

Computer Games Development

CW208

TDD

Year IV

Darragh McKernan

C00271651

|  |  |
| --- | --- |
|  |  |
| 11/05/2025 | |

**Acknowledgements**

I would like to thank Martin for helping me every week plan my project out and for guiding me in what features to focus my time on and for keeping me on track timewise by encouraging me to have features planned to be finished each week. This kept my project development very stress free as I always knew what to work on next. Martin also helped me focus on the next few features I was going to work on which had the effect of making the features I was currently working on be designed to work with the next few features. This sped up development a lot as I was future proofing my code.

I would also like to thank Lei and Noel for their feedback during the 2 demos as this feedback was invaluable in helping me decide what features to focus on and where I should go with my project.

I would like to thank all of my classmates, we have had a lot of deadlines and we have consistently reminded each other of deadlines and have been able to help each other out if anyone has questions or issues. We also frequently played each others games and gave each other feedback which was very helpful.

**Project Abstract**

War of Attrition is a top down 2D turn based strategy game. It is built using C++ and SFML. The game focuses on having the player make tactical planned out decisions when deciding what they should do during their turn. The player can buy and sell units in an attempt to turn the tide of the war. The players also have access to a research lab allowing them to make their own custom units that can be built to make a unit fit to any purpose and adding an element of uncertainty to the game as that means all players can hold on to their money and create a supertank or maybe they could make a vast amount of weaker tanks to act as a barrier. The core development focused on making formations that had followers that were dynamic and capable of working independently of the formation in the event that they can not get to their designated position. All of the units the player can create are able to be added into the formation. The world consists of a grid of tiles, these are used for the pathfinding algorithm so that the game can scan the world for valid tiles and generate paths for the units and formations that need them. The game also has an automatic texture setting world editor, this allows the players to simply paint on the background where they want tiles to go and the game will set the textures automatically. The game also features a particle system that is used for the explosion effects on the destruction of units.

**Project Introduction**

The overall goal of my project was to create an interesting turn based game that wasn’t completely predictable and had some variance in the gameplay. I wanted to focus on formations and units that will interact with the formation in an intelligent way so that they can still somewhat navigate the world independently but will still attempt to get to their designated formation position. I wanted to keep the formation abstract and have it function independently of the units that it consisted of so that it could still get to its objective even with missing units. I also wanted to give the player as much information as I had access to as a developer when it came to the background numbers that made up each unit. Part of the variance in the game that I wanted was a custom unit that the player would be able to create so that the game isn’t boring and wont have the same outcome over and over and to make sure each stays interesting. The player is allowed to modify most of the background stats that make the units unique from one another. I also wanted to make a level editor that got rid of the need for entire spritesheets that could be selected on screen and simply have the game handle all of this for the player. I wanted the player to have full control of my game but not have to have any knowledge of the game, I wanted to make the game do all of the hard work while all the player had to do was click units for automatic formations or draw in tiles on the map and watch the game handle the rest.

**Literature Review**

The original game concept was sort of inspired to be like a chess-like game that had a grid based map and loads of units and decisions to be made on where to put them. I wanted it to be strategic and force players to not just charge ahead but maybe try draw units out and then attack them during their own turn. To prevent the game from getting stale the custom units would allow for anyone to throw a wrench into learning off a strategy. Custom units also force players to fight as if they don’t they can just sit back and make a really strong custom unit and decimate the other team with it. A core inspiration for the visual design and overall gameplay was a game called Advance Wars. Its style and movement of its units was nearly swarm like but they still had a clear and discernible route that they would take to get to their end goal. I also use BFS to explore the tile grid so that I can generate a path between the target and the unit that requested the path. The entire game was built to be as modular as possible making future additions easy. The unit class was slow to make but after completing it I could then very easily simply assign a new sprite and modify the squadData struct so that the unit could have its own unique stats. The explosion particle system was also made with modularity in mind and can be easily expanded upon and upgraded.

**Evaluation and Discussion**

The system for creating a formation of units turned out very close to how I originally set out to create. It is very easy for the players to use and the units are able to find their own way if they are not able to get to their designated formation position. The units typically just follow a path but once added to a formation they sway to the use of behaviours. These behaviours are what allow the units to function independently from the overall formation, they are simply given a formation position and they can do the rest of the work themselves. The units will swap between appropriate behaviours depending on what specific scenario or issue they have ran into, it was designed with several redundancies to prevent the units from getting stuck as this would soft lock the game by preventing the round from ending. The world editor turned out perfectly though it is lacking sprites for certain specific outcomes as the spritesheet I used did not cover all of them however the vast majority of outcomes are implemented and it can be seen working as expected. I intended on making it as easy as possible for the player and I feel that I achieved that, all the player has to do is click on the tiles to set them to the type they want. The custom unit creator also looks as I planned, the player can very easily modify a struct that contains the custom squadData and that is then given to the current player and they can then use that to make the unit that the player designed themselves. It again is as easy as possible for the user to interact with. The particle system was a more recent addition but turned out well though it is lacking sprites and is only used for the unit explosion effect though it is very easy to modify and could be used standalone for other projects.

**Project Milestones**

For milestones I tried to follow a week by week feature addition so that as I got closer to the end of the year and other modules had work stacking up that I could give them more time as I had front loaded myself with as much of the work as possible. Whenever there was a lack of other projects for my modules I used that time to work on this project giving me more breathing room for the last month of college. During my weekly project meetings with Martin we would discuss what I had done in the last week and what I planned to have done for the next week. This was massively helpful as I not only was getting instant feedback but I was getting an invaluable insight into my project that I often couldn’t see myself. This ensured that my features functioned well together and always kept me with a target I had in mind for the next week. It also kept me thinking about future additions which was incredibly helpful as it let me plan out the groundworks of features like my units knowing that down the line they would need to be able to be highly customisable so I could make different units from the same class. It kept my classes modular and I always built them with the intention of adding to them in the future. This future proofed code allowed me to rapidly iterate and create new features after a few months of work as now all I had to do for creating new units for example was just specify what texture I wanted and I could also just modify the squadData struct that was already in each unit and I then had multiple unique units that all used identical code.

**Timeline**

**October:**  
I first created my world out of a grid of sf::RectangleShape and then proceeded to make a super basic version of the game. I originally made a load of units from circleshapes and had their strength as a value display on top of them, you could drag and drop them onto each other and they would take damage and the weaker unit would be destroyed. This formed the basic groundwork for the combat system.

**November:**

Converted the game to be turn based and had the player go first.  
Modified the players units so that they will now move to a set point after the player has moved 3 units.  
Prevented the players units from accidentally selecting the tile it is already on.  
Added a basic version of the HUD.  
Added zooming in or out on the map.  
Highlight the current tile player is hovering over after a unit has been selected to be moved.  
Added in the basic version of the GameManager.  
Change update to use deltaTime.  
Highlight if a tile is too far away to move to.  
Implemented end turn button so round no longer immediately ends when the player moves 3 units.  
Added in menu to create new units.  
Added start menu and template for create tower menu.  
Fully implemented first iteration of collisions.  
Implemented end screen if there is only 1 team left alive.

**January:**

Basic version of the world editing added allowing for wall and ground tiles to be added.  
Can now draw water tiles on the map.  
Automatic Texture setting partially implemented.  
Added in large amount of sprites for texture setter.  
Added an outline for units to show what units belong to what team.  
New unit type added.  
Added in squad stat display box.  
Added in cost to buy new units.  
Added ability to sell units at a loss.  
Another new unit added to the game.  
Gold tower added to the game.

**February:**

Added variance to damage so unit can take more or less damage from an attack.  
Collision system completely redone to account for multiple players having multiple units.  
Updated damage system fully implemented.  
Groundwork for formation class added.  
Basic formation added.  
Formation following units can now function alongside non formation units.  
Began implementation of button to reset the players turn back to the beginning.

**March:**

Changes to formation movement so that they function as intended. Formation will move with the leader and has a slow rotation to prevent issues with followers going though walls.  
Added in pathfinding so that formation and units can get the exact path they need to follow when going to their target.  
Modified the viewport to keep the camera within the bounds of the map.  
Updated the unit data readout box to display all stats.  
Changed units to remember recent tile and to move back to it after a fight to prevent clipping.  
Changed collision system again to properly apply damage units in the correct order after an attack has ended.  
Movement is now more consistent and units move from tile to tile.  
Units that are not in a formation will now get a path for themselves to go to their target as they do not need to worry about obstacles.  
Formation will no longer move until end turn has been pressed.  
Fixes to when the player selects multiple units at once to move, wrong paths were being assigned.  
Basic groundwork for unit behaviours added.  
Changes to how invalid tiles are handled and processed allow for consistent collision detection for modified tiles.  
Basics for steering behaviour added.  
Basic reading and writing of invalid tiles to file added.  
Behaviour for follower unit taking leaders path added.

**April:**

Colliders for steering implemented.  
Basic version of steering added.  
Steering fully implemented.  
More fixes to unit steering.  
Logic added for the custom unit.  
Moved all UI interactions to a UI class.  
UI for custom unit upgrade menu added.  
Custom unit data now properly assigned to custom unit.  
Fixed tower placement.  
Research tower added to the game.  
Upgrade menu now works with the research tower.  
Icons for create unit buttons added.  
Icons for create tower buttons added.  
Custom unit fully implemented.  
Seek nearby cell behaviour added for follower units as a default backup state.  
Pathfinding now avoids unit in the world.  
Can no longer create units on top of existing units.  
Collision system completely reworked again to be more consistent and to take into account all units. The new system now uses 2 hitboxes for vertical and horizontal collisions, this was done due to sf::CircleShape’s getGlobalBounds collision check simply being AABB which did not work as intended as I did not want collisions in the corners of cells so a + shaped collider was my solution.  
Highlight units that are added to the formation so player has more visual feedback on the current state of the game.

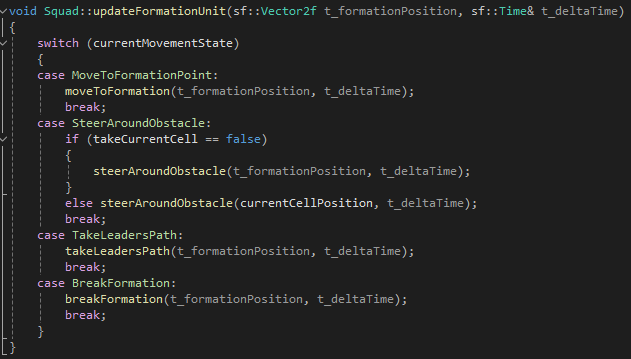
**May:**

Dynamic texture setting of modified cells fully implemented. Code is significantly more streamlined and readable and now works with a spritesheet instead of 20+ separate sprites.  
Break formation follower behaviour added, used to give the seek nearby cell behaviour some variance.  
Particle system added for explosions on unit death.  
Units now face who they have dealt damage to.

**Major Technical Achievements**

**Modular Behaviour Driven Units**

All of the units in the game were designed to be as easy to modify as possible so that while I developed the game I could easily add in new units by changing just the information in the squadData struct. These units are also built specifically to take into account the formation system. The units will take in the position that the formation is telling it to move to and if for any reason they cannot get to this position in the formation they are specifically designed to be able to try several different methods to get to or at least near to the formation position.  
Their behaviours are:

  
MoveToFormation  
This behaviour calls the moveToFormation function and it is the default state that units are set into when they begin moving. This function will have the unit ensure that the point is actually valid and if not it will select a different behaviour. Otherwise it will simply move towards the formation point that was given to it.

SteerAroundObstacle  
This behaviour is responsible for seeing if we can get back to the formation point after losing it. It will take into account the tiles that make up the world and the units that are also in the game world. Since the world is made up of a grid of tiles we actually can make a steering behaviour built to take advantage of the nature of the grid based map. The 3 colliders can be locked to the cardinal directions so that we can avoid various issues that could arise. If the unit were to rotate and the left and right colliders followed the unit could end up with both colliders contacting different walls and locking it in place. This can be completely negated by locking to cardinal directions and works because the world is a grid and there are no unexpected angles or curves we need to conform to. The front collider is used to let the unit stop if it is going into a friendly or if it has hit a wall it can change its behaviour. If we have decided we want the unit to just take the cell directly in front of it we can also call this behaviour but specify it only has to move to the position of the cell directly in front of it.

TakeLeadersPath  
This behaviour will have the unit simply check through the leaders path and begin moving to the nearest position on the path, this should not be far as if we break from the formation we should be near the path anyway. Every time the unit reaches the next cell in the path it again checks if the formation position is valid. This is done to prevent repeated unnecessary calls to the check position valid function.

BreakFormation  
This behaviour will have the unit give up on following the formation altogether and will have it simply seek somewhere nearby to go to, this is only done if the leader has reached its goal and we cannot. If the leader has reached its goal we should not be too far away unless we ran into a lot of obstacles in which case it is just better to stop nearby. This behaviour at first has a chance to just pick the cell in front of it and stay there and otherwise will keep scanning the nearby cells until it picks a valid cell and a path can be generated to it. The unit will then just follow this path and then stay in place.  
  
**Custom Units**  
This system allows the player to create a research lab tower and then proceed to make a fully custom unit. This is done using the basis of the Modular Units and creating a dumb custom unit before then passing in all of the values in a squadData struct that was made specifically for the custom unit. The custom unit combines the UI class and the Tower class to open up the UI for the upgrade menu and then whenever there is a change made to the custom units new data we take the struct for squadData and pass that into the player so that they can then give it to their custom unit. We also specify what texture the player selected from the menu and let the unit use that texture. This allows the player to make whatever type of unit they want in the game and helps show off how modular my units are.  
  
A screenshot of a video game

AI-generated content may be incorrect.

**Formation System**  
Whenever a unit is selected to be moved they are automatically prepped to be the leader of the formation. If another unit is selected the formation is created and it is added to the formation alongside any other units that are selected. The formation gets the path that the leader is to follow and once the end turn button is pressed the formation begins moving from point to point along the path. The leader is technically just a follower and it just moves to the leader point. If the formation has to turn it is set up so that it will slowly rotate to prevent issues with units moving too abruptly. The formation is set up in a way that it can be very easily modified by bringing the entire formation closer or further from each point by changing the x and y spread that is applied to the formation points. This was intended to allow the formation to automatically change its shape and size depending on the obstacles around it and how far or close certain units are to their points. The formation is entirely abstract and is separate from the units meaning that all the formation does is follow its given path and report back the positions that the units need to go to. It is up to the units to find their own way to the points after that. What makes the formation system work so well is actually more down to how the units use their behaviours to get to the point and navigate the world.  
  
**World Editor and Automatic Texture Setter**  
This custom world editor can be accessed from in the game and is used to set the obstacles in the world and makes it extremely for the players and myself to make the map that the game is to take place on. It simply changes the type of the cell that the mouse is clicking on and the game handles the rest automatically. The water tiles are currently saved to a text file though it would be easy to add more tiles assuming the spritesheet for them was completed. The automatic texture setting system is made to be as easy as possible for the player to use. The player just draws on the map and the texture setter does the rest. This is done by first taking in the players input for the type of tile they want to set the tile to, then the tile we are hovering over is got and we set that tile to the specified type, we then clear and re-populate the stored walls data which is just the impassable tiles. Then we call on the game to set the correct texture and specify this is the first time we call this function as it designed to run recursively though this was reduced to just 1 additional call to prevent unnecessary calculations. In updateTileTexture we first get the key for where in the spritesheet we need to set the texture by calling the checkSurroundingTiles function which returns the sprite we need to use. The checkSurroundingTiles function uses a lambda to verify if the tiles in the 8 immediate neighbours are the same as the current tile or not. We then use this information to check what texture needs to be used as we now know where the connections to the neighbouring cells are. We then return the exact texture we need to set after checking the neighbouring cells. Using this info we can finally set the texture in updateTileTexture be just using the key we got for what texture we need. Now we need to call this function again so that we can check the 8 neighbours for changes as they might need to modify themselves since we are also neighbours of them. We have to ensure when we call the function that we specify the depth is 1 so we don’t need to go any deeper and check any more cells though this could be done very easily if needed for whatever reason by simply taking current depth and adding 1 and then making sure we don’t go further than we need so we make sure total depth is below 5 for example.

**Particle System**  
The particle effect system was created entirely for visual feedback from the destruction of a unit but was still built as modularly as possible so that it could not only be used for other particles in my game but could actually included as just a header file in other projects where all they would need to specify would be the spritesheet, size of sprites and how many sprites there are. Its built to be readable and customisable so that anyone can use it for any reason. Currently how it is implemented is the player will create several instances of the class when a unit is destroyed and so long as the size of the containing vector is not 0 I loop through them all and update and draw them accordingly. Most of the variables it uses are slightly randomised such as scale, lifetime and duration so that the particles are not all identical and have a nice visual variety.

**Dynamic UI**All of UI in the game was made to fit the screen by being moved in increments and placed in position using the SCREEN\_WIDTH and SCREEN\_HEIGHT variables in globals. This allows for the screen to be easily resized without all of the UI getting completely broken. The UI is also designed to give the user as much information as I had for the units. While debugging at one point I was printing all the values for the units into the terminal and I decided that would actually be a really good feature for the players to be able to see. I take the player we are currently hovering over and grab their squadData and then pass that into the UI class. Then I update all the strings on the data readout box and render the unit data readout info box. It is automatically positioned on the mouse and is constrained to the map bounds. The box allows the users to see all of the relevant information from the units that they hover over so that they can plan out attacks or strategize the best way to position their units in the world.  
  
**Dynamic World Map Layout**  
The world is made using individual world tiles and are managed by the TileGrid. The map is designed in the same way as the UI being built to fit into the constraints in the globals. The tiles will be automatically created and positioned so that they fill in the screen and so the world size must be a multiple of the TILE\_SIZE to prevent any issues. This class also handles the generation of the paths that the formations and units use to navigate the world. It already knows about all of the invalid tiles but it is also given the positions of the units in the world so that paths can be created that will avoid stationary units. The tiles are designed to be modified on the fly by the world editor. The entire grid based map was designed with a world editor in mind so the cells are easy to swap between types and are also built to be scanned through using the global ROWS and COLUMNS variables so that the paths are not strangely trying to wrap around the map. This is another reason it is so important to keep SCREEN\_HEIGHT and SCREEN\_WIDTH multiples of TILE\_SIZE.  
  
**Collision System**  
The collision system went through several iterations, my game and how I handled my units and players changed quite extensively and so the collisions had to change alongside them. A significant amount of time was spent getting them to the state they are now currently in. My most recent issue was SFMLs built in getGlobalBounds which simply returns the smallest possible square that the shape would fit in. This meant using circles was out of the question and rotating a rectangle 45 degrees did not help either as the hitbox returned does not account for rotations and so the hitbox actually ended up being significantly bigger than before. The core issue was that I did not want my units fighting anything other than units either NESW of their position and so my final solution was to use 2 rectangle hitboxes in a + shape. This allowed me to get exactly the kind of collision I want and also opens the door for units that have longer range if needed.  
A video game of a tank

AI-generated content may be incorrect.

**Project Review**

I feel like I hit most of the goals I set out for myself at the start of the project, some features did get scrapped early on but typically when I decided on a feature I stuck with it until it was done. The weekly meetings served as a really helpful goal for a time to have things done and helped me from going overboard with features early on and I was able to set a really good foundation for the rest of my project to be built upon.

I had collision issues several times throughout the project and the main issue was simply that I changed how my units were handled by the player in slight ways and so it threw off the way that I had been calculating my collisions.

A lot of features that I had put into the backlog did not make it in but they were more of a wishlist than expected features. If there was a point that I wanted to add something in I would pick it out of this list. That is how I ended up with the automatic texture setting being implemented. The formations still needed a bit of work with being automatically rescaled and having units automatically take the next position in the formation if there was a unit missing. I would also have like to have given the units more behaviours to make them more varied. I also did not implement the tower units that would have acted as a HQ building and others to act as turrets. The tilemap was also originally going to have mud to slow enemies down and to provide a heuristic for A\* implementation. The only feature that was removed was the reset turn button as I was trying to add it while I was doing formations and they kept breaking each other so I dropped it but I would have liked to have had that implemented.

If was starting again I don’t know if I would actually change much, I feel like I paced myself well and the weekly meetings with Martin kept me on track. I think ensuring that your code is future proofed in such a way that if you next week had to add onto it that its left in a really good state to do so. Also don’t be afraid to completely delete chunks of code and redo them again like I did with my collisions. Each iteration of my collisions was better than the last and I have left them in a state that is super easy to add to or modify should I ever had needed to.

I do think I did make the correct technology choices as using C++ and SFML while slower to develop it meant that I extensively know how my code works and I got very used to maintaining large amounts of code and ensuring that any code I did add in did not change or break any existing code.

**Conclusions**

This project turned out mostly as I had envisioned and I’m very happy with the progress that I had achieved over the last few months. I stuck to my schedule of week by week progress which kept me relatively stress free throughout the year and made sure that when it came to crunch at the end of the year I was able to focus on other modules as I had the bulk of my work on the project already done. Since I also built the game on a solid foundation it meant that the additions near the end were way easier on myself to implement.

**Future Work**

I think that exploring an in-depth AI opponent would have been a really cool and interesting addition to the game but would need to be explored entirely separately as it would need to be very complex for it to actually be interesting to play against.  
Combinations of formations in formations would have also been a really interesting next step though the map would have to be significantly expanded. The way units interact with each other would have to be completely redesigned in order to achieve this.

**References**

https://en.wikipedia.org/wiki/Advance\_Wars