slows and damages detected players.
- Made different cooldowns for each shape.

Feb 18th:

- Configured a CI pipeline to automatically builds the project on every pull request, catching any compilation errors before merging

Feb 21st:

- Developed a score function that measures how regular a shape is. The score then determines the intensity of the damage and size of the colliders of the abilities.

Feb 23rd:

- Integrated specialised sounds that match actions (dash, shoot, death…) and edited duration and speed accordingly using Audacity. The sounds are played on clients if the source is in screen view of the player. Created an Audio Manager object that persists across scenes and plays lobby and in-game music.

Feb 25th:

- Made a questionnaire based on the Game Experience questionnaire which was used in the user testing sessions. Participated in the four sessions conducted in the last month of the project and talked directly to players, gathering impressions firsthand.

March 4th:

- Wrote a Python script using the Blender API to automate the 3D buildings generation. The script: ensures the Blender OSM addon is installed, fetches the OSM data for given latitude and longitude bounds, applies the Buildify blend file that generates the 3D buildings based on the buildings footprints and then exports it as a GLB file that Unity can import and spawn during runtime.
- Aligning the 2D map to the 3D buildings scole so they match perfectly when combined.

March 17th:

- Created an html page that uses the Leaflet and the Leaflet.draw plugins to select area and respawn points. The coordinates can be copied and used as input in the Lobby for which the map should be drawn. Implemented buttons in the lobby scene where the Host can choose between “Default Map” and “Generate Map”.
- Added RPC logic so sync generation, running the Python script across all clients. Added ready checks for players generating the map.

Software, Tools, and Development

Game Engine

We used the Unity game engine to develop our game, since one person in our group already had experience using it and it fit the needs for our game. For the rest of us that hadn't used Unity before, we spent the first week of the project getting familiar with the engine and following some basic tutorials to learn how it works.

Networking

Within Unity, we used the Photon Fusion 2 networking library to handle all networking, using it in host mode, where one player acts as the server and other players connect to the host's computer as clients. Players can provide a session name that Fusion uses to establish a connection via Photon's relay network, which provides simple and easy matchmaking, allowing players to connect to each other regardless of where they are in the world. This made development easier as we could playtest with each other with a click of a button, even if we were in separate locations on separate networks.

Fusion made it easy to sync state between the host and all clients in real time, such as player positions, activated shape vertices, health, mana and flag positions. It also provided more advanced features that we made use of, such as client-side prediction for player movement and lag-compensated hit detection for firing projectiles, to ensure a smooth and fair playing experience.

Blender

We made a Python script that interacts with the Blender API to utilise the Blosm and Buildify plugins to generate 3D buildings from OpenStreetMap data for our game, given the GPS bounding box coordinates. The host chooses a map location via the GUI, a script passes the relevant parameters into Blender which is running in headless mode. Blender uses the parameters to produce a 3D model containing the buildings for our map, which is imported into the game in order to have dynamically generated 3D buildings at runtime

Art and Sound

The smaller 2D sprites were made in Aseprite, a pixel art image editor, which allowed us to easily and quickly create simple pixel art images that we could use in our game, such as and pickups. The larger 2D sprites were made in GIMP, an image manipulation program, which made the production of larger sprites, with more details and number of colours simple as well as the production of sprite sheets for animations. Any custom sounds were made in the Audacity audio editor. The in-game music consists of 2 tracks made by a composer.

OpenStreetMap Overpass API

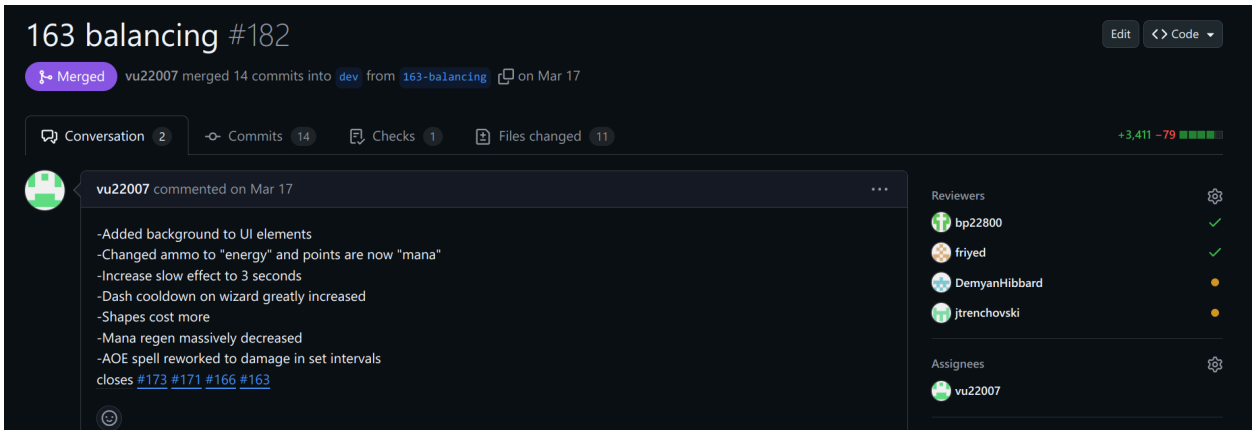
Despite the Blender plugins doing the API requests for us for the 3D building generation, the 2D map generation is done using a custom C# script we made in Unity that manually makes a request to the OpenStreetMap Overpass API and parses the returned JSON to generate scene objects for the 2D map.

The API accepts requests that use the Overpass Query Language, a special query language that allows you to specify what map elements you want to fetch, taking coordinates of the bounding box to fetch, and using filters such as element types and tag names to narrow down the map elements returned. We used the Overpass Turbo online tool at https://overpass-turbo.eu/ for interactively building and testing different requests, and observing the output, to understand the request format we needed to use in our script.

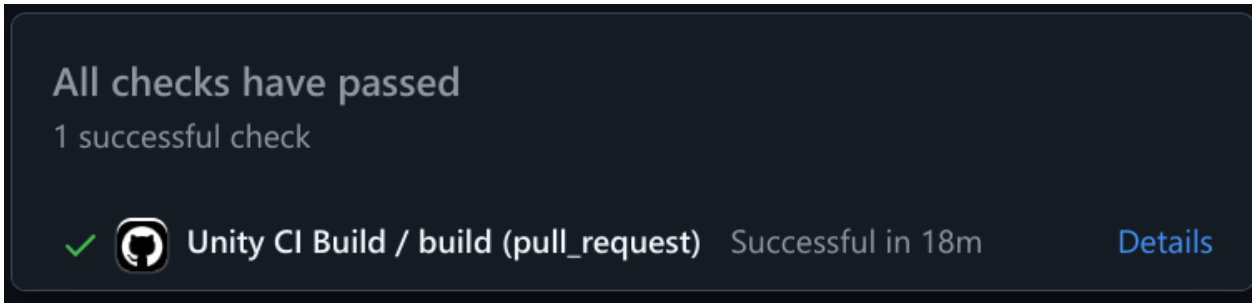
Version Control and Collaboration

We used git for version control, with our repository hosted on GitHub, which allowed us to easily collaborate on the project together. We utilised the kanban board feature in GitHub to organise tasks. Branches were heavily utilised, and we had main and dev protected, so we could only put changes into them via approved pull requests. Each feature we worked on was done within its branch, named with the issue number along with a short title, and when the feature was completed, a pull request was made, which required at least two other team members' approval, before it could be merged into dev. This ensured that our team was understanding and reviewing each other's changes throughout the project.

An example merged PR showing a list of changes, the issues it closes, and two approvals from other team members:



On top of this, we had continuous integration (CI) also set up in GitHub, which built and ran our project automatically on every pull request. This added extra assurance that any feature compiles and runs with no errors before making its way into dev or main. The image below shows the CI passing for the PR shown above.



Development Process

We worked in weekly agile sprints, organising our tasks on the GitHub kanban board and creating a new feature branch for each task each of us planned to do. Once a feature was complete, we would create a pull request documenting our changes, and we would have at least two people review the changes before we could merge them into the protected dev branch. The CI check would also have to pass before allowing us to merge into dev, ensuring our code could compile and run with no errors. If there was an issue or suggestion to be made, we would do so in the pull request, and we would make sure all issues were addressed before merging. Every week we would review what we have done and plan for the next week.

Adding New Functionality

Our code is divided logically into a game controller for managing general game state, classes for network and input management and individual classes for each component of our game, e.g., players, projectiles, and pickups. We also have a shape controller class for each player that manages the several shape components. This clear separation of our code would make it easy for someone to find the relevant part of our codebase they need to work on. The code also contains lots of comments throughout, so it would be relatively easy for someone to understand code written by another member and be able to expand on any functionality.

The vast majority of game objects are instantiated from prefabs, rather than already existing in the scene, so it is easy to adjust properties of e.g. the player by simply changing the parameters of the prefab. The characters that the player can choose to play as are specified as scriptable objects, which are easy to modify in the Unity editor, so character properties such as health and speed can be easily modified. These changes only need to be made in one place (the prefab or scriptable object) and the game will instantiate them at runtime, making adjustments to game objects simple.

2D map elements are simply Unity Sprite Shape objects that have their vertices set to what is specified in the map data for that map element, with the visuals handled by a tileset that is automatically mapped over the shape. So modifying the generated 2D map’s design can easily be done by editing the sprite shape controller for the desired map element, and selecting different tiles for the outer and fill tiles of the sprite shape. The game will dynamically create the object at runtime as specified by the map element’s sprite shape controller in its prefab, making the map design easy to change.

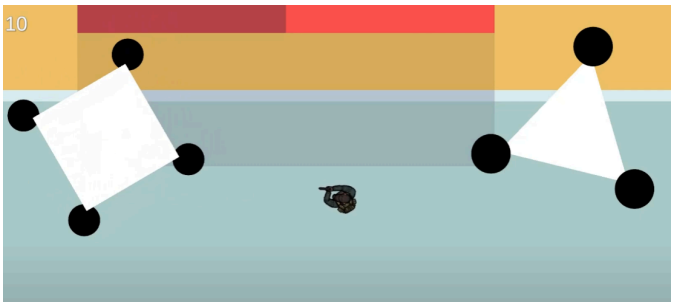
Technical Content

Game Concept

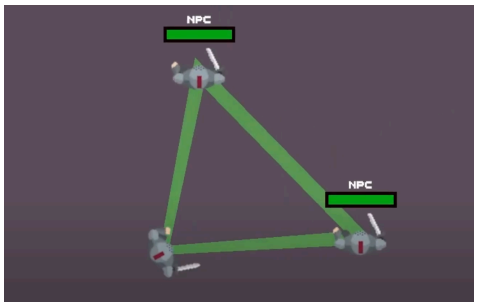
Game Theme

The game's initial concept was an army-themed shooter with static shape mechanics. Initially, there was a big debate as to how we would manage the shape activation as it was the distinguishing feature of our game initially as represented in both the name “Geometa” i.e geometry and meta (strategy) and our initial pitch. Our first iteration of the shape making involved pressing a key and placing down the shape with your mouse for your team to move onto the corners, activating the ability. This implementation of shapes, as pointed out by panelists during our first

presentation, was very rigid, boring and did not keep with the fast pace of the gameplay as it required players to stay still, where they could then but shot at etc.



Original theme with old shape mechanics



New theme with new shape mechanics

This led us to develop a new method of shape activation as detailed in the game concept section of technical content which involved simply holding down a key to preview the shape, automatically created with nearby players, and then releasing the key to activate the ability. Not only was this method more intuitive and easy to use, it also looked much better and fit the gameplay more. This is because you can move and shoot while taking part in the ability, making it more desirable to perform. Without the feedback from users testing the game, we might not have come to this realization and the shape activation would be left stiff and unenjoyable.

The new theme that we decided is medieval, with a wizard and a knight as the characters. The characters also have different attributes and abilities. The wizard has less health and mobility than the knight as it has a ranged attack while the knight is healthier and more mobile with a melee attack. The abilities for the wizard are an AoE fireball for triangle and an ogre summon for square, where it will attack the nearest enemy. On the other hand, the knight deals damage in the area of the triangle made and a stun at the vertices of the square made.

Shape-making System

The new shape-making system finds the nearest alive team member based on Euclidean distance. Holding the ability button shows a preview of the shape by instantiating a line renderer that connects the players. The positions of the closest players are constantly updated on every network tick giving real time information to the players needed in the dynamic gameplay environment. To prevent the line renderer's vertices from criss-crossing diagonals, the player's positions are sorted by the polar angle around the centroid resulting in the correct "around" order. The shape-making also has a scoring system that affects the abilities in many ways, such as the size of the ability and the damage. The convexity of the shape is determined by looking at the sum of angles. The sum of all the interior angles in a polygon is $(n - 2) \times 180$ where n is the number of sides. The function used to calculate the angles always returns the minor between the two vectors (i.e ≤ 180). If a polygon contains an interior angle >180 the sum will be smaller than $(n - 2) \times 180$ as the angle function will return the explementary angle which is ≤ 180. If the shape is concave the ability isn't activated while in the convex case a regularity score is calculated determining the strength of the ability. For example, the wizard's AoE ability will be bigger when it is activated by a more regular triangle. The score function states as: $regular\ angle\ sum \div (total\ deviation + regular\ angle\ sum)$, where the regular angle sum is $(n - 2) \times 180$ and the total deviation is the difference of the angles sum from the regular angle sum. This means that a regular polygon achieves a score of 1 while the values smoothly decrease as the shape gets distorted.

Game Mode

Since the game has a base concept of a known game, which is ‘Capture the Flag’, the initial winning condition of the game was that if both the enemy and the ally flag were near the respawn point, the team wins the game. This winning condition made the game end too quickly without the need of making shape or the game ends with a draw which doesn't align with its competitive nature. In certain situations, some of the players just hid their flag and went on shooting other players throughout the map, causing the game to last the maximum 8 minutes and still end in a draw..

Then, once the game concept was updated to the medieval wizard and knight theme, a points system was introduced. In the point system, the team will get 10 points for bringing the enemy flag to their respawn point and 2 points for killing an enemy. The reason these changes are made is based on the user feedback that we got from the user testing session and to give the player more motivation to actually use the shape's abilities to kill an enemy and capture their flag throughout the game. Some UI was also added to ensure that the players can keep up with their team points and the enemy points. The points are clearly shown on the top left of the player view, which will enhance the overall player’s gaming experience.

Networking

There have been a few challenges regarding the networking code of our game. The first notable example is to do with the projectiles. These are a key element of the game if you play as a wizard, with a large number of networked synced projectiles moving around the game scene. We originally had projectiles that would have a small delay before spawning on the screen from the client’s perspective, since the client would wait for the host to verify and sync the state and spawn the projectile across all clients. This was very noticeable to the players during some of our testing sessions, since players expect immediate response especially when engaged in fast-paced combat and the delay can put them off. This issue was remedied by having the client create a local “dummy” projectile as soon as the shoot action initiates, which only the client can see. The host still responds to the shoot action, spawning the projectile across all clients, when the client that fired receives the update from the host, it replaces the local copy with the network-synced version. This fills the gap in time between pressing the shoot button and seeing the projectile, making the game feel more responsive for the player.

Originally, the projectile’s position was being synced across all clients on every server tick, which we realised was unnecessary since the projectile’s rotation never changes, also its position is predictable given its starting position, starting tick and its constant velocity. So as an optimisation, we removed the syncing of the projectile’s position, instead only syncing the tick it was spawned on by the host, the velocity and the rotation. Once each client receives the request to spawn the projectile from the host, from there it locally calculates the position on each tick based on its starting state and the time since then (number of ticks passed multiplied by time per tick) to essentially simulate the bullet’s trajectory. With a lot of projectiles in the scene, this cuts back a lot on the data sent over the network, with data only sent on bullet spawn and despawn.

Another issue with the networked projectiles was to do with bullet despawn. When the projectile hits a player or wall it despawns, which is an action initiated by the host when it detects that the projectile has collided, and only when the clients receive the update to despawn the projectile, did it actually disappear on the clients’ screens. We had made the projectile stop moving when collision is detected locally, so that they don’t pass through objects while waiting for the host to despawn them, but this still meant that they were visible but stationary at the point of impact while waiting. This gave the effect that the projectiles were “stuck” in the player or wall for a brief moment. We fixed this by locally disabling the projectile game object when a collision is detected locally, so clients see an immediate reaction to the impact.

Due to an initial lack of understanding of how Photon Fusion 2 works regarding client side prediction, we were putting all game object update code for networked game objects inside of the Fusion-provided FixedUpdateNetwork method, which gets called on each tick like Unity's Update method does, except it is called and handled by Fusion. We were putting both network state updates and visuals (e.g. health bar UI updates) all in the same method, which was causing visual bugs such as having the health bar jitter back and forth when health changes, or having on screen notification for a player reloading showing up multiple times at once. After doing more research and reading more Fusion documentation, it became clear what the issue was. Fusion can call FixedUpdateNetwork multiple times per tick for the purposes of client-side prediction, with the duplicate calls being for simulating earlier game state, and one of them being for the current tick (the one we actually want visual updates for).

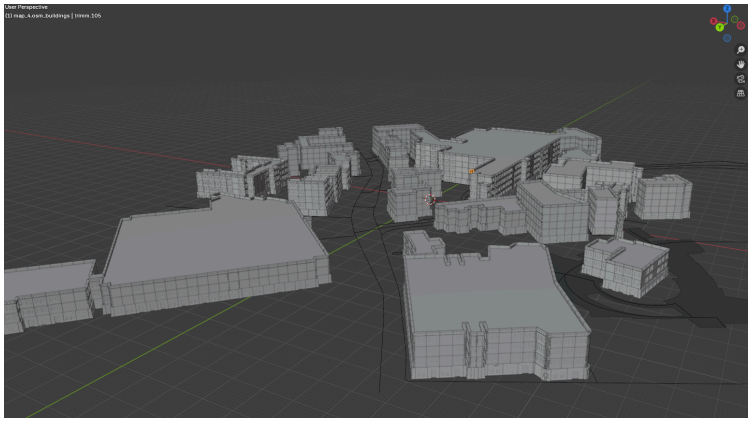
The client-side simulation as provided by the Fusion library works by having the client respond to player inputs and move or do key press actions locally, but then once the host receives this state and verifies/corrects anything, the client that provided the input needs to look back in time to when it started predicting, and it needs to recalculate the state each tick from then until now, using the information it now knows from the server e.g. other player's positions. The code in FixedUpdateNetwork is supposed to only modify networked properties and is supposed to give the same output given the same networked property values, to allow for it to be used for resimulation without side effects. So in order to fix our code, we needed to do a refactor, going through and separating all visual side effects from the code that deals with networked state. Fusion provides an UpdateRender method that only gets called once per tick for this exact purpose, and also provides the ability to set callback methods that call a method to do side effects on network state change only once per tick. By refactoring the code to utilise these render methods, the visual bugs we were experiencing went away, and the game felt more stable. The refactor also fixed some other networking-related bugs we had, especially with the shape system.

Map Selection

Before the game starts the player can choose between a default or a procedurally generated map for a given area. If they choose the generation option the area can be selected by opening an html page powered by Leaflet and the Leaflet-Draw plugin. Leaflet fetches the OSM tiles and adds an interactive layer. The Leaflet Draw is used to select a rectangular area and respawning points which can then be copied and fed into the map-generation pipeline.

3D Map Generation

These coordinates are then used by Unity to run Blender in headless mode. The Python script uses the Blender API that: ensures the BLOSM add-on is installed and installs it if not, BLOSM fetches the OSM data and creates a building footprint for only the buildings inside the given coordinates box and using the Builify blend-file a Geometry Node modifier is attached to each imported 2D footprints and the 3D building meshes are created. The buildings are then exported as a glb file in the Streaming Assets folder where it can be accessed by Unity during runtime.

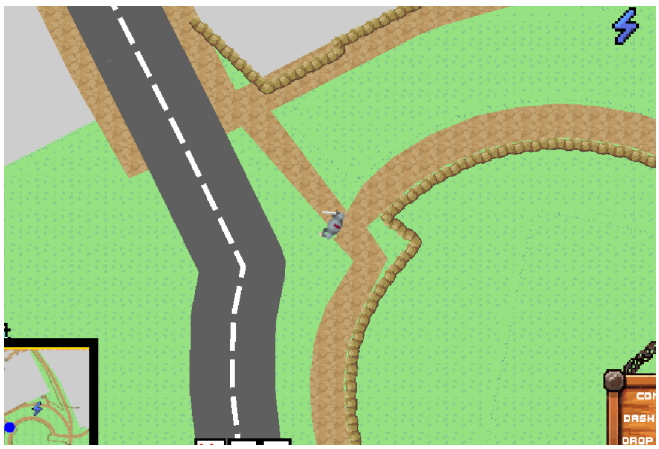


3D Buildings in Blender generated using BLOSM and Buildify

A problem that we came upon was that Unity only supports importing FBX files in the editor and not during runtime. Because there wasn't a third-party package offering this, we switched to exporting a GLB file instead and used the gITFast plugin to import the data during runtime. This caused a big performance drop - a small map area that would normally take 30 sec to generate was taking more than 5 mins. Debugging involved detailed analysing of how Buildify handles its geometry nodes as well as how the different exporting ways differ. Experimenting with the export function parameters (materials, vertex colors, draco mesh compression, etc.) didn't result in any improvements. Later we compared the exporting functions and found that the GLB exporter printed thousands of log lines describing every mesh while FBX only printed a couple. This caused the OS buffer to fill up, blocking Blender until the C# process drains it, significantly slowing the export process. Canceling the redirected output brought the runtime of the export process as before.

Another problem was that the roofs of the buildings were missing when we switched the file type. This was because Buildify's handles roofs differently by placing it into its own helper collection and is only generated when the Geometry Nodes modifier is applied. We changed the GLB export function to also access objects outside of the main Selection and apply the modifiers before exporting, something that the FBX one did by default.

2D Map Generation



The 2D map generation was achieved by creating a Unity C# script that fetches from the OpenStreetMap Overpass API. The API request uses the Overpass Query Language to specify a bounding box and several filters to select which map elements are wanted from the map database that lie in the bounding box. The API returns JSON that contains a list of all map elements that match the query. Each element is one of three types: a node, a way or a relation. Nodes are just single points on a map, with associated tags that describe what it is. Ways are a collection of nodes, with tags associated with the way as a whole that describe what it represents. Ways can be open-ended or close-ended, where the list of nodes in the latter start and end with the same node. Roads and paths are open-ended ways, and polygons on the map such as buildings, grass and water are close-ended ways. Relations are a collection of ways, e.g. a building

that consists of an outer polygon with an inner “hole” polygon. The script runs during the lobby scene once the host selects the bounding box from the map GUI.

The script we made uses Unity’s JsonUtility library to parse the JSON into a list of objects mapped to our MapElement class. Each MapElement contains the type (node, way or relation), the latitude and longitude coordinates if it's a node, a list of nodes (lat and lon pairs) if it’s a way and a list of members if it's a relation. It also has a set of tags that we are interested in for our map, such as whether it's a building, a highway (road or path), etc.

The coordinates for all map elements are GPS coordinates, which require a conversion to the latitude to make it the correct proportion in comparison to the longitude when rendered in 2D. After doing that, all coordinates are shifted so that the centre of the map is at the scene’s origin, and the points are scaled up to a suitable size.

Initially the 3D buildings weren’t exactly matching the 2D map. Because Blender and Unity have the same measuring units and 1 meter in Blender translates to 1 meter in Unity, this meant that in order to match the maps we only needed to match the manipulation of the OSM data to the one of the BLOSM add-on. Initially we had a fixed scale and noticed that the alignment error varied depending on the location of the selected area. Suggesting that we need to account for the fact that one degree of longitude covers fewer meters the farther you are from the equator. The scale we used is $cos(midLat) \times 111319.488$ where 111319.488 is the meters length of one degree of longitude at the equator and $midLat$ is the central latitude, adjusted the equatorial to actual meters-per-degree perfectly aligning the 2D map with the 3D buildings.

For each MapElement object, the script uses the tags to decide if it's a road, path, grass, water, building, etc. and instantiates the prefab for that object type. The prefab contains a Sprite Shape Controller, which renders a sprite based with the given tiles based on the nodes that make up the shape outline. The coordinates of each node is set in the sprite shape controller to define the sprite’s actual shape and the Sprite Shape Controller in the prefab is preset to be either open-ended or close-ended depending on what the object is. The Sprite Shape Controller in the prefab has a Sprite Shape Profile assigned to it, which specifies the tiles to use for the edge and the fill of the shape, plus other settings. The controller can also automatically sync the same coordinates to a polygon collider (e.g. for water or buildings) or to an edge collider (e.g. for walls/fences). Each prefab has a different profile that defines how it will look once the vertices of the shape are set. This makes turning a MapElement from the parsed JSON into a scene object as easy as instantiating a prefab and setting the vertices.

One complication we had was that using an open-ended sprite shape for paths didn’t give good looking results, since it tiles along the shape’s line and stretches it to the desired thickness, so for wide paths the tiles looked too distorted. To resolve this, we created a function that takes an open-ended line of vertices and outputs a close-ended list of vertices of a desired thickness, by taking each point on the line and finding the two points either side at the specified distance. Getting it to work properly for all paths took some work, but it allowed us to have a close-ended polygon that we could use a close-ended sprite shape with, which allowed us to fill the shape with smaller unstretched dirt tiles that gave the result we were looking for. Roads on the other hand were simpler since the tile we were using is a plain block of colour with a white line in the centre, so this being stretched to a given thickness looked fine.

Another complication was with walls and fences. These two map elements are both rendered simply as a stone wall in our game and is done by using an open-ended sprite shape containing the list of vertices that make up that wall or fence, which tiles a stone wall sprite along the line. The issue is that these walls can create inaccessible areas where there is supposed to be a gate or gap in the wall, but isn’t present in the game world. Gates are marked with a node along the wall in the OpenStreetMap map data, but the wall itself doesn’t have a gap in it, so converting the wall data into an object in the scene will not include this, which also means that the wall’s edge collider will not contain a gap which can block players from reaching places that they should be able to according to the map data. To fix this, we had to make a function that takes the list of wall vertices, the gate node, and the desired gap thickness, and it

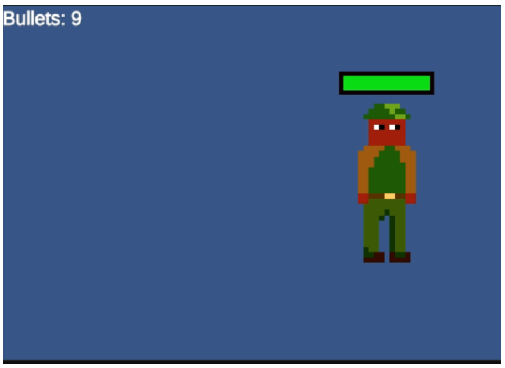
returns two separate wall objects separated by the gap specified. This is done by finding the closest point on the line to the gate, removing any vertices on the line that are too close to this point (which splits the wall in two), and inserting a point at the end of each half that is as close as possible while maintaining a gap.

Asset Creation

All character, pickup, flag and spawned sprites were made in house. The larger sprites, such as the characters, ogre and fireball were made in GIMP in a 64x64 format. To achieve animation, these were redrawn for each individual frame, put together into a sprite sheet and spliced in Unity for easy use adoption into the animation system. Smaller sprites, such as the bullets, mini fireballs and pickups were made in Aseprite, as the lighter software was all we needed for the 8x8 sprites.

In-Game Settings

The UI initially only had a health bar on top of the character and the ammo counter at the top left corner. Based on the user feedback, the UI is a little too empty, and the bullet counter is hard to see.



On the new UI, the main changes from the old UI are that the player health bar and ammo counter are now at the bottom middle of the player's view. The health bar is now much bigger for the player to always be aware of their current health, and the ammo counter is now simplified to just a number with a ring around it, which will slowly disappear as the ammo is used. As the game changed, the overall UI of the game also changed to ensure that it presents enough information and the player can easily navigate. All the cooldown icons for the abilities are placed on top of the health bar and the mana (used to make shapes) bar is under the health bar. On the left side of the player view, have both teams' current points at the top and a mini map that will change according to the player's position at the bottom. Then on the right side, there are the control options at the bottom, which will be useful for a new player and there is a game timer at the top. This overall change to the UI does helps the player have a better gaming experience. The new UI is more useful and packed for the player as it shows all the icons and has clear indications of everything that the players need to know.

With the implementation of the 3D building into the map, the layering of the asset needs to change, as the ability does not have any fixed layer settings. The characters and abilities are now on the same layer to ensure that the damage taken and attacking are interacting with each other, and other layers can be ignored.

Testing

Every time a team member is assigned to add more features, an integration test is made to ensure that the feature works properly according to the description made on the issue. The team member usually will work on the feature on their own branch and create a pull request once the feature is done. The other team member will then try the feature on their machine and request some changes if needed.

In the Unity editor, a separate scene is created to use as the testing ground. Once the new feature is complete, the feature is tested on the scene to make sure that the feature works, but also to make sure that the feature works together with the pre-existing feature that is available in the ‘dev’ branch. The new feature also has a few criteria that need to be met before approving the pull request. One of them is to make sure that the feature is working in both single-player mode and multiplayer mode. To make the testing easier, a commented code is created on our game controller script. With this section of the code, the feature can be easily tested for its multiplayer functionality without the need for multiple players, by simply uncommenting the section of the code.

User Testing

Once we had an MVP version of the game with which we could test with, we started user testing weekly in order to get the necessary balancing changes needed to make the multiplayer experience as enjoyable as possible. The details as to how we carried out the testing sessions were outlined earlier so this section will be detailing specific changes we made to the game as a result of the user testing sessions.

Knights vs Wizards

The balancing of the characters in terms of how powerful they felt to play relative to each other was critical as the team gameplay depended on the diversity of composition within teams, each character had to feel desirable to pick. To balance the characters initially we went with the very commonly used system where you have three main classes, these being tanks (to shield their team from damage while providing objective support), damage/DPS (damage per second) who are meant to be the damage dealers and lastly, the support whose responsibility is to provide bonuses to their own team while not doing much to the other team. In this sense, the knight was designed to be the tank - having more health while dealing less damage in comparison to the wizard, designed to be the DPS class - being more frail and dealing more damage. This, in theory, would encourage cooperative play and diverse teams leading to a more varied experience, allowing repeat players to find the game enjoyable over multiple gaming sessions. We developed this initial class system through our own experience playing similar team class-based games such as Overwatch, League of Legends and Marvel Rivals.

Through user testing, we found that the initial character balancing was far from perfect as, when asked which character felt more powerful to play, the testers almost always said the wizard. This was mainly due to the fact that wizards could attack from a range and the knights could not, meaning the wizards could damage knights while being safe from damage - not very balanced. To combat this, for our next testing session, we implemented dashing for knights and wizards, with wizards having a much greater cooldown for dashes so that a knight could get in range to attack the wizard successfully. Sure enough, during the next testing session we had a more diverse range of opinions as to which character was stronger.

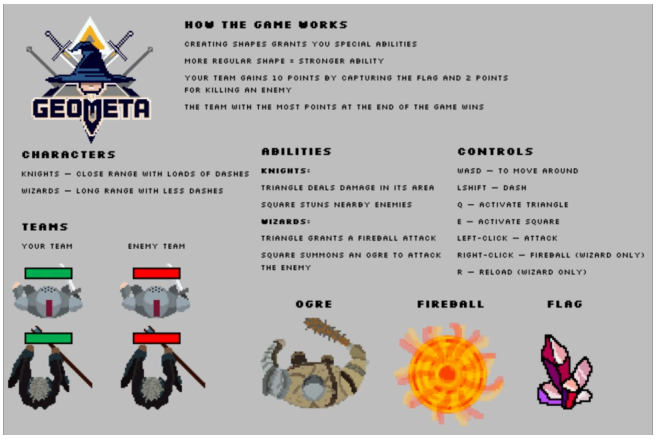
The shape abilities also played a big part in the balancing of the two characters as they were unique to the caster’s class. Initially, the knight's shape abilities were obviously quite underwhelming, with testing users saying that it's more effective to just focus on attacking with left click as the abilities were not potent enough. The knight’s abilities at the time for the triangle and square respectively were damaging the players on the edges, up to 5 damage, and

slowing down enemies at the corners for a duration of a couple of seconds. Looking back it's easy to see that this would not encourage use of the abilities, especially considering they had a cooldown and needed "mana" to use which you gather relatively slowly. Because of this, we decided to make the triangle do 3 times the damage and make the square now stun enemy players at the corners for 4 seconds, during which they are unable to move or attack etc. This made the abilities actually desirable to use and made the knight more desirable to pick.

Game Playability

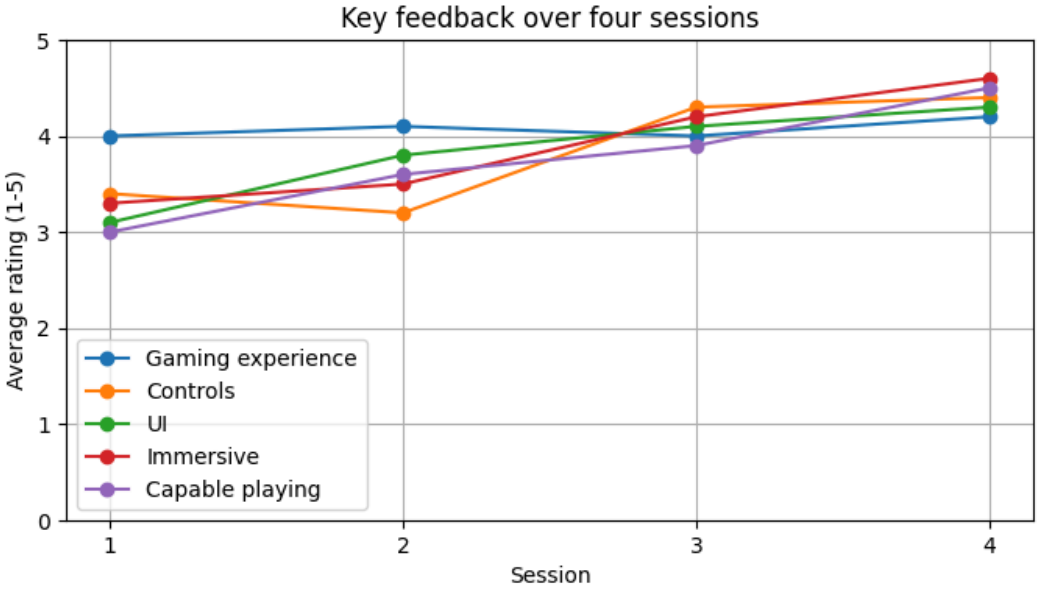
Our game was designed to be challenging, with a learning curve so that it doesn't get mundane after a few times playing. This meant that there was a lot of necessary information to convey to a first-time tester such as: the characters and their traits, the objectives of the game (i.e capturing the enemy flag while killing as many as possible) and how the shape making abilities function. To combat this, we first considered implementing a tutorial as it was suggested during one of the initial panels; however, we concluded that a tutorial would not be viable as the games were confined to 8 minute playing sessions. With 12 players in total, the time it would take for each player to first learn the game via a tutorial would mean the actual game session would take much longer to commence.

We needed to implement some form of information display for the new users though as we had many responses on our feedback form saying that controls and gameplay features were not immediately obvious. To combat this, we added a control list to the bottom right so they at least knew the basics in playing the game. However, across a couple more testing sessions it became clear that we needed more so we decided to create a short demo on how abilities are used and what they do along with an information sheet with all the game rules to display in the room on the screens available, so that we could reference them when giving the initial talk on how the game works. This is the information sheet that we used, alongside the link to the shapes demo video:



[Shapes Demo Video]

This proved quite effective, as we saw the "game controls were responsive and intuitive" response in our google form for collective feedback, improve across testing sessions as detailed by this graph plotted from the response:


Key feedback over four sessions
Average rating (1-5)
Session
0
1
2
3
4
5
1
2
3
4
Gaming experience
Controls
UI
Immersive
Capable playing